Informàtica per a físics i enginyers

Sònia Estradé, José M. Gómez, Ricardo Graciani, Manuel López, Xavier Luri

UNIVERSITAT DE BARCELONA

Índex

Deposit Legal B.XXXX-2019

Capítol 1

L'ordinador

1.1 Introducció

Informàtica és una paraula d'origen francès formada per la contracció dels vocables *informació* i *automàtica*. El diccionari defineix la informàtica com el conjunt de coneixements científics i tècniques que fan possible el tractament automàtic de la informació per mitjà d'ordinadors.

Seguint amb les definicions, podem dir que *computador* o *ordinador* és una màquina capaç d'acceptar unes dades d'entrada, efectuar amb elles operacions lògiques i aritmètiques, i proporcionar la informació resultant a través d'un mitjà de sortida, tot això sense la intervenció d'un operador humà i sota el control d'una sèrie d'instruccions, anomenades *programa*, prèviament emmagatzemat a l'ordinador.

El primer ordinador va ser dissenyat al segle XIX per Charles Babbage amb el propòsit d'automatitzar els càlculs de taules de logaritmes, encara que mai va arribar a funcionar a la pràctica. No obstant això, la primera computadora electrònica de propòsit general va ser l'ENIAC (*Electronic Numerical Integrator and Computer*), presentada el 1946. Va ser desenvolupada i construïda per l'exèrcit americà per al seu laboratori d'investigació balística. Tenia com a objectiu calcular taules balístiques de projectils. Va ser concebuda i dissenyada per J. Presper Eckert i John William Mauchly de la Universitat de Pennsilvània. Ocupava una superfície de 167 m² i operava amb un total de 17.468 vàlvules electròniques o tubs de buit. Físicament, l'ENIAC tenia 17.468 tubs de buit, 7.200 díodes de vidre, 1.500 relés, 70.000 resistències, 10.000 condensadors i 5 milions de soldadures. Pesava 27 tones, mesurava 2,4 m x 0,9 m x 30 m; utilitzava 1.500 commutadors electromagnètics i relés; requeria l'operació manual d'uns 6.000 interruptors, i el seu programa o programari, quan requeria modificacions, trigava setmanes a instal · lar-se manualment. La seva potència de càlcul li permetia fer unes 5.000 sumes o restes per segon. L'ENIAC va ser programada fonamentalment per Kay McNulty, Betty Jennings, Betty Snyder, Marlyn Wescoff, Fran Bilas i Ruth Lichterman. Totes elles van ser les encarregades de crear les primeres biblioteques de rutines, i d'elaborar els primers programes per a l'ENIAC. Podríem dir que van ser les primeres informàtiques.

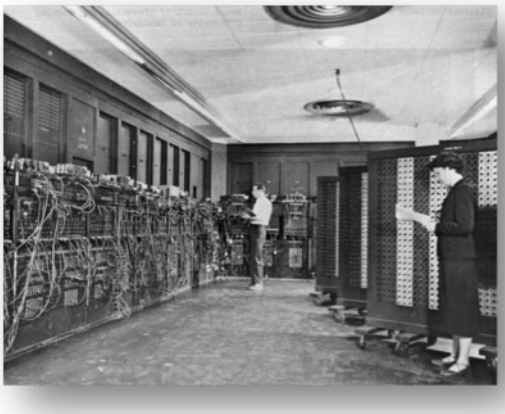


Figura 1.1: Ordinador ENIAC.

Una de les característiques de l'ENIAC era el seu gran consum, uns 160 kW. Arran d'això corria el mite que la ciutat de Filadèlfia, on es trobava instal · lada, patia apagades quan l'ENIAC entrava en funcionament. Aquest consum elevava la temperatura del local a 50 ºC. Per efectuar les diferents operacions calia canviar, connectar i reconnectar els cables com es feia, en aquesta època, en les centrals telefòniques. Aquest treball podia demorar-se diversos dies depenent del càlcul que s'havia de fer, sense oblidar que la vida mitjana entre fallades era d'una hora. L'ENIAC va ser el pioner dels actuals ordinadors, i va permetre obrir un camp que encara està

evolucionant, tot i que molts dels aspectes originals de l'ENIAC es mantenen.

Un aspecte molt important dels ordinadors, des de l'ENIAC fins als actuals, és que utilitzen codi binari per emmagatzemar i processar la informació. Els nombres binaris, com el nom indica, estan representats en la base de dos valors, 0 i 1, i la unitat mínima d'informació és el bit (*binary digit*). El motiu es deu al fet d'establir els estats totalment obert o totalment tancat que és més senzill de definir que els estats intermedis.

La situació vindria a ser semblant a quan un nen compta amb els dits. Un dit aixecat indica un 1, dos dits aixecats indica un 2 i així successivament (figura 1.2):

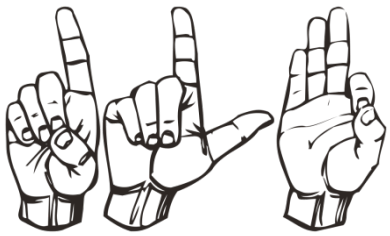


Figura 1.2: Signes de les xifres 1, 2 i 3 amb els dits.

Ara bé, quan intentem mostrar un quatre, si ho fem amagant el polze, cap problema, però si ho fem amb el dit petit, és molt probable que se'ns quedi a mitges l'anular, i en aquest cas, és un tres o un quatre? Perquè el dit anular es pugui estendre completament, necessitem ajudar el petit amb el polze, fet que ens torna a mostrar un tres (figura 1.3).

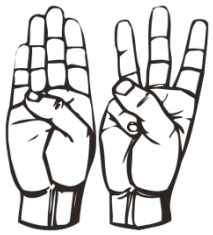


Figura 1.3: Signe de la xifra 4 i de 3 amb els dits.

De la mateixa manera, quan treballem amb un ordinador és millor treballar amb encès/apagat (estès/doblegat). Si treballem amb estats intermedis, ens pot passar com quan s'intenta doblegar el dit petit per mostrar un quatre, ja que l'anular quedarà en un estat que no sabrem quin número està mostrant. En el cas d'un ordinador, els dits serien transistors, i estès/doblegat passaria a ser obert/tancat (una possibilitat seria totalment obert, que és un 1, i totalment tancat, un 0).

Per aquest motiu, qualsevol informació que vulguem introduir a l'ordinador ha de ser traduïda amb anterioritat a codi binari (digitalitzada) perquè pugui ser guardada o processada. En els temps de l'ENIAC això era relativament senzill, ja que s'utilitzaven interruptors l'estat dels quals representava directament un valor d'un dígit binari. Això tenia un petit inconvenient: les programadores no podien cometre errors, ja que identificar en un panell d'interruptors quin d'ells estava malament podia ser una tasca molt tediosa. Avui en dia, la situació ha millorat notablement i podem arribar fins i tot a parlar a l'ordinador perquè faci operacions bàsiques.

Com que el llenguatge natural dels ordinadors és el codi binari, és important familiaritzar-se amb alguns termes propis d'aquest codi. Com hem dit, un bit és un dígit en codi binari i es representa com a unitat amb una *b* minúscula. Un byte són 8 bits i es representa amb una *B* majúscula. Per referir-se a un major nombre de bits se solen utilitzar els prefixos habituals definits pel Sistema Internacional de Mesures (quilo-, mega-, giga-, tera-, etc.) encara que s'ha de tenir en

compte que, per motius històrics (veges l'enllaç per a una discussió en detall del problema), hi ha una certa confusió en l'ús i coexisteixen (almenys) dues interpretacions d'aquests prefixos, com es recull a la taula 1.1. Per aquest motiu la Comissió Internacional d'Electrotècnia (IEC) ha proposat una alternativa.

Taula 1.1: Prefixos establerts pel SI i la IEC

Prefix	Símbol	Decimal (SI)	Prefix	Símbol	Binari (IEC)
kilo-	k-	10^3	**kibi-**	Ki-	1024^1
mega-	M-	10^6	**mebi-**	Mi-	1024^2
giga-	G-	10^9	**gibi-**	Gi-	1024^3
tera-	T-	10^{12}	**tebi-**	Ti-	1024^4
peta-	P-	10^{15}	**pebi-**	Pi-	1024^5
exa-	E-	10^{18}	**exbi-**	Ei-	1024^6
zetta-	Z-	10^{21}	**zebi-**	Zi-	1024^7
yotta-	Y-	10^{24}	**yobi-**	Yi-	1024^8

És important remarcar la diferència entre el factor decimal i el binari, ja que aquest últim és sempre superior al primer. Quan treballem amb un ordinador, normalment es treballa amb factors binaris. No obstant això, en comprar certs productes (com en el cas dels discos durs) és possible que ens indiquin la capacitat en factor decimal, i per tant l'ordinador indiqui una mida inferior de la que apareix a la capsa del producte.

1.2 Components d'un ordinador

L'estructura bàsica dels ordinadors actuals es basa en l'arquitectura proposada per John von Neumann el 1945. Aquesta arquitectura té tres components bàsics que interactuen entre ells. Aquests components són una unitat de processament, una memòria on s'emmagatzemen tant instruccions com dades i els dispositius d'entrada/sortida. Si ens fixem en una calculadora, les dades serien els números que introduïm, i les instruccions, les operacions que s'han de fer amb aquestes dades. La figura 1.4 mostra un diagrama una mica més detallat de l'estructura dels ordinadors actuals. Estan formats pels elements o unitats funcionals següents: unitats d'entrada, unitats de sortida, memòria externa, memòria interna, unitat aritmeticològica i unitat de control.

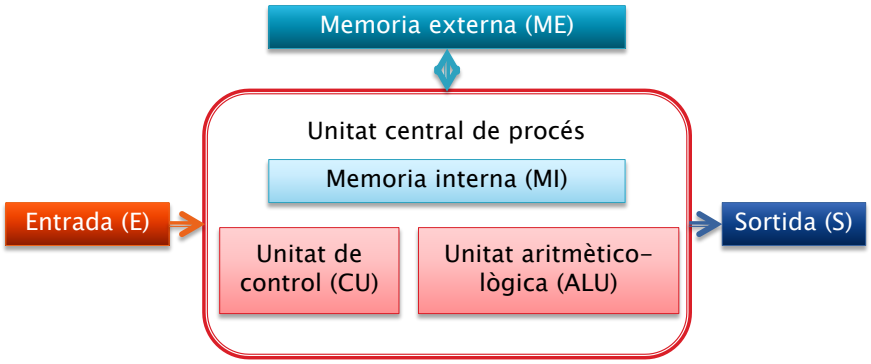


Figura 1.4: Estructura funcional d'un ordinador.

Cadascun d'aquests elements acostuma a equivaler a un xip físic o un dispositiu. Així, la unitat central de procés és el xip central d'un ordinador, mentre que l'entrada, la sortida i la memòria externa solen ser dispositius que es poden afegir o treure a voluntat.

- Unitat central de procés (CPU, *central processing unit*). És el centre neuràlgic de l'ordinador i gestiona tots els elements. Es compon de tres elements bàsics:

  - Unitat de control (CU, *control unit*). Forma part de la CPU i s'encarrega d'interpretar les instruccions dels programes i generar els senyals de control dirigits a totes les unitats de l'ordinador per executar la instrucció. La unitat de control també rep senyals dels altres dispositius per comunicar-li el seu estat (disponibles, ocupats, etc.). Aquests senyals es coneixen com a *senyals d'estat*. És la responsable que l'ordinador funcioni.

  - Unitat aritmeticològica (ALU, *arithmetic-logic unit*). Duu a terme operacions aritmètiques (sumes, restes, etc.) i operacions lògiques (comparació, AND, OR, XOR). Aquest mòdul rep les dades per ser processades.

  - Memòria interna (MI). Inicialment guarda la informació bàsica per poder fer les operacions, en el que es denominen *registres*. Serien equivalents a les memòries d'una calculadora. Amb el temps, i en augmentar la capacitat d'integració, ha aparegut la memòria cau (*cache memory*), que pot arribar a emmagatzemar megabytes d'informació per accelerar el treball de la CPU. En tots dos casos, la memòria és volàtil, també coneguda com a *memòria d'accés aleatori* (RAM, *random access memory*), i per tant en apagar l'ordinador es perd la informació. Utilitzen un tipus de memòria que s'anomena *estàtica* (SRAM, *static RAM*) i que requereix 6 transistors per guardar cada unitat d'informació.

- Memòria externa (ME). La memòria interna és molt ràpida (desenes de milions de paraules transmeses per segon), però no té gran capacitat per emmagatzemar dades. Per desar una quantitat d'informació més gran s'utilitzen dos tipus de memòries més:

  - Memòria principal. És la memòria on es guarda la informació que utilitza l'ordinador durant l'execució de processos i en l'actualitat sol tenir mides al voltant del gigabyte. La característica principal és que utilitza memòria dinàmica (DRAM, *dynamic RAM*), la qual té l'avantatge d'ocupar menys espai que la seva homòloga estàtica, ja que només necessita un transistor per a cada bit, però, en canvi, requereix anar-la refrescant de tant en tant perquè no perdi les dades.

  - Dispositius d'emmagatzematge. Són memòries basades en altres tecnologies com discos magnètics, discos òptics, cintes magnètiques, memòria flash, etc., que encara que són més lents permeten emmagatzemar més informació (entorn d'un milió de vegades més lents però amb cent o mil vegades més capacitat).

- Unitat d'entrada (E). És un dispositiu pel qual s'introdueixen a l'ordinador les dades i instruccions. En aquestes unitats es transforma la informació d'entrada en senyals binaris de naturalesa elèctrica. Un mateix ordinador pot tenir diverses unitats d'entrada: teclat, ratolí, escàner d'imatges, lector de targetes.

- Unitat de sortida (S). Permet obtenir els resultats dels programes executats a l'ordinador. La major part d'aquestes unitats transformen els senyals binaris en senyals perceptibles per l'usuari. Els dispositius de sortida inclouen pantalles, impressores i altaveus.

Exercici 1.2.1 *Característiques d'un ordinador*
En el quadre inferior pots veure les característiques d'un ordinador de sobretaula. Digues a quin tipus d'unitat funcional correspon cadascuna d'aquestes característiques.

Avui en dia, les CPU solen incloure diversos nuclis, fet que significa que dins d'un mateix xip hi ha diverses CPU interconnectades, tot i que, des del punt de vista extern, és com si només n'hi hagués una. Això permet executar diversos programes en paral · lel, la qual cosa incrementa les prestacions de l'ordinador.

1.3 Sistemes operatius

El sistema operatiu és el programa principal de l'ordinador i té per objectiu facilitar el seu ús i aconseguir que s'utilitzi eficientment. És un programa de control que s'encarrega de gestionar, assignar i accedir als recursos del maquinari que requereixen els altres programes. Aquests recursos de maquinari són: el processador, la memòria principal, els discos i els perifèrics. Si diversos usuaris utilitzen el mateix ordinador al mateix temps, el sistema operatiu s'encarrega d'assignar recursos, i evitar els conflictes que poden sorgir quan dos programes volen accedir als mateixos recursos simultàniament (la unitat de disc o la impressora, per exemple). Diem que el sistema operatiu fa transparent a l'usuari les característiques del maquinari sobre el qual treballa.

El sistema operatiu també permet que l'ordinador s'utilitzi eficientment. Així, per exemple, un sistema operatiu que permeti multiprogramació s'encarrega d'assignar els temps d'execució de dos programes que estan actius de manera que durant els temps d'espera d'un, l'altre programa s'estigui executant. I fins i tot si correm sobre una plataforma multiprocessador (amb diverses CPU i diversos nuclis) pot assignar a cada programa un dels processadors.

Les principals funcions, encara que no les úniques, del sistema operatiu són la gestió dels recursos del computador:

- Processador
- Memòria principal
- Dispositius
- Sistema d'arxius

A continuació passem a descriure-les.

1.3.1 Gestió del processador

La gestió i administració del processador consisteix a determinar quina és la utilització que es va a fer d'ell en cada moment. Això dependrà de quins programes hagin d'executar, de la disponibilitat de dades per executar i de les peticions prioritàries d'execució (interrupcions) que rebi el sistema. La gestió del processador per part del sistema operatiu se centra en el concepte de *procés*. Un procés és un programa l'execució del qual s'ha iniciat, i que està format no només pel codi (instruccions) sinó també pel seu estat d'execució (valors de registre de CPU) i la seva memòria de treball. Un programa només és un ens passiu mentre que un procés és un ens actiu. Per tant, el sistema operatiu s'encarrega de planificar el temps del processador determinant quin procés s'executa en cada instant.

Els primers sistemes operatius només permetien un procés actiu alhora, de manera que fins que el procés no s'acabava d'executar completament no es començava el següent. Això es coneix com a *monoprogramació*. L'ordre de l'execució dels processos es determinava per l'ordre d'arribada de les peticions d'execució i per la prioritat dels processos. Els processos de més prioritat s'executaven abans. La monoprogramació té l'inconvenient de desaprofitar temps de processador, ja que els processos estan esperant que les operacions d'entrada/sortida (lectura/escriptura en memòria, disc) s'executin. Aquest problema es va solucionar amb la multiprogramació.

Els sistemes operatius més recents permeten tenir diversos processos actius a la vegada (multiprogramació) de manera que l'execució dels processos es va intercalant en el processador sense necessitat que acabi completament un procés abans de donar pas al següent. Això permet que durant els temps d'espera d'un procés es canviï de procés en execució i no es malgasti temps de processador. La multiprogramació permet donar la sensació a l'usuari que diversos programes s'estan executant simultàniament encara que l'ordinador només tingui un processador, amb un únic nucli, i per tant només es pot estar executant un procés alhora.

1.3.2 Gestió de la memòria

La memòria principal de l'ordinador s'utilitza per guardar les instruccions i dades dels programes que s'han d'executar. El contingut d'aquesta memòria varia a mesura que els programes que hi ha en execució acaben, i alliberen l'espai de memòria que ocupen, i nous programes es carreguen en memòria per ser executats. La gestió de la memòria consisteix a fer un seguiment de l'ocupació de les diferents posicions i el gens trivial problema de determinar en quines posicions de memòria es carreguen els programes nous sense reduir l'eficiència dels accessos a memòria.

Amb aquesta finalitat, se solen utilitzar diferents estratègies per a l'assignació de memòria principal: particions estàtiques, particions dinàmiques, segmentació. A les particions estàtiques es divideix la memòria en particions de mides variables i s'assigna un procés a cadascuna d'elles. Les particions no es canvien i per això es diu *segmentació estàtica*. Com que és molt difícil que els programes tinguin exactament la mateixa longitud que alguna de les particions de memòria i se'ls assigna una partició de mida més gran al programa, la part buida de la partició queda inutilitzada. Aquest és el principal inconvenient d'aquest mètode. La partició dinàmica evita aquest problema ja que assigna exactament la mateixa quantitat de memòria que cada programa requereix. Es genera una partició de la mida del programa que cal carregar en memòria. No obstant això, aquest mètode de gestió de memòria té l'inconvenient de la fragmentació de la memòria. S'omple la memòria amb diferents processos. En un moment determinat es buida espai de dos processos no contigus. Quan un procés nou vol entrar, si aquest és més gran que qualsevol dels processos antics, no podrà entrar, encara que realment hi ha memòria suficient en el sistema. El mètode de segmentació és semblant al de partició dinàmica només que el programa en lloc de guardar-se en un sol bloc se separa en els segments de dades, codi i pila, que es guarden en particions separades.

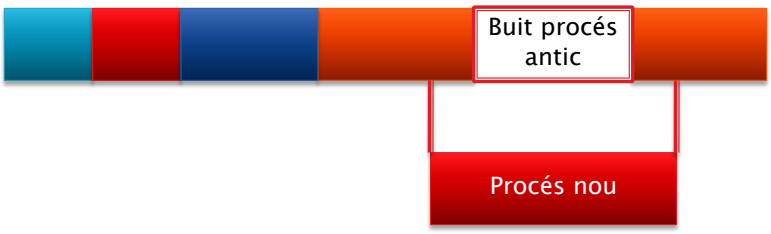


Figura 1.5: Fragmentació de la memòria.

El mètode de paginació es basa a dividir els programes en segments de longitud fixa anomenats *pàgines* i dividir la memòria en marcs de pàgina de la mateixa mida, de manera que cada pàgina es podrà posicionar en un marc de pàgina. La mida de les pàgines sol ser petita (512 bytes) de manera que l'espai inutilitzat que pot quedar per posicionar l'últim segment del programa és petit. A més, la fragmentació de la memòria evita tenir particions fixes. Com a contrapartida, aquest mètode exigeix una càrrega elevada d'activitat de processador per gestionar les pàgines, els marcs de pàgina i l'assignació de pàgines a marcs. Aquest és un dels mètodes més utilitzats pels sistemes operatius, per exemple, les últimes versions de Windows o Unix.

1.3.3 Controladors

Tal com hem dit, el sistema operatiu ha de fer al més transparent possible el maquinari de la màquina on treballa al programa. D'aquesta manera s'aconsegueix que un mateix programa funcioni amb dispositius completament diferents, i doni resultats equivalents. Alguns exemples d'aquesta situació van dels teclats i ratolins a les pantalles i targetes gràfiques, o les impressores.

Per aconseguir aquest objectiu, el sistema operatiu defineix un dispositiu virtual amb unes característiques genèriques. Evidentment, no tots els dispositius compleixen de la mateixa manera aquestes característiques, motiu pel qual és necessari un traductor. Aquest traductor s'anomena *controlador*.

El controlador duu a terme tres tasques bàsiques:

- Gestiona el dispositiu, l'inicialitza quan és necessari i mira l'estat en què es troba.
- Indica quines característiques concretes del dispositiu virtual compleix el dispositiu real.
- Tradueix la informació del dispositiu real perquè compleixi les especificacions del dispositiu virtual.

Amb tot això s'aconsegueix que un programa sempre vegi de la mateixa manera un dispositiu, la qual cosa simplifica molt la tasca de la persona que desenvolupa. Un exemple seria escriure en un fitxer, que independentment del suport que estiguem utilitzant (un disc dur intern, una memòria USB, un disc dur extern...) sempre segueix els mateixos passos.

Un altre exemple serien les impressores. Totes solen treballar en un pseudollenguatge de composició de pàgines que és diferent per a cada fabricant. Per solucionar-ho, el sistema operatiu sol utilitzar una impressora genèrica amb un llenguatge de paginació. Quan un programa vol enviar alguna cosa a imprimir, utilitza aquest llenguatge, i posteriorment, és el controlador el que fa la traducció, tal com es mostra a la figura 1.6.

Finalment, tenim el cas de les targetes gràfiques. Avui dia se solen utilitzar dos pseudollenguatges per gestionar-les, l'OpenGL i les Direct X. En aquest cas, la mateixa targeta sol incorporar el traductor d'aquest llenguatge, i el controlador es limita a transferir la informació del sistema operatiu al dispositiu. Com veiem, les opcions són múltiples, però el resultat final és sempre el

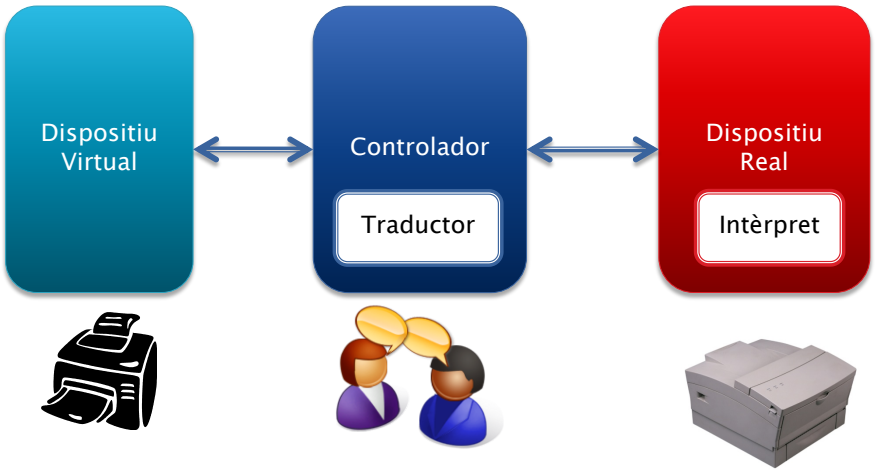


Figura 1.6: Diagrama de blocs d'un controlador d'impressora.

mateix: el programa deixa de ser dependent dels perifèrics, la qual cosa permet al programador dedicar-se a optimitzar-ne el funcionament.

1.3.4 El sistema de fitxers

Les dades que es troben a la memòria externa solen organitzar-se en arxius, també coneguts com a *fitxers*. El concepte d'*arxiu* possibilita aïllar l'usuari dels problemes físics d'emmagatzematge específics de cada unitat de memòria externa. Cada arxiu té assignat un nom, atributs, descriptor de seguretat (qui pot accedir a l'arxiu i amb quins privilegis) i adreça (on es troben les dades). Els atributs inclouen informació com ara la data i l'hora de creació, la data i l'hora de l'última actualització, els bits de protecció (només lectura, o lectura i escriptura), la contrasenya d'accés, el nombre de bytes per registre o la capacitat de l'arxiu.

Per accedir al sistema d'arxius i fer operacions com crear, llegir o escriure, el sistema operatiu té una sèrie de crides al sistema com les recollides a la taula 1.2.

Taula 1.2: Crides al sistema per a l'accés a arxius

Crida al sistema	**UNIX**	**Windows XP**
Crea un arxiu	Open	CreateFile
Esborra l'arxiu	Unlink	DeleteFile
Tanca l'arxiu	Close	CloseHandle
Llegeix les dades de l'arxiu	Read	ReadFile
Escriu les dades a l'arxiu	Write	WriteFile
Obté les propietats de l'arxiu	Stat	GetFileAtributes

La segona abstracció que utilitza el sistema operatiu per gestionar volums de dades és la de directori o carpeta. Els directoris o carpetes són conjunts d'arxius agrupats. L'estructura global del sistema d'arxius sol tenir forma d'arbre, en el qual els nodes interiors són els directoris i els nodes exteriors són arxius. A la taula 1.3 s'inclouen les crides al sistema més comunes relacionades amb la gestió de directoris o carpetes.

Taula 1.3: Crides al sistema per a l'accés a directoris

Crida al sistema	**UNIX**	**Windows XP**
Crea un directori	Mkdir	CreateDirectory
Esborra el directori buit	Rmdir	RemoveDirectory
Obre el directori per a lectura	Opendir	FindFirstFile
Llegeix l'element següent del directori	Readdir	FindNextFile
Elimina un arxiu d'un directori	Unlink	
Mou un arxiu d'un directori a un altre		MoveFile

1.3.5 Sistemes operatius de Microsoft

Són els sistemes operatius dominants en el mercat de microcomputadors. El primer d'ells va ser el Microsoft Disk Operating System i deu la seva difusió al fet que va ser adoptat per IBM al principi de la dècada dels 80 com el sistema operatiu estàndard per a l'IBM-PC. A la figura 7 es mostra un esquema de l'evolució **MS-DOS** d'aquest sistema operatiu. L'MS-DOS inicial era un sistema operatiu per a microprocessadors de 16 bits (d'Intel), monousuari i tenia una interfície d'usuari de línia de comandes. A inicis dels 90 es va comercialitzar el Microsoft Windows 3.0 que disposava d'una interfície gràfica d'usuari (*graphic user interface*, GUI) que permetia mostrar finestres amb gestió de menús, i era capaç de carregar en memòria més d'un programa alhora. El 1995 Microsoft va comercialitzar el Windows 95, que conté una GUI basada en icones, i és un sistema operatiu de 16/32 bits amb multiprogramació apropiativa (pot suspendre temporalment l'execució d'un treball per executar-ne un altre) i amb memòria virtual. Posteriorment es van comercialitzar les versions Windows 98, Windows ME (Millennium Edition) el 1998 i 2000, respectivament. Paral · lelament al desenvolupament d'aquestes versions de Windows, Microsoft va comercialitzar el Windows NT (Windows New Technology), dissenyat fonamentalment per a estacions de treball potents i servidors de xarxa amb processadors de 32 bits. El 2001 es disposava de tres alternatives: Windows 2000 professional, Windows NT Server i Windows CE (Consumer Electronics). L'any 2001 es va comercialitzar el Windows XP, que va suposar la unificació de la línia clàssica dels sistemes operatius de Microsoft (Windows 3.1 / 95/98 / 98SE / ME) amb la de **Windows NT** (NT 3.1, 4.1 i Windows 2000) (figura 1.7). El Windows XP conté el nucli del Windows NT i es va comercialitzar en dues versions: Home i Professional.

Posteriorment va aparèixer el Windows 2003, destinat a servidors de bases de dades, aplicacions, Internet, etc. El nucli era fonamentalment el mateix que el de l'XP, però optimitzat per dur a terme les tasques anteriorment esmentades, la qual cosa implica una alta taxa de transferències d'informació per xarxa, seguretat, multiusuari, etc. Més recentment va arribar el Windows Vista, que va tenir molts problemes inicials, a causa fonamentalment d'algunes incompatibilitats amb el Windows XP; del nou sistema de seguretat, que era bastant molest, i de la falta de controladors per a alguns dispositius. Això va portar moltes empreses a retrotraure al Windows XP els seus nous equips. L'últim sistema operatiu de Microsoft és el Windows 10, que s'espera que doni resposta a tots els problemes trobats en el Vista, Windows 7 o Windows 8.

1.3.6 Sistemes operatius UNIX

El sistema operatiu UNIX inicialment es va desenvolupar als Bell Labs de la companyia AT & T el 1970 i va ser comercialitzat en els inicis per Honeywell. A principis de la dècada dels 90 Novell en passa a ser el propietari. A la figura 8 s'inclou un esquema simplificat de l'evolució d'aquest sistema operatiu. Una de les versions més conegudes és la desenvolupada al Campus de Berkeley de la Universitat de Califòrnia.

Des d'un punt de vista tècnic, UNIX pot considerar-se com un conjunt de famílies de sistemes

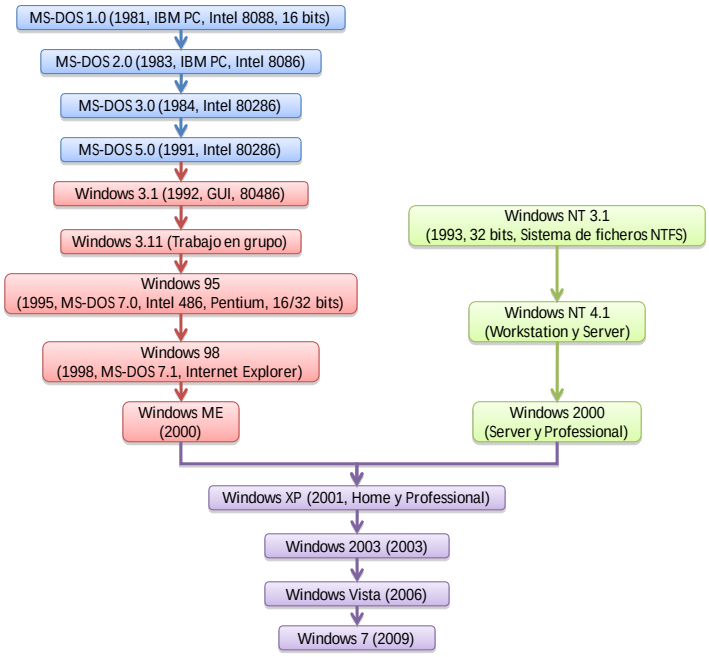


Figura 1.7: Evolució dels sistemes operatius de Microsoft.

operatius que comparteixen certs criteris de disseny i interoperabilitat. Aquesta família inclou més de 100 sistemes operatius desenvolupats al llarg de 35 anys. Poden funcionar en multitud de computadors, des d'un supercomputador com el Mare Nostrum, passant per un PC de sobretaula, i arribant a una agenda electrònica o un telèfon mòbil d'última generació. A causa d'això pot considerar-se com un autèntic sistema operatiu estàndard i la columna vertebral d'Internet. Permet multiprogramació, multiusuari i multiprocessament, i es pot utilitzar també en entorns de temps real.

Un estudiant d'informàtica de la Universitat d'Hèlsinki (Finlàndia) anomenat Linus Torvalds va concloure el 1991 (amb 23 anys d'edat) un sistema operatiu que va denominar Linux, versió 0.01. Aquest sistema operatiu va ser concebut com una versió per a PC compatible i substancialment millorada del sistema operatiu Minix (molt semblant a l'UNIX), descrit pel professor Tanenbaum en els seus primers textos de sistemes operatius. Encara que Linux no és un programa de domini públic, es distribueix amb una llicència GPL (*general public license*) del GNU, de manera que els autors no han renunciat als seus drets, però l'obtenció de la seva llicència és gratuïta, i qualsevol pot disposar de tots els programes font, modificar-los i desenvolupar noves aplicacions basades en Linux.

El 1999 Apple va presentar un sistema operatiu basat en BSD, amb un nucli Mach. Aquest es distribueix sota llicència de codi obert com Darwin. Per sobre corre una interfície gràfica molt elaborada, que és el que el diferencia fonamentalment de la resta d'UNIX. En la resta d'aspectes, Darwin és un UNIX més i comparteix un gran nombre d'aplicacions.

Finalment, convé indicar que en l'actualitat, les versions d'UNIX més esteses són Mac OS X, Linux i Solaris.

1.4 Perifèrics

Quan parlem de perifèrics, solem referir-nos als que ens permeten interactuar amb l'ordinador:

- Pantalla

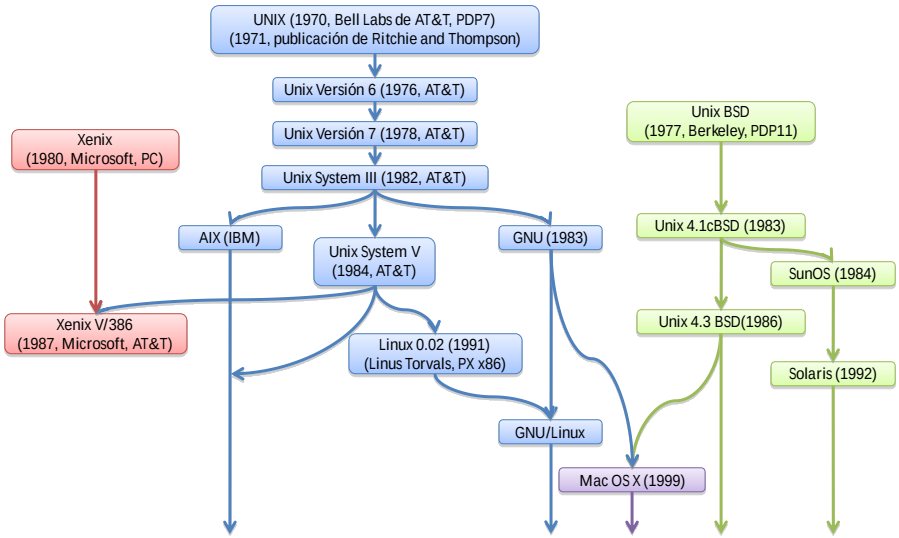


Figura 1.8: Evolució dels sistemes operatius UNIX.

- Teclat
- Ratolí

No obstant això, hi ha tot un conjunt de dispositius que solen estar inclosos dins de l'ordinador, i que generalment donem per descomptat que funcionen. En aquesta secció veurem les seves principals característiques per entendre'n el funcionament. Ens centrarem en dos perifèrics bàsics, com són el disc magnètic i el disc òptic. També estudiarem els principals busos de connexió cablejats de perifèrics (USB i Firewire) així com sense fil (Bluetooth).

1.4.1 Discos magnètics

El disc dur és el mitjà més utilitzat per a l'emmagatzematge de grans volums d'informació, atès el baix cost, alta velocitat i gran capacitat. Aquests discos utilitzen un apilament de plats de discos magnètics muntats en estructures rígides i aïllats de la possible contaminació ambiental i atmosfèrica. Aquests discos també són coneguts com a *tecnologia Winchester*, a causa del nom que va donar IBM al primer model que tenia els discos i els capçals muntats en una estructura rígida per donar-li més velocitat, capacitat i fiabilitat. Aquest primer disc tenia una capacitat de 30 MB. En l'actualitat un disc dur pot superar els terabytes (unes 105 vegades la grandària del primer).

Tots els sistemes basats en discos guarden la informació en superfícies circulars dividides en pistes concèntriques, com es mostra a la figura 1.9. Cada pista es divideix al seu torn en sectors. Per accedir a un sector determinat, el cap lector/gravadora pivota sobre els discos en rotació fins a trobar la pista correcta. Quan s'hi troba a sobre, espera fins que el disc rota fins al sector correcte. En aquest punt pot començar a llegir l'orientació magnètica dels punts del sector, o bé grava una nova orientació, en funció de la necessitat d'aquest moment.

El propòsit de muntar tot el sistema electromecànic en un contenidor hermètic és evitar la contaminació de partícules en els plats. D'aquesta forma els caps lectors poden volar a una distància mínima respecte del disc sense possibilitat de xocar contra cap objecte. Aquesta menor alçada permet llegir i escriure informació amb menors dimensions, alhora que permet un funcionament més ràpid, en poder passar més bits en una unitat de temps.

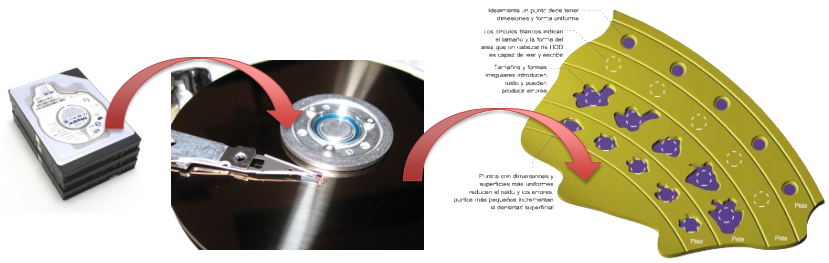


Figura 1.9: Del disc magnètic als punts al plat.

1.4.2 Discos òptics

Un altre possible mètode d'emmagatzematge és utilitzar dispositius òptics. En aquest cas, en lloc de modificar propietats magnètiques d'un material, es modifica la forma en què interacciona amb la llum. Aquest és el cas de tecnologies com el CD o el DVD. Hi ha diversos tipus d'emmagatzematge en aquests dispositius. La primera són els CD-ROM o DVD-ROM. En aquest cas, la informació es guarda en forma de microsolcs en un disc transparent. Els canvis d'alçada, obtinguts per zones amb forats (*pits*) i zones sense, permeten indicar la informació guardada. Per crear aquests microsolcs s'utilitza un mètode d'estampació sobre un substrat de policarbonat (PC).

Per aconseguir discos amb la superfície escrivible s'utilitzen materials amb canvi de fase. En la fase cristal · lina són perfectes reflectors de la llum, i en la fase amorfa són mals reflectors. Aquest material s'introdueix com una capa de tint en el disc, tal com es mostra a la figura 1.10. Aquest canvi de fase és detectat per un làser, i la diferència és llegida com un 1 o un 0.

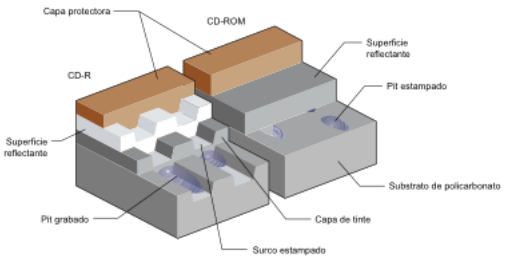


Figura 1.10: Capes d'un CD-ROM i un CD-R.

Els CD gravables (CD-R), els DVD-R i els DVD + R utilitzen un material que solament pot ser modificat un cop d'estat cristal · lí a estat amorf. En el cas dels discos regravables (CD-RW, DVD-RW i DWD + RW) s'utilitzen materials que poden modificar el seu estat milers de vegades (els defensors dels sistemes magnètics diuen que unes 100.000 vegades). En general, les tecnologies del CD i del DVD per a la seva lectura i escriptura són molt semblants, tot i que el DVD aprofita els anys d'experiència del CD i inclou algunes millores respecte a aquest últim. En primer lloc, el DVD utilitza un lector làser vermell que té una longitud d'ona inferior a la de l'infraroig amb la qual treballa el CD. Aquesta reducció de la longitud d'ona permet reduir la mida del *pit*. A la figura 1.11 es mostra una comparació entre la mida del *pit* i la distància entre pistes.

Amb aquesta reducció de dimensions s'aconsegueix incrementar la capacitat del DVD en un 2,56. Això, unit a millores en les tècniques de codificació i en el sistema de fitxers, han permès incrementar la capacitat per capa a 6:1.

A la figura 1.12 es pot veure que el substrat del DVD està dividit en dues capes. Això permet reduir les deformacions, i proporciona la possibilitat d'escriure en els dos substrats. D'aquesta manera es pot incrementar la capacitat del DVD per dos. Finalment, cal indicar que cada

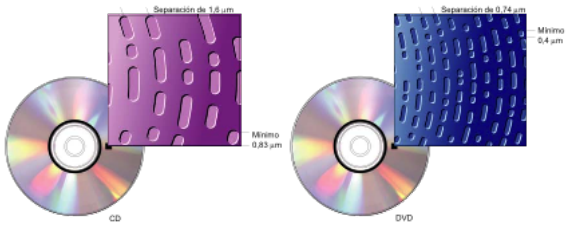


Figura 1.11: Comparació de mida de *pit* entre CD i DVD.

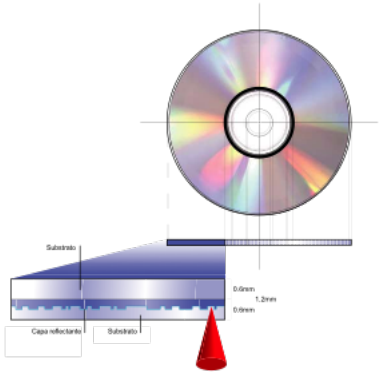


Figura 1.12: Estructura d'un DVD de cara simple i una capa.

substrat es pot dividir en dos, de manera que s'aconsegueix incrementar la capacitat per un altre factor dos. Utilitzant un DVD de doble cara i doble capa es pot arribar a obtenir una capacitat de 18 GB.

En l'actualitat, s'està popularitzant el format Blu-ray, basat en un làser blau, fet que permet incrementar la capacitat fins als 25 GB (una cara) i 50 GB (doble cara).

1.4.3 USB

L'estàndard USB (*universal serial bus*) va ser definit per Intel amb la intenció de substituir tot el conjunt d'estàndards per a la connexió de dispositius de baixa velocitat a l'ordinador (sèrie, paral · lel, teclat, ps/2...). Aquest protocol treballa a 12 Mbps i pot ser utilitzat com a connexió de dispositius multimèdia com ara càmeres web.

A causa dels requeriments actuals d'alguns perifèrics, com les impressores d'alta qualitat o els sistemes d'emmagatzematge massiu com els DVD, Intel va decidir ampliar l'estàndard USB per donar servei a aquest tipus d'aplicacions. A aquest estàndard li va donar el nom d'USB 2.0 i permet treballar a 480 Mbps (40 vegades més ràpid que el seu predecessor), mantenint la compatibilitat amb el seu predecessor. Recentment s'ha incorporat la versió USB 3.0 que aconsegueix els 5 Gbps de velocitat i que modifica el connector per ser compatible amb les versions anteriors.

El comportament de l'USB és equivalent al d'una xarxa, amb una gestió basada en el model mestre-esclau. Un dispositiu és la capçalera de la xarxa, i el que estableix quin dispositiu ha d'accedir a aquesta xarxa en cada moment. Aquesta capçalera sol ser un PC, que, fent ús de controladors, estableix el comportament del dispositiu, i gestiona la utilització de la xarxa. Això permet reduir els costos associats al maquinari, ja que el dispositiu necessita menys intel · ligència.

Per interconnectar diversos dispositius s'utilitzen concentradors, que en funció dels requeriments de velocitat poden ser USB 3.0 o USB 2.0 per a altes prestacions, o USB 1.1 quan les necessitats són inferiors. A la figura 1.13 es mostra la topologia d'aquesta xarxa, que té sempre un ordinador

com a arrel de l'arbre.

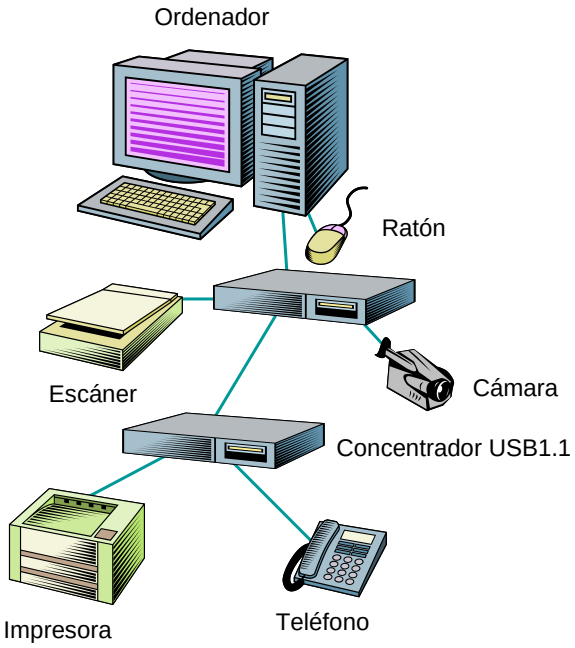


Figura 1.13: Topologia en arbre de l'USB 2.0.

Inclou una transferència isòcrona, pensada per a la transferència d'àudio i vídeo. Envia informació amb un control temporal molt precís, però en el qual no es fa cap comprovació d'errors. Per poder-lo utilitzar, el PC ha de donar la seva conformitat.

En l'actualitat aquest bus està molt introduït al mercat, i és un dels més utilitzats per a la interconnexió digital de dispositius.

1.4.4 Firewire

El Firewire o IEEE 1394 és un estàndard que dissenya inicialment Apple com a xarxa per a la transmissió d'informació entre dispositius multimèdia. Va ser estandarditzat el 1995. La seva principal característica és que permet treballar de forma isòcrona, assegurant que tots els nodes reben la mateixa informació a la cadència desitjada. Per aquest motiu és una xarxa òptima per a aplicacions de vídeo i àudio digital, tot i que permet intercanviar tot tipus d'informació.

Per aconseguir un control precís per al mode isòcron, utilitza una topologia en arbre. A la figura 1.14 es mostra un exemple de la topologia, on es pot veure que els diferents dispositius s'interconnecten a través dels seus superiors el node arrel dels quals és l'STB, encara que en molts casos és un ordinador.

Des del punt de vista d'un dispositiu connectat a la xarxa, aquesta és com un gran espai de memòria virtual al qual pot accedir. Cada dispositiu ocupa una determinada adreça d'aquesta memòria, amb adreces de 48 bits de longitud (256 tebibytes). A cada tram de xarxa poden haver-hi fins a 64 dispositius connectats, i la xarxa pot contenir un total de 1024 segments. En aquest cas es pot accedir a 16 exbibytes de memòria, molt superior a les necessitats de qualsevol sistema actual.

Com en el cas de l'USB, hi ha un mode de transmissió isòcrona que permet enviar informació d'un node a un altre o a tots. En aquest mode, no hi ha correcció d'errors o retransmissió de la informació, i pot arribar a ocupar un 80% de la capacitat del bus, que és de 800 Mbps. Un node

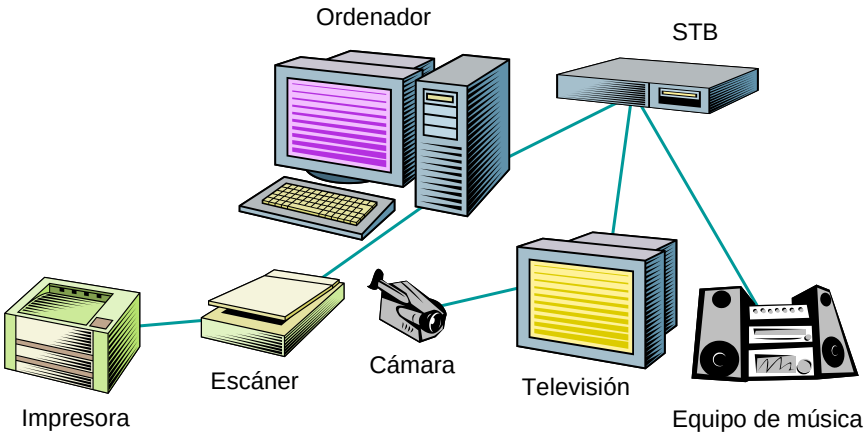


Figura 1.14: Topologia en arbre de l'estàndard IEEE 1394.

pot sol · licitar en qualsevol moment la utilització del mode isòcron, però només se li concedeix quan la xarxa té disponible capacitat de bus per fer-ho. En cas contrari, el node s'ha d'esperar.

En mode asíncron, un node pot enviar informació quan el canal no està ocupat. Aquesta manera serveix de control d'errors i de repetició de la trama, i està especialment indicat per a aplicacions que no permeten errors, com l'accés a un disc dur.

En l'actualitat els dispositius que utilitzen aquest bus són càmeres digitals, vídeos digitals i gravadores de DVD. A més, també s'utilitzen amb PC. Permet que qualsevol dispositiu es connecti amb qualsevol altre, ja que els considera iguals. A tall d'exemple, una càmera pot connectar-se amb una gravadora de DVD i ser controlada des d'un televisor, tot això sense necessitat d'un ordinador. No obstant això, ha anat perdent força enfront de l'USB, i alguns ordinadors d'Apple ja no l'incorporen.

1.4.5 Bluetooth

La xarxa Bluetooth o IEEE 802.15.1 és l'equivalent sense fil de l'USB 1.1. Va néixer amb l'objectiu de permetre que petits dispositius s'interconnectessin entre si en el que s'ha denominat una *xarxa d'àrea personal* (*personal area network*, PA). Possibles aplicacions d'aquesta xarxa són la intercomunicació d'un telèfon mòbil amb uns auriculars, d'una PDA amb un mòbil o d'un teclat amb un PC. La distància màxima entre nodes és de 10 metres.

Transmet en la banda ISM de 2,4 GHz, fent ús de l'esquema de transmissió d'espectre eixamplat per salt de freqüència (*frequency hopping spread spectrum*, FHSS). Aquest esquema divideix l'espectre de freqüències en subbandes d'ample equivalent a la taxa de bits a transmetre. Durant la transmissió del senyal, es va saltant d'una subbanda a una altra, de manera que s'utilitza tot l'espectre, i es disminueix la possibilitat d'interferències.

Hi ha dues modalitats de FHSS: la ràpida, que fa diversos salts per bit transmès, i la lenta, que transmet diversos bits per salt. En el cas del Bluetooth s'utilitza el mètode ràpid per incrementar la seguretat.

Un dispositiu Bluetooth pot comunicar-se amb vuit dispositius. Quan es creen els enllaços entre els diferents dispositius, es crea una picoxarxa o PA, la qual és supervisada per un node mestre. Tots ells fan ús de la mateixa seqüència de salt. Un node esclau pot pertànyer a diverses picoxarxes, tal com es mostra a la figura 1.15.

Cada picoxarxa es connecta a una velocitat de fins a 721 Kbps, i és el mestre qui determina com es distribueix l'ample de banda. Poden existir fins a 10 picoxarxes de 8 dispositius, de

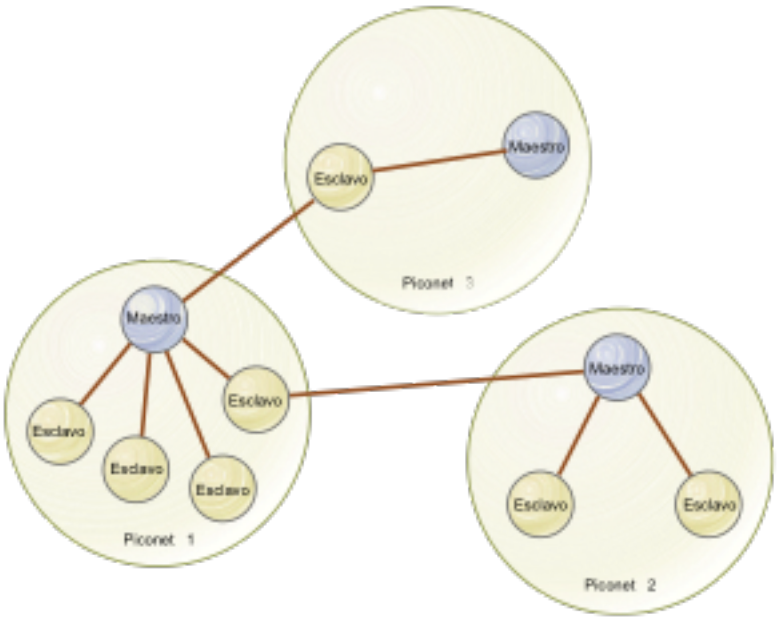


Figura 1.15: Arquitectura d'una xarxa Bluetooth.

manera que l'ample de banda en què s'aconsegueix transferència útil és d'uns 6 Mbps. Per incrementar la seguretat, a més de fer salts 1.600 vegades per segon, utilitza encriptació de 64 bits i autenticació per dispositiu.

En l'actualitat ja hi ha multitud de dispositius preparats per connectar-se a través de Bluetooth, com càmeres fotogràfiques, PDA, telèfons mòbils, mans lliures, teclats, ratolins, PC...

1.5 Xarxes

Avui dia, la majoria d'ordinadors es connecten entre si a través d'una xarxa. D'aquesta manera poden intercanviar informació, veure l'estat d'un determinat procés de forma remota o enviar missatges. La majoria de les xarxes estan interconnectades, i formen una xarxa mundial coneguda com a *Internet*. Internet està constituïda per equips de capacitat molt diversa (des de supercomputadors fins a equips mòbils), i proporciona a desenes de milions de persones la utilització de múltiples serveis, com ara l'accés a recursos i maquinari compartits, correu electrònic, transferència d'arxius, notícies, accés a documents multimèdia, transaccions bancàries i comerç electrònic.

Podem definir una *xarxa d'ordinadors* com un conjunt d'equips autònoms interconnectats a través d'un mitjà pel qual s'intercanvien informació. Entenem per *mitjà* el sistema (material o no) que ens serveix per transmetre la informació. Entre els mitjans de comunicació més destacats tenim el fil o conjunt de fils de coure, el cable coaxial, l'aire i la fibra òptica.

Tot ordinador o equip que pretengui connectar-se a una xarxa necessita un maquinari específic, mòdem o targeta d'interfície de xarxa (NIC) i un programa especial de comunicacions. Tots dos elements s'han d'adequar a les característiques i protocols de la xarxa.

Les xarxes d'ordinadors augmenten notablement el quocient prestacions/preu dels sistemes informàtics. Els principals avantatges que s'obtenen són:

- Possibiliten un servei remot d'utilització d'aplicacions, sense necessitat que l'usuari disposi realment d'elles.

- Es poden establir processos en diferents màquines, actuant d'aquesta forma en paral · lel i, per tant, millorant les prestacions en col · laborar per fer una tasca determinada.
- Permeten l'accés a documents de tipus text o multimèdia en general. Amb aquests serveis es pot accedir a informació de bases de dades i arxius des d'equips localitzats a grans distàncies.
- Admeten la gestió de bases de dades distribuïdes, de manera que un únic sistema no necessiti tenir la capacitat suficient per allotjar la base de dades completa.
- Augmenten la seguretat del sistema des de la redundància.
- Si l'equip local al qual es té accés directe no disposa de les prestacions adequades (poca memòria, o bé està molt carregat de feina, no disposa d'impressores ràpides o de suficient qualitat d'impressió, etc.), l'usuari pot connectar-se a un altre equip de la xarxa que reuneixi les característiques pertinents.
- Permeten la utilització de la xarxa d'ordinadors com a mitjà de comunicació: correu electrònic, ràdio i televisió a través d'Internet, etc.

Els ordinadors s'interconnecten a través d'una xarxa de comunicacions que pot ser una xarxa d'àrea àmplia (WAN) l'àmbit geogràfic de la qual és un entorn regional, nacional o internacional; una xarxa d'àrea metropolitana (MAN) que interconnecta equips d'una ciutat, o una xarxa d'àrea local (LAN) l'àmbit de la qual és un departament, un edifici o un campus. En l'actualitat també estan adquirint gran rellevància un tipus de xarxes sense fil d'abast molt curt. Aquest tipus de xarxa se sol denominar *xarxa d'àrea personal* (PAN).

1.5.1 El mòdem

Durant els últims anys la transmissió analògica ha dominat les comunicacions. Avui dia, però, des de la implantació de la XDSI, i més recentment l'xDSL i FTTH, les comunicacions es basen en transmissions digitals, que ens permeten velocitats de transferència molt superiors a les que es podien aconseguir mitjançant els sistemes basats en transmissió analògica. En particular, el sistema telefònic es va basar inicialment en la senyalització analògica. En l'actualitat les línies troncals metropolitanes i de llarga distància són digitals, tot i que existeixen encara llaços locals que són analògics i probablement seguiran així durant un temps pel cost de convertir-los al sistema digital. En conseqüència, quan tenim un equip connectat a Internet des de casa, el sistema envia dades digitals per una línia analògica. Les dades es modulen i desmodulen per poder-se transmetre pel llaç analògic. Posteriorment, quan les dades arriben al proveïdor de serveis, es tornen a convertir en senyals digitals que es dirigeixen a la destinació corresponent. El mòdem convencional permetia velocitats de comunicació relativament petites, uns 56 Kbits/s. En l'actualitat els mòdems ADSL 2+ permeten velocitats molt superiors: 25 Mbits/s de baixada i fins a 3,5 Mbit/s de pujada.

1.5.2 La targeta d'interfície de xarxa (NIC)

Una targeta d'interfície de xarxa és una placa de circuit imprès la funció de la qual és la connexió del PC a una xarxa local de comunicacions (LAN). És per aquesta raó que també se la coneix pel nom d'*adaptador LAN*.

La connexió de la NIC amb la placa base del PC es fa a través del port d'expansió, tot i que actualment es pot trobar integrada a la placa base de l'ordinador. Encara que existeixen diversos tipus de NIC per als diferents protocols de comunicacions, les més utilitzades són les NIC Ethernet.

Usualment, una NIC es comunica amb la xarxa d'àrea local a través d'una connexió sèrie (és a dir, les dades es transmeten bit a bit). Per a això sol utilitzar-se un cablejat estàndard, definit per l'organització EIA-TIA com l'UTP Categoria 5 o 6. La connexió entre el cablejat i el PC es fa a través d'un connector estàndard: l'RJ-45. No obstant això, cada vegada més les connexions entre PC i LAN tenen lloc a través del mitjà sense fil; és el que es coneix com el *Wireless Fidelity* o wifi. Una LAN cablejada pot arribar a transmetre dades a velocitats de 1000 Mbits/s. Pel que fa a les comunicacions sense fil, les velocitats arriben fins als 54 Mbits/s.

Les dades que ens arriben des de la LAN (connexió sèrie) són transmeses cap al PC a través d'una connexió paral · lela. De la mateixa manera, les dades transmeses des del PC cap a la LAN es reben a través d'una connexió paral · lela i s'envien a través d'una connexió sèrie.

Les targetes NIC porten un número d'identificació conegut com a *adreça maquinari* o *adreça MAC*. Aquest número és de 48 bits i és diferent per a cada targeta de xarxa. Els primers 24 bits identifiquen el fabricant de la NIC. Els 24 bits restants són la identificació del producte. Quan un ordinador transmet una sèrie de dades (un correu electrònic, per exemple) cap a un determinat destí, l'equip origen ha de conèixer l'adreça MAC de la destinació i introduir-la al missatge juntament amb la seva pròpia MAC. Quan la NIC de l'ordinador destí rep les dades, primer farà una verificació de l'adreça MAC, i posteriorment determinarà si les dades rebudes són correctes o si hi ha hagut algun problema durant la transmissió (dades errònies). Un cop verificat tot això, les dades es transmetran al PC per processar-les posteriorment.

Exercici 1.5.1 *Identificar l'adreça MAC del nostre equip*
Volem determinar l'adreça MAC del nostre equip. Per fer això:

- *Obrim una consola.*
- *Escrivim* `ipconfig` *(Windows) o* `ifconfig` *(Linux-MAC).*
- *Recordem que l'adreça MAC és de 48 bits. Quina MAC teniu?*

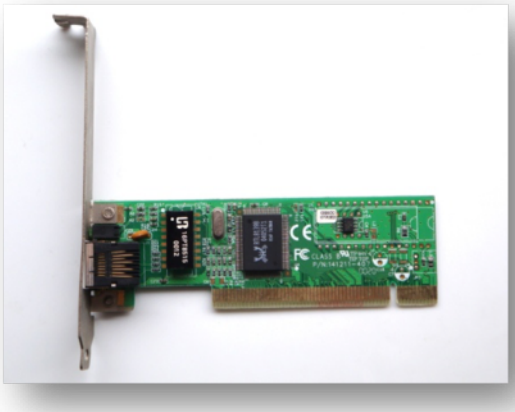


Figura 1.16: Targeta NIC de connexió per a PC a xarxes d'àrea local.

En el cas de les connexions sense fil (generalment wifi), la velocitat que s'aconsegueix és molt inferior al cas d'una LAN cablejada. Per contra, es guanya en mobilitat, ja que no cal connectar cap cable per tenir accés a la xarxa. Aquest fet ha permès portar Internet a llocs com un parc o una platja, perquè les persones puguin treballar i distreure's en un ambient més relaxat.

Exercici 1.5.2 *Identificar l'adreça MAC d'altres equips a la nostra taula local:*

- *Obrim la consola i executem la comanda* `ifconfig`.
- *Apuntem l'adreça IP del nostre ordinador i la passem a diversos companys.*
- *Accedim als ordinadors dels nostres companys per veure si estan connectats. Executem la comanda* `Ping` *+ IP del company.*
- *Repetim l'apartat anterior en diversos companys.*
- *Finalment executem la comanda* `arp -a`. *Això ens mostrarà les adreces MAC de les diferents màquines a les quals hem contactat.*

La taula on es guarden les diferents adreces MAC està en una memòria volàtil que es va actualitzant a mesura que es necessita. És possible que es vulgui enviar informació a un determinat PC del qual coneixem la IP però del qual es desconeix l'adreça MAC. Quan es produeix aquest cas (més habitual del que puguis pensar), s'executa un protocol conegut com a *ARP*. L'ordinador origen envia un missatge amb les seves adreces MAC i IP en què indica que vol conèixer l'adreça MAC d'un equip la IP del qual és coneguda. Quan l'ordinador destí rep aquest missatge, completa el camp de la seva adreça MAC perquè l'origen pugui actualitzar la seva taula ARP. Aquest format s'utilitza per a xarxes locals. Si la comunicació s'estableix amb un PC que pertany a una altra xarxa, el funcionament és diferent.

1.5.3 Un model per a les comunicacions de dades: la pila TCP/IP

El departament de defensa dels EUA va crear el model de referència TCP/IP ja que necessitava una xarxa robusta, capaç de sobreposar-se a qualsevol problema i que permetés la transmissió de dades entre diferents ordinadors. TCP/IP és l'acrònim de *transmission control protocol / Internet protocol*. Aquest protocol està format per 4 nivells o capes, que cobreixen des de la part més física (maquinari i cablejat) fins a la part de programari que s'encarrega de la comunicació (http, ftp...).

El TCP/IP és la base d'Internet i serveix per enllaçar ordinadors que utilitzen diferents sistemes operatius, incloent-hi PC, minicomputadores i computadores centrals sobre xarxes d'àrea local (LAN) i àrea extensa (WAN). Els quatre nivells que formen el protocol TCP/IP són:

- Capa d'aplicació. És la capa de més alt nivell en el protocol. S'encarrega de la presentació i la codificació de les dades i eines de control. Alguns dels protocols que conformen aquesta capa són SMTP (correu electrònic), FTP (transferència de fitxers), TELNET (connexions remotes) i HTTP (*hypertext transfer protocol*).

- Capa de transport. La capa de transport típicament duu a terme tasques de proporció de la fiabilitat, necessària per al transport de dades; control de flux de dades, i retransmissions en cas d'errors. Tenim dos protocols dins d'aquesta capa: TCP i UDP. EL TCP és un protocol orientat a la connexió mentre que l'UDP és un protocol no orientat a la connexió. El TCP és el protocol més comunament usat i encapsula HTTP, FTP i SMTP, entre d'altres.

- Capa de xarxa. El propòsit d'aquesta capa és l'enviament de paquets des de qualsevol ordinador de qualsevol xarxa connectada a Internet amb qualsevol destinació que es trobi connectada. El protocol específic que governa aquesta capa és l'*Internet protocol* (IP). Per aconseguir la seva comesa, el protocol IP s'ajuda de protocols d'enrutament, que li permeten trobar el camí per arribar d'un punt a un altre (aquests protocols serien com el navegador per a un conductor).

- Capa d'accés al medi. El nom d'aquesta capa pot donar lloc a confusió. També se la coneix com a *capa d'interconnexió amb el medi físic*. Normalment es compon de dues capes: la capa d'enllaç, encarregada de muntar la trama de dades i assegurar el control d'errors i

flux en cada salt, i la capa física, on es defineixen els senyals utilitzats per codificar els bits. És en aquesta capa on intervenen la NIC o el mòdem. També trobem els concentradors i commutadors, que permeten interconnectar segments de xarxa.

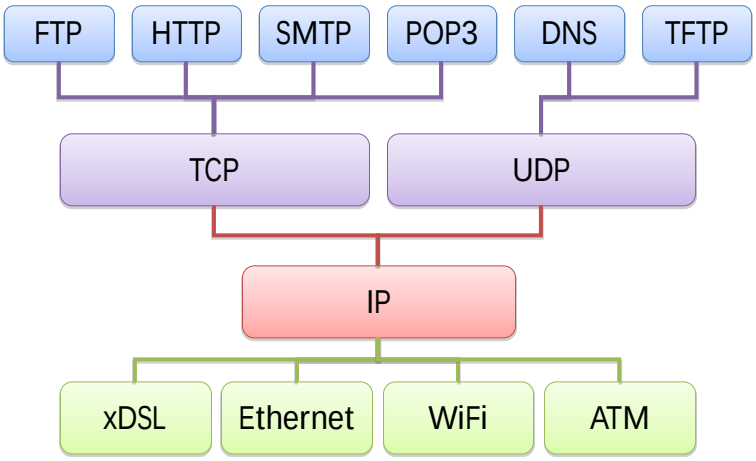


Figura 1.17: Representació gràfica de la pila de comunicacions TCP/IP.

A la figura 1.17 es mostra un diagrama on apareixen els diferents protocols que es tenen en TCP/IP. A la capa d'aplicació es troben alguns dels diferents protocols més utilitzats a Internet. És probable que el lector els faci servir diverses vegades cada dia.

A Internet un ordinador individual s'identifica mitjançant una adreça IP que està formada per dues parts: el codi d'identificació de la xarxa i el codi identificador de l'equip. L'adreça IP està formada per 4 bytes, que se solen donar en decimal i separats per punts.

Exercici 1.5.3 *Determinar l'adreça IP del teu ordinador*

Utilitza la instrucció `ipconfig` *(Windows) o* `ifconfig` *(Linux-MAC). Un cop descoberta la teva IP, determina una de les IP de Google executant la comanda* `traceroute` *www.google.com*

1.5.4 Aplicacions d'Internet

En aquest apartat s'expliquen de forma general algunes de les aplicacions d'Internet més utilitzades i que ja han estat comentades en l'apartat anterior.

1.5.4.1 Connexió de terminal remot (TELNET)

Possibilita a un usuari connectar-se a qualsevol equip remot en qualsevol de les xarxes interconnectades utilitzant el conjunt de protocols TCP/IP. La connexió es fa emulant, en l'equip de l'usuari, un terminal virtual. D'aquesta manera pot utilitzar el terminal com si hi estigués connectat directament. En l'actualitat, aquest tipus de connexió ha evolucionat amb l'objectiu d'aconseguir connexions segures, a través del protocol SSH.

1.5.4.2 Transferència d'arxius

La transferència d'arxius a través d'Internet utilitza un protocol per a la transferència de fitxers conegut com a FTP (*file transfer protocol*). Aquest protocol permet transferir fitxers des d'un determinat servidor cap a aquella persona que hagi fet la petició de transferència (client). Hi ha programes específics que implementen el protocol FTP, encara que també pot executar-se des de la consola de qualsevol ordinador. La instrucció a implementar és:

- ftp adreça IP

A partir d'aquí la captura de fitxers es fa mitjançant la instrucció `get`. La pujada de fitxers al servidor es fa mitjançant la instrucció `put`.

La connexió entre un client i un servidor via FTP és una connexió no segura ja que no porta cap tipus de xifratge. Per transferir fitxers de forma segura s'utilitzen aplicacions com ara SFTP (*Secure FTP*) que afegeixen les tècniques necessàries per establir la connexió de forma segura.

1.5.4.3 Sistema de fitxers a la xarxa (NFS)

L'NFS és un protocol que permet al client accedir a un fitxer remot, com si estigués en la màquina del servidor, de manera que pot editar-lo, o fins i tot transferir-lo a una tercera màquina si fos necessari.

1.5.4.4 Correu electrònic

El correu electrònic potser és el servei més utilitzat a Internet, ja que amb ell un usuari pot enviar missatges o fitxers a un altre ordinador. El sistema es fonamenta en la utilització del concepte de *bústia* i d'*adreça de bústia*. La bústia està lligada a una persona o ens determinat. Una adreça de correu es compon de dues parts diferenciades i separades pel caràcter @. La primera part correspon a l'adreça de la bústia dins del domini mentre que la segona correspon al nom del domini.

El correu electrònic a Internet s'implementa amb el protocol SMTP (*simple mail transfer protocol*).

1.5.4.5 Tertúlies a Internet o xats (IRC)

L'IRC o xat és una aplicació mitjançant la qual diferents usuaris d'Internet poden comunicar-se entre si en temps real. La diferència fonamental amb el correu electrònic rau en el fet que la comunicació és instantània.

1.5.4.6 Aplicacions multimèdia

En aquest grup s'inclouen aplicacions com ara videoconferència o transmissió d'àudio o vídeo sota demanda feta utilitzant el protocol IP. Aquest conjunt d'aplicacions es caracteritza per la necessitat d'utilitzar un gran ample de banda i fer execucions en temps real, amb retards molt baixos i una alta qualitat. Solen utilitzar tècniques de compressió de dades per reduir l'ample de banda.

1.5.5 El web

La xarxa mundial World Wide Web (abreujadament WWW o simplement web) és simplement un conjunt de documents (arxius) distribuïts al llarg de tot el món i amb enllaços entre ells. Els documents es visualitzen amb paraules, frases o imatges ressaltades (en negreta o en un color determinat) sota les quals s'amaguen punters a altres documents ubicats en el propi servidor o un altre. Seleccionant i activant amb el ratolí un element amb punter, automàticament es fa una crida al document apuntat. Un punter a un altre document es denomina *enllaç* i un text amb hiperenllaços s'anomena *hipertext*. Un hipertext disponible a la WWW s'anomena *pàgina*. La primera pàgina que apareix quan obrim el navegador es denomina *pàgina principal*.

1.5.5.1 Adreçament al web (URL)

Un dels conceptes fonamentals de la web és la forma de localitzar documents. Això es fa utilitzant un localitzador uniforme de recursos (URL), que pot referenciar qualsevol tipus de document a la xarxa. Un localitzador URL es compon de tres parts, separades amb ratlles inclinades:

- Protocol-de-recuperació://computador/ruta-i-nom-de-l-arxiu

Hi ha diferents protocols de recuperació de documents. El més conegut és HTTP, però també destaquem FTP, SMTP i TELNET. *Computador* és el nom DNS de l'equip, i finalment el nom complet de l'arxiu que conté la pàgina buscada.

Exemple 1. Una adreça d'una pàgina web és
http://www.ub.edu/homeub
que significa el següent:

- http: el protocol de transferència usat és HTTP.
- www: la pàgina forma part de la World Wide Web.
- ub.edu: la pàgina pertany a la Universitat de Barcelona.
- homeub: document sol · licitat. Inici de la UB

Els hiperenllaços són senzillament localitzadors URL, ja que determinen exactament el nom del fitxer, el computador on es troba la pàgina, així com el protocol utilitzat.

Exemple 2. L'adreça del Departament d'Electrònica de la Universitat de Barcelona és:
http://el.ub.es
S'utilitza de nou el protocol HTTP. El nom de l'equip és el.ub.es, i com que no indiquem cap nom d'arxiu, s'accedirà a la pàgina principal del Departament.
D'altra banda, l'adreça URL del servidor web del Departament d'Electrònica és:
http://www.el.ub.es
De manera que el computador on es troba és www.el.ub.es (WWW sol ser un àlies del nom de l'ordinador).

1.5.5.2 Protocol HTTP

Un dels punts clau en l'expansió de la WWW és el protocol HTTP (protocol de transferència d'hipertextos o *hipertext transfer protocol*). L'HTTP segueix el model client servidor, usat en general entre un navegador web i un servidor web. L'HTTP estableix com recuperar hiperdocuments distribuïts i enllaçats a través del web. El protocol estableix per al client tres possibles paquets:

- Sol · licitar dades del servidor, l'objectiu fonamental.
- Sol · licitar la recepció d'informació sobre característiques del servidor.
- Enviar informació al servidor.

El servidor té definit un únic paquet de resposta que es limita a enviar la informació de capçaleres o l'arxiu sol · licitat.

Quan un usuari fa una petició d'accés a un determinat document d'una adreça donada, l'HTTP sol · licita una connexió TCP, amb la qual es transfereix al servidor la petició HTTP, en la qual s'inclou la URL, informació sobre el client i l'ordre concreta amb els seus paràmetres associats. El servidor respon duent a terme l'operació requerida i enviant una resposta HTTP, que conté informació sobre el servidor i de control, i la resposta pròpiament dita a la sol · licitud. Un cop donada la resposta es tanca la connexió TCP.

1.5.5.3 Navegadors web

Un navegador (*browser* en anglès) és un programa que permet captar, interpretar i visualitzar documents web. La captació d'aquests documents es fa amb l'ajuda d'un dels protocols vistos amb anterioritat i que permeten l'obtenció d'arxius a través d'Internet: HTTP, FTP, TELNET, etc. Els documents web poden classificar-se en tres tipus: estàtics, dinàmics i actius.

- Documents estàtics. Són documents amb informació fixa. El servidor web conté el document en llenguatge HTML i l'envia al client a petició seva. El client processa la informació que genera el document que serà visualitzat.
- Documents dinàmics. Són documents la informació dels quals canvia constantment. Un exemple és la pàgina web que mostra els resultats de la jornada de futbol en temps real, o l'hora i la temperatura d'una determinada localitat. Cada vegada que un usuari sol · licita una d'aquestes pàgines, es llança al servidor l'execució d'un programa que genera el document a transmetre o n'actualitza les parts dinàmiques. Un cop generada la pàgina es transmet al client. Per a això s'utilitza un estàndard anomenat *CGI* o *interfície comuna de passarel · la*. Els programes CGI generadors de documents dinàmics poden ser escrits en llenguatges orientats a objectes, com ara Perl, Phyton, C ++ o d'altres.
- Documents actius. Són documents que també es generen en el moment de visualitzar-se, però en lloc de fer-ho al servidor, es creen en el propi client. En aquest cas, el servidor remet el programa i les dades, el client executa el programa que genera la pàgina i el propi programa va actualitzant els canvis. Els documents actius se solen programar en Java. Els compiladors de Java generen codi màquina per a un processador virtual conegut com a *JVM*, de manera que executar aquest codi en una màquina concreta simplement requereix que es disposi d'un emulador de la JVM. La majoria dels navegadors disposen d'un emulador de JVM.

Capítol 2

Introducció a la programació

La programació és el procés d'escriure instruccions per tal de crear programari per a un usuari final. Per sobre de tot, és una activitat creativa que preten solucionar, mitjançant programari, un determinat problema.

Actualment, la programació forma part de les habilitats bàsiques que ha de proporcionar qualsevol grau d'enginyeria o de base científica.

Les instruccions d'un programa s'escriuen fent servir un **llenguatge de programació**. Per tant, un **llenguatge de programació** és el llenguatge formal amb què comuniquem a una màquina les instruccions que ha d'executar (programes o algorismes). Un **llenguatge de programació** consisteix en una sèrie de regles sintàctiques (forma) i semàntiques (significat).

Tots els llenguatges de programació tenen elements bàsics per a la descripció de les dades i dels processos o transformacions que se'ls poden aplicar (exemples d'aquests processos són la suma de dos nombres o la selecció d'un element concret d'una col · lecció). Aquests elements bàsics es defineixen per una sèrie de regles sintàctiques i semàntiques, que en descriuen l'estructura i funcionalitat, respectivament.

La sintaxi es refereix a la forma en què un llenguatge està escrit. La majoria dels llenguatges de programació són textuals. Això vol dir que fan servir seqüències de text, paraules, nombres i puntuació com la majoria de llenguatges naturals que existeixen. Hi ha, però, altres llenguatges de programació que treballen de forma gràfica, fent servir relacions visuals entre símbols per definir i crear el programa final. La sintaxi d'un llenguatge descriu les possibles combinacions de símbols que formen un programa sintàcticament correcte.

La semàntica es refereix al contingut, al significat. Existeixen una sèrie de restriccions en l'estructura d'un programa vàlid. En el cas dels llenguatges compilats, la semàntica del llenguatge inclou aquelles regles que s'han de complir en el procés de compilació del programa. Per exemple, necessitem comprovar que tot identificador ha estat declarat abans d'utilitzar-se (en aquells llenguatges que requereixen la declaració d'aquests identificadors). Vegem l'exemple següent:

```
1 print (x)
2 x = 3
```

i

```
1 x = 3
2 print (x)
```

Tot i que els dos exemples són sintàcticament correctes, són semànticament diferents, ja que s'ha de tenir en compte l'ordre d'execució. En funció de l'ordre en què fem servir les instruccions d'un programa el seu significat pot variar. Això té a veure amb l'ordre d'execució, la precedència dels operadors, etc.

Depenent del nivell d'abstracció que el llenguatge presenta respecte als detalls físics de l'ordinador que s'està fent servir (tipus de processador, memòria, accés a registres,...), els llenguatges es classifiquen en **llenguatges d'alt nivell** i **llenguatges de baix nivell**. Els llenguatges de més baix nivell són aquells que treballen en llenguatge màquina o assemblador, que ataquen directament registres generals i específics del processador, que s'adrecen la memòria, etc.

2.1 Llenguatges de programació d'alt nivell

Tal com hem comentat en l'apartat anterior, els llenguatges de programació d'alt nivell són aquells que presenten un alt nivell d'abstracció respecte als detalls del maquinari del PC on treballem. Fan servir elements del llenguatge natural (anglès) i són normalment més senzills d'utilitzar, ja que amaguen al programador elements importants dels ordinadors com és el cas de la gestió de la memòria.

En la dècada de 1960, els llenguatges de programació d'alt nivell feien servir els compiladors per "traduir" aquests llenguatges formals en seqüències de codi màquina. Aquests llenguatges eren anomenats **autocodes**. Exemples d'*autocodes* són COBOL i Fortram. Fortram va ser el primer llenguatge d'alt nivell àmpliament utilitzat (de fet, encara hi ha gent avui dia que el fa servir...) que proporcionava un veritable sistema de desenvolupament independent de la màquina, en aquella època, als primers sistemes IBM. Un dels grans avantatges d'aquests programes era que **el mateix programa es podia fer servir en diferents màquines**.

En lloc de treballar amb registres, adreços de memòria i *call stacks*, els llenguatges d'alt nivell feien servir variables, *arrays*, objectes, complexes expressions aritmètiques o booleanes, subrutines i funcions, bucles, fils, *locks* i altres conceptes de la ciència de la programació abstracta, que es focalitzava en la (re)usabilitat del codi en lloc de buscar l'eficiència del programa.

Mentre que els programes d'alt nivell tracten de fer més senzilla la programació, els llenguatges de baix nivell sovint generen codi més eficient (quant a velocitat d'execució i espai en memòria). Ara bé, amb l'evolució de les modernes i complexes arquitectures de processadors actuals, el desenvolupament de compiladors ha evolucionat en concordança amb aquestes arquitectures, i ha generat codi comparable en eficiència i mida al que poden generar la majoria de programadors de baix nivell.

Segons el mode d'execució, els llenguatges de programació d'alt nivell es poden subdividir en: **compilats** i **interpretats**. En el primer cas, el codi és transformat en un fitxer executable abans de fer-lo córrer. Això es fa produint directament "codi màquina" a partir del programa d'alt nivell, o, més recentment, produint una representació optimitzada intermèdia com és el *byte code*, que es pot guardar per a futurs processos sense haver d'accedir al codi original. Aquesta representació intermèdia requereix ser posteriorment compilada o interpretada per poder-se executar.

2.1.1 Llenguatges interpretats

Quan el llenguatge és interpretat, la seva sintaxi és llegida i executada directament, sense necessitat de ser compilat. Hi ha un programa anomenat **intèrpret** que llegeix cada sentència del programa, i seguint la seva seqüència natural, l'executa. Un híbrid d'un intèrpret i un compilador compilarà la sentència en codi màquina i l'executarà. Un cop executat, el codi màquina es descartarà, per interpretar la sentència de programa següent.

El pas de compilació usualment introdueix una comprovació extensiva de la sintaxi de tot el codi, assegurant amb un alt nivell de precisió, la consistència del codi abans de produir l'executable. Al mateix temps s'introdueixen procediments automàtics que optimitzen l'ús de recursos. Quan el codi és intepretat, aquesta comprovació només es pot fer en aquella sentència que s'està executant.

Nota. Els llenguatges no són estrictament o "interpretats" o "compilats"; més aviat, algunes implementacions d'aquests llenguatges fan servir la compilació o la interpretació.

2.1.2 Python, IPython i Notebook

Python és un llenguatge de programació d'alt nivell, multiplataforma, de propòsit general i interpretat. La seva filosofia rau en la reutilització de codi, amb una sintaxi que permet als programadors expressar conceptes en menys línies de codi que les que es farien servir en altres llenguatges com ara C++ o Java.

Els intèrprets de Python estan disponibles per a la instal · lació en la majoria de sistemes operatius, fet que permet l'ús de Python en una àmplia varietat de sistemes.

En lloc d'introduir tota la funcionalitat requerida en el nucli del llenguatge, Python incorpora un conjunt de biblioteques que poden ser importades en cas de ser necessàries. Python pot ser incrustat en aplicacions ja existents que necessiten una interfície programable. Aquest disseny d'un nucli d'instruccions relativament petit amb una gran quantitat de biblioteques estàndards i un senzill i extensible intèrpret forma part de la filosofia de Python des dels seus orígens.

Es tracta d'un llenguatge de programació multiparadigma, ja que permet la programació orientada a objectes (OOP), la programació imperativa i la programació funcional. Com ja s'ha comentat, és un llenguatge interpretat, que fa servir tipus de dades dinàmic i és multiplataforma. Una característica important de Python és la resolució dinàmica de noms (associa noms de variables i funcions durant l'execució del programa).

Un neologisme prou comú en la comunitat de Python és ***pythonic***. Diem que un codi és *pythonic* perquè fa servir correctament el llenguatge, és senzill de llegir, és reutilitzable i segueix la filosofia de Python. Al contrari, quan veiem un codi que és difícil de seguir o llegir, o sembla una traducció complicada d'un altre llenguatge de programació i que és difícilment reutilitzable, diem que és ***unpythonic***.

Python pot ser empaquetat en programes executables per alguns dels sistemes operatius més utilitzats fent servir eines d'altres distribuïdors o fabricants (i, per tant, de pagament), fet que permet la distribució de programari basat en Python per a l'ús d'aquests programes sense requerir la instal · lació d'un intèrpret de Python.

2.1.2.1 IPython

IPyhton (Python interactiu) és un projecte de programari lliure que proporciona:

- Una consola de Python millorada.
- Un model de comunicació de dos processos desacoblat, que permet a diversos clients connectar-se al *kernel* de computació, fent servir l'eina Notebook.
- Una arquitectura que permet la computació interactiva paral · lela.

IPython té característiques com ara: terminació de sentències fent servir el tabulador, introspecció d'objectes, accés a la consola de sistema, historial de comandaments de recuperació entre sessions, i sistema de comandes propi per afegir funcionalitats quan es treballa de forma interactiva. Proporciona també un entorn de treball força eficient per al desenvolupament de codi Python i per solucionar problemes fent servir objectes de Python.

Les quatre comandes més útils d'IPython es mostren a l'inici de l'intèrpret:

```
?         -> Introducció o visió en conjunt de les característiques d'IPython
%quickref -> Quick reference
help      -> Sistema d'ajuda de Python
object?   -> Característiques d'un determinat objecte. Podem fer servir 'object??' per a més
```

Altres característiques interessants són:

- Acaba les sentències fent servir el tabulador. Especialment per a atributs, és interessant per explorar l'estructura i característiques de qualsevol objecte, simplement escrivint `object_name.<TAB>`. També funciona amb noms de fitxers i directoris.
- Escrivint `object_name?` imprimirà les característiques i detalls sobre qualsevol objecte.
- Emmagatzema les comandes introduïdes i el resultat que generen. Es pot anar fàcilment a les comandes prèvies simplement clicant les fletxes amunt i avall, o podem accedir a l'històric de forma més sofisticada.
- Permet moure qualsevol comanda al terminal de sistema, simplement amb el prefix `!`.

2.1.2.2 IPython Notebook

IPython Notebook millora la proposta de programació interactiva proporcionada per la consola o terminal, explicada en el punt anterior, introduint un entorn de programació més agradable fent servir el propi navegador. Aquesta aplicació és adequada per treballar el procés de programari global: desenvolupament, documentació i codi executable, així com un entorn per preparar i documentar els resultats.

Les principals característiques són:

- Editor de codi al navegador, amb identificació de comandes de forma automàtica; identificació i terminació de sentències fent servir el tabulador, i introspecció.
- Execució de codi des del navegador, amb el resultat de l'execució annexada al codi que l'ha generat.
- Mostra del resultat de l'execució del programa utilitzant representació enriquida, com és HTML, LaTeX, PNG, SVG, etc.
- Editor integrat al navegador per a text enriquit fent servir l'opció **Markdown**, que proporciona comentaris de text per al codi, no limitat a text pla.
- Opció d'incloure equacions matemàtiques i afegir notació matemàtica a les cel · les definides com a Markdown, utilitzant LaTeX, proporcionat de forma nativa per MathJax.

2.1.3 El primer programa

Normalment el primer codi que es genera quan s'aprèn un llenguatge de programació consisteix a imprimir una sentència. El codi bàsicament imprimeix una cadena de text, fent servir la consola o sortida estàndard. La sentència que imprimim és "Hola, món!".

```
In [1]: print("Hola, món!")
```

```
Hola, món!
```

Nota. `<TAB>+<Intro>` executa el codi que hi ha a la cel · la.

2.1.3.1 Comentant el codi

Els comentaris són essencials per ajudar altres usuaris (o a nosaltres mateixos passat un temps) a entendre el codi. Farem servir tres formes de comentar el codi:

- Inline comment (`#`)
- Multi-line comment(`"""`)
- Markdown

```
In [1]: # Aquest és el primer programa Hola, món
        print("Hola, món!")
        """ Les cometes defineixen un
            comentari de múltiples
```

```
        línies i es pot fer servir
        per a comentaris més llargs.
        Ara imprimim "Adeu!".
    """
    print("Adeu!")
```

```
Hola, món!
Adeu!
```

2.1.4 Markdown

Markdown és una eina que permet la conversió de text a HTML. Markdown permet escriure de forma senzilla text pla i convertir-lo a un format estructurat HTML.

L'avantatge principal de fer servir Markdown és que la lectura i escriptura del seu codi font és molt senzilla. La idea és que un document amb format Markdown es pot presentar tal com és, com a text pla, sense semblar que ha estat marcat amb etiquetes o instruccions de format.

La millor manera de veure com funciona Markdown i la seva sintaxi és simplement fer una ullada a un document amb format Markdown.

IPython Notebook permet combinar cel · les de codi (que fan servir l'intèrpret d'IPython) amb cel · les de text (fent servir l'intèrpret de Markdown).

Exercici 2.1.1 *Impressió de text:*

a) Imprimeix: El meu primer exercici.

b) El caràcter \ permet escriure una sentència més gran que una línia; comprovar-ho:

```
print("%Això␣es␣un␣text␣\textbackslash{}%
%que␣ocupa␣dues␣l\'{\i}nies%")
```

Exercici 2.1.2 *Si no posem \ per separar una cadena entre dues línies ens dona error.*

a) Comprova:

```
print("%Això␣es␣un␣text%
que␣ocupa␣dues␣%l\'{\i}nies%")
```

Exercici 2.1.3 *Operacions aritmètiques:*

a) També ens val per a operacions aritmètiques. Comprova:

```
print(34-15+16\
-5-20)
```

b) Però com que fem servir parèntesi no cal posar-lo. Comprova:

```
print(34-15+16
-5-20)
```

2.1.4.1 Negreta i itàlica

Es poden generar a partir de la sintaxi següent:

- `**Negreta**`: **Negreta** o `__Negreta__`: **Negreta**
- `*Itàlica*`: *Itàlica* o `_Itàlica_`: *Itàlica*

2.1.4.2 Enllaços externs

Es poden crear enllaços a pàgines web:

```
[Universitat de Barcelona](http://www.ub.edu)
```

El resultat és:

```
\href{http://www.ub.edu/}{{Universitat de Barcelona}}
```

I també s'hi poden incloure imatges:

```
![UB](http://www.ub.edu/web/ub/galeries/imatges/logo_home_nou.png)
```

El resultat és:

1

Exercici 2.1.4 *Enllaços:*

a) Crea un enllaç a una pàgina web.

b) Inclou-hi una imatge.

2.1.4.3 Estructura de text

El text pot ser estructurat en seccions i subseccions:

```
# Títol de primer nivell

Text de la secció principal

## Títol de segon nivell

Text de la subsecció

### Títol de la subsubsecció

Text de la subsubsecció
```

El resultat és:

1. Títol de primer nivell

Text de la secció principal

1.1 Títol de segon nivell

Text de la subsecció

1.1.1 Títol de la subsubsecció

Text de la subsubsecció

Exercici 2.1.5 *Format de seccions:*

a) Escriu com a títol de primer nivell: Els homes primitius

b) Com a títol de segon nivell: L'home de neandertal

c) Com a títol de subsecció: Informe sobre el clima

d) Repeteix els apartats anteriors amb lletra itàlica.

2.1.4.4 Format de citació

Per fer citacions es pot usar el format de citació:

```
> Això és una citació literal: Això és un exemple
```

El resultat és:

> Això és una citació literal: Això és un exemple

o com a text d'amplada fixa (freqüentment usat per codi):

````
```
indicar
print("x + y = " + str(x + ))
```
````

El resultat és:

```
indicar
print("x + y = " + str(x + ))
```

Exercici 2.1.6 *Ús de* ***print****:*

a) Indica que el resultat de la suma de a = 3, b = 2 és 5. Comprova-ho escrivint:

```
print("El resultat de sumar \textit{a} i \textit{b} %\'{e}s: \textit{a}
```

b) Indica que la seva diferència és 1.

2.1.4.5 Llistes

Es poden crear llistes iniciant el text amb - o amb *:

```
* Ítem a
- Ítem a
```

Resultat:

- Ítem a
- Ítem a

Aquesta llista conté una subllista:

```
    * Subítem 1
    * Subítem 2
* Ítem b
* Ítem c
```

El resultat és:

- Ítem a
  - Subítem 1
  - Subítem 2
- Ítem b
- Ítem c

I llistes numerades:

```
2 1. Ítem 1
3 2. Ítem 2
4 3. Ítem 3
```

```
1 El resultat és:
2    1) Ítem 1
3    2) Ítem 2
4    3) Ítem 3
```

Exercici 2.1.7 *Llistes i enumeracions:*

a) Crea una llista dels dies de la setmana utilitzant les dues opcions.

b) Crea una llista dels dies de la setmana i una subllista que indiqui els tres àpats principals.

c) Crea una llista numerada dels set dies de la setmana.

2.1.4.6 Equacions matemàtiques

També es poden escriure fórmules. Si fem servir:

```
1 $\vec{F} = -G \frac{mM}{R^{2}}\hat{r}$
```

Obtenim:

```
1 $\vec{F} = -G\frac{mM}{R^2}\hat{r}$
```

Fent servir `$$equació$$` introdueix un canvi de línia. Per exemple, executem ara això:

```
1 $$\vec{F} = -G \frac{mM}{R^{2}}\hat{r}$$
```

El resultat és:

$$\vec{F} = -G\frac{mM}{R^2}\hat{r}$$

Exercici 2.1.8 *Escriu l'equació de la velocitat en un moviment rectilini uniforme. Comprova el canvi de línia segons si utilitzes* `$` *o* `$$`*.*

2.2 Tipus de dades

Tots els llenguatges de programació tenen tipus de dades bàsiques. Aquests tipus bàsics permeten emmagatzemar, manipular i intercanviar informació dins el programa o entre ells. Els tipus bàsics estan molt relacionats amb la CPU, i cada llenguatge intenta utilitzar-los tantes vegades com sigui possible. Això és el que fa Python.

D'altra banda, és necessari emmagatzemar la informació entre operacions en algun lloc. És el que s'anomena `variables`, permeten emmagatzemar els resultats intermedis mentre estem processant un algoritme.

2.2.1 Tipus numèrics

Representen valors matemàtics, i s'utilitzen per fer càlculs. Els tipus utilitzats són:

2.2.1.1 *Integer*

S'utilitza per operar amb nombres enters. Treballen amb una precisió infinita, és a dir, poden treballar amb qualsevol nombre de dígits. En són exemples:

- Suma:

```
In [2]: 2**100
```

```
Out[2]: 1267650600228229401496703205376
```

- Producte:

```
In [2]: 3 * 5
```

```
Out[2]: 15
```

- Creació d'una variable d'enter:

```
In [3]: # Creant una variable d'enter per testar nombres enters.
        integer_var = 4

        integer_var
```

```
Out[3]: 4
```

- Determinació del tipus d'una variable d'enter:

```
In [4]: type(integer_var)
```

```
Out[4]: int
```

```
In [5]: integer_var + 5
```

```
Out[5]: 9
```

2.2.1.2 *Float*

Són nombres de coma flotant, i s'utilitzen per representar nombres reals. Tenen una part entera, una de fraccional i exponent. En són exemples:

- Creació d'una variable de coma flotant:

```
In [6]: float_var = 2.1

        float_var
```

```
Out[6]: 2.1
```

- Creació d'una variable amb un nombre representat en notació exponencial:

```
In [7]: float_exp = 6.023e23

        float_exp

Out[7]: 6.023e+23
```

- Determinació del tipus:

```
In [8]: type(float_exp)

Out[8]: float
```

- Divisió de variables:

```
In [9]: float_var/float_exp

Out[9]: 3.4866345674912835e-24
```

2.2.1.3 *Complex*

Els nombres complexos són natius a Python, i s'utilitzen com els altres tipus (`j` representa la part imaginària):

- Creant una variable:

```
In [10]: complex_var = 1.5 + 0.5j
```

- Demanant la part real:

```
In [11]: complex_var.real

Out[11]: 1.5
```

- Demanant la part imaginària:

```
In [12]: complex_var.imag

Out[12]: 0.5
```

- Determinant el tipus:

```
In [13]: type(1 + 0j)

Out[13]: complex
```

Important: tingues en compte que $1 + 0j$ es pot considerar un nombre real, ja que no té part imaginària. Python considera que és complex.

Exercici 2.2.1 *Tipus de variables*

Donats:

$a = 3$

$b = 7$

$c = 8 + 3j$

$d = complex(0,5)$

Determina de quin tipus de variable es tracta.

Exercici 2.2.2 *Càlculs d'enters*

Donats:

$x = 13$

$y = 5$

Expressa el seu quocient com a variable: entera, float i complexa.

Exercici 2.2.3 *Càlculs amb diferents tipus*

Donats: $x = 6 + 3j$, $y = 5$, $z = 3$

Calcula: $a = xy$, $b = xz$, $c = yz$, $d = x + y$, $e = y + z$

Determina, en cada cas, el tipus de variable.

Exercici 2.2.4 *Parts d'un nombre complex*

Donat: $x = 6 + 3j$

Determina'n la part real i la part imaginària.

Exercici 2.2.5 *Determina el complex conjugat:*

a) Donat: $x = 6 + 3j$

b) Donat: x = complex(6,3)

2.2.1.4 Booleans

```
In [14]: 3 > 4
```

```
Out[14]: False
```

```
In [15]: test = (3 > 4)
```

```
In [16]: test
```

```
Out[16]: False
```

```
In [17]: type(test)
```

```
Out[17]: bool
```

Exercici 2.2.6 *Comprova si és cert o fals:*

3 != 2 + 1; 3 > 7; 3 < 7; 0 == 2 - 2; 0 == 3. - 3

Exercici 2.2.7 *Comprova el tipus de variable:*

3 != 2 + 1; 0 == 2 - 2

2.2.2 Operadors numèrics

Així l'entorn Python pot ser la teva calculadora, amb les operacions aritmètiques bàsiques `+`, `-`, `*`, `/`, `%` (mòdul) i `**`(exponent) implementades de forma natural. Per a `+`, `-`, `*` i `%`, si el nombre del mateix tipus és utilitzat, els resultats són sempre del mateix tipus. En canvi, `/` i `**` canviaran el tipus així que sigui necessari per representar el resultat esperat de forma correcta:

```
In [18]: 7/3
```

```
Out[18]: 2.3333333333333335
```

En aquest cas el resultat és de coma flotant, per donar el resultat que millor s'ajusta.

Si es requereix la divisió d'un enter, es pot forçar utilitzant l'operador `//`:

```
In [19]: 7//3
```

```
Out[19]: 2
```

El romanent es pot calcular utilitzant l'operador mòdul:

```
In [20]: 7%3
```

```
Out[20]: 1
```

Com a exemple, podem comprovar la fórmula: $dividend = divisor * quocient + romanent$

```
In [21]: dividend = 7
         divisor = 3

         quotient = dividend//divisor

         quotient
```

```
Out[21]: 2
```

```
In [22]: reminder = dividend%divisor

         reminder
```

```
Out[22]: 1
```

```
In [23]: divisor*quotient+reminder
```

```
Out[23]: 7
```

El resultat és **ok**.

Un exemple de canviar el tipus d'un resultat amb l'operador `**`:

```
In [24]: 5 ** -2
```

```
Out[24]: 0.04
```

Exercici 2.2.8 *Comprova les operacions suma, resta, multiplicació i potència*

Donats: x = 13, y = 5

*Calcula: a = x + y, b = x - y, c = x * y, d = x ** y*

Per calcular la potenciació podem fer servir la funció `pow`.

Comprova: e = pow(x,y)

Exercici 2.2.9 *Operador divisió*

Donats: x = 17, y = 6

Calcula i comenta el resultat: a = x/y, b = x//y, c = int(x)//int(y), d = int(x)%int(y)

Exercici 2.2.10 *Comprova les operacions divisió entera i residu:*

```
1 c = 38//6
2
3 r = 38%6
4
```

```
5 print('La␣part␣entera␣del␣quocient␣és:␣', c)
6
7 print('El␣residu␣és:␣', r)
```

Exercici 2.2.11 *La funció* `divmod` *ens permet obtenir el mateix resultat. Comprova:*

```
1 c, r = divmod(38, 6)
2
3 print(c, r)
```

És possible operar amb diferents tipus:

- Enter i coma flotant: resultat, coma flotant

```
In [25]: 2 * 3e4
Out[25]: 60000.0
```

- Enter i complex: resultat, complex

```
In [26]: integer_var * complex_var
Out[26]: (6+2j)
```

- Coma flotant i complex: resultat, complex

```
In [27]: 8 * 3j
Out[27]: 24j
```

És possible forçar el tipus utilitzant un *casting*:

```
In [28]: int(3.14)
Out[28]: 3
In [29]: float(43)
Out[29]: 43.0
In [30]: complex(3)
Out[30]: (3+0j)
In [31]: complex(4.23)
Out[31]: (4.23+0j)
```

Però no és possible forçar el tipus d'un complex a un altre tipus numèric:

```
In [32]: float(4 + 0j)
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-32-567fbb0947cd> in <module>()
----> 1 float(4 + 0j)

TypeError: can't convert complex to float
```

```
In [33]: int(2 + 0j)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-33-f5d8093ffa9e> in <module>()
----> 1 int(2 + 0j)


TypeError: can't convert complex to int
```

Si un complex s'ha de convertir al quadrat del valor absolut, i després llegir la part real:

```
In [34]: complex_var
```

```
Out[34]: (1.5+0.5j)
```

Primer, el conjugat complex del `complex_var` és calculat:

```
In [35]: complex_var_conj = complex_var.conjugate()

         complex_var_conj
```

```
Out[35]: (1.5-0.5j)
```

En aquest cas, el conjugat té parèntesis `()` que volen dir que és una funció associada amb el tipus de complex.

Aleshores el quadrat del valor absolut de la variable complexa és calculat: $\left(|Complex|^2 = Complex \cdot Complex^*\right)$:

```
In [36]: absolute_square_var = complex_var*complex_var.conjugate()

         absolute_square_var
```

```
Out[36]: (2.5+0j)
```

Aleshores la part real s'obté:

```
In [37]: absolute_square_var.real
```

```
Out[37]: 2.5
```

El resultat és ja un valor de coma flotant.

Exercici 2.2.12 *Càlcul amb diferents formats*

Calcula indicant el tipus de variable:

*a = 3 * 4*

*b = 3 * (5 + 2j)*

*c = 3,2 * (5 + 2j)*

Exercici 2.2.13 *Canvi de tipus:*

a) Calcula la part entera de 5,01.

b) Transforma a float el nombre enter 5.

c) Recorda: no podem convertir un complex a float. Comprova-ho amb: 3 + 6j.

Exercici 2.2.14 *Calcula el mòdul del nombre complex: 3 - 4j.*

2.2.3 Cadenes

Les cadenes s'utilitzen per representar el text. Les diferents lletres es codifiquen en el que es coneix com a *characters*.

2.2.3.1 Creant una cadena

Diferents sintaxis es poden utilitzar per crear una cadena:

- Cometes simples:

```
In [38]: s = 'Hello, how are you?'

         s

Out[38]: 'Hello, how are you?'
```

- Cometes dobles:

```
In [39]: s = "Hi, what's up?"

         s

Out[39]: "Hi, what's up?"
```

- Cometes triples:

Permeten expandir el text en diferents línies, però s'ha de tenir una cura especial amb el format:

```
In [40]: s = '''Hello,
             how are you?'''

         s

Out[40]: 'Hello,\n    how are you?'
```

En aquest cas, hem inclòs implícitament caràcters de format, com la nova línia `\n`. El resultat es pot veure utilitzant la funció `print`:

```
In [41]: print(s)

Hello,
    how are you?
```

Els espais abans de *how* s'han inclòs automàticament, i molt probablament això no es volia. Una altra opció és eliminar els espais:

```
In [42]: s = """Hi,
         what's up?"""

         print(s)

Hi,
what's up?
```

Però en aquest cas, el codi Python no és fàcil de llegir.

Tres vegades cometes dobles (""") i tres vegades cometes simples (''') són correctes sintàcticament a Python. Però, les cometes dobles són recomanades per a documentació (també conegut com *docstrings*), encara que només sigui per a una línia.

Nota. *Docstrings* són comentaris immediatament a continuació de la definició d'un mòdul, funció, classe o mètode, que n'expliquen l'ús i propòsit.

També és possible incloure explícitament caràcters de format:

```
In [4]: s = "Hi guys\nHow are you?\na\tWe are fine! Thanks!"

        type(s)
```

```
Out[4]: str
```

En aquest cas, hem utilitzat la nova línia de caràcter `\n`, i el caràcter tabulat `\t`. El text resultant és:

```
In [5]: print(s)
```

```
Hi guys
How are you?
a        We are fine! Thanks!
```

I també dividim les cadenes entre línies:

```
In [45]: s = "Hi, " \
             "what's up?"

         print(s)
```

```
Hi, what's up?
```

I les barregem:

```
In [46]: s = "Hi,\n" \
             "what's up?"

         print(s)
```

```
Hi,
what's up?
```

2.2.3.2 Indexant una cadena

Es pot accedir als diferents caràcters dins una cadena utilitzant el que s'anomena *indexat*:

```
In [47]: a = "hello"
```

L'índex s'introdueix dins els claudàtors `[]`:

```
In [48]: a[0]
```

```
Out[48]: 'h'
```

Important: el primer caràcter és el que està indexat 0. A Python, els índexs sempre comencen amb el valor 0.

```
In [49]: a[1]

Out[49]: 'e'
```

El darrer caràcter és el quart índex:

```
In [50]: a[4]

Out[50]: 'o'
```

Si intentem accedir al cinquè índex, el resultat pot ser un error:

```
In [51]: a[5]

---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)

<ipython-input-51-b6a934feab86> in <module>()
----> 1 a[5]


IndexError: string index out of range
```

Això indica que hem intentat accedir a un índex incorrecte.

És possible accedir al darrer caràcter directament, només utilitzant l'índex -1.

```
In [52]: a[-1]

Out[52]: 'o'
```

Aquesta notació es pot utilitzar fins a assolir l'índex 0, que en aquest cas seria el -5:

```
In [53]: a[-5]

Out[53]: 'h'
```

Pregunta: Què passaria si intentéssim accedir a l'índex -6?
Breument, l'índex negatiu correspon a comptar des del final de la dreta.

Exercici 2.2.15 *Ús de les cometes:*

a) Escriu entre cometes simples, dobles i triples la cadena de caràcters: "Escric una cadena".

b) Cerca el tipus en cada cas.

Exercici 2.2.16 *Les cometes dobles permeten introduir en el text paraules "entre cometes".*

Escriu la frase anterior, indicant el tipus.

Exercici 2.2.17 *Ús de les cometes triples:*

a) Escriu entre cometes triples una cadena de més d'una línia (no més de cinc paraules per línia).

b) Repeteix-ho tal com s'ha indicat a l'exercici 2.2.15.

c) Comenta el resultat obtingut.

2.2.3.3 *Slicing*

De vegades volem llegir parts d'un text. Això es conegut com a *slicing*:

```
In [54]: a = "hello, world!"
```

Per exemple, volem accedir als elements 3, 4, 5. Aleshores els hem d'escriure com:

```
In [55]: a[3:6]
```

```
Out[55]: 'lo,'
```

Important: el darrer índex és sempre el previ al requerit.

És possible preguntar la longitud de la subcadena:

```
In [56]: len(a[3:6])
```

```
Out[56]: 3
```

El primer índex i el darrer ho poden ser implícitament, si no hi estan inclosos:

```
In [57]: a[:4]
```

```
Out[57]: 'hell'
```

```
In [58]: a[4:]
```

```
Out[58]: 'o, world!'
```

Podem preguntar també pels caràcters a les posicions senars. Això es fa afegint dos punts més i un nombre després, en aquest cas 2:

```
In [59]: a[::2]
```

```
Out[59]: 'hlo ol!'
```

O des del quart caràcter, cada tres:

```
In [60]: a[4::3]
```

```
Out[60]: 'owl'
```

Els accents i caràcters especials també es poden fer a mà amb cadenes Unicode (veges https://docs.python.org/3.4/howto/unicode.html).

Exercici 2.2.18 *Indexació de cadenes de caràcters*

Donat: a = "Volem tenir accés a uns caràcters determinats"

a) Llegeix els caràcters de 3 a 20 ambdós inclosos.

b) Llegeix els caràcters de 0 a 24 ambdós inclosos de 3 en 3.

c) Escriu els onze primers caràcters.

d) Escriu els caràcters des de l'onzè.

Exercici 2.2.19 *Accés a la cadena amb un índex negatiu*

Com ja sabem, podem comptar des del final.

Donat: a = "Volem tenir accés a uns caràcters determinats"

a) Escriu l'últim caràcter.

b) Escriu els onze últims caràcters.

c) Escriu els caràcters compresos entre els onze primers i els onze últims.

2.2.3.4 Immutabilitat

Una cadena és un **objecte immutable** i no se'n poden modificar els continguts.

```
In [61]: a = "hello, world!"

         a

Out[61]: 'hello, world!'

In [62]: a[2] = 'z'

        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-62-d57c4312feba> in <module>()
    ----> 1 a[2] = 'z'


        TypeError: 'str' object does not support item assignment
```

Es pot però crear noves cadenes des de l'original. Podem substituir la primera aparició d'un caràcter:

```
In [63]: b = a.replace('l', 'z', 1)

         b

Out[63]: 'hezlo, world!'

In [64]: a

Out[64]: 'hello, world!'
```

O totes les aparicions:

```
In [65]: a.replace('l', 'z')

Out[65]: 'hezzo, worzd!'
```

Les cadenes tenen molts mètodes útils, com ara `a.replace` com acabem de veure. Recorda la notació orientada a objecte `a.` i l'ús del tabulador final o `help(str)` per cercar nous mètodes.

Python ofereix possibilitats avançades per manipular cadenes, buscant patrons o formatant. Si hi estàs interessat: https://docs.python.org/3.4/library/stdtypes.html#text-sequence-type-str i https://docs.python.org/3.4/library/string.html#formatstrings

2.2.3.5 Conversió de cadenes

Les cadenes es poden convertir a altres tipus, igual que els tipus numèrics:

```
In [66]: int('1')

Out[66]: 1
```

Però han de seguir el format correcte:

```
In [67]: int('1.')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-67-f18b6136395b> in <module>()
----> 1 int('1.')

ValueError: invalid literal for int() with base 10: '1.'
```

Altres exemples:

```
In [68]: complex('3.+4j')

Out[68]: (3+4j)

In [69]: complex('3. + 4j')

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-69-84986758f0fb> in <module>()
----> 1 complex('3. + 4j')

ValueError: complex() arg is a malformed string
```

Això ho podem solucionar utilitzant la funció de substitució:

```
In [70]: complex('3. + 4j'.replace(' ', ''))

Out[70]: (3+4j)
```

Nota. En aquest cas, hem aplicat la funció `replace` a la cadena directament, però també podem assignar la cadena a una variable i operar després. El darrer és el preferit.

2.2.3.6 Substitució de cadenes

És possible convertir una variable en una cadena directament:

```
In [71]: str(4j)

Out[71]: '4j'
```

Però el mètode preferit és utilitzant la substitució de la cadena i formatant, ja que controla millor la cadena resultant. En són exemples bàsics:

```
In [72]: 'An integer: {}; a float: {}; another string: {}'.format(1, 0.1, 'string')

Out[72]: 'An integer: 1; a float: 0.1; another string: string'

In [73]: i = 102

In [74]: filename = 'processing_of_dataset_{}.txt'.format(i)

In [75]: filename

Out[75]: 'processing_of_dataset_102.txt'
```

Es pot aplicar un format més complex, però en parlarem a entrades i sortides.

Exercici 2.2.20 *Accedir a cadenes de caràcters*

Donat: a = "Volem tenir accés a uns caràcters determinats"

a) Comprova que en podem calcular la longitud fent:

```
1 print(len(a))
```

b) Comprova que el caràcter que ocupa el lloc cinquè és un espai fent:

```
1 p = a[4]
2
3 b = a[5]
4
5 q = a[6]
6
7 print(p, b, q)
8
9 print('Aquest␣caràcter␣és:', b)
```

c) Substitueix les "e" per "u".

Exercici 2.2.21 *Converteix en cadena:*

a) 3 + 4j

b) -0,305

c) El preu era 3,5 euros.

2.2.4 Dades binàries

Els ordinadors treballen sempre amb dades booleanes (`1` i `0`), també anomenades *bits*, i operacions booleanes (`and`, `or` i `not`), també anomenats *operadors lògics*. Com a resultat, per codificar nombres i cadenes, és necessari definir com agrupar els booleans en paquets que permetin codificar més informació.

Com a exemple:

1 bit (booleà) pot emmagatzemar dos valors: (`1` i `0`).

2 bits poden emmagatzemar quatre valors: (`11`, `10`, `01` i `00`).

3 bits poden emmagatzemar vuit valors: (`111`, `110`, `101`, `100`, `011`, `010`, `001` i `000`).

I així. El nombre de valors (o símbols) es pot determinar mitjançant la fórmula:

$$m = 2^n$$

On `n` és el nombre de bits i `m` el nombre de símbols que es poden representar.

2.2.4.1 *Integers*

Ara tenim el problema de com codificar un enter com un nombre binari. Això es pot fer fàcilment utilitzant el procediment següent amb un nombre com el 7. Primer el dividim per 2 (ja que podem codificar dos valors amb un bit):

$$\frac{7}{2} \rightarrow 3$$

Per tant:

$$7 \rightarrow 3 \cdot 2 + 1$$

El romanent és 1. Aquest nombre serà el bit menys significant o el bit 0. El 3 resultant ara s'haurà també de dividir:

$$\frac{3}{2} \rightarrow 1$$

Per tant:

$$3 \rightarrow 1 \cdot 2 + 1$$

Aquest romanent 1 serà el següent bit o bit 1. Ara podem dividir altre cop el resultat (1):

$$\frac{1}{2} \rightarrow 0$$

Per tant:

$$1 \rightarrow 0 \cdot 2 + 1$$

Aquest romanent 1 serà el següent bit o bit 2. El nombre resultant és:

b_2	b_1	b_0
1	1	1

Pregunta: Quina serà la representació binària de 5? I de 4?

Els nombres més grans poden ser representats utilitzant un nombre més gran de bits. Python permet calcular-los automàticament. Per exemple, el nombre 234 és representat pel nombre:

```
In [76]: bin(234)
```

```
Out[76]: '0b11101010'
```

El resultat és una cadena de Python que inclou el prefix '0b' per indicar que és binari.

El valor en aquest cas s'ha de representar per un grup de, com a mínim, 8 bits:

b_7	b_6	b_5	b_4	b_3	b_2	b_1	b_0
1	1	1	0	1	0	1	0

Un grup de 8 bits s'anomena *byte*, i pot representar:

$$\left(2^8 \rightarrow 256\right)$$

El rang va del nombre 0 al 255. La dificultat més gran és que aquests nombres només són positius. Si el nombre és negatiu, Python afegeix un signe abans del nombre per representar-lo. Per exemple:

```
In [77]: bin(-234)
```

```
Out[77]: '-0b11101010'
```

Pregunta: Quina serà la representació binària de -9? I de -15?

La representació d'aquesta codificació dins l'ordinador es basa en el complement a dos. Però en aquesta assignatura no hi entrarem.

Per simplificar la representació és possible utilitzar el format hexadecimal. En aquest cas els bits s'agrupen de 4 en 4. Com que 4 bits poden representar 16 símbols, els 10 dígits no són suficients, i per aquesta raó, s'afegeixen lletres:

dec	bin	hex
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	a
11	1011	b
12	1100	c
13	1101	d
14	1110	e
15	1111	f

Com a resultat, el nombre 234 és 0b11101010 i en hexadecimal és:

1110 és e

1010 és a

Per tant, 234 és 0xea (*0x* indica que és un nombre hexadecimal).

La funció `hex` fa la conversió:

```
In [78]: hex(234)
```

```
Out[78]: '0xea'
```

2.2.4.2 Nombres reals

En el cas dels nombres reals o nombres de coma flotant, Python utilitza la representació IEEE 754. Aquest estàndard permet representar nombres entre:

$$\pm 4.9406564584124654 \cdot 10^{-324} \leq x \leq \pm 1.7976931348623157 \cdot 10^{308}$$

Un nombre menor que $\pm 4.9406564584124654 \cdot 10^{-324}$ és considerat 0 amb el mateix signe, i un de major és considerat un infinit:

```
In [79]: -1e-325
```

```
Out[79]: -0.0
```

```
In [80]: -1e309
```

```
Out[80]: -inf
```

Els nombres també es poden mostrar en format hexadecimal utilitzant la funció **`float.hex()`**:

```
In [81]: float.hex(1.4e5)
```

```
Out[81]: '0x1.1170000000000p+17'
```

Nota que l'exponent s'escriu en decimal en lloc d'hexadecimal, i que dona la potència de 2 per la qual es multiplica el coeficient. Per exemple, la cadena hexadecimal 0x1.117p+17 representa el nombre en coma flotant:

$$x = \left(1 \cdot 16^0 + 1 \cdot 16^{-1} + 1 \cdot 16^{-2} + 1 \cdot 16^{-3}\right) \cdot 2^{17} = 14000 = 1.4 \cdot 10^5$$

Tots els dígits després de la coma s'han multiplicat per la potència de 16 corresponent. Aquesta notació només és utilitzada quan es requereix una precisió elevada en la representació. En els altres casos, es prefereix la notació decimal.

Exercici 2.2.22 *Converteix de natural a binari el nombre 25.*

Exercici 2.2.23 *Reconverteix el resultat anterior.*

Exercici 2.2.24 *Converteix a hexadecimal i a binari el nombre 325.*

Exercici 2.2.25 *Reconverteix el 0x145 a decimal i a binari.*

2.3 Operadors

Els operadors permeten dur a terme diferents accions amb les variables. En general, són símbols especials que permeten fer operacions específiques amb un, dos o tres operands, i retornar el resultat associat. Com que els diferents operadors es poden encadenar, és important conèixer quin ordre se segueix per aplicar-los (precedència).

A continuació es detallen els diferents operadors.

2.3.1 Operacions bàsiques (aritmètiques)

Python implementa les sis operacions aritmètiques bàsiques amb els símbols següents:

- Suma: +
- Resta: -
- Producte *
- Divisió: /
- Exponenciació: **
- Mòdul: % (residu de la divisió)

Aquestes operacions requereixen dos operands i retornen un resultat numèric el tipus del qual depèn dels anteriors, com hem vist en el tema anterior. El programa següent conté exemples d'ús d'aquests operadors:

```
In [1]: # Definim dues variables de treball
        x = 10
        y = 20

        # Operacions aritmètiques bàsiques
        print(x+y)  # Suma
        print(x-y)  # Resta
        print(x*y)  # Producte
        print(x/y)  # Divisió
        print(x**y) # Exponenciació
        print(y%x)  # Mòdul
```

```
30
-10
200
0.5
100000000000000000000
0
```

Exercici 2.3.1 *Fes un programa que resolgui equacions de segon grau*

$$ax^2 + bx + c = 0$$

Solucions:

$$\begin{aligned} x_1 &= \frac{-b+\sqrt{b^2-4ac}}{2a} \\ x_2 &= \frac{-b-\sqrt{b^2-4ac}}{2a} \end{aligned}$$

Nota. *Per calcular arrels quadrades en Python usa l'expressió d'aquestes com una potència* $\sqrt{x} \equiv x^{\frac{1}{2}}$

Exercici 2.3.2 *Una implementació alternativa de les solucions d'una equació de segon grau és la solució quadràtica:*

$$q \equiv -\frac{1}{2}\left[b + sgn(b)\sqrt{b^2 - 4ac}\right]$$
$$x_1 = \frac{q}{a}$$
$$x_2 = \frac{c}{q}$$

Aquesta solució és numèricament més estable quan s'usen valors de coma flotant. Implementa-la i compara els resultats amb l'anterior.

Exercici 2.3.3 *Un cop fets els dos exercicis anteriors, calcula:*

En l'equació $x^2 - 12x + c = 0$ *troba* c *de manera que una arrel sigui el doble de l'altra.*

Exercici 2.3.4 *Quin és el nombre la tercera part més 5 del qual dona 62?*

Exercici 2.3.5 *Donada l'equació* $2x^3 - 3x + c = 0$*, determina* c *quan una de les seves arrels és 2.*

2.3.2 Particularitats de la divisió d'enters

Quan s'usa l'operador de divisió "/" amb enters el resultat en general no és un enter. En aquest cas Python converteix el resultat en un *float* en comptes de retornar un enter.

```
In [2]: x = 4
        y = 3
        print(x/y)
```

```
1.3333333333333333
```

Nota important. En versions anteriors de Python el comportament de la divisió d'enters no era aquest. El resultat era un enter i podies trobar casos com $4/3 = 1$. Ves amb compte amb aquesta peculiaritat si uses versions anteriors de Python o altres llenguatges de programació com ara Java.

Experimenta amb la divisió i els altres operadors amb els propis exemples.

2.3.3 Operacions addicionals

El signe "-" davant d'un valor serveix per canviar-li el signe:

```
In [3]: x = 10
        y = -x
        print(y)
```

```
-10
```

L'operador "//" és la **divisó d'enters**. En aquest cas Python sempre arrodoneix el resultat al valor inferior més proper.

```
In [4]: # Divisió normal
        a = 1
```

```
b = 2
c = a/b
print(c)
print(type(c))

# Divisió entera
d= a//b
print(d)
print(type(d))

# Divisió normal
a = -1
b = 2
c = a/b
print(c)
print(type(c))

# Divisió entera
d= a//b
print(d)
print(type(d))
```

```
0.5
<class 'float'>
0
<class 'int'>
-0.5
<class 'float'>
-1
<class 'int'>
```

Exercici 2.3.6 *Calcula:* $(-1)^{20}$; -1^{20}; $(-1)^{21}$; -1^{21}.

Exercici 2.3.7 *Calcula:* $(-2)^0$; -2^0, $-2^2-(-2)^0$.

2.3.4 Operacions a nivell de bit

Aquestes operacions treballen amb les variables tractant el seu contingut a nivell de bit. Solen usar-se en programes que treballen a baix nivell, tot i que ocasionalment poden ser útils per a programes científics.

- and: &
- or: |
- xor: ^
- not: ~
- Desplaçament a l'esquerra: <<
- Desplaçament a la dreta: >>

A continuació se'n detalla el comportament.

2.3.4.1 `And`

Fa un `and` lògic bit a bit.

$$\begin{aligned} 0\&0 &= 0 \\ 0\&1 &= 0 \\ 1\&0 &= 0 \\ 1\&1 &= 1 \end{aligned}$$

```
In [5]: a = 1 # 00000001
        b = 1 # 00000001
        c = 2 # 00000010
        print(bin(a & b)) # 00000001
        print(bin(a & c)) # 00000000
```

```
0b1
0b0
```

2.3.4.2 `Or`

Fa un `or` lògic bit a bit.

$$\begin{aligned} 0|0 &= 0 \\ 0|1 &= 1 \\ 1|0 &= 1 \\ 1|1 &= 1 \end{aligned}$$

```
In [6]: a = 1 # 00000001
        b = 1 # 00000001
        c = 2 # 00000010

        print(bin(a | b)) # 00000001
        print(bin(a | c)) # 00000011
```

```
0b1
0b11
```

2.3.4.3 `Xor`

Fa un `or` exclusiu bit a bit.

$$\begin{aligned} 0\overline{0} &= 0 \\ 0\overline{1} &= 1 \\ 1\overline{0} &= 1 \\ 1\overline{1} &= 0 \end{aligned}$$

```
In [7]: a = 1 # 00000001
        b = 1 # 00000001
        c = 2 # 00000010

        print(bin(a^b)) # 00000000
        print(bin(a^c)) # 00000011
```

```
0b0
0b11
```

2.3.4.4 `Not`

Fa una negació bit a bit d'un valor.

$$\begin{aligned} \sim 0 &= 1 \\ \sim 1 &= 0 \end{aligned}$$

Nota que en aquest exemple es veu l'efecte de la representació dels enters en complement a 2.

```
In [8]: a = 1 # 00000001
        b = 2 # 00000010

        print(bin(~a)) # 11111110
        print(bin(~b)) # 11111101

        # Representació fent servir una màscara
        print(bin(~a & 0xff)) # 11111110
        print(bin(~b & 0xff)) # 11111101
```

```
-0b10
-0b11
0b11111110
0b11111101
```

2.3.4.5 Desplaçament

Desplaça n bits a la dreta (>>) o l'esquerra (<<).

Matemàticament seria com:

$$a << n \Rightarrow a \cdot 2^n$$
$$a >> n \Rightarrow \frac{a}{2^n}$$

Nota. Els bits de l'extrem es perden.

```
In [9]: a = 1 # 00000001
        b = 2 # 00000010

        print(a << 1) # 00000010
        print(b >> 1) # 00000001

        print(a << 2) # 00000100
        print(b >> 2) # 00000000
```

```
2
1
4
0
```

Exercici 2.3.8 *Càlcul amb operadors binaris*

Donats:

```
1 a = 1
2 b = 2
3 c = 1
```

Calcula expressant el resultat com a nombre enter i com a binari:

a) `a&b`

b) `a&c`

c) `a|b`

d) `a|c`

e) `a&(b|c)`

f) `a|(b&c)`

g) `b&(a|c)`

h) `~(a&b)`

i) `~(~b)`

j) `~c|(~(a&b))`

k) `~(a^b)`

l) `~a^~b`

m) `~c|~(a^b)`

Comenta els resultats.

Exercici 2.3.9 *Desplaçaments*

Donats:

```
1 a = 1
2 b = 2
3 c = 3
```

Calcula expressant el resultat com a nombre enter i com a binari:

a) `a<<1`

b) `~a<<1`

c) `(c|~a)>>2`

d) `b<<2`

e) `b>>2`

Comenta els resultats.

2.3.5 Operacions amb cadenes (*strings*)

És important tenir en compte que les operacions anteriors no es poden aplicar a cadenes (valors tipus *string*), encara que el seu contingut sigui la transcripció d'un nombre. Pots veure com l'exemple següent dona un error quan s'executa:

```
In [10]: cadena = "1.0"

         print(cadena - 1)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)

<ipython-input-10-840b66641cd6> in <module>
  1 cadena = "1.0"
  2
----> 3 print(cadena - 1)


TypeError: unsupported operand type(s) for -: 'str' and 'int'
```

Si tens una cadena que conté un text que representa un nombre i vols fer operacions amb ell, cal transformar la cadena abans de poder fer-ho:

```
In [11]: cadena="1.0"
         valor= float(cadena)

         print(valor-1)

         cadena2= "1"
         valor2= int(cadena2)
         print(valor2-1)
```

```
0.0
0
```

Ves amb compte, però, perquè si la cadena no representa correctament un valor numèric **simple**, si intentes transformar-la et donarà un error.

```
In [12]: cadena= "1+2"
         valor= int(cadena)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)

<ipython-input-12-9595cff600aa> in <module>
  1 cadena= "1+2"
----> 2 valor= int(cadena)
```

```
ValueError: invalid literal for int() with base 10: '1+2'
```

Tanmateix, l'operador suma sí que es pot aplicar a cadenes, però Python no l'interpreta com una operació aritmètica sinó com una concatenació:

```
In [13]: cadena1= "ABC"
         cadena2= "DEF"

         print(cadena1+cadena2)
```

```
ABCDEF
```

De forma similar, l'operador multiplicació aplicat a cadenes s'interpreta com una repetició:

```
In [14]: cadena= "12345 "

         print(cadena*5)
```

```
12345 12345 12345 12345 12345
```

Exercici 2.3.10 *Operadors de cadenes*

Donades les cadenes:

```
1 a = 'Podem␣operar␣amb␣cadenes'
2 b = 'Primer␣hem␣de␣llegir␣els␣exemples'
```

Calcula:

1. *a+b*
2. *a[:6]*
3. *b[14:]*
4. *a[:6]+b[14:]*
5. *5*a[:6]*

Exercici 2.3.11 *Operadors de cadenes amb representacions numèriques*

Donades les cadenes:

```
1 a = '3.0'
2 b = '2'
```

Calcula la suma, diferència, producte i quocient dels seus valors.

2.3.6 Operacions booleanes (lògiques)

Permeten fer comparacions de valors o avaluació de condicions més complexes sobre un conjunt de valors. El resultat és un valor booleà *True* o *False*.

Seran especialment útils quan introduïm les comandes de control de flux. Permetran condicionar l'execució d'un programa en funció dels valors dels seus resultats. De moment aprendrem com funcionen.

2.3.6.1 Operadors relacionals

Els operadors d'igualtat i relacionals permeten comparar els operands. La major part dels operadors són semblants als utilitzats en matemàtiques:

Operador	Funció
x == y	**Igualtat:** retorna *True* si el valor x és igual al valor y. Altrament retorna *False*.
x != y	**Desigualtat:** retorna *False* si el valor x és igual al valor y. Altrament retorna *True*.
x > y	**Major:** retorna *True* si el valor x és més gran que el valor y. Altrament retorna *False*.
x < y	**Menor:** retorna 'True' si el valor x és més petit que el valor y. Altrament retorna 'False'.
x >= y	**Major o igual:** retorna *True* si el valor x és més gran o igual que el valor y. Altrament retorna *False*.
x <= y	**Menor o igual:** retorna *True* si el valor x és més petit o igual que el valor y. Altrament retorna *False*.

Alguns exemples de la seva utilització són:

```
In [15]: a = 1
         b = 1
         c = 2

         print("a == b",a==b)
         print("a != b",a!=b)
         print("a == c",a==c)
         print("a != c",a!=c)
         print("a > b",a>b)
         print("a >= b",a>=b)
         print("a < c",a<c)
         print("a <= c",a<=c)
```

```
a == b True
a != b False
a == c False
a != c True
a > b False
a >= b True
a < c True
a <= c True
```

Exercici 2.3.12 *Introdueix un nombre enter pel teclat i comprova si és divisible per 5. Per això cal fer:*

```
1 a = int(input('escriu␣un␣nombre:' ))
```

```
2 if a\%5==0:
3   print('El␣nombre␣és␣divisible␣per␣5')
4 else:
5   print('El␣nombre␣no␣és␣divisible␣per␣5')
```

Exercici 2.3.13 *Vist el procediment en l'exercici anterior, escriu tres nombres enters, introduint-los pel teclat i comprova, suposant que estan expressats en les mateixes unitats, si podem construir o no un triangle.*

Exercici 2.3.14 *Escriu un nombre enter de dues xifres, introduint-lo pel teclat, i comprova si la suma d'aquestes és parella o senar.*

2.3.6.2 Operadors lògics

Serveixen per combinar diverses expressions lògiques que donen al final un valor *True* o *False*. Permeten formar condicions complexes a partir de diverses comparacions individuals.

Operador	Funció
and	**I lògic:** retorna *True* si els dos valors booleans als quals s'aplica són *True*. Altrament retorna *False*.
or	**O lògic:** retorna *True* si almenys un dels dos valors booleans als quals s'aplica és *True*. Altrament retorna *False*.
not	**Negacio:** canvia el valor booleà al qual s'aplica pel seu oposat.

2.3.6.2.1 *And*

Opera de forma semblant a l'**and** binari, però en aquest cas sobre un booleà. En aquest cas, l'expressió fa servir la paraula, i no un símbol.

True **and** *True*	*True*
True **and** *False*	*False*
False **and** *True*	*False*
False **and** *False*	*False*

```
In [16]: a = 0
         b = 1

         # Expressions boleanes: and
         print("a==0 and b==1", a==0 and b==1)
         print("a==0 and b==0", a==0 and b==0)
         print("a>=0 and a<=2", a>=0 and a<=2)
         print("a>=0 and a<=2 and b>1", a>=0 and a<=2 and b>1)
```

```
a==0 and b==1 True
a==0 and b==0 False
a>=0 and a<=2 True
a>=0 and a<=2 and b>1 False
```

2.3.6.2.2 *Or*

Equivalent al cas anterior, però per a `or`.

True **or** *True*	*True*
True **or** *False*	*True*
False **or** *True*	*True*
False **or** *False*	*False*

```
In [17]: a = 0
         b = 1

         # Expressions booleanes: or
         print("a==0 or b==1", a==0 or b==1)
         print("a==0 or b==0", a==0 or b==0)
         print("a>=0 or a<=2", a>=0 or a<=2)
         print("a==2 or b==2", a==2 or b==2)
         print("a>=0 or a<=2 or b>1", a>=0 or a<=2 or b>1)
```

```
a==0 or b==1 True
a==0 or b==0 True
a>=0 or a<=2 True
a==2 or b==2 False
a>=0 or a<=2 or b>1 True
```

2.3.6.2.3 *Not*

Negació de l'expressió anterior.

not *True*	*False*
not *False*	*True*

```
In [18]: a = 0
         b = 1

         # Expressions booleanes: not
         print("not a==0", not a==0)
         print("not a==1", not a==1)
```

```
not a==0 False
not a==1 True
```

Notes:

- `or` només avalua el segon argument si el primer és fals (operador curt-circuit).
- `and` només avalua el segon argument si el primer és cert (operador curt-circuit).
- `not` té menys prioritat que els operadors no booleans, de manera que $not\ a == b$ s'interpreta com $not(a == b)$, i $a == not\ b$ és un error de sintaxi si bé no és booleà.

2.3.6.3 Operacions compostes: compte amb l'ordre d'avaluació!

Els operadors de comparació i lògics es poden combinar per formar sentències lògiques complexes, però **cal anar amb compte amb l'ordre d'avaluació perquè pot canviar el resultat**.

Es poden usar parèntesis per especificar com s'ha d'avaluar l'expressió.

```
In [19]: a = 0
         b = 1

         # Compte amb l'ordre d'avaluació dels operadors
         print("not a==1 and b==1",not a==0 and b==0)
         print("not (a==1 and b==1)",not (a==0 and b==0))
         print("(not a==1) and b==1",(not a==0) and b==0)
```

```
not a==1 and b==1 False
not (a==1 and b==1) True
(not a==1) and b==1 False
```

En resum, Python proporciona un conjunt d'operadors que permeten fer operacions amb els diferents tipus de dades. És important tenir cura dels tipus de dades que fem servir per no tenir errors.

Exercici 2.3.15 *Importa la biblioteca* `math` *on hi ha la funció* `sqrt` *que permet calcular arrels quadrades.*

Calcula la hipotenusa d'un triangle rectangle de catets 3 i 4 expressats en la mateixa unitat, u.

Exercici 2.3.16 *Calcula la resistència d'un fil de coure d'un metre de llarg i 0,5 mm de diàmetre. La resistivitat del coure és: 1,6E-6 ohms/cm. Especifica amb claredat les variables i les constants.*

Exercici 2.3.17 *Defineix una funció que permeti calcular la superfície d'un rombe i calcula la superfície d'un rombe de diagonals: $D = 8$ m i $d = 5$ m.*

2.4 Codis d'estil

En aquest document llistem les diferents convencions utilitzades en Python a l'hora de programar. Es pot trobar més informació en la *Python Developer's Guide*. Python és l'exemple més clar a l'hora d'implementar el que es coneix com a ***programació estructurada***.

2.4.1 Programació estructurada

La programació estructurada és un paradigma de programació orientada a millorar la claredat, qualitat i temps de desenvolupament en la realització d'un programa. Programar de forma estructurada facilita la lectura de codi i la seva reutilització.

2.4.1.0.1 Indentació

La indentació consisteix a introduir ***quatre espais*** per delimitar l'estructura del programa, fet que permet establir blocs de codi. La indentació en Python és **obligatòria**. Un codi no indentat genera l'error següent:

```
  File "<ipython-input-1-35a3266bacb1>", line 3
    print(n)
        ^
IndentationError: expected an indented block
```

Però, que vol dir la sentència "delimitar l'estructura del programa, fet que permet establir blocs de codi"?

El significat d'aquesta sentència és que un conjunt d'instruccions determinat que trobem indentat pertany o s'executa depenent del que hi hagi a la instrucció anterior. Vegem-ne un exemple:

```
1 comptador = 0
2 variable = 1
3 mentre comptador < 5
4     Si variable < 3 llavors
5         variable = variable + 1
6         comptador = comptador + 1
7     En cas contrari
8         variable = 1
9         print(variable)
```

Tal com es veu en aquest pseudocodi, les dues comparacions s'executaran sempre que el valor de la variable `comptador` sigui menor que 5. Dintre d'aquesta estructura tenim dues comparacions: `Si variable < 3 llavors` i `En cas contrari`. El codi que ve a continuació només s'executarà si la comparació és correcta.

2.4.2 Mida màxima de la línia

La mida màxima de les línies és 79 caràcters.

```
In [14]: print('introdueix el valor màxim per executar la \
          seqüència anterior')
```

```
introdueix el valor màxim per executar la seqüència anterior
```

Es pot fer servir la barra invertida (\) per indicar que la línia que estem llegint és la mateixa.

2.4.3 Importació de biblioteques

Quan es volen importar dues biblioteques o més, haurien de col · locar-se en línies diferents:

Correcte

```
1 import sys
2 import os
```

Incorrecte

```
1 import sys, os
```

Tot i que sí que és correcte importar parts d'una biblioteca d'aquesta forma:

```
1 from llibreria import A, B
```

Les importacions s'han de col · locar sempre a la part superior de l'arxiu o la cel · la on treballem, just després de qualsevol comentari o cadena de documentació del codi i just abans de la definició de variables i constants.

Has de garantir que fas servir les biblioteques que importes.

2.4.4 Comentaris

Un comentari és una construcció del llenguatge de programació destinada a introduir anotacions llegibles al programador en el codi font d'un programa informàtic. Aquestes anotacions són útils per als programadors ja que donen informació de què, quan i com s'executa un codi determinat. Són per tant afegits usualment amb el propòsit de fer el codi font més fàcil d'entendre pel que fa al manteniment o reutilització.

Per aquest motiu, un comentari que contradiu el codi és molt pitjor que no tenir cap comentari. Els comentaris han de ser frases completes i si és possible en anglès, per afavorir que pugui ser llegit per qualsevol persona independentment de la seva nacionalitat.

Tenim dos tipus de comentaris:

- Comentaris de bloc. Són aquells que s'apliquen al codi que es troba a continuació, i s'indenten al mateix nivell que el codi que s'està comentant. Cada línia d'un comentari de bloc comença amb un #.
- Comentaris de línia. Són els que es troben en la mateixa línia on tenim una sentència. Els comentaris en línia haurien de separar-se com a mínim per dos espais de la sentència que comenten. Haurien de començar amb un # i un espai.

Tot seguit es donen exemples de comentaris de bloc i comentaris de línia:

```
In [15]: # Aquest codi calcula el temps
         # i la velocitat final quan tenim
         # una caiguda lliure i mostra el
         # resultat en la consola

         g = -9.8
         alcada = float(input("Introduir l'alçada: "))
```

```
        v0 = float(input("Introduir la velocitat inicial: "))
        v = (v0**2 - 2*g*alcada)**0.5
        t = (-v0 + (v0**2 -2*g*alcada)**0.5)/alcada    # compte amb els valors i els signe
        print("La velocitat final és", v," i el temps és", t)
```

```
Introduir l'alçada: 3
Introduir la velocitat inicial: 2
La velocitat final és 7.92464510246358 i el temps és 1.9748817008211932
```

Utilitza els comentaris en línia de forma moderada. Aquests comentaris no són útils si indiquen coses òbvies i només serveixen per distreure el programador. No cal fer coses com aquesta:

```
1 x = x + 1     # incrementa el valor de x en 1
```

2.4.5 Cadenes de documentació

Les cadenes de documentació o ***docstrings*** permeten documentar funcions o classes per conèixer què són i què fan aquests elements. Un *docstring* s'inicia introduint tres cometes (simples o dobles), després s'hi insereix el text i finalitza de nou amb tres cometes.

Així, donada una funció:

```
1 def Funcio():
2     """
3     Aquí introduïm el comentari
4     que considerem oportú, definint
5     les característiques, el que fa
6     i com ho fa
7     """
8     return 0
```

Posteriorment, quan demanem informació sobre les funcions i classes s'obtindrà aquesta informació:

```
In [16]: def Funcio():
             """
             Funció exemple on introduïm les
             característiques principals.
             Normalment s'hi inclouen els paràmetres
             d'entrada i els paràmetres
             de sortida
             """
             return 0
         help(Funcio)
```

```
Help on function Funcio in module __main__:

Funcio()
    Funció exemple on introduïm les
    característiques principals.
    Normalment s'hi inclouen els paràmetres
```

```
    d'entrada i els paràmetres
    de sortida
```

2.4.5.1 Definició de variables i funcions

Defineix el nom de les variables de manera que indiquin el seu significat. Si la definició està formada per diverses paraules, la primera paraula escriu-la tota en minúscules i, tot seguit, sense espais, les paraules següents separades amb un guió baix (_).

Per exemple:

- **Correcte**

```
1 velocitat = 0
2 velocitat_inicial = 4    # variable definida en m/s
3 acceleracio = 2          # acceleració en m/s2
```

- *Incorrecte*

```
1 a = 0
2 b = 4
3 c = 2
```

Fer servir variables com a, b o c, que no donen informació, dificulta després la interpretació i l'enteniment del codi. No facis servir accents ni altres signes gràfics que puguin donar errors sintàctics.

Pel que fa a les funcions, es defineixen també amb minúscules separades amb guió baix (_) i preferentment amb un nom que indiqui l'acció. Així tindrem:

```
1 def modificar_variable(velocitat):
2     return velocitat*2
```

Igual que en el cas de les variables, no facis servir ni accents ni altres signes gràfics, ja que donaran errors sintàctics.

2.4.5.2 Definició de constants

A diferència d'altres llenguatges de programació, on es pot definir explícitament una constant, en Python no es poden definir. Simplement el que hem de fer és crear la variable i no modificar-la. Com podem saber que el que estem utilitzant és una constant? La regla d'estil ens diu que quan tenim una variable, la definim tota en majúscules. En cas que la definició estigui formada per més d'una paraula, separarem les paraules per un guió baix (_).

Exemple:

```
In [17]: # **********************
         # Definició de constants
         # **********************
         GRAVETAT = 9.8                # en m/s2
         K_BOLTZMANN = 1.380648E-23    # en J/K

         # **********************
```

```
# Definició de variables
# **********************
acceleracio = 1
temperatura = 300
velocitat_inicial = 0
distancia = 0
```

En resum, hem vist que hi ha un codi d'estil per a Python que manté una coherència en tot el programa. El punt més important és que el codi del programa ha de ser al més autoexplicatiu possible, de forma que una ullada en proporcioni la màxima informació possible.

Exercici 2.4.1 *Importa la biblioteca* `math` *on hi ha la funció* `sqrt` *que permet calcular arrels quadrades.*

Calcula la hipotenusa d'un triangle rectangle de catets 3 i 4 expressats en la mateixa unitat, u.

Exercici 2.4.2 *Calcula la resistència d'un fil de coure d'un metre de llarg i 0,5 mm de diàmetre. La resistivitat del coure és: 1.6E-6 ohms/cm. Especifica amb claredat les variables i les constants.*

Exercici 2.4.3 *Defineix una funció que permeti calcular la superfície d'un rombe i calcula la superfície d'un rombe de diagonals: $D = 8$ m i $d = 5$ m.*

2.5 Entrada i sortida

Els programes han d'interaccionar amb l'usuari per obtenir les entrades que necessiten i retornar els resultats. Per aquest motiu, els diferents llenguatges de programació proporcionen mètodes per poder intercanviar informació amb l'usuari. En aquest tema analitzarem les diferents alternatives que ens dona Python per intercanviar informació per consola o, en el nostre cas, l'IPython Notebook.

2.5.1 Sortida per pantalla

El contingut de les variables, literals, o el resultat d'operacions poden ser visualitzats per pantalla amb la funció **`print()`**. Ens centrarem en la versió de Python 3.x.

Comencem per alguns exemples amb cadenes de text:

```
In [1]: # Definim variables i els assignem valors del tipus cadena de text
        string_1 = "Aquest és un text estrany: "
        string_2 = "La vida és tan bonica..."

        # Impressió bàsica i concatenació
        print(string_1)
        print(string_2)
        print('-'*10)
```

```
Aquest és un text estrany:
La vida és tan bonica...
----------
```

També podem mostrar dues cadenes de text o més concatenant-les:

```
In [2]: print('Resultat concatenació: ' + string_1 + string_2)
```

```
Resultat de la concatenació: Aquest és un text estrany: La vida és tan bonica...
```

O podem indicar un nombre de cadenes de text separades per`,`. En aquest cas també podem indicar els caràcters separadors a mostrar per pantalla entre els diferents *strings* especificats. Per defecte, aquest és l'espai en blanc:

```
In [3]: #On a sep indiquem el caràcter separador
        #entre els diferents arguments del print
        print('Resultat sense concat.: ', string_1, string_2, sep='')
```

```
Resultat sense concat.: Aquest és un text estrany: La vida és tan bonica...
```

O també:

```
In [4]: print('Resultat sense concat.: ', string_1, string_2, sep='\n-----\n')
```

```
Resultat sense concat.:
-----
Aquest és un text estrany:
-----
La vida és tan bonica...
```

Podem combinar *strings* i valors numèrics de dues formes diferents:

1. `print (<text literal>, <valor numèric>)`
2. `contatenant dos strings: print (<text literal> + str(<valor numèric>)).`

En el segon cas, cal convertir prèviament el nombre a *string* mitjançant **`str()`**:

```
In [5]: sqrt_2_value = 2**(1/2)

        # Es pot fer utilitzant la sintaxi següent del print ja
        # vista anteriorment.
        # Si no especifiquem separador, aquest serà l'espai en blanc.
        print("[0] 2**(1/2)", sqrt_2_value)

        # Podem fer-ho també de la manera següent,
        # però necessitarem fer una conversió de tipus numèric a
        # string amb 'str'
        print("[1] 2**(1/2) " + str(sqrt_2_value))
```

```
[0] 2**(1/2) 1.4142135623730951
[1] 2**(1/2) 1.4142135623730951
```

El valor de $\sqrt{2}$ es mostra amb un nombre de decimals per defecte. Més endavant veurem com modificar el comportament del **`print`** per especificar altres opcions.

Podem mostrar també *strings* prèviament compostos mitjançant la concatenació:

```
In [6]: x = 10 * 3.25
        y = 200 ** 2
        s = 'El resultat de x és' + str(x) + 'i y és' + str(y) + '...'
        print(s)
```

```
El resultat de x és 32,5 i y és 40.000...
```

També podem mostrar més d'un valor de diferents tipus per pantalla amb un únic **`print()`** i sense fer les conversions corresponents. Com pots veure, és possible combinar variables amb literals.

En aquest cas, aprofitarem per canviar el separador entre els valors a mostrar per `' - '`:

```
In [7]: value_a = 2.5
        value_b = 90
        value_c = "Això és el que vols"
        print(value_a, value_b, 'valor de la cadena:', 'es',
              value_c, 15, sep=' - ')
```

```
2.5 - 90 - valor de la cadena: - es - Això és el que vols - 15
```

En un *string* podem incloure una sèrie de caràcters especials. Aquestes sèries es distingeixen perquè comencen pel caràcter **`\`**. N'hi ha de diversos entre els quals destaquem el salt de línia **`\n`** i el tabulador **`\t`**. Més endavant tornaran a aparèixer.

Si directament imprimim el contingut d'una variable *string* via **`print()`**, aquest interpretarà els caràcters especials i mostrarà el resultat de la interpretació. En canvi, si el que desitgem és

mostrar exactament els caràcters que conformen una seqüència de text sense que els caràcters especials siguin interpretats ho podem fer mitjançant la funció **repr()**:

```
In [8]: s = 'Hola, benvolgut, \n\t Alexandre.'
        # Traiem per pantalla la interpretació del car. especial:
        print(s)
        # Traiem per pantalla el contingut literal:
        print(repr(s))
```

```
Hola, benvolgut,
         Alexandre.
'Hola, benvolgut, \n\t Alexandre.'
```

Exercici 2.5.1 *Donades les cadenes:*

```
1 a = "Van␣existir␣els␣Tasaday?"
2 b = "Es␣diu␣que␣vivien,␣en␣ple␣segle␣\textsc{xx},␣com␣a␣l'edat␣de␣pedra."
```

Repeteix els exemples anteriors amb aquestes cadenes efectuant: impressió bàsica, concatenació i no concatenació.

Exercici 2.5.2 *Sabent que podem combinar* ***strings*** *amb valors numèrics, efectua-ho, de les dues maneres possibles, per:*

```
1 a = "Es␣va␣veure␣que␣era␣un␣engany␣l'any:"
2 b = 1980
```

Exercici 2.5.3 *Donats $b = 5$ i $e = 2$, calcula: $p = b\hat{}e$*

Escriu mitjançant un ***print*** *les dades i especifica el resultat de la potència indicat: "El resultat de $b\hat{}e$ és:"*

Exercici 2.5.4 *Donat el text:*

"Farem una prova, per entendre bé el que estem fent: 1 2 3 4 5 6 7 8 9 10 11 12"

Divideix la part literal en dues línies i amb la part numèrica forma una taula de tres columnes.

2.5.2 Donant format a les dades que es mostren per pantalla

Aquí veurem alguns exemples de com donar format especial al text que ha d'aparèixer per pantalla amb el **print()**. En la versió *3.x* de Python es pot fer de dues maneres:

- **Seguint les especificacions del llenguatge C**: de forma similar a com es fa quan s'utilitza la funció **sprintf()** del llenguatge de programació C, usant el signe % seguit de nombres i lletres.
- **A la Python**: usant la funció **format()** i usant **{}**.

Podeu trobar més informació a: www.python-course.eu o bé a la pàgina de l'especificació de Python 3.4: docs.python.org.

2.5.2.1 Donant format segons l'estil C

Tot seguit veurem uns exemples de com donar format segons la vella escola, usant l'estil del llenguatge C.

Cal tenir en compte que el tipus de la dada que volem mostrar ve indicat pels caràcters següents:

Tipus	Especificador
Cadena de text	%s
Enter	%d
Real	%f, %e, %g

On el format és el següent:

```
<string>%(value_1, value_2,..., value_n)
```

Quan la funció **print()** troba dins d'un *string* algun dels codis mostrats a la taula anterior, cerca el valor corresponent dins del parèntesi. Allí on aparegui el primer codi, aquest serà substituït pel primer valor, el segon codi pel segon valor...

```
In [9]: 'Em dic %s, tinc %d anys, i mesuro %.2f m' % \
        ('Joan', 22, 1.75)
```

```
Out[9]: 'Em dic Joan, tinc 22 anys, i mesuro 1,75 m'
```

El primer codi indica una cadena, i fa servir la primera cadena, el segon espera un enter i el darrer un *float*.

2.5.2.1.1 Especificador de cadenes de text %s

Podem incloure el contingut d'una variable de cadena mitjançant aquest especificador.

De forma ordenada, es mostrarà el resultat per pantalla, substituint cadascun d'aquests especificadors per la variable o literal de tipus *string* corresponent que apareix a dins dels parèntesis de la dreta.

Vegem-ne un exemple:

```
In [ ]:
```

```
In [10]: Text_1 = "Lorem"
         Text_2 = "Ipsum"

         print("El primer text és %s i el segon és %s" % (Text_1,Text_2))
```

```
El primer text és Lorem i el segon és Ipsum
```

```
In [11]: Number_1 = 3
         Number_2 = 2**(1/2)

         print("El primer text és %s i el segon és %s" % (Number_1, Number_2))
```

```
El primer text és 3 i el segon és 1,4142135623730951
```

2.5.2.1.2 Especificador d'enters %d

Permet imprimir el valor d'un enter. Es pot controlar el nombre de dígits i d'espais i si cal incloure-hi zeros al davant.

La forma més senzilla és:

```
In [12]: Var_1 = 12
         Var_2 = 1234
         Var_3 = 1234567890

         # Sense format
         print("Var_1 =", Var_1)
         print("Var_2 =", Var_2)
         print("Var_3 =", Var_3)
```

```
Var_1 = 12
Var_2 = 1234
Var_3 = 1234567890
```

Una altra opció és alinear a la dreta el valor especificant el nombre de dígits total a mostrar per pantalla. En aquest cas, el nombre de dígits s'ha d'indicar entre els carècters **%** i **d**. En l'exemple següent usarem com a mínim 8 dígits:

```
In [13]: # Format fixant la longitud: davant del %d especifiquem
         # el nombre de caràcters amb els quals s'ha de mostrar
         print("Var_1 = %8d" % Var_1)
         print("Var_2 = %8d" % Var_2)
         print("Var_3 = %8d" % Var_3)
```

```
Var_1 =       12
Var_2 =     1234
Var_3 = 1234567890
```

En els primers dos casos els nombres s'alineen de forma correcta ja que els valors corresponents a mostrar contenen menys de 8 dígits. En el tercer, el valor del nombre consta de 10 dígits. En aquest cas especial, per no perdre precisió, no es té en compte l'alineació.

També és possible inserir zeros a l'esquerra d'un valor en comptes d'espais en blanc. Aquesta vegada cal inserir un 0 just abans de **%**, tot seguit el nombre total de dígits a mostrar i finalment **d**.

```
In [14]: # Format fixant la longitud i completant amb zeros
         print("Var_1 = %010d" % Var_1)
         print("Var_2 = %010d" % Var_2)
         print("Var_3 = %010d" % Var_3)
```

```
Var_1 = 0000000012
Var_2 = 0000001234
Var_3 = 1234567890
```

En aquest cas el nombre de dígits és 10 i aleshores tots els nombres estan correctament alineats.

2.5.2.1.3 Especificadors de reals %f, %e i %g

Permet imprimir un real. Es pot controlar el nombre de dígits davant i darrere la coma i si cal incloure-hi zeros al davant.

La forma més senzilla és:

```
In [15]: sqrt_2 = 2**(1/2)

         print('Valor de 2**(1/2) és %f' % (sqrt_2))
```

```
El valor de 2**(1/2) és 1,414214
```

I també podem controlar el nombre de dígits (abans i després de la coma) a mostrar per pantalla, així com especificar el caràcter de signe:

```
In [16]: print("Valor de 2**(1/2) és %+15.15f" % (sqrt_2))
         print("Valor de 2**(1/2) és %15.4f" % (sqrt_2))
```

```
El valor de 2**(1/2) és +1,414213562373095
El valor de 2**(1/2) és          1,4142
```

En el primer cas, l'alineació a la dreta té en compte tots els dígits, incloent-hi la coma i el signe.

El format **%e** mostra sempre l'exponent, tot seguint les mateixes regles pel que fa a nombre de decimals i alineació:

```
In [17]: avogadro = 6.023e23

         print("Un altre valor és %+15.4e" % (avogadro))
         print("Una altra opció és %+15.4g" % (avogadro))
         print("Un altre cas   %+15.4g" % (sqrt_2))
```

```
Un altre valor és     +6.0230e+23
Una altra opció és      +6.023e+23
Un altre cas            +1.414
```

El comportament del format **%g** és similar al de **%e** si l'exponent és menor de -4 o major que la precisió. Altrament, es comporta com **%f**. Podem apreciar com el nombre de dígits en el cas de **%g** es correspon amb la precisió.

2.5.2.1.4 Altres formats: octal (%o) i hexadecimal (%x)

Permeten escriure valors enters en format octal o hexadecimal.

```
In [18]: Val_1 = 2**8
         Val_2 = 2**10
         Val_3 = 242942

         print("[Val_1] enter:%d octal:%o  hexa:0x%x" % (Val_1, Val_1, Val_1))
         print("[Val_2] enter:%d octal:%o  hexa:0x%x" % (Val_2, Val_2, Val_2))
         print("[Val_3] enter:%d octal:%o  hexa:0x%x" % (Val_3, Val_3, Val_3))
```

```
[Val_1] enter:256 octal:400  hexa:0x100
[Val_2] enter:1024 octal:2000  hexa:0x400
[Val_3] enter:242942 octal:732376  hexa:0x3b4fe
```

2.5.2.1.5 Caràcters especials

Es poden imprimir o incloure en cadenes de text alguns caràcters especials. Els més usats, com ja hem vist en alguns exemples anteriors, són:

Descripció	Caràcter especial
Salt de línia	\n
Tabulador	\t
\	\\
Percentatge	%

```
In [19]: print("Això és un text\n\tpartit en dues línies\n\n")
         print("Tab0\tTab1\tTab2")
         print("A\tB\tC")
```

```
Això és un text
    partit en dues línies


Tab0    Tab1    Tab2
A       B       C
```

I ara un exemple on combinem diferents tipus de dades:

```
In [20]: # Un altre exemple en el qual ho combinem tot
         v1 = 100
         v2 = 123456789.123456789
         s = '-'*10 + "\n"

         print("v1:\n\tdec (ent) - %d\n\tdec (real) - %f\n\thexa - 0x%X\n%s" %(v1, v1, v1, s
         print("Limitant el nombre de dígits als decimals d'un float: %.3f\n" %v2)
         print("Afegint zeros a la dreta si no arribem a un nombre de xifres determinat: %.5
         print("Imprimint un real com un enter: %d\n" % v2)
         print("Afegint zeros a l'esquerra si no arribem a un nombre de xifres determinat: %
```

```
v1:
        dec (ent) - 100
        dec (real) - 100.000000
        hexa - 0x64
----------

Limitant el nombre de dígits als decimals d'un float: 123456789.123

Afegint zeros a la dreta si no arribem a un nombre de xifres determinat: 123.12300

Imprimint un real com un enter: 123456789

Afegint zeros a l'esquerra si no arribem a un nombre de xifres determinat: 00123
```

Exercici 2.5.5 *Donats: a = 'base', b = 3, c = 2, d = 5, e = 1,8*

Omple els espais de la frase següent utilitzant el format segons l'estil C:

La potència de ___ igual a ___ i exponent ___ dividida per ___ dona com a resultat ___.

Exercici 2.5.6 *Donada l'equació: $x^2 - x - 1 = 0$, calcula'n les arrels i expressa el seu valor especificant-ne el signe i donant deu decimals.*

Exercici 2.5.7 *La distància mitjana Terra-Sol és d'uns 1.5e8 km. Expressa aquest valor com a nombre enter, en format octal i en hexadecimal.*

Exercici 2.5.8 *Donat el text:*

"Apliquem la majoria de les vacunes per eliminar determinades malalties. Tant la diftèria com el tètanus s'apliquen: en els 2, 4, 6 i 18 mesos, entre 4-6 i 14-16 anys, i cada 10 anys."

Divideix el text fins als dos punts en tres línies i amb la resta construeix una taula en què s'indiqui edat i vacuna.

2.5.2.2 Donant format a la Python

Especifiquem un *string* dins del qual apareixen parelles de **{}** i a continuació la funció **format()** dins de la qual adjuntem tants valors (o variables) com conjunts **{}** tinguem a l'*string*.

A l'hora de mostrar per pantalla dins de l'*string* cadascuna de les parelles **{}** se substitueix, per ordre, amb els valors dels objectes de dins del parèntesi que acompanya el format.

Cal tenir en compte que, igual que donant format seguint la vella escola, el tipus de la dada que volem mostrar ve indicada pels caràcters següents:

Tipus	Especificador
Cadena de text	s
Enter	d
Real	f, e, g
Locale	n

```
In [21]: nom = 'Pere'
         v = 10.0

         # Deixem que el format determini automàticament el tipus dels
         # valors a mostrar.
         s = "Lorem Ipsum:\n\t1r val: {}\n\t2n val: {}\n\t3r val: {}".format(nom, v, 12+12)
         print(s)
```

```
Lorem Ipsum:
        1r val: Pere
        2n val: 10.0
        3r val: 24
```

En aquest cas, les dades es mostren de la mateixa manera que usant la funció **str()**.

Podem alterar l'ordre d'aparició dels valors a mostrar, tot indicant explícitament dins del **{}** l'índex del valor que volem que aparegui:

```
In [22]: nom = 'Pere'
         separa = '-'*10 + "\n"
         v = 10.0

         s = "Lorem Ipsum:\n\t1r val: {1}\n\t2n val: {0}\n\t3r val: {2}".format(nom, v, 12+1
         print(s)
         print(separa)
```

```
        print('{} and {}'.format('dogs', 'cats'))
        print('{0} and {1}'.format('dogs', 'cats'))
        print('{1} and {0}'.format('dogs', 'cats'))
```

```
Lorem Ipsum:
        1r val: 10.0
        2n val: Pere
        3r val: 24
----------

dogs and cats
dogs and cats
cats and dogs
```

Podem usar paraules clau dins del **format()** per tal de referir-nos als diferents valors:

```
In [23]: print('En {person} és {adjective}.'.format(person='Pau',
                                                    adjective='molt divertit'))
```

```
En Pau és molt divertit.
```

Podem expressar valors en forma de taula:

```
In [24]: print('{0:2d} {1:10d} {2:3d}'.format(1, 1024, 2389))
         print('{0:2d} {1:10d} {2:3d}'.format(12, 1234567890, 123))
```

```
 1       1024 2389
12 1234567890 123
```

El nombre que apareix entre claus i abans dels dos punts indica la posició del valor que es troba dins dels parèntesis que es mostrarà en cada cas. El que hi ha darrere dels dos punts indica el comportament de forma similar a l'estil C.

El darrer exemple en **estil C**, el podem reproduir ara tot seguint el format **a la Python**. Fixeu-vos que podem especificar dins de les parelles **{}** el format concret amb el qual volem que surti per pantalla el valor corresponent:

```
In [25]: v1 = 100
         v2 = 123456789.123456789
         s = '-'*10 + "\n"

         # Componem els strings als quals volem donar format
         str_format_1 = "v1:\n\tdec (ent) - {0:d}\n\tdec (real) - {1:f}\n\thexa - 0x{2:X}\n
         str_format_2 = "Limitant el nombre de dígits als decimals d'un float: {0:.3f}\n"
         str_format_3 = "Afegint zeros a la dreta si no arribem a un nombre de xifres deterr
         str_format_4 = "Imprimint un real com un enter: {0}\n"
         str_format_5 = "Afegint zeros a l'esquerra si no arribem a un nombre de xifres dete

         # Apliquem format i traiem per pantalla
         print(str_format_1.format(v1, v1, v1, s))
         print(str_format_2.format(v2))
         print(str_format_3.format(123.123))
```

```
        print(str_format_4.format(int(v2)))
        print(str_format_5.format(123))
```

```
v1:
        dec (ent) - 100
        dec (real) - 100.000000
        hexa - 0x64
----------

Limitant el nombre de dígits als decimals d'un float: 123456789.123

Afegint zeros a la dreta si no arribem a un nombre de xifres determinat: 123.12300

Imprimint un real com un enter: 123456789

Afegint zeros a l'esquerra si no arribem a un nombre de xifres determinat: 00123
```

Finalment, introduirem un nou component: el **locale**.

Primerament cal importar el que hi ha configurat per defecte (en un futur veurem l'ús específic de la paraula ***import***):

```
In [26]: import locale

         locale.setlocale(locale.LC_ALL, 'ES_es')
         locale.localeconv()

Out[26]: {'int_curr_symbol': 'EUR ',
          'currency_symbol': 'Eu',
          'mon_decimal_point': ',',
          'mon_thousands_sep': '.',
          'mon_grouping': [3, 3, 0],
          'positive_sign': '',
          'negative_sign': '-',
          'int_frac_digits': 2,
          'frac_digits': 2,
          'p_cs_precedes': 0,
          'p_sep_by_space': 1,
          'n_cs_precedes': 0,
          'n_sep_by_space': 1,
          'p_sign_posn': 1,
          'n_sign_posn': 1,
          'decimal_point': ',',
          'thousands_sep': '',
          'grouping': [127]}

In [27]: help(locale.setlocale)

Help on function setlocale in module locale:

setlocale(category, locale=None)
    Set the locale for the given category.  The locale can be
    a string, an iterable of two strings (language code and encoding),
```

```
    or None.

    Iterables are converted to strings using the locale aliasing
    engine.  Locale strings are passed directly to the C lib.

    category may be given as one of the LC_* values.
```

```
In [28]: print("{:n}".format(v2))
```

```
1,23457e+08
```

En aquest cas, el separador decimal és **‘,’** (veges el camp `decimal_point`) ja que hem definit que volem treballar amb la configuració espanyola.

Exercici 2.5.9 *Donats:*

```
1 a = 'En␣base␣binària␣el␣nombre'
2 b = 8
3 c = 'és:'
```

Escriu el nombre 8 en base binària, especificant la transformació en el format i escriu la frase completa deixant que sigui el format qui determini el tipus de valor a mostrar.

Exercici 2.5.10 *Donades les lletres a, b, c, escriu, efectuant els print corresponents, totes les permutacions possibles sense alterar el format.*

Exercici 2.5.11 *Escriu en forma de taula de dues files i tres columnes: 2097654, 372, 7, 5437, 67, 111.*

2.5.3 Entrada per teclat

`input():` retorna una **cadena de text** introduïda per l’usuari a través del teclat (o consola).

```
In [29]: nom = input("Introdueix el teu nom: ")
         print('Hola, {}!'.format(nom) )
```

```
Introdueix el teu nom: Joan
Hola, Joan!
```

Podem demanar més dades cridant **`input():`** diverses vegades:

```
In [30]: nom = input("Com et dius? ")
         edat = input("Quants anys tens? ")
         pes = input("Quant peses? ")

         print(type(nom), type(edat), type(pes))

         print("Et dius {} tens {} anys i peses {}kg.".format(nom, edat, pes))
```

```
Com et dius? Joan
Quants anys tens? 23
Quant peses? 73
<class 'str'> <class 'str'> <class 'str'>
```

```
Et dius Joan tens 23 anys i peses 73 kg.
```

Si volem obtenir l'edat i el pes com a valors numèrics (fixa't que són de tipus cadena de text), cal convertir-los.

Això ho podem fer de dues maneres. Les funcions **`int()`** o **`float()`** retornen el valor en format enter o real contingut en una cadena. En combinació amb **`input()`** permeten llegir dades numèriques introduïdes per l'usuari a través del teclat.

```
In [31]: s_oper1 =  input("Introdueix el primer operant:")
         s_oper2 =  input("Introdueix el segon operant:")
         text_oper1 = "Has entrat el nombre {} i és del tipus {}".format(s_oper1, type(s_ope
         text_oper2 = "Has entrat el nombre {} i és del tipus {}".format(s_oper2, type(s_ope
         separa = '-'*10 + "\n"

         print(separa+text_oper1+"\n"+text_oper2)

         # Abans de fer l'operació numèrica cal convertir els strings a floats
         f_oper1 = float(s_oper1)
         f_oper2 = float(s_oper2)
         print('El tipus de f_oper1 és {} i el de f_oper1 és {}'.format(type(f_oper1), type
         print(separa, f_oper1, '+', f_oper2, '=', f_oper1+f_oper2)
```

```
Introdueix el primer operant:1
Introdueix el segon operant:3
----------
Has entrat el nombre 1 i és del tipus <class 'str'>
Has entrat el nombre 3 i és del tipus <class 'str'>
El tipus de f_oper1 és <class 'float'> i el de f_oper1 és <class 'float'>
----------
 1.0 + 3.0 = 4.0
```

Podem també fer directament la conversió:

```
In [32]: # Conversió numèrica immediata

         x = int(input("Entra un nombre: "))
         y = int(input("Entra un segon nombre: "))
         print('La suma de ', x, ' i ', y, ' és ', x+y, '.', sep='')
```

```
Entra un nombre: 3
Entra un segon nombre: 4
La suma de 3 i 4 és 7.
```

Una altra opció és emprar el mòdul **locale** prèviament introduït. D'aquesta manera Python tindrà en compte el separador decimal amb el qual treballi el nostre sistema: '.' o ','. És a dir, si el nostre sistema funciona en català o en espanyol, podrem introduir valors reals on aparegui una ','.

En aquest cas farem servir les funcions **`locale.atoi()`** i **`locale.atof()`** segons es tracti d'un enter o bé un valor real:

```
In [33]: import locale
```

```
locale.setlocale(locale.LC_ALL, 'ES_es')

s_oper1 =  input("Escriu el primer operand:")
s_oper2 =  input("Escriu el segon operand:")
text_oper1 = "El primer operand és {}"\
                " i el seu tipus és {}".format(s_oper1,
                                               type(s_oper1))
text_oper2 = "El segon operand és {}"\
                " i el seu tipus és {}".format(s_oper2,
                                              type(s_oper2))
separator = '-'*10 + "\n"

print(separator+text_oper1+"\n"+text_oper2)

f_oper1 = locale.atof(s_oper1)
f_oper2 = locale.atof(s_oper2)

print("El tipus de f_oper1 és {}"\
      " i el de f_oper1 és {}".format(type(f_oper1),
                                      type(f_oper2)))
print(separator, "{:n} + {:n} = {:n}".format(f_oper1,
                                             f_oper2,
                                             f_oper1+f_oper2))
```

```
Escriu el primer operand: 2
Escriu el segon operand: 4,3
----------
El primer operand és 2 i el seu tipus és <class 'str'>
El segon operand és 4,3 i el seu tipus és <class 'str'>
El tipus de f_oper1 és <class 'float'> i el de f_oper1 és <class 'float'>
----------
 2 + 4,3 = 6,3
```

I ara amb enters:

```
In [34]: x = locale.atoi(input("Escriu un enter: "))
         y = locale.atoi(input("Escriu un segon enter: "))
         print('Sumant {:n} i {:n} = {:n}'.format(x, y, x+y))
```

```
Escriu un enter: 2
Escriu un segon enter: 4
Sumant 2 i 4 = 6
```

En aquest darrer exemple, el resultat és sempre el mateix, ja que no s'usa el separador decimal, però aquesta solució mitjançant el `locale` ens permet que el nostre codi sigui portable a qualsevol sistema.

Exercici 2.5.12 *Sense efectuar una transformació immediata, utilitzant dos* ***`input()`*** *i un sol* ***`print`*** *escriu el text següent omplint els espais: Podem dividir exactament ___ per ___, el resultat és: ___ .*

Indica, efectuant un altre ***`print`****, de quin tipus són les variables que has introduït.*

Exercici 2.5.13 *Repeteix l'exercici anterior de manera que el primer nombre sigui* `float` *i el segon enter, efectua la transformació immediata i per a un altre* `print` *indica de quin tipus són les variables que has introduït.*

Exercici 2.5.14 *Repeteix l'exercici anterior utilitzant el mòdul* `locale`*.*

Exercici 2.5.15 *Repeteix l'exercici anterior, però ara tan sols pots utilitzar un* `print` *i separadors.*

2.6 Mòduls: el mòdul `tkinter`

Com s'ha indicat al començament, l'intèrpret de Python només proporciona la funcionalitat bàsica del llenguatge. Aquesta funcionalitat inclou els tipus bàsics (com els tipus numèrics que ja hem vist) i algunes funcions elementals (com **`print()`**, **`help()`** o **`type()`**).

La major part de la funcionalitat de Python està disponible mitjançant l'ús de **mòduls** (el que en altres llenguatges es diria biblioteques). Python inclou una gran quantitat de [mòduls estàndard] (https://docs.python.org/3/py-modindex.html) que es distribueixen juntament amb l'intèrpret. Diversos grups han desenvolupat i distribueixen un gran nombre de mòduls amb funcionalitats molt diverses.

La distribució de Python que fa Anaconda inclou una selecció de mòduls addicionals que complementen la funcionalitat dels mòduls estàndard de Python.

2.6.1 La importació de mòduls

La manera de fer ús d'aquests mòduls en Python és utilitzant la sentència **`import`**. Si volem utilitzar el mòdul estàndard **`math`**, que ens dona accés a diverses funcions matemàtiques, cal escriure:

```
In [1]: # Importem el mòdul math
        import math
```

Un cop importat el mòdul, podem fer ús de les funcions que inclou usant la sintaxi:

```
[nom del mòdul]. [nom de la funció] ()
```

Dins de l'entorn Jypiter (o d'IPython) s'inclou una funcionalitat d'ajuda en línia que ens permet conèixer les funcions (i constants) disponibles en un mòdul. Només cal escriure **`math.`** i prémer la tecla <TAB> perquè aparegui un menú amb totes les funcions disponibles:

```
In [ ]: math.
```

En l'exemple següent es pot veure com utilitzar les funcions trigonomètriques de `math`.

```
In [2]: # math defineix una constant amb el valor de pi
        angle= math.pi
        print(angle)
        # sinus
        print(math.sin(angle))
        # cosinus
        print(math.cos(angle))
        # logarithm
        print(math.log(angle))
        # squared root
        print(math.sqrt(angle))
```

```
3.141592653589793
1.2246467991473532e-16
-1.0
1.1447298858494002
1.7724538509055159
```

En **`math`** les funcions trigonomètriques tenen els arguments en radiants. El mateix mòdul té funcions per convertir entre radiants i graus:

```
In [3]: # Converting radians
        deg = math.degrees(math.pi / 2.)

        # Converting degrees
        rad = math.radians(deg)

        print ('Value in degrees {} and in radians {}.'.format(deg, rad) )
```

```
Value in degrees 90.0 and in radians 1.5707963267948966.
```

La funció **help ()** (de Python) permet obtenir informació més detallada sobre un mòdul o les seves funcions:

```
In [4]: help(math.cos)
```

```
Help on built-in function cos in module math:

cos(...)
    cos(x)

    Return the cosine of x (measured in radians).
```

```
In [5]: help(math.degrees)
```

```
Help on built-in function degrees in module math:

degrees(...)
    degrees(x)

    Convert angle x from radians to degrees.
```

2.6.1.1 Importació de funcions específiques

Python ofereix la possibilitat de no importar tot el mòdul sinó només funcions específiques. En aquest cas la funció es pot fer servir directament, sense el nom del mòdul. Això és útil si en un programa donat hem de fer servir una funció amb molta freqüència.

```
In [6]: from math import sin  # import sin method from math module
        from math import pi   # import pi data member from math module

        sin( pi / 2. )
```

```
Out[6]: 1.0
```

2.6.1.2 Importació global de totes les funcions

Python permet importar individualment totes les funcions d'un mòdul en una única sentència (*wild import*). La diferència amb la importació genèrica és que això permet usar totes i cadascuna de les funcions sense usar com a prefix el nom del mòdul.

Important: aquesta opció no és recomanable ja que pot definir moltes funcions i variables de manera incontrolada en el nostre codi.

```
In [7]: from math import *
        print(sin(angle))
        print(cos(angle))
        print(log(angle))
        print(sqrt(angle))
```

```
1.2246467991473532e-16
-1.0
1.1447298858494002
1.7724538509055159
```

2.6.1.3 Importació amb un nom alternatiu

Python permet importar mòduls especificant un nom alternatiu. Això permet, per exemple, fer servir noms més curts per cridar les funcions del mòdul.

```
In [8]: import math as m

        print(m.sin(m.pi))
```

```
1.2246467991473532e-16
```

2.6.1.4 Llistat del contingut d'un mòdul

La funció **dir ()** permet llistar el contingut d'un mòdul (les variables i funcions que s'hi defineixen dins).

```
In [9]: # Llistant el contingut del mòdul math
        print(dir(math))
```

```
['__doc__', '__file__', '__loader__', '__name__',
'__package__', '__spec__', 'acos', 'acosh', 'asin', 'asinh',
'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos', 'cosh',
'degrees', 'e', 'erf', 'erfc', 'exp', 'expm1', 'fabs', 'factorial',
'floor', 'fmod', 'frexp', 'fsum', 'gamma', 'gcd', 'hypot', 'inf',
'isclose', 'isfinite', 'isinf', 'isnan', 'ldexp', 'lgamma', 'log',
'log10', 'log1p', 'log2', 'modf', 'nan', 'pi', 'pow', 'radians',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Exercici 2.6.1 *Una corda forma un arc de 50^o en una circumferència de 15 m de radi. Quina és la distància entre els seus extrems?*

Exercici 2.6.2 *Calcula la inclinació dels rajos del Sol si un pal de 2 m de longitud projecta una ombra de 5 m.*

Expressa el resultat tant amb radiants com amb graus hexadecimals.

Exercici 2.6.3 *Donat un rectangle de base 10 u i alçada 6 u, calcula la longitud de les seves diagonals i l'angle que formen amb la base. Especifica mitjançant un sol* ***print*** *els resultats.*

Exercici 2.6.4 *La suma dels valors recíprocs de dos nombres enters consecutius és 5/6. Calcula'ls i discuteix el resultat.*

Exercici 2.6.5 *Per a un nombre x, a determinar, en efectuar 3^x dona com a resultat 6561. Quin és el valor x?*

Exercici 2.6.6 *Quin és el logaritme natural de 69?*

2.6.2 El mòdul `tkinter`

El mòdul de Python **`tkinter`** permet la creació d'interfícies gràfiques d'usuari (GUI per la denominació en anglès). Està basat en la funcionalitat de la biblioteca **Tk**, un joc d'eines multiplataforma per desenvolupar interfícies gràfiques d'usuari.

El Tk va ser creat al voltant de 1988 per Juan Ousterhout, en aquell moment un professor d'informàtica a UC Berkeley. Va ser desenvolupat com una forma de construir fàcilment GUI amb el seu llenguatge d'*scripting* **Tcl**. El Tcl només era accessible per a Unix, i el Tk corria sota X11. El primer llançament de codi obert va ser al voltant de 1991, amb una adopció bastant ràpida aproximadament un any després. Pots trobar els records de John dels primers anys en el lloc principal de desenvolupament de Tcl/Tk.

La funcionalitat que proporciona **`tkinter`** és molt extensa, i en aquest curs només farem servir una mínima part com a ajuda visual a la comprensió d'alguns conceptes inicials de programació. Per a això utilitzarem únicament els *widgets* **`canvas`** (i **`button`**).

L'exemple següent permet comprovar el funcionament correcte del mòdul **`tkinter`** al teu ordinador. En executar-lo s'obre una finestra amb dos botons funcionals: un que cada vegada que es prem afegeix claus al text i un altre per tancar la finestra.

```
In [10]: import tkinter
         tkinter._test()
```

El resultat es mostra a la figura 2.1.

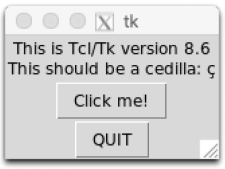


Figura 2.1: Test de `tkinter`.

2.6.2.1 Un programa bàsic amb `tkinter`

Un programa mínim que faci servir el mòdul **`tkinter`** ha de tenir tres parts:

1. Importació del mòdul.
2. Inici de la finestra gràfica de treball.
3. Instruccions per posar elements a la nostra finestra.

L'exemple següent implementa aquestes tres etapes per dibuixar dues línies. El *canvas* o llenç representa l'àrea de la finestra on podem dibuixar, com es comenta més endavant.

Nota. La finestra de treball de `tkinter` (que es diu *Top-level window*) s'obre de forma separada al Notebook, i **pot quedar darrere del navegador**.

```
In [11]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal


         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem
                                 height=200,     # un llenç de 200 x 200 píxels
                                 width=200)
         canvas.pack()                           # I el situem a la finestra principal

         canvas.create_line(0, 0, 100, 200)      # Tracem dues línies des de l'origen
         canvas.create_line(0, 0, 200, 200)      # fins a les coordenades (en píxels) indicades


         root.mainloop()                         # Aquesta comanda mostra el resultat i ens
                                                 # permet interactuar amb la GUI. El programa
                                                 # es queda executant aquí fins que la
                                                 # tanquem
```

El resultat es mostra a la figura 2.2.

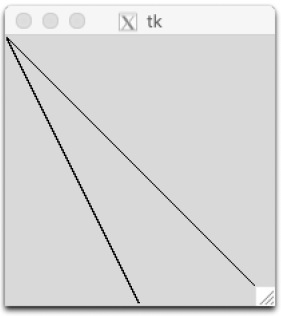


Figura 2.2: Test de dues línies.

```
In [12]: help(canvas.create_line)
Help on method create_line in module tkinter:

create_line(*args, **kw) method of tkinter.Canvas instance
    Create line with coordinates x1,y1,...,xn,yn.
```

Important: com es pot veure en aquest exemple, el sistema de coordenades de **Tk** té l'origen a la cantonada superior esquerra amb l'eix `x` cap a la dreta i l'eix `y` cap avall.

El mòdul **`tkinter`** inclou múltiples funcions per controlar tant el **`canvas`** com els **`widgets`** (o elements interactius) que hi situem i les seves propietats. En aquesta introducció veurem algunes funcions bàsiques. Durant una pràctica a l'aula veurem algunes funcions més avançades.

2.6.2.2 El giny `canvas`

Ens permet crear el *canvas* (o llenç) sobre el qual volem situar la resta d'elements (o *widgets*) amb els quals treballarem. El llenç és una superfície rectangular sobre la qual podrem dibuixar i posicionar la resta d'objectes que utilitzarem.

Per crear el llenç farem servir la sintaxi:

```
canvas = tkinter.Canvas(root, option=value, ... )
```

Algunes de les opcions que podem definir són:

- `Bd` | `Borderwidth`: gruix del marc al voltant del llenç.
- `Bg` | `Background`: color del fons del llenç.
- `Cursor`: forma del cursor al llenç.
- `Height`: alçada (dimensió I) del llenç.
- `Width`: amplada (dimensió X) del llenç.

Més detalls a http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/canvas.html

Vegem el mateix exemple anterior donant valors a totes aquestes opcions:

```
In [13]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal

         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem un "llenç de" 20
                                 borderwidth = 20,
                                 background = 'yellow',
                                 cursor = 'circle',
                                 height = 300,
                                 width = 300)
         canvas.pack()                           # I el situem a la finestra principal


         canvas.create_line(0, 0, 100, 200)      # Tracem dues línies des de l'origen
         canvas.create_line(0, 0, 200, 200)

         root.mainloop()                         # Aquesta comanda mostra el resultat i ens
                                                 # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.3.

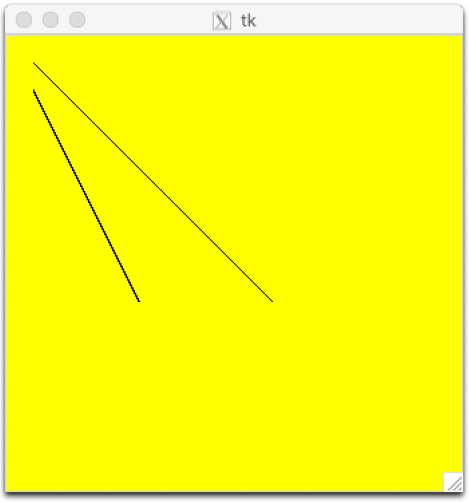


Figura 2.3: Test amb canvi de fons, mida i cursor.

Un cop creat el nostre llenç, podem "dibuixar-hi" a sobre. Per a això tenim una sèrie de funcions que ens permetran crear diversos tipus d'objectes (línies, ovals, arcs, rectangles, polígons o text). Si hem anomenat `canvas` al llenç, la forma bàsica de crear aquests objectes seria:

- `Canvas.create_line (x0, y0, x1, y1, ..., xn, i n, option, ...)`
- `Canvas.create_oval (x0, y0, x1, y1, option, ...)`
- `Canvas.create_arc (x0, y0, x1, y1, option, ...)`
- `Canvas.create_rectangle (x0, y0, x1, y1, option, ...)`
- `Canvas.create_polygon (x0, y0, x1, y1, ..., option, ...)`
- `Canvas.create_text (x, i, option, ...)`

Veurem cadascun d'ells en els apartats següents.

2.6.2.3 Creant una línia

`canvas.create_line(x0, y0, x1, y1, ..., xn, yn, option, ...)`

Crea un línia, és a dir, un conjunt de segments (rectes o corbs) que uneixen els punts `(x0, y0)`, `(x1, y1)`, ..., `(xn, yn)`. Les principals opcions que admet aquesta funció són:

- `Dash`: defineix el tipus de línia.
- `Fill`: defineix el color de la línia.
- `Width`: defineix l'amplada de la línia.
- `Activedash`: defineix el tipus de línia quan està activa.
- `Activefill`: defineix el color de la línia quan està activa.
- `Activewidth`: defineix l'amplada de la línia quan està activa.
- `Smooth`: si és *True* fa servir segments corbs (*splines*).

Més detalls en aquest enllaç.

Vegem-ne un exemple:

```
In [14]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal

         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem
                                 height=300,     # un llenç de 300 x 300
                                 width=300)
         canvas.pack()                           # I el situem a la finestra principal

         canvas.create_line(20, 25,
                            50, 30,
                            65, 40,
                            80, 65,
                            100, 150,
                            120, 220,
                            135, 230,
                            200, 280,
                            fill = 'red',
                            width = '5')

         canvas.create_line(120, 25,
                            150, 30,
                            165, 40,
                            180, 65,
```

```
                200, 150,
                220, 220,
                235, 230,
                300, 280,
                fill = 'blue',
                width = '4',
                smooth = True,
                activefill = 'cyan',
                activedash = (5,))

root.mainloop()                 # Aquesta comanda mostra el resultat i ens
                                # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.4.

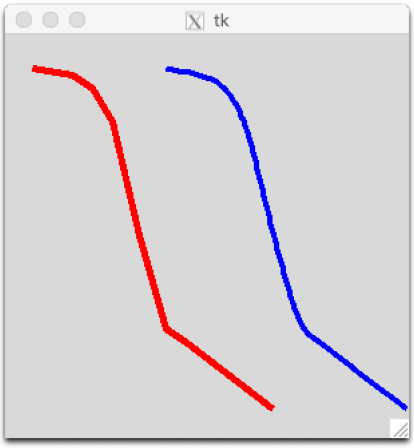


Figura 2.4: Test amb diferents línies.

2.6.2.4 Creant un òval o un arc

```
canvas.create_oval(x0, y0, x1, y1, option, ...)
canvas.create_arc(x0, y0, x1, y1, start=angle, extent=angle, option, ...)
```

Aquestes funcions permeten traçar una el · lipse (oval) o arc d'el · lipse (arc) en el llenç. Els punts `(x0, y0)` i `(x1, y1)` defineixen els vèrtexs superior esquerre i inferior dret del rectangle en el qual s'inscriu l'el · lipse.

Les principals opcions comunes a aquestes funcions són:

- `Dash`: defineix el tipus de línia.
- `Fill`: defineix el color de l'interior.
- `Outline`: defineix el color de la vora.
- `Width`: defineix l'amplada de la línia.
- `Activedash`: defineix el tipus de línia quan està actiu.
- `Activefill`: defineix el color de la línia quan està actiu.
- `Activeoutline`: defineix el color de la vora quan està actiu.
- `Activewidth`: defineix l'amplada de la línia quan està actiu.

I en el cas de l'arc, aquestes opcions determinen els angles d'inici i final:

- `Start`: angle en graus on comença l'arc.
- `Extent`: amplada en graus de la falca.
- `Style`: tipus d'arc: `tkinter.PIESLICE, tkinter.ARC, tkinter.CHORD`.

Més detalls en aquests enllaços: òval i arc.

Vegem-ne un exemple:

```
In [15]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal



         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem
                                 height=300,     # un llenç de 300 x 300
                                 width=300)
         canvas.pack()                           # I el situem a la finestra principal


         # Dos òvals al centre de la finestra
         canvas.create_oval(100, 130,
                            120, 170,
                            fill = 'blue',
                            width = 5)

         canvas.create_oval(170, 130,
                            230, 170,
                            fill = 'red',
                            width = 5)


         # Arcs sense rang d'angles; per defecte és start = 0 extent = 90
         canvas.create_arc(20, 20,
                           120, 120,
                           fill = 'red',
                           width = 5)

         canvas.create_arc(20, 20,
                           120, 120,
                           fill = 'red',
                           width = 5)

         canvas.create_arc(170, 40,
                           270, 80,
                           fill = 'yellow',
                           width = 5)

         # Arcs amb rang d'angles especificat
         canvas.create_arc(20, 190,
                           120, 290,
```

```
                        fill = 'red',
                        outline = 'cyan',
                        width = 5,
                        start = 90,
                        extent = 130,
                        style = tkinter.CHORD,
                        activefill = 'pink',
                        activewidth = 10)

canvas.create_arc(170, 190,
                        270, 270,
                        fill = 'yellow',
                        width = 5,
                        start = 45,
                        extent = 250,
                        style = tkinter.ARC,
                        activedash = (8, 4) )


root.mainloop()                         # Aquesta comanda mostra el resultat i ens
                                        # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.5.

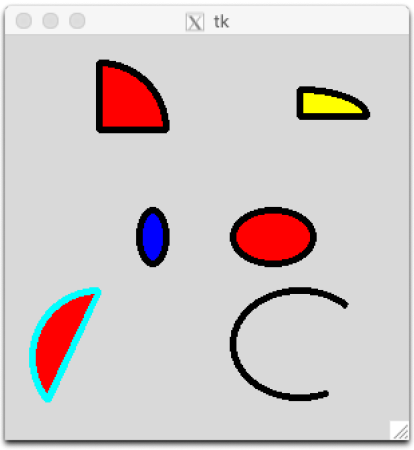


Figura 2.5: Test amb ovals i arcs.

2.6.2.5 Creant un rectangle

`canvas.create_rectangle(x0, y0, x1, y1, option, ...)`

Aquesta funció traça un rectangle en el llenç. De forma similar a les anteriors, els punts `(x0, y0)` i `(x1, y1)` defineixen els vèrtexs superior esquerre i inferior dret del rectangle.

Les principals opcions d'aquesta funció són similars a les de les anteriors:

- `Dash`: defineix el tipus de línia.
- `Fill`: defineix el color de l'interior.
- `Outline`: defineix el color de la vora.
- `Width`: defineix l'amplada de la línia.

- `Activedash`: defineix el tipus de línia quan està actiu.
- `Activefill`: defineix el color de la línia quan està actiu.
- `Activeoutline`: defineix el color de la vora quan està actiu.
- `Activewidth`: defineix l'amplada de la línia quan està actiu.

Més detalls en aquest enllaç: http://infohost.nmt.edu/tcc/help/pubs/tkinter/web/create_rectangle.html

Vegem-ne un exemple:

```
In [16]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal

         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem
                                 height=300, # un llenç de 300 x 300
                                 width=350)
         canvas.pack()                           # I el situem a la finestra principal


         # Rectangle groc
         canvas.create_rectangle(50, 50,
                                 275, 250,
                                 fill = 'yellow',
                                 width = 5)

         # Rectangles vermells sense vora

         canvas.create_rectangle(75, 50,
                                 100, 250,
                                 fill = 'red',
                                 width = 0)

         canvas.create_rectangle(125, 50,
                                 150, 250,
                                 fill = 'red',
                                 width = 0)

         canvas.create_rectangle(175, 50,
                                 200, 250,
                                 fill = 'red',
                                 width = 0)

         canvas.create_rectangle(225, 50,
                                 250, 250,
                                 fill = 'red',
                                 width = 0)

         root.mainloop()                          # Aquesta comanda mostra el resultat i ens
                                                  # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.6.

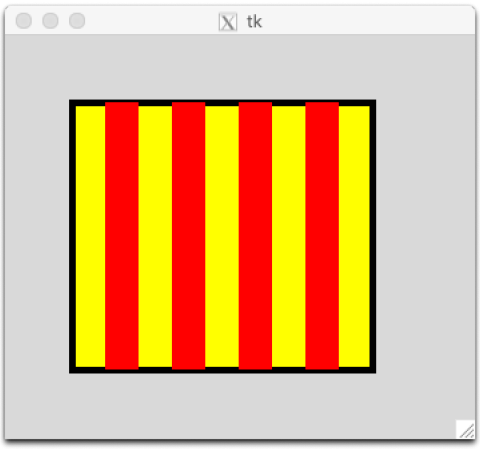


Figura 2.6: Test amb rectangles.

2.6.2.6 Creant un polígon

`canvas.create_polygon(x0, y0, x1, y1, ..., option, ...)`

Aquesta funció traça un polígon unint els punts (x0, y0), (x1, y1), (x2, y2), etc.

Les opcions principals d'aquesta funció són similars a les de les anteriors:

- `Dash`: defineix el tipus de línia.
- `Fill`: defineix el color de l'interior.
- `Outline`: defineix el color de la vora.
- `Width`: defineix l'amplada de la línia.
- `Activedash`: defineix el tipus de línia quan està actiu.
- `Activefill`: defineix el color de la línia quan està actiu.
- `Activeoutline`: defineix el color de la vora quan està actiu.
- `Activewidth`: defineix l'amplada de la línia quan està actiu.

Més detalls en aquest enllaç.

Vegem-ne un exemple:

```
In [17]: import tkinter                          # Importem el mòdul

         root = tkinter.Tk()                     # Creem la finestra principal

         canvas = tkinter.Canvas(root,           # A la nostra finestra, creem
                                 height=300,     # un llenç de 300 x 300
                                 width=350)
         canvas.pack()                           # I el situem a la finestra principal


         # Polígon en traç negre i interior verd
         canvas.create_polygon(50,50, 80,99,  80,170,  150,170, 275, 250,
                                 outline='black',
                                 fill = 'green',
```

```
                                width = 5)


        # Polígon en traç vermell gruixut
        canvas.create_polygon(180,150, 200,200, 270,50,
                              outline='red',
                              width = 10)

        root.mainloop()                         # Aquesta comanda mostra el resultat i ens
                                                # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.7.

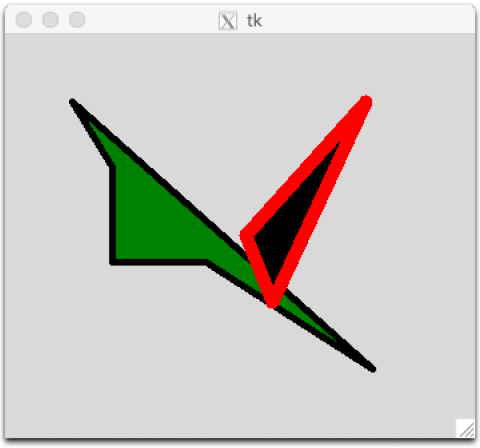


Figura 2.7: Test amb polígon.

2.6.2.7 Creant un text

`canvas.create_text(x, y, text="....", option, ...)`

Aquesta funció escriu un text centrat en la posició (`x`, `y`) del llenç.

Les principals opcions d'aquesta funció són:

- `Fill`: color del text.
- `Width`: amplada de la caixa de text.
- `Font`: tipus de lletra (font i mida).
- `Justify`: alineació del text (`tk.LEFT` per defecte, `tk.CENTER` o `tk.RIGHT`).

Més informació a l'enllaç.

Vegem-ne un exemple:

```
In [18]: import tkinter                         # Importem el mòdul

         root = tkinter.Tk()                    # Creem la finestra principal

         canvas = tkinter.Canvas(root,          # A la nostra finestra, creem
                                 height=300,    # un llenç de 300 x 300
                                 width=350)
         canvas.pack()                          # I el situem a la finestra principal
```

```
# Text vermell, mida per defecte, a la part baixa
canvas.create_text(50,225,
                   text="Hola, món",
                   fill='red')

# Text vermell, lletra diferent, a la part baixa
canvas.create_text(50,250,
                   text="Hola, món",
                   font=("Times", 16),
                   fill='green')

# Text blau, molt llarg, no cap a la finestra
canvas.create_text(50,50,
                   text="Lorem ipsum dolor sit amet, consectetur adipiscing \
elit. Curabitur eu arcu in risus consectetur elementum. \
Suspendisse cursus vestibulum nunc vitae facilisis. \
Aliquam mollis commodo mauris venenatis euismod. ",
                   fill='blue')

# Text blau, molt llarg, en una caixa de 200 píxels
canvas.create_text(175,150,
                   text="Lorem ipsum dolor sit amet, consectetur adipiscing \
elit. Curabitur eu arcu in risus consectetur elementum. \
Suspendisse cursus vestibulum nunc vitae facilisis. \
Aliquam mollis commodo mauris venenatis euismod. ",
                   width=200,
                   fill='blue')

root.mainloop()          # Aquesta comanda mostra el resultat i ens
                         # permet interactuar amb la GUI.
```

El resultat es pot veure a la figura 2.8.

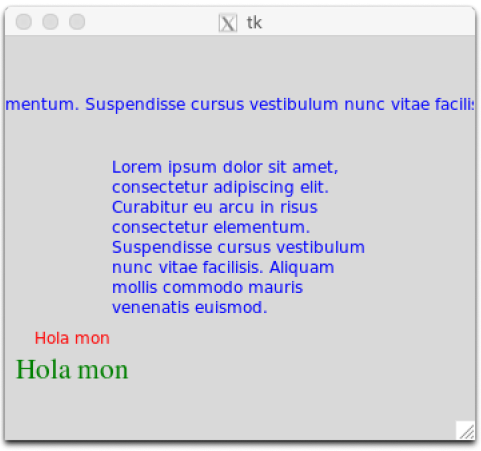
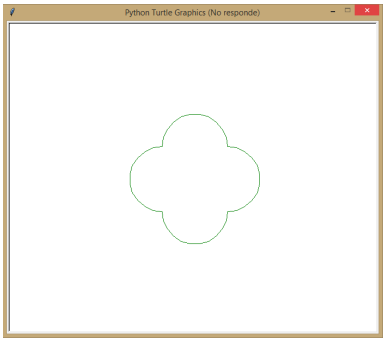
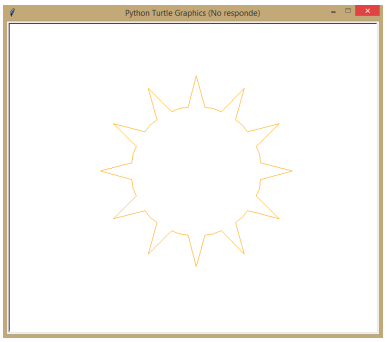


Figura 2.8: Test amb text.

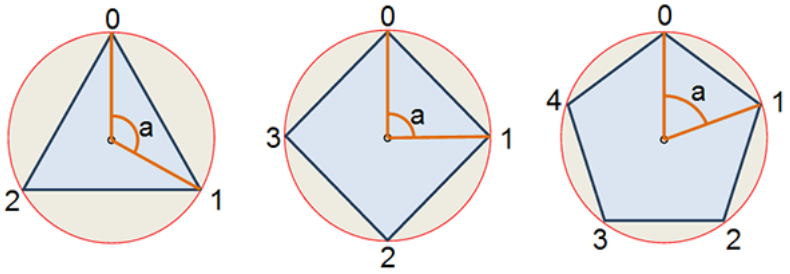
Exercici 2.6.7 *Volem dissenyar amb* `tkinter` *un jardí. En una zona hi volem posar flors del camp i la volem delimitar amb quatre semicercles iguals de 60 punts de radi. Centra la figura al mig de la finestra.*



Exercici 2.6.8 *Volem dibuixar amb* `tkinter` *el Sol. Volem posar-hi dotze raigs, cada un dels quals correspon als dos costats iguals d'un triangle isòsceles de longitud igual a la meitat del radi del Sol que formen un angle de 30º. Calcula la figura perquè quedi totalment simètrica i centrada. El radi és de 120 punts.*



Exercici 2.6.9 *Reprodueix una d'aquestes figures en una finestra amb* `tkinter`.



2.7 Funcions

Les funcions permeten la reutilització de codi: escriure codi que fa una tasca definida i que es pot utilitzar moltes vegades. D'aquesta manera el codi es pot fer més compacte (evitant repeticions tedioses) i més llegible.

2.7.1 Definició d'una funció

Per implementar una funció en un programa Python cal:

1. Una línia on es defineix la funció usant la paraula clau **def** seguida del nom de la funció.
2. Una descripció de la funció, marcada entre cometes triples.
3. Les sentències que componen la funció. **Important:** les línies de codi de la sentència han d'estar indentades per indicar que pertanyen a la funció. Si s'acaba la indentació s'entén que s'acaba la funció.

   def Name(Parameters): "' docstring "' Statements return

El nom pot ser una paraula o un grup de paraules. L'única restricció és que no pot ser Python reserved words.

```
identifier ::=  (letter|"_") (letter | digit | "_")*
letter     ::=  lowercase | uppercase
lowercase  ::=  "a"..."z"
uppercase  ::=  "A"..."Z"
digit      ::=  "0"..."9"
```

`Docstring` i `return` no són necessaris.

2.7.2 Cridant una funció

Definint una funció només dona el seu nom, especifica els paràmetres que s'han d'incloure a la funció i estructura els blocs de codi. Pots executar-la cridant-la des d'una altra funció o directament des de `prompt`.

Exemple:

```
In [1]: def my_function(s):
            print("My string: {}".format(s))

        my_function("Hello")
```

```
My string: Hello
```

Si volem dibuixar un quadrat amb **`tkinter`**:

```
In [3]: import tkinter

        def drawSquare():
            ''' Dibuixa un quadrat usant tkinter
            '''
            root = tkinter.Tk()                  # Creem la finestra principal
            canvas = tkinter.Canvas(root,        # A la finestra, creem
                                    height=200, # un llenç de 200 x 200 píxels
                                    width=200)
```

```
        canvas.pack()                          # I el situem a la finestra principal

        canvas.create_line( 50,  50,  50, 150)
        canvas.create_line( 50, 150, 150, 150)
        canvas.create_line(150, 150, 150,  50)
        canvas.create_line(150,  50,  50,  50)
        root.mainloop()

drawSquare()
```

Hem dibuixat el quadrat (figura 2.9), però què passa si volem dibuixar el quadrat i després fer una nova acció?

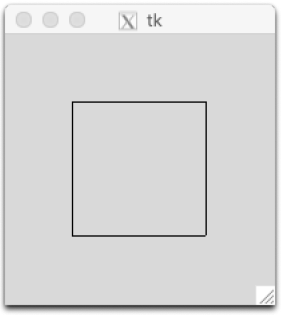


Figura 2.9: Test amb text.

Les variables `root` i `canvas`, que fan referència a la finestra i al llenç que hem creat, no estan definides fora de la funció per la qual cosa no és possible usar la mateixa finestra en altres funcions.

Per poder-ho fer, podem crear un llenç fora de la nostra funció i passar-lo com a argument (o paràmetre):

```
In [4]: def drawSquare(c):
            ''' Dibuixa un quadrat usant tkinter
                c: ha d'estar inicialitzada com a tkinter.Canvas
            '''
            c.create_line( 50,  50,  50, 150)
            c.create_line( 50, 150, 150, 150)
            c.create_line(150, 150, 150,  50)
            c.create_line(150,  50,  50,  50)

In [5]: import tkinter

        root = tkinter.Tk()                    # Creem la finestra principal
        canvas = tkinter.Canvas(root,          # A la finestra, creem
                                height=200,    # un llenç de 200 x 200 píxels
                                width=200)

        canvas.pack()                          # I el situem a la finestra principal
        drawSquare(canvas)
        root.mainloop()
```

Podem modificar la nostra funció per poder indicar l'origen i grandària del quadrat:

```
In [6]: def drawSquare(c, x0, y0, side):
            ''' Dibuixa un quadrat usant tkinter
                c: ha d'estar inicialitzada com a tkinter.Canvas
                x0, y0: és la coordenada del vèrtex superior esquerre
                side: és la mida del costat
            '''
            c.create_line( x0, y0,  x0, y0+side)
            c.create_line( x0, y0+side, x0+side, y0+side)
            c.create_line( x0+side, y0+side, x0+side, y0)
            c.create_line( x0+side, y0, x0, y0)
```

I ara usar-la per dibuixar altres figures:

```
In [7]: import tkinter

        root = tkinter.Tk()                 # Creem la finestra principal
        canvas = tkinter.Canvas(root,       # A la finestra, creem
                                height=250, # un llenç de 250 x 250 píxels
                                width=250)

        canvas.pack()                       # I el situem a la finestra principal

        x = 25
        y = 25
        lado = 50
        delta = 10

        drawSquare(canvas, x, y, lado)
        x = x + delta
        y = y + delta
        lado = lado + 2 * delta
        drawSquare(canvas, x, y, lado)
        x = x + delta
        y = y + delta
        lado = lado + 2 * delta
        drawSquare(canvas, x, y, lado)
        x = x + delta
        y = y + delta
        lado = lado + 2 * delta
        drawSquare(canvas, x, y, lado)
        x = x + delta
        y = y + delta
        lado = lado + 2 * delta
        drawSquare(canvas, x, y, lado)

        root.mainloop()
```

El resultat es pot veure a la figura 2.10.

O bé figures més complexes:

```
In [8]: import tkinter

        root = tkinter.Tk()                 # Creem la finestra principal
```

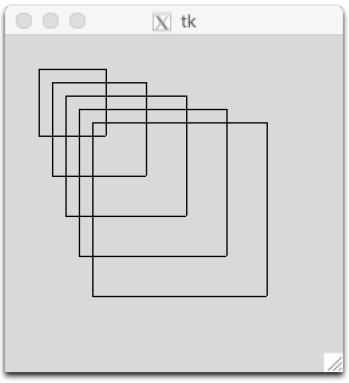


Figura 2.10: Múltiples quadrats.

```
canvas = tkinter.Canvas(root,        # A la finestra, creem
                        height=250, # un llenç de de 250 x 250 píxels
                        width=250)

canvas.pack()                        # I el situem a la finestra principal

x = 50
y = 75
costat = 100
drawSquare(canvas, x, y, costat)

x = x + costat/4
y = y + costat/4
drawSquare(canvas, x, y, costat)

x = x + costat/4
y = y - costat/4
drawSquare(canvas, x, y, costat)

x = x - costat/4
y = y - costat/4
drawSquare(canvas, x, y, costat)

x = x - costat/4
y = y + costat/4
drawSquare(canvas, x, y, costat)

x = x + costat/4
y = y + costat/4
drawSquare(canvas, x, y, costat)

root.mainloop()
```

El resultat es pot veure a la figura 2.11.

Important: les funcions han d'estar definides prèviament en el codi per poder-les usar.

L'exemple següent mostra un error típic. Nota l'ordre erroni:

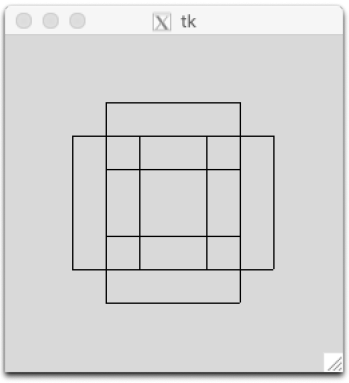


Figura 2.11: Múltiples quadrats.

```
In [9]: helloWorld()
```

```
        def helloWorld():
            print("Hello World!")

        ---------------------------------------------------------------------------

        NameError                                 Traceback (most recent call last)

        <ipython-input-9-367bbb3feb6b> in <module>
    ----> 1 helloWorld()
          2
          3 def helloWorld():
          4     print("Hello World!")


        NameError: name 'helloWorld' is not defined
```

Com que l'intèrpret de Python va executant línia a línia, no és suficient que la funció estigui definida en el mateix bloc. Ha d'estar definida prèviament (bé en el mateix bloc, bé en un bloc executat prèviament):

```
In [10]: def helloWorld():
             ''' Hello world function
             '''
             print("Hello World!")
```

```
In [11]: helloWorld()
```

```
Hello World!
```

Atenció: un problema típic quan s'usa Notebook és que una vegada s'ha definit una funció, aquesta queda "en memòria" fins que es tanca el programa o es reinicialitza el *kernel*. Això fa que encara que s'esborri el codi de la funció o es canviï el nom, l'original continuï "disponible", de manera que ens trobem amb exercicis que funcionaven a l'aula però que en rebre'ls per a correcció ja no funcionen a causa dels canvis que s'hi han fet.

Una funció també pot cridar-ne unes altres, com la `drawSquare()` en el nostre exemple. D'aquesta manera el dibuix anterior pot estar creat per una altra funció:

```
In [12]: def fewSquares(c, x0, y0, size):
             ''' Dibuixa en el llenç usant la funció drawSquare
                 c: ha d'estar inicialitzada com tkinter.Canvas
                 size: mida del costat
             '''


             x = x0
             y = y0
             costat = size
             drawSquare(canvas, x, y, costat)

             x = x + costat/4
             y = y + costat/4
             drawSquare(canvas, x, y, costat)

             x = x + costat/4
             y = y - costat/4
             drawSquare(canvas, x, y, costat)

             x = x - costat/4
             y = y - costat/4
             drawSquare(canvas, x, y, costat)

             x = x - costat/4
             y = y + costat/4
             drawSquare(canvas, x, y, costat)

             x = x + costat/4
             y = y + costat/4
             drawSquare(canvas, x, y, costat)
```

```
In [13]: import tkinter

         root = tkinter.Tk()                  # Creem la finestra principal
         canvas = tkinter.Canvas(root,        # A la finestra, creem
                                 height=250, # un llenç de 250 x 250 píxels
                                 width=250)

         canvas.pack()                        # I el posem a la finestra principal

         x = 50
         y = 75
         size = 20
         fewSquares(canvas, x, y, size)

         x = 100
         y = 150
         size = 60
         fewSquares(canvas, x, y, size)
```

```
root.mainloop()
```

El resultat es pot veure a la figura 2.12.

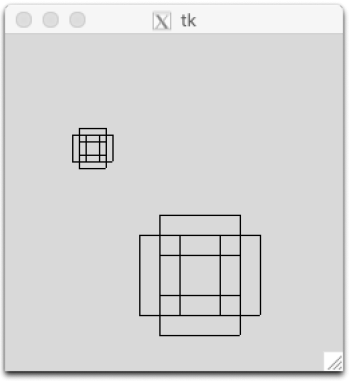


Figura 2.12: Múltiples quadrats amb jerarquia de funcions.

En analitzar un programa no sempre s'ha de llegir simplement de dalt a baix; s'ha de tenir en compte el flux d'execució per saber en quines condicions s'executa cada part.

Encara que les funcions poden tenir qualsevol nombre de paràmetres (o arguments) s'ha d'anar amb compte a l'hora de definir-los i usar només els necessaris.

Exercici 2.7.1 *Defineix una funció que sigui una equació de segon grau on es puguin variar els coeficients a, b, c i que permeti obtenir el seu valor numèric donant un valor a x. Fes una comprovació.*

2.7.3 La sentència `return`

Una funció pot fer accessible un resultat a l'entorn que la crida. Per a això és necessari utilitzar la sentència **`return`**:

```
In [14]: def parabola(x, a, b, c):
             y = a*x**2 + b*x + c

             return y

         valor = parabola(5, 1, 3, 2)
         print(valor)
```

```
42
```

Podem utilitzar el resultat de la funció guardat en una variable:

```
In [14]: result = parabola(5, 1, 3, 2)
         second_value = result * 4
         print(second_value)
```

```
168
```

Una funció acaba quan la sentència **`return`** és trobada, o quan la indentació finalitza. Si no es troba la sentència **`return`**, una funció donarà **`None`**.

`None` és freqüentment utilitzat a Python per representar l'absència d'un valor.

Exercici 2.7.2 *Defineix una funció f(x, y) que calculi la hipotenusa d'un triangle rectangle i una funció g(h, z) que calculi la superfície d'un triangle rectangle en què un catet és la hipotenusa calculada per f(x, y) i l'altre és z. Resol el problema per a $x = 3u$, $y = 4u$ i $z = 6u$.*

2.7.4 Com podem saber què fa una funció?

Primer cal posar un nom clar. Però no és suficient. Per aquesta raó el que hem d'incloure-hi és un comentari just després de la declaració. Això s'anomena **docstring**:

```
In [17]: def myFunction(parameter0, parameter1):
             """My function is used to perform a sequence of statements.
             The parameters that it uses are:
             parameter0: It is of type real and gives the...
             parameter1: It is a string that describes...
             My function returns a value of type int..."""
             result = 0
             print("My nice function...")

             return result
```

Al Notebook, aquesta informació és donada posant el nom de la funció seguit de l'interrogant.

```
In [18]: myFunction?
```

Python també mostrarà aquest *docstring* si utilitzem la funció **help**:

```
In [19]: help( myFunction )
```

```
Help on function myFunction in module __main__:

myFunction(parameter0, parameter1)
    My function is used to perform a sequence of statements.
    The parameters that it uses are:
    parameter0: It is of type real and gives the...
    parameter1: It is a string that describes...
    My function returns a value of type int...
```

Exercici 2.7.3 *Donada una sèrie de terme general a(n)= 3n^2-1 defineix la funció sèrie(n), i el seu* ***docstring****.*

Calcula el terme a(8) i recupera tot seguit el contingut del ***docstring****.*

2.7.5 Identificació de funcions

La identificació de funcions és un aspecte important de la programació, i una de les tasques més confuses. En aquesta secció, intentarem definir una estratègia per identificar les funcions i millorar la codificació.

L'estratègia proposada segueix una aproximació de dalt a baix i de baix a dalt. El primer pas és identificar el problema a solucionar. Usualment un problema és massa complex per ser solucionat en un pas. Per aquesta raó, el problema s'ha de dividir en petits problemes fins que puguin ser solucionats de cop.

Aquest procés és equivalent a solucionar un problema matemàtic. Si hem de trobar la solució d'un problema, primer analitzarem la informació de la seva expressió. D'acord amb aquesta

informació analitzarem quins passos o tasques són necessaris per solucionar el problema. Si una tasca o pas és massa gran, l'haurem de subdividir en problemes més petits. Finalment, tots els subproblemes seran a un nivell atòmic. Això vol dir que seran al suficientment simples per ser solucionats mitjançant eines que tinguem a mà. Això s'anomena *aproximació de dalt a baix*.

A continuació començarem a construir la solució solucionant cada subproblema atòmic, i donant els resultats als passos o tasques següents. Això ens permetrà trobar la solució final. Això és una aproximació de baix a dalt, on construirem la solució des dels fonaments.

És important tenir en compte que les dues aproximacions usualment no es fan una després de l'altra, ja que són barrejades, i a vegades un problema que sembla ser atòmic s'ha de subdividir altre cop. I també poden haver-hi biblioteques que donaran solucions a un problema.

Considerarem cada subproblema una funció. Per tant, una funció pot ser una extensió d'una funció si soluciona un problema atòmic, o una funció nòdul si soluciona un problema utilitzant altres funcions. Com a resultat, solucionar un problema requereix un arbre de funcions que intercanviï la informació per assolir el resultat desitjat.

Un exemple podria ser dibuixar una *dummy*.

2.7.5.1 Definició del problema

Com hem dit volem dibuixar una *dummy*. Per tant, hem de definir la funció **`dummy()`**. Hem de pensar també en els arguments. Com a primera aproximació posarem la tortuga. A continuació pensarem quines tasques ha de fer. Com a primera aproximació dibuixarem una *dummy* bàsica (figura 2.13):

```
In [25]: import turtle

         def dummy(t):
             '''dummy draws a dummy on the screen using the turtle t.'''
             # TOT: draw the dummy
             pass #Indicates that nothing has to be done
```

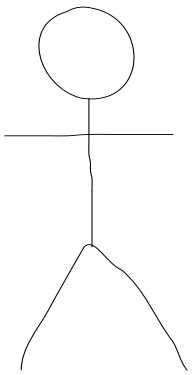


Figura 2.13: *Dummy*.

Podem veure que està composta per un cap fet amb un cercle, el cos que és una línia, els braços dibuixats amb una sola línia, i les cames utilitzant una v invertida:

- Head
- Body
- Arms
- Legs

La funció `dummy` esdevé un arbre que es mostra a la figura 2.14.

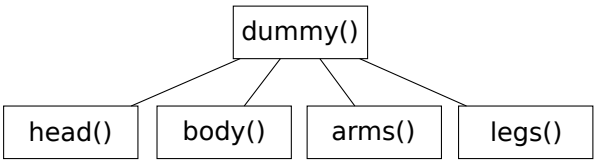


Figura 2.14: *Dummy tree.*

Els arguments tindran en compte la tortuga i la mida, per assegurar una relació correcta:

```
In [28]: def head(t, size):
             '''head draws a head.'''
             # TOT: draw the head
             pass

         def body(t, size):
             '''head draws a body.'''
             # TOT: draw the body
             pass


         def arms(t, size):
             '''head draws a head.'''
             # TOT: draw the arms
             pass


         def legs(t, size):
             '''head draws a head.'''
             # TOT: draw the legs
             pass
```

Ara hem d'omplir les funcions per dibuixar els diferents elements.

Exercici 2.7.4 *Omple la funció per dibuixar els diferents elements.*

Nota. Per simplificar, dins la funció `dummy` mou la tortuga cap al punt inicial on les subtasques han de començar. Dins les funcions cridades fes els moviments necessaris, i recorda de tornar a la posició inicial.

Ara, imaginem que volem dibuixar un somriure al *dummy* inicial com el mostrat a la figura 2.15.

Per fer-ho inclourem noves funcions dins la funció **`head()`**:

- Eyes
- Smile

Exercici 2.7.5 *Escriu les funcions que permetran dibuixar una cara somrient.*

Seguint aquest procés, podem millorar el *dummy* fins a assolir el nivell de detall desitjat. També podem afegir l'argument mida a la funció `dummy`, i calcular les mides d'altres elements proporcionalmet.

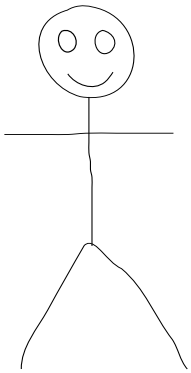


Figura 2.15: *Dummy smiling.*

En resum, el procés de solucionar un problema en un ordinador és un procés iteratiu, ja que el problema és solucionat en passos diferents seguint els processos de dalt a baix i de baix a dalt per assolir la qualitat desitjada.

2.7.6 Arguments de la funció

Hem vist la manera més simple de definir els arguments d'una funció en Python:

```
def drawSquare(t):
```

La funció té un nombre fix d'arguments que s'han de donar tots en el mateix ordre quan siguin cridats. Aquests s'anomenen **Required positional arguments**.

```
In [30]: # Function definition is here
         def printme(sentence):
            "This prints a passed string into this function"
            print(sentence)
            return

         # Now you can call printme function
         printme('Hello World!')

         # Now you can call printme function
         printme()
```

```
Hello World!
```

```
        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-30-1a586cecd3a4> in <module>()
          9
         10 # Now you can call printme function
    ---> 11 printme()


        TypeError: printme() missing 1 required positional argument: 'sentence'
```

2.7.6.1 Arguments Keyword

A vegades és convenient passar els arguments en un ordre diferent del que hem utilitzat en la definició de funció. Això es pot fer identificant els arguments amb els noms dels paràmetres. Per exemple:

```
In [31]: # Function definition is here
         def printme(sentence1, sentence2):
            "This prints a passed string into this function"
            print(sentence1, sentence2)
            return

         # Now you can call printme function
         printme('Hello World!', 'Hey again!')
         print()
         printme(sentence2 = 'Hello World!', sentence1 = 'Hey again!')
```

```
Hello World! Hey again!

Hey again! Hello World!
```

2.7.6.2 Arguments per defecte

Un argument per defecte és un argument que assumeix un valor per defecte si el valor no és donat en cridar la funció per aquest argument. Si qualsevol dels paràmetres formals en la definició de funció són declarats amb el format `arg = value`, aleshores tindràs l'opció de no especificar un valor per a aquests arguments en cridar la funció. Si no especifiques un valor, aleshores aquest paràmetre tindrà el valor per defecte donat quan s'executi la funció.

L'exemple següent dona una idea dels arguments per defecte. Imprimirà una edat per defecte si no s'ha superat:

```
In [32]: # Function definition is here
         def printinfo( name, age = 35 ):
            "This prints a passed info into this function"
            print("Name: ", name)
            print("Age ", age)
            return

         # Now you can call printinfo function
         printinfo("miki", 50)
         printinfo("miki")
```

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Els arguments per defecte s'han de definir després dels arguments de posició que s'han de donar sempre en cridar una funció. En cridar el `keyword` d'un codi, els arguments el poden usar per donar els arguments de posició i defecte.

```
In [33]: # Now you can call printinfo function
         printinfo(age=50, name="miki")
```

```
         printinfo(name="miki")
```

```
Name:  miki
Age  50
Name:  miki
Age  35
```

Nota. Els arguments `non-keyword` s'han de donar sempre abans de qualsevol argument `keyword`.

```
In [34]: printinfo(name="miki", 40)
```

```
          File "<ipython-input-34-fd2f475e39fa>", line 1
        printinfo(name="miki", 40)
                               ^
    SyntaxError: non-keyword arg after keyword arg
```

Un ús correcte dels arguments simplifica la reutilització de codi.

2.7.7 Variables locals i globals

Les variables definides dins d'una funció només són disponibles dins la funció:

```
In [35]: def test_function():
             a = 3

         test_function()

         print(a)
```

```
        ---------------------------------------------------------------------------
        NameError                                 Traceback (most recent call last)

        <ipython-input-35-bd3b9212ddb6> in <module>()
          4 test_function()
          5
    ----> 6 print(a)


        NameError: name 'a' is not defined
```

D'altra banda, les variables definides fora d'una funció es poden veure dins una funció:

```
In [36]: b = 5
         def print_b():
             print(b)
```

```
        print_b()
```

```
5
```

Però no poden ser modificades:

```
In [37]: c = 3
         def modify_c():
             c = 6
             print(c)
         modify_c()
         print(c)
```

```
6
3
```

Si és necessari modificar una variable global, la funció ha d'incloure la sentència global:

```
In [38]: d = 4
         def modify_global_d():
             global d
             d = 7
             print(d)

         modify_global_d()
         print(d)
```

```
7
7
```

La utilització de la global és requerida per minimitzar el risc d'errors.

Exercici 2.7.6 *Defineix la funció* `f(x, y=1)` *que retorna el valor numèric per a un binomi* $p = x^2 - y$. *Calcula el valor de la funció per a* $x = 3$. *Repeteix el càlcul per a* $x = 3$ *i* $y = 2$, *tenint en compte que la* y *és un argument per defecte.*

Exercici 2.7.7 *Modifica la variable* x *de la funció anterior, de manera que en lloc de calcular el valor per a* $x = 3$ *ho faci per a* $x = 5$. *Utilitza la sentència global.*

2.8 Control de flux (I). Les sentències `if` i `while`

Els programes que hem fet fins ara s'executaven en una **seqüència lineal**, seguint les sentències en ordre de la primera a l'última, però en programes reals sol ser necessari **modificar el flux d'execució a partir de sentències condicionals basades en l'estat i valors de les dades de programa**.

2.8.1 Control de flux: conceptes

Bàsicament hi ha dos tipus de sentències que ens permeten modificar el flux d'execució:

- La sentència condicional. Permet executar les instruccions que hi ha sota amb la indentació si l'operació booleana és certa (`True`).
- Els bucles. Permeten executar unes determinades línies de codi, indentades sota la sentència de bucle, de forma repetitiva mentre l'operació booleana que hi ha al bucle sigui certa.

Per a aquest fi, Python proporciona sentències per a bifurcacions i bucles que discutirem en aquesta secció.

2.8.1.1 Exemple de bifurcació

Llença una moneda a l'aire. Si surt cara guanyes, si surt creu perds. La implementació d'aquest codi requereix bifurcacions.

2.8.1.2 Exemple de bucle

La realització del sumatori següent

$$\sum_{i=1}^{N} i$$

Sense sentències de control de flux no té gaire sentit implementar un sumatori; el codi no és flexible, no té gaire utilitat.

2.8.2 Bifurcacions

Intentem implementar el codi que simula el llançament de la moneda.

Podem fer servir la biblioteca **`random`** per tal de generar un valor aleatori. La funció **`randint(inici, final)`** genera un enter aleatori.

$$\texttt{inici} <= \mathit{EnterGenerat} <= \texttt{final}$$

```
In [1]: import random as rnd
        n = rnd.randint(0,1)

        # Fem l'associació
        #    CARA = 0
        #    CREU = 1

        # Per tant,

        #    Si n és igual a cara, guanyo
```

```
        #    En cas contrari, n és igual a creu, perdo
        print(n)
```

```
0
```

Un altre mètode de la biblioteca **random** és **choice(<sequence>)**. Ens retorna un element aleatori d'una seqüència no buida:

```
In [2]: import random as rnd

        # Selecció aleatòria d'un element de la seqüència
        sequence = 'Ab5,.9K-_m'
        ch = rnd.choice(sequence)
        print( "Random choice from '{}' is '{}'".format(sequence, ch) )
```

```
Random choice from 'Ab5,.9K-_m' is '.'
```

Podem generar un caràcter aleatori amb **choice(<sequence>)** fent servir la cadena **string.ascii_letters** com a argument:

```
In [3]: import string
        import random as rnd

        print("Characters contained in 'string.ascii_letters':\n\t{}\n".format(string.ascii

        # Draw a letter in string.ascii_letters
        rand_letter = rnd.choice(string.ascii_letters)
        print("Random letter generated: ", rand_letter)
```

```
Characters contained in 'string.ascii_letters':
        abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ

Random letter generated:  c
```

També podem seleccionar un element aleatori d'un conjunt:

```
In [4]: import random

        elements = ('cat', 'dog', 'turtle', 'iguana', 1, 5, 8, 'pig')

        # Draw an element
        rand_elem = random.choice(elements)
        print("Random choice: ", rand_elem)
```

```
Random choice:  cat
```

La biblioteca **random** conté altres mètodes. Per a més detalls, pots consultar la seva documentació.

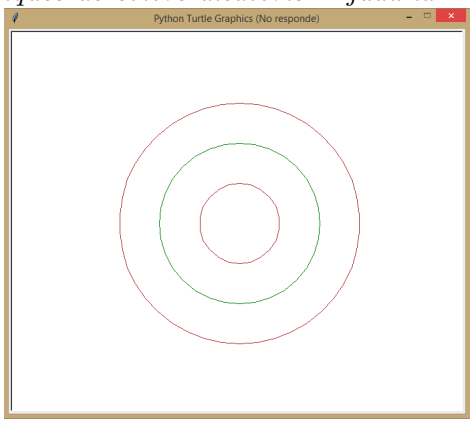
Exercici 2.8.1 *Ús de les funcions* ***range*** *i* ***random****: calcula* $c = n(n+1)$ *per a* n *pertanyent a l'interval (3,9). Comprova-ho amb:*

```
1 a = range(3,9)
```

```
2 n = random.choice(a)
3 c =n*(n+1)
4 print(n, c)
```

Exercici 2.8.2 *Introduint el rang per teclat calcula per a un valor de x elegit aleatòriament: a = x^2, i b = quàdruple de l'extrem superior. Fes un print(x, a, b).*

Exercici 2.8.3 *Disposem d'una paleta amb els colors c = (red, orange, blue, green). Vol dibuixar tres circumferències concèntriques de colors aleatoris. Ajuda-la.*



2.8.2.1 Sentència `if`

Les sentències **`if`** permeten l'execució condicional d'un bloc de codi basant-se en expressions booleanes. Si l'expressió booleana dona com a resultat **`True`**, el bloc de codi s'executa; si dona com a resultat **`False`**, no s'executa.

```
if (condició):
    # bloc de codi que s'executa si l'expressió és True

# El flux del codi continua
```

El bloc de codi al qual s'aplica la condició es marca mitjançant la indentació.

```
In [5]: # Primera aproximació a la solució del problema
        CARA = 0
        CREU = 1
        if n == CARA:
            print("Guanyo!")
        if n == CREU:
            print ("Ooooh, he perdut")
```

```
Guanyo!
```

En aquest cas anterior, el codi executa les dues sentències **`if`**. Comprova primer si **`n`** és igual a **`CARA`**. Si aquesta condició és certa, imprimeix "Guanyo!".

Tot seguit executa la instrucció següent **`if`** i comprova si **`n`** és igual a **`CREU`**. Si aquesta condició és falsa, no imprimeix "Ooooh, he perdut".

Existeix l'opció que un cop s'ha detectat que una condició és certa, no verifiqui la resta de condicions.

```
if (condició1):
    # Bloc de codi 1
```

```
elif (condició2):
    # Bloc de codi 2

else:
    # Bloc de codi 3

# Altre codi
```

En aquest cas el flux és el següent:

- Si **condició1** és **True**: s'executa el **bloc de codi 1** i després continua amb **Altre codi**.
- Si **condició1** és **False**: s'avalua la **condició2**.
  - Si **condició2** és **True**: s'executa el **bloc de codi 2** i després continua amb **Altre codi**.
  - Si **condició2** és **False**: s'executa el **bloc de codi 3** i després continua amb **Altre codi**.

Vegem-ne un exemple:

```
In [6]: import random as rnd
        n = rnd.randint(0,2)
        CARA = 0
        CREU = 1
        if n == CARA:
            print ("surt cara")
            n = CREU
        elif n == CREU:
            print ("surt creu")
            n = 2
        else:
            print("aguantar-se de cantell")
```

```
aguantar-se de cantell
```

Nota que independentment del valor de **n**, només s'executa una bifurcació, tot i que explícitament fem canviar el valor de **n** per donar-li altres valors. Vegem què passa si posem aquesta seqüència només amb **if**:

```
In [7]: import random as rnd
        n = rnd.randint(0,2)
        CARA = 0
        CREU = 1
        if n == CARA:
            print ("surt cara")
            n = CREU
        if n == CREU:
            print ("surt creu")
            n = 2
        if n == 2:
            print("aguantar-se de cantell")
```

```
surt cara
```

```
surt creu
aguantar-se de cantell
```

Tenim doncs aquestes dues opcions. En certs casos necessitarem que un cop el programa entra en una de les bifurcacions, no executi la resta d'opcions. En altres casos necessitarem que es comprovin totes les opcions. Dependrà del tipus de problema amb el qual ens enfrontem.

Les comprovacions poden ser tan complicades com sigui necessari. Es poden fer servir operadors booleans com **and** i **or**, i comparadors com: * A == B A és igual que B? * A < B A és menor que B? * A > B A és major que B? * A <= B A és menor o igual que B? * A >= B A és major o igual que B? * A != B A és diferent de B?

Vegem-ne un exemple senzill:

```
In [8]: a = 1
        b = 2
        c = 3
        d = 4

        if a==1 and b==2 and ( c!=5 or d>4 ) :
            print ("Condició certa")
        else:
            print ("Condició falsa")
```

```
Condició certa
```

Exercici 2.8.4 *Fes un programa que generi un número aleatori entre 1 i 10 i que demani el nom del jugador. El programa preguntarà al jugador un número entre 1 i 10. Es guanya si s'encerta el valor. Es poden introduir tres valors com a màxim.*

```
In [9]: #El codi aquí
        import random as rnd
        import sys

        # Ordena el codi correctament...

        '''
        y = int(input("introdueix un número entre 1 i 10 "))
        if y == numeroMagic:
            print("Encert a la segona! ")
            sys.exit(0)
        x = int(input("introdueix un número entre 1 i 10 "))
        if x == numeroMagic:
            print("Encert a la primera! ")
            sys.exit(0)
        print ("no has encertat! Torna-ho a provar ")
        print ("no has encertat! Torna-ho a provar ")
        z = int(input("introdueix un número entre 1 i 10 "))
        if z == numeroMagic:
            print("Encert a la tercera! ")
            sys.exit(0)
        jugador1 = input("nom del jugador ")
        numeroMagic = rnd.randint(1,10)
```

```
        print("Ooooh! {} no has encertat el número. Era {}. Els teus valors eren {}, {} i {
        '''

        print('')
```

Es podria millorar aquest codi fent servir `if - elif - else`? Fes l'exercici implementant ara `if` i `else`.

Exercici 2.8.5 *Tirem dos daus n vegades; quantes vegades els valors obtinguts difereixen en una unitat? Introdueix el valor n per teclat i expressa'n clarament els resultats.*

Exercici 2.8.6 *Introdueix un nombre enter pel teclat, fes un programa que calculi tots els seus divisors. Recorda que tot nombre és divisor de si mateix i la unitat és divisor de qualsevol nombre.*

2.8.3 Bucles

Els bucles ens permeten fer operacions iteratives sempre que una condició determinada sigui certa. En el cas del sumatori,

$$\sum_{i=1}^{N} i$$

la condició d'iteració seria que `i` ha de ser menor o igual que `N`.

2.8.3.1 Sentència `while`

Repeteix un bloc de sentències mentre la condició de control associada sigui certa. La condició de control és avaluada cada vegada, abans d'executar el bloc de codi. Si la condició és falsa, d'entrada no s'executa mai el bloc.

Nota. Compte amb els bucles infinits!

La sintaxi bàsica és:

```
while condició :
    # executa si condició és certa
    sentència 1
    ...

else:
    # executa quan condició és falsa
    sentència 2
```

La sentència **`else`** és pròpia de Python (existeix en pocs llenguatges de programació) i permet executar codi quan sortim del bucle amb la condició falsa.

Exemple:

Solucionem el problema del sumatori

```
In [10]: # Assignem a i el valor incial 1 i demanem el valor de N

         i = 1
         N = int(input("introdueix el valor final del sumatori"))
```

```
        suma = 0
        while i <= N:
            suma = i + suma
            i = i + 1
        print("el valor del sumatori és",suma)
```

```
introdueix el valor final del sumatori 2
el valor del sumatori és 3
```

2.8.3.2 break

Dins els bucles **while** també es pot usar la sentència **break**. Aquesta sentència permet interrompre l'execució d'un bloc **while**. Quan s'executa aquesta comanda l'execució salta el que queda del bloc **while** i segueix amb el programa. Tot i que pot anar tota sola, normalment, aquesta sentència va lligada amb la instrucció **if**. En aquest cas el codi després de la sentència **else** no s'executa i la condició de control no es torna a avaluar.

La sintaxi bàsica és:

```
while (condició de control):
    # bloc de codi

    if (condició de sortida):
        break

    # més codi

else:
    # Si la condició de control s'ha avaluat False

# I més codi
```

Exemple:

```
In [11]: # Seguint amb l'exemple anterior, fem un sumatori però en cas que la
         # suma sigui superior a un determinat valor, voldrem que el programa
         # deixi de fer les operacions.

         i = 1
         N = int(input("introdueix el valor final del sumatori "))
         M = int(input("introdueix el valor màxim que pot assolir la suma: "))
         suma = 0

         while i <= N:
             suma = i + suma
             i = i + 1
             if suma > M:
                 print("límit assolit")
                 break
         if suma < M:
             print("el valor del sumatori és", suma)
         else:
             print ("per al terme {}-èsim hem superat el màxim {}".format(i,M))
```

```
introdueix el valor final del sumatori 4
introdueix el valor màxim que pot assolir la suma: 5
límit assolit
per al terme 4-èsim hem superat el màxim 5
```

2.8.3.3 `continue`

Una altra sentència que es pot usar dins els bucles **`while`** és **`continue`**. La sentència **`continue`** permet interrompre l'execució del bloc **`while`** en curs i saltar directament a l'iteració següent. Igual que en el cas anterior, la sentència **`continue`** pot executar-se directament, però normalment va lligada a una instrucció **`if`**.

Exemple:

Calculem els valors de `y` per a la funció:

$$y = \frac{1}{x - a}$$

Primer pas: introdueix per consola un valor enter per a `a`.

Segon pas: demana els valors enters de `x` per als quals vols saber el valor de `y`.

Repeteix aquest procés 3 vegades.

```
In [12]: a = int(input("introdueix el paràmetre a: "))
         MAXIM = 3
         contador = 0

         while contador < MAXIM:
             x = int(input("introdueix el valor de x: "))
             if x == a:
                 print ("per a aquest valor tenim un infinit")
                 continue
             else:
                 y = 1/(x-a)
                 print (y)
                 contador = contador + 1
```

```
introdueix el paràmetre a: 2
introdueix el valor de x: 2
per a aquest valor tenim un infinit
introdueix el valor de x: 1
-1.0
introdueix el valor de x: 5
0.3333333333333333
introdueix el valor de x: 3
1.0
```

Exercici 2.8.7 *Sentència `if` + `while`*

Volem provar la nostra punteria llançant una pedra a una diana d'1 metre de diàmetre i que es troba a una alçada d'1 metre des del centre fins al terra. Ens trobem a una distància de 10

metres. Considera només el cas de dues dimensions i troba la velocitat inicial i l'angle necessari per fer diana, tenint en compte que l'alçada inicial del llançament és igual a 1 metre.

```
In [13]: # La resposta aquí...

         from math import sin, cos
         PI = 3.1416
         G = 9.8
         dianaEixX = 10
         dianaSuperiorEixY = 1.5
         dianaInferiorEixY = 0.5
         y0 = 1
         encert = False
         contador = 0

         '''
             Poseu les instruccions següents en l'ordre correcte

         y = y0+Vy0*tDiana-0.5*G*tDiana**2
         tDiana = dianaEixX/Vx
         if y > dianaInferiorEixY and y < dianaSuperiorEixY:
         Vx = velInicial*cos(angleRadians)
         Vy0 = velInicial*sin(angleRadians)
         print("Objectiu assolit")
         angle = float(input("introdueix l'angle en graus: "))
         angleRadians = angle * PI/180
         encert = True
         else:
         print("torna-ho a provar!")
         contador = contador + 1
         velInicial = float(input("Introdueix la velocitat m/s: "))
         while encert == False:
         '''

         print (x,y)
         print ("has encertat en {} llançaments".format(contador))

         # Millora el codi per dir a quina alçada de la diana hem encertat
```

```
3 1.0
has encertat en 0 llançaments
```

Exercici 2.8.8 *Considerant els nombres d'1 fins a 44, ambdós inclosos, calcula la seva suma ignorant els que són múltiples de dos.*

Exercici 2.8.9 *Podem passar un nombre natural a binari utilitzant la funció* `bin` *de Python, o escrivint un petit programa. Calcula-ho de les dues maneres introduint el nombre enter a convertir pel teclat.*

Exercici 2.8.10 *Per a dos nombres, a pertanyent al rang (3,8) i b al (4,6), calcula c = a^2+b^2 i efectua els print(a, b, c), però si c>50, indica que has superat aquest valor i surt del programa.*

Exercici 2.8.11 *Per a dos nombres* `a` *i* `b` *de l'1 al 5, efectua tots els productes possibles sense*

que hi hagi resultats repetits.

Exercici 2.8.12 *Per a un nombre* x *elegit aleatòriament a l'interval (3,12), imprimeix el seu valor fins que aquest valor sigui 5. El programa ha d'indicar el nombre d'iteracions efectuades.*

Exercici 2.8.13 *Introdueix un nombre enter, n, per pantalla i cerca aleatòriament un nombre k dintre del rang (1,n). Si k és múltiple de 3, suma-li 5; si no, multiplica'l per 2 i resta-li una unitat. Repeteix les operacions sempre que el valor resultat sigui més petit que n + 10. Expressa clarament els resultats.*

2.9 Tipus de dades estructurades

En seccions anteriors hem estudiat els tipus de dades bàsics, numèrics (`int, float, complex` i `bool`) i cadenes de caràcters (*strings*), inclosos a l'especificació de Python. En aquest capítol ens centrarem en altres tipus de dades que reben el nom de *col · leccions de dades estructurades*. Així com els tipus bàsics només emmagatzemen un únic valor o cadena, totes les col · leccions de dades es caracteritzen per poder emmagatzemar més d'un valor i per ser també iterables. Com ja veurem, les col · leccions són molt flexibles i el seu ús és molt freqüent en els programes.

Els tipus de dades estructurades que estudiarem a continuació són:

1. Llistes.
2. Tuples i tuples amb nom.
3. Diccionaris.
4. Conjunts (*sets*).

2.9.1 Llistes

Una llista (**list** en Python), de forma similar a un *string*, és una seqüència ordenada de valors. Cada dada (o valor) individual s'anomena ***element*** de la llista i **s'hi pot accedir mitjançant índexs**. Un índex és un valor enter que indica la posició de l'element en la seqüència ordenada a la qual es vol accedir. Aquest índex especifica un desplaçament, com ja veurem, des de l'origen (esquerra) o el final (dreta) de la llista.

Una llista es diferencia d'un *string* (que només pot contenir caràcters) en:

- Una llista pot contenir una seqüència composta per qualsevol tipus de valors. Poden conviure diferents tipus de dades bàsics dins d'una mateixa llista.
- Una llista és **mutable**. Això vol dir que, a diferència d'un *string*, podem modificar-ne el contingut després de la inicialització.

2.9.1.1 Creació de llistes

Com els tipus bàsics de dades que ja hem vist, una llista es pot associar a un nom de variable, el qual permetrà després usar-la i accedir als seus diferents elements.

Les llistes poden ser creades mitjançant el constructor (**list**):

```
In [1]: l_week_days = list()
        # Creem una llista d'strings mitjançant el constructor
        l_week_days = list(('Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'))

        # Mostrem el contingut
        print('l_week_days =', l_week_days)
```

```
l_week_days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

O amb claudàtors [ ]:

```
In [2]: l_numbers = ["Hola", 1, 2, 3, 4, 5, 6, 7, 8, 9]

        # Mostrem el contingut
        print('l_numbers =', l_numbers)
```

```
l_numbers = ['Hola', 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Per facilitar la creació de llistes numèriques d'enters Python proporciona la funció **range()**. Podem reescriure l'exemple anterior usant aquesta funció de la forma següent:

```
In [3]: l_numbers_01 = list ( range(1, 10) )

        # Mostrem el contingut
        print('l_numbers_01 =', l_numbers_01)
```

```
l_numbers_01 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

La sintaxi d'aquesta funció és: **`range(inici, final[, pas])`**

- Els arguments de la funció **range** han de ser nombres enters. Si no s'especifica `pas`, el seu valor serà 1. Si no s'especifica `inici`, el primer valor serà el 0
- Si `pas` és positiu, el contingut ve determinat per la fórmula:

$$r[i] = start + step * i \quad on \quad i >= 0 \; and \; r[i] < stop$$

- Si `pas` és negatiu, el contingut ve determinat per la fórmula:

$$r[i] = start + step * i \quad on \quad i >= 0 \; and \; r[i] > stop$$

Vegem-ne un exemple on s'especifica `pas`:

```
In [4]: l_numbers_02 = list ( range(10, 110, 10) )

        # Mostrem el contingut
        print('l_numbers =', l_numbers_02)
```

```
l_numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
```

De forma similar, podem construir una llista amb els mateixos valors que l'exemple anterior, però en ordre decreixent:

```
In [5]: l_numbers_03 = list ( range(100, 0, -10) )

        # Mostrem el contingut
        print('l_numbers =', l_numbers_03)
```

```
l_numbers = [100, 90, 80, 70, 60, 50, 40, 30, 20, 10]
```

Es poden crear llistes buides, a les quals posteriorment es poden afegir elements:

```
In [6]: empty_list = []
        empty_plus = list()

        print (empty_list)
        print (empty_plus)
```

```
[]
[]
```

Podem també crear llistes que continguin valors de diferents tipus:

```
In [7]: # Llista mixta
        l_mixed = [134, 2, 'a', 3.14159, True, 'Julius', 3.0+6j]

        # Mostrem el contingut
        print('l_mixed =', l_mixed)
```

```
l_mixed = [134, 2, 'a', 3.14159, True, 'Julius', (3+6j)]
```

2.9.1.2 Operacions bàsiques

Com que les llistes són seqüències com els **string**, suporten moltes de les operacions que podem dur a terme amb les cadenes de text:

Operació	Descripció
`<list>` + `<list>`	Unió o concatenació de llistes
`<list>` * `<int>`	Repetició dels elements de la llista
len(`<list>`)	Nombre d'elements que conté la llista
`<value>` in `<list>`	Indica si el valor indicat és a la llista

2.9.1.2.1 Nombre d'elements o mida d'una llista (`len`)

```
In [8]: l_00 = ['Hi', 'Bye', 'See You', 'Hello']
        l_01 = list ( range(8) )

        # Càlcul de la longitud
        n_l_00 = len( l_00 )
        n_l_01 = len( l_01 )

        text = 'La llista {} té {} elements.'
        print( text.format(l_00, n_l_00) )
        print( text.format(l_01, n_l_01) )
```

```
La llista ['Hi', 'Bye', 'See You', 'Hello'] té 4 elements.
La llista [0, 1, 2, 3, 4, 5, 6, 7] té 8 elements.
```

2.9.1.2.2 Unió o concatenació (+)

```
In [9]: l_02 = list ( range(5) )
        l_03 = list ( range(100, 105) )
        l_04 = ['Ernie', 'Paul', 'Lucy']

        # Concatenació de llistes
        print ('Concatenant', l_02, 'i', l_03, '::', l_02 + l_03)
        print ('Concatenant', l_04, 'i', l_02, '::', l_04 + l_02)
```

```
Concatenant [0, 1, 2, 3, 4] i [100, 101, 102, 103, 104] :: [0, 1, 2, 3, 4, 100, 101, 102, 10
Concatenant ['Ernie', 'Paul', 'Lucy'] i [0, 1, 2, 3, 4] :: ['Ernie', 'Paul', 'Lucy', 0, 1, 2
```

2.9.1.2.3 Repetició (*)

```
In [10]: print ( ['Hallelujah', 'Brother!'] * 3 )
```

```
['Hallelujah', 'Brother!', 'Hallelujah', 'Brother!', 'Hallelujah', 'Brother!']
```

2.9.1.2.4 in

Retorna `True` o `False` en funció de si l'element especificat es troba o no contingut en la llista donada:

```
In [11]: transport = ['truck', 'car', 'bike', 'cicle', 'bus', 'train', 'rocket']

         print ( 'train' in transport )
         print( 'jet' in transport )
```

```
True
False
```

2.9.1.3 Accedint al contingut d'una llista

Tal com ja vàrem veure amb el tipus **string**, podrem també accedir als diferents elements d'una llista mitjançant l'operador claudàtor []. Podem establir la classificació següent en funció del nombre d'elements als quals accedirem:

1. **Indexació**. Accedim únicament a un element de la llista. Especifiquem un desplaçament o índex, el qual ha de ser un valor enter.

   - Mitjançant valors **positius** d'índex s'accedeix des de l'inici (esquerra) de la llista.
   - Mitjançant valors **negatius** d'índex s'accedeix des del final (dreta) de la llista.

   Sintaxi:

   ```
   <nom_llista>[<valor_índex>]
   ```

2. **`Slicing` o subconjunts**. En Python un segment (conjunt de valors) d'una llista s'anomena *`slice`*. Per tant, emprarem `slicing` si volem accedir a la vegada a més d'un element d'una llista. Com en el cas de la indexació, també admet l'ús d'enters positius i negatius en funció de si hi accedim des del principi o des del final de la llista.

   Sintaxi:

   ```
   <nom_llista>[<índex_inicial>:<índex_final>:<pas>]
   ```

   - On:
     - Si s'omet el valor d'`índex_inicial`, aquest pren el valor de **0** (inici de la llista).
     - Si s'omet el valor d'`índex_final`, aquest pren el valor de **`len(<nom_llista>)`**.
     - Si s'omet el valor del `pas`, aquest pren el valor de **1**.
     - Igual que el **range()**, l'element especificat per a l'`índex_final` mai està inclòs.

Important:

- L'índex del primer element d'una llista des de l'esquerra, igual que en un *string*, és el *0*.
- Nota però que en cap cas es poden usar índexs més enllà de la mida de la llista (genera un error).

2.9.1.3.1 Indexació

Accedim a cadascun dels elements d'una llista especificant-ne la posició des de l'inici:

```
In [12]: # Creem la llista
         latin_list = ['Lorem', 'ipsum', 'dolor sit amet']
         print ("Contingut: ", latin_list)

         # S'accedeix al valor usant el seu índex, començant per zero
         print ('Valor índex 0: ', latin_list[0])
         print ('Valor índex 1: ', latin_list[1])
         print ('Valor índex 2: ', latin_list[2])
```

```
Contingut:  ['Lorem', 'ipsum', 'dolor sit amet']
Valor índex 0:  Lorem
Valor índex 1:  ipsum
Valor índex 2:  dolor sit amet
```

Podem accedir al darrer element de dues maneres diferents: des de l'inici (desplaçament positiu) i des del final (desplaçament negatiu).

```
In [13]: print ('[0] Darrer element: ', latin_list[ len(latin_list)-1 ])
         print ('[1] Darrer element: ', latin_list[-1])
```

```
[0] Darrer element:  dolor sit amet
[1] Darrer element:  dolor sit amet
```

I el penúltim el podem obtenir així:

```
In [14]: print ('[0] Penúltim element: ', latin_list[ len(latin_list)-2 ])
         print ('[1] Penúltim element: ', latin_list[-2])
```

```
[0] Penúltim element:  ipsum
[1] Penúltim element:  ipsum
```

A més, en cas que els elements d'una llista siguin *strings*, és possible accedir a una posició determinada d'un *string* determinat. Vegem-ne un exemple:

```
In [15]: fruits = ['Cherry', 'Banana', 'Grape', 'Kiwi', 'Lemon']

         print ("La darrera lletra de '{}' és una '{}'.".format( fruits[2], fruits[2][-1]) )
```

```
La darrera lletra de 'Grape' és una 'e'.
```

2.9.1.3.2 `Slicing`

Utilitzant `slicing` podem accedir a tots els elements d'una llista (començant per l'inici, esquerra) que es trobin en posicions senars:

```
In [16]: # Llista de ciutats
         cities = ['Vladivostok', 'Lyon', 'Glasgow', 'Rome', 'Munich', 'Miami', 'Kyoto', 'Za

         print ( cities[0:len( cities):2] )
```

```
['Vladivostok', 'Glasgow', 'Munich', 'Kyoto']
```

Nota que el resultat és també una llista, en aquest cas de 4 elements.

Podem obtenir el mateix resultat ometent els índexs inicials i finals (els quals prendran el seu valor per defecte):

```
In [17]: print ( cities[::2] )
```

```
['Vladivostok', 'Glasgow', 'Munich', 'Kyoto']
```

Podem obtenir la mateixa subllista de ciutats però en ordre invers tot accedint a la llista de ciutats des del final i mitjançant un pas negatiu:

```
In [18]: print ( cities [len(cities)-2::-2] )
```

```
['Kyoto', 'Munich', 'Glasgow', 'Vladivostok']
```

Podem obtenir tota la llista en ordre invers si la recorrem de dreta a esquerra de la següent manera:

```
In [19]: print ( cities [::-1] )
```

```
['Zagreb', 'Kyoto', 'Miami', 'Munich', 'Rome', 'Glasgow', 'Lyon', 'Vladivostok']
```

Podem obtenir subllistes de valors consecutius (pas 1, valor per defecte si no especifiquem res):

```
In [20]: print ( cities[:3] )
         print ( cities[3:] )
         print ( cities[2:4] )
```

```
['Vladivostok', 'Lyon', 'Glasgow']
['Rome', 'Munich', 'Miami', 'Kyoto', 'Zagreb']
['Glasgow', 'Rome']
```

2.9.1.4 Ús dels elements d'una llista

Els elements d'una llista poden tractar-se com a variables normals, per fer qualsevol operació permesa:

```
In [21]: integers = list( range(2,20,3) )

         # Sumem alguns elements de la llista
         add_value = integers[0]-integers[3]+integers[4]
         print ("Enters: ", integers)
         print ("Suma dels índexs 0,3,4: ", add_value)
```

```
Enters:  [2, 5, 8, 11, 14, 17]
Suma dels índexs 0,3,4:  5
```

2.9.1.5 Afegir i eliminar elements d'una llista

Python proporciona diversos mètodes per manipular el contigut d'una llista:

Mètode	Descripció
`extend(values)`	Amplia la llista afegint tots els elements de `value` al final de la llista
`append(value)`	Afegeix un element al final de la llista
`pop(index)`	Elimina i **retorna** l'element amb l'índex donat
`insert(index, value)`	Afegeix un valor en una posició determinada
`remove(value)`	Elimina el primer element que conté el valor donat

En el cas del mètode **pop()**, si no especifiquem l'índex es pren el darrer element.

Tots aquests mètodes s'apliquen sobre una llista usant la sintaxi **<nom_llista>.mètode()**

- **Important**: les modificacions s'efectuen directament sobre la llista indicada. Excepte `pop`, els altres mètodes retornen `None`. Cal evitar construccions de la forma: **list_test = list_test.append(X)**

Vegem en acció tots els mètodes anteriorment descrits:

```
In [22]: # Creem la llista original
         sports = ["cycling", "basketball"]
         print ("Llista original: ", sports)

         # Afegim més d'un element al final de la llista
         sports.extend( ["taekwondo", "rugby"] )
         print ("Extending: ", sports)

         # Afegim un únic element al final de la llista
         sports.append( ["running", "gymnastics"] )
         print ("Appending: ", sports)

         # Traiem un element donant l'índex i guardem el valor esborrar a 'el'
         el = sports.pop(3);
         print ("Popping '{}': {}".format(el, sports))

         # Afegim un element al mig de la llista
         sports.insert(2, "handball") # Cal indicar l'índex on s'insereix el valor
         print ("Inserting index 2 ", sports)

         # Eliminem un element a partir del seu contingut
         sports.remove("basketball") # Elimina el primer element que contingui "X"
         print ("Removing 'basketball': ", sports)

         # Traiem el darrer element i guardem el valor esborrar a 'el'
         el = sports.pop();
         print ("Popping {}: {}".format(el, sports))
```

```
Llista original:  ['cycling', 'basketball']
Extending:  ['cycling', 'basketball', 'taekwondo', 'rugby']
```

```
Appending:  ['cycling', 'basketball', 'taekwondo', 'rugby', ['running', 'gymnastics']]
Popping 'rugby': ['cycling', 'basketball', 'taekwondo', ['running', 'gymnastics']]
Inserting index 2  ['cycling', 'basketball', 'handball', 'taekwondo', ['running', 'gymnastic
Removing 'basketball':  ['cycling', 'handball', 'taekwondo', ['running', 'gymnastics']]
Popping ['running', 'gymnastics']: ['cycling', 'handball', 'taekwondo']
```

Nota les diferències entre el mètode `extend` i l'`append`. El primer permet inserir en una llista una sèrie de valors que són en una altra llista. En canvi, el segon insereix únicament un element nou al final, la segona llista de valors (com a llista).

2.9.1.6 Sentència `del`

També és possible esborrar elements d'una llista directament mitjançant la sentència **del**, ja sigui indicant-li un índex o bé un conjunt amb `slicing`:

```
In [23]: test_list = list ( range(5, 55, 5) )
         print('Llista original: ', test_list)

         del test_list[3]
         print('Esborrem l'element 3: ', test_list)

         del test_list[::2]
         print('Esborrem els elements senars: ', test_list)
```

```
Llista original:  [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]
Esborrem l'element 3:  [5, 10, 15, 25, 30, 35, 40, 45, 50]
Esborrem els elements senars:  [10, 25, 35, 45]
```

2.9.1.7 Modificar elements d'una llista

En tractar-se d'un tipus de dada mutable podem modificar-ne el contingut directament especificant `indexació` o `slicing`:

```
In [24]: cereals = ['maize', 'rice', 'wheat', 'barley', 'millet', 'quinoa']
         print('Llista original de cereals: ', cereals)

         # Modificant el contingut d'un únic element
         cereals[1] = 'sorghum'
         print('Llista cereals: ', cereals)

         # Modificant el contingut d'un subconjunt d'elements
         cereals[2:5:2] = ['oats', 'fonio']
         print('Llista de cereals: ', cereals)
```

```
Llista original de cereals:  ['maize', 'rice', 'wheat', 'barley', 'millet', 'quinoa']
Llista de cereals:  ['maize', 'sorghum', 'wheat', 'barley', 'millet', 'quinoa']
Llista de cereals:  ['maize', 'sorghum', 'oats', 'barley', 'fonio', 'quinoa']
```

2.9.1.8 Funcions avançades

Presentem aquí algunes funcions avançades per treballar amb llistes, però et remetem a la documentació de Python per a un llistat complet de les eines disponibles.

2.9.1.8.1 Cerca d'índex

La funció `index()` permet buscar l'índex d'un element que contingui un patró donat.

```
In [25]: llista = ["A","B","C","D","E","A"]

         # Es pot buscar l'índex del primer element de la llista amb la funció "index"
         print("Índex de 'A': ",llista.index("A")) # Només dona el primer!
         print("Un altre índex de 'A': ",llista[1:6].index("A")) # Ara dona l'últim
         print("Índex de 'B': ",llista.index("B"))

         # Atenció: si l'element no existeix dona un error
         #print("Índex de 'p': ",llista.index("p"))
```

```
Índex de 'A':  0
Un altre índex de 'A':  4
Índex de 'B':  1
```

2.9.1.8.2 Ordenació

Una llista es pot ordenar, de forma ascendent o descendent, amb la funció `sort()`:

```
In [26]: llista = [5,6,2,9,6,1,0,4]
         print ("Abans d'ordenar: ", llista)

         llista.sort() # Nota: el contingut de la llista és substituït
         print ("Després d'ordenar: ", llista)

         # També es pot fer amb llistes de cadenes de text
         animals = ['lion', 'elefant', 'rat', 'jackal', 'gecko', 'lemur']
         print ("Abans d'ordenar: ", animals)

         animals.sort()
         print ("Després d'ordenar (ordre creixent): ", animals)

         animals.sort(reverse=True)
         print ("Després d'ordenar (ordre decreixent): ", animals)
```

```
Abans d'ordenar:  [5, 6, 2, 9, 6, 1, 0, 4]
Després d'ordenar:  [0, 1, 2, 4, 5, 6, 6, 9]
Abans d'ordenar:  ['lion', 'elefant', 'rat', 'jackal', 'gecko', 'lemur']
Després d'ordenar (ordre creixent):  ['elefant', 'gecko', 'jackal', 'lemur', 'lion', 'rat']
Després d'ordenar (ordre decreixent):  ['rat', 'lion', 'lemur', 'jackal', 'gecko', 'elefant
```

2.9.1.8.3 Capgirar una llista

Una llista es pot capgirar amb la funció `reverse()`:

```
In [27]: data_00 = list( range(100, 200, 10) )
         print ("Llista original: ", data_00)
         data_00.reverse()
         print ("Llista capgirada: ", data_00)
```

```
Llista original:  [100, 110, 120, 130, 140, 150, 160, 170, 180, 190]
Llista capgirada:  [190, 180, 170, 160, 150, 140, 130, 120, 110, 100]
```

Exercici 2.9.1 *Donada la llista x = [1, 5, 'a', 9, 'ab', 72, 4.2, 0.3]; efectua: print(x[0]), print(x[3]), print(x[3:6]), print(x[3:]), print(x[:3]), print(x[-1]), print(x[-3]), print(x[-3:]), print(x[:-3]). Modifica la llista canviant la a per z i mostra el resultat; continua l'exercici obtenint print(x[3:4]), print(x[3]), print(x[1:-1:2]), print(x[::-1]) i, finalment, print(x[11]). Comenta les operacions indicades a cada print i els resultats.*

Exercici 2.9.2 *Donada la llista x =[1,3,14,4,5,6,7,8,5,9], a/ Afegeix-hi l'element 10. b/ Insereix l'element 99 en el tercer lloc. c/ Elimina l'element 7. d/ Quantes vegades el nombre 5 està contingut a la llista? e/ Ordena la llista en ordre creixent. f/ Inverteix l'ordre de la llista. g/ Buida la llista. Mostra els resultats efectuant els prints corresponents.*

Exercici 2.9.3 *Amb les llistes es poden utilitzar operadors de relació. Donades les llistes: x =[1,2,3] i y = [2,4,6,8], comprova si són verdaderes o falses les relacions: a = x < y, b = x > y, i c = x! = y.*

Exercici 2.9.4 *Podem eliminar elements d'un llista segons el seu índex. Per tant, donada la llista a = [7, 'a', 'mn',0, 0.4], fes print(a) després d'eliminar el quart element, tot seguit i també segons el seu índex, elimina l'element 'mn'. Quina és la llista resultat?*

Exercici 2.9.5 *Repeteix l'exercici anterior utilitzant la funció* `remove` *i calcula la longitud de la llista obtinguda.*

Exercici 2.9.6 *Reprodueix la llista x = ['ab', 7, 'r', 2, 0] tres vegades i calcula la longitud de la llista resultant.*

Exercici 2.9.7 *Com sabem, l'operador + és alternatiu a la funció* `append` *per concatenar llistes. a/ Donada la llista x = [3,5,7] i la llista [3,2], concatena-les fent servir l'operador + de la manera seguent: x = x + [3,2] i fes print(x). Compara'n les dimensions. b/ Donades les llistes a = [7,'a', 6, 'b', 1] i b = [2, 7], fes c = a + b i print(c). Hi ha alguna diferència?*

Exercici 2.9.8 *Crea una llista de nombres aleatoris compresos entre 5 i 22, de longitud 10, i una llista de zeros de longitud 3.*

2.9.1.9 Llistes multidimensionals

Fins ara hem vist llistes "unidimensionals", on cada element queda identificat per un únic índex. Es poden definir llistes multidimensionals senzillament creant llistes els elements de les quals siguin altres llistes.

Per exemple

```
list_2D = [ [1,2] , [3,4] ]
```

Aquesta és una llista amb dos elements: `[1,2]` i `[3,4]`, però cadascun d'aquests elements és una altra llista de dos elements. El resultat és de fet una matriu 2 x 2 i es pot accedir als seus elements mitjançant dos índexs:

```
print(list_2D[0][1])
```

Vegem-ho a la pràctica usant una llista per guardar una matriu 3 x 3:

```
In [28]: # Creem una llista que conté una matriu 3 x 3
         matrix_3x3 = [ [1,2,3], [4,5,6], [7,8,9] ]

         sep = '------------\n'
```

```
        print("La meva matriu:", matrix_3x3)
        print(sep)

        print ( "Element (0): {} - {}".format(matrix_3x3[0], type(matrix_3x3[0])) )
        print ( "Element (1): {} - {}".format(matrix_3x3[1], type(matrix_3x3[1])) )
        print ( "Element (2): {} - {}".format(matrix_3x3[2], type(matrix_3x3[2])) )
        print(sep)

        # Imprimim alguns elements
        print ( "Element (0,0): {} - {}".format(matrix_3x3[0][0], type(matrix_3x3[0][0])) )
        print ( "Element (1,1): {} - {}".format(matrix_3x3[1][1], type(matrix_3x3[1][1])) )
        print ( "Element (2,2): {} - {}".format(matrix_3x3[2][2], type(matrix_3x3[2][2])) )
        print(sep)
```

```
La meva matriu: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
------------

Element (0): [1, 2, 3] - <class 'list'>
Element (1): [4, 5, 6] - <class 'list'>
Element (2): [7, 8, 9] - <class 'list'>
------------

Element (0,0): 1 - <class 'int'>
Element (1,1): 5 - <class 'int'>
Element (2,2): 9 - <class 'int'>
------------
```

Nota que les llistes multidimensionals, al contrari que les matrius algebràiques, no tenen per què ser regulars, com es veu en l'exemple següent.

```
In [29]: # Creem una llista 2D irregular
         list_2D = [ [1,2,3], [4,5], [7,8,9,0], [1] ]

         print ( "Element (0): {} - {}".format(list_2D[0], type(list_2D[0])) )
         print ( "Element (1): {} - {}".format(list_2D[1], type(list_2D[1])) )
         print ( "Element (2): {} - {}".format(list_2D[2], type(list_2D[2])) )
         print ( "Element (3): {} - {}".format(list_2D[3], type(list_2D[3])) )
```

```
Element (0): [1, 2, 3] - <class 'list'>
Element (1): [4, 5] - <class 'list'>
Element (2): [7, 8, 9, 0] - <class 'list'>
Element (3): [1] - <class 'list'>
```

```
In [30]: # Llista 1D
         list_1D = [1,2,3,4,5]
         print ("Llista: ", list_1D)
         print ("Mida de la llista: ", len(list_1D))
         print('------------\n')

         # Llista 2D
         list_2D = [[1,2], [3,4,5], [5]]
```

```
        print ("Llista: ", list_1D)
        print ("Nombre de files: ", len(list_2D))  # 3
        print ("Nre. de columnes de la fila 0: ", len(list_2D[0]))  # 2
        print ("Nre. de columnes de la fila 1: ", len(list_2D[1]))  # 3
        print ("Nre. de columnes de la fila 2: ", len(list_2D[2]))  # 1
```

```
Llista:  [1, 2, 3, 4, 5]
Mida de la llista:  5
------------

Llista:  [1, 2, 3, 4, 5]
Nombre files:  3
Nre. de columnes de la fila 0:  2
Nre. de columnes de la fila 1:  3
Nre. de columnes de la fila 2:  1
```

2.9.1.9.1 Llista de llistes heterogènies

Podem també treballar amb llistes els elements de les quals a la vegada siguin una altra llista de valors heterogenis. S'accedeix als seus elements de la mateixa manera que hem vist en l'apartat anterior:

```
In [31]: # Definim una llista cada element de la qual conté ['nom', 'edat', 'pes']
         person_list = [ ['Patty', 22, 58], ['Ricky', 44 , 82.5], ['Johnny', 18, 75] ]

         row = person_list[1]
         print('Fila 1: ', row)

         column = row[2]
         print('Element[1][2]: ', column)
```

```
Fila 1:  ['Ricky', 44, 82.5]
Element[1][2]:  82.5
```

Exercici 2.9.9 *Repassem algunes funcions per a llistes numèriques. Donada la llista x = [0,1,2,3,4,5,6,7,8,9], efectua:* `print(len(x))`, `print(max(x))`, `print(min(x))`, `print(sum(x))`. *En cada cas, què has obtingut?*

Exercici 2.9.10 *Hi ha llistes formades per altres llistes; per tant, donada la llista x = ['a', 3, ['m', 6, 7], 0.9, 5, ['e', 4]], efectua:*

a) `print(x[0])`

b) `print(x[1])`

c) `print(x[2])`

d) `print(x[2][2])`

e) substitueix l'element x[2][2] per la lletra k.

Comenta els resultats.

Exercici 2.9.11 *Podem procedir igualment per a llistes de nivells superiors. Sigui la llista x = [[[1,4,'s'],7],[1,8,5,['k',9]],'t', [9,'a']], efectua:*

a) `print(x[0])`

b) `print(x[1])`

c) `print(x[0][0])`

d) `print(x[0][1])`

e) `print(x[0][0][2])`

f) substitueix l'element x[0][0][1] per la lletra ñ.

Comenta els resultats.

2.9.2 Tuples

Les tuples són una estructura de dades molt similar a les llistes. La diferència entre els dos tipus de dades i la decisió de quan usar una o altra pot ser una mica subtil, possiblement més enllà dels objectius d'aquest curs.

En qualsevol cas, la principal diferència és:

- Les **tuples**, igual que els *strings* (i al contrari que les llistes i els diccionaris), són **immutables**: un cop definides no és possible alterar-ne el contingut.

Una tupla (**tuple** en Python), igual que un *string* i una llista, és una col · lecció de valors. Cada dada (o valor) individual s'anomena ***part*** de la tupla i **s'hi pot accedir mitjançant índexs**. Com ja hem vist per a les llistes, un índex és un valor enter que especifica la posició de l'element de la seqüència ordenada al qual es vol accedir. Aquest índex especifica un desplaçament, com ja veurem, des de l'origen (esquerra) o el final (dreta) de la llista.

2.9.2.1 Creació de tuples

Les llistes poden ser creades mitjançant el seu constructor (`tuple`). El contingut de la tupla ha d'anar entre parèntesi, i els diferents camps separats per comes:

```
In [32]: tuple_one = tuple( ("John", "Smith", 45, 1.70, ('name', 'last_name')) )
         print (tuple_one)
```

```
('John', 'Smith', 45, 1.7, ('name', 'last_name'))
```

Nota que el símbol que caracteritza una tupla són els **( )**, així com els [ ] caracteritzen una llista.

Com pots veure en l'exemple anterior, qualsevol de les parts d'una llista pot ser a la vegada un tipus de dada estructurat.

De forma similar a les llistes, una tupla pot crear-se mitjançant l'operador parèntesi (**( )**):

```
In [33]: tuple_two = ('cookies', 10, 'ham', 45, 'eggs', 5, 'apples', 8)
         print(tuple_two)
```

```
('cookies', 10, 'ham', 45, 'eggs', 5, 'apples', 8)
```

Podem també crear tuples buides tot i que després, a diferència de les llistes, no podrem actualitzar-ne el contingut:

```
In [34]: empty_tuple = ()
         empty_tuple_plus = tuple()
```

```
print(empty_tuple)
print(empty_tuple_plus)
```

```
()
()
```

2.9.2.2 Cas a tenir en compte

Si el que volem és una tupla que contingui un únic element, cal afegir al final i abans del parèntesi una coma:

```
In [35]: value = (56)
         tuple_one_value = (56,)

         print('Un valor:', value)
         print('Una tupla:', tuple_one_value)
```

```
Un valor: 56
Una tupla: (56,)
```

2.9.2.3 Operacions bàsiques

Igual que les llistes, les tuples suporten moltes de les operacions que podem fer amb les cadenes de text:

Operació	Descripció
<tuple> + <tuple>	Unió o concatenació de tuples
<tuple> * <int>	Repetició dels elements de la tupla
len(<tuple>)	Nombre d'elements que conté la tupla
<value> in <tuple>	Indica si el valor indicat és a la tupla

2.9.2.3.1 Nombre de parts o mida d'una tupla (len)

```
In [36]: print( "El nombre de parts de {} és {}".format(tuple_one, len(tuple_one)) )
         print( "El nombre d'elements de\n\t{}\n\tés {}".format(tuple_two, len(tuple_two)) )
```

```
El nombre de parts de ('John', 'Smith', 45, 1.7, ('name', 'last_name')) és 5
El nombre d'elements de
        ('cookies', 10, 'ham', 45, 'eggs', 5, 'apples', 8)
        és 8
```

2.9.2.3.2 Unió o concatenació (+)

```
In [37]: tuple_name = ('Ernie', 'Paul', 'Lucy')
         tuple_last_name = ('Williams', 'Howard', 'Johnson')

         new_tuple = tuple_name + tuple_last_name
         print(new_tuple)
```

```
('Ernie', 'Paul', 'Lucy', 'Williams', 'Howard', 'Johnson')
```

2.9.2.3.3 Repetició (*)

```
In [38]: tuple_ora = ('ora', 'pro', 'nobis', 123) * 3
         print(tuple_ora)
```

```
('ora', 'pro', 'nobis', 123, 'ora', 'pro', 'nobis', 123, 'ora', 'pro', 'nobis', 123)
```

2.9.2.3.4 in

Retorna `True` o `False` en funció de si l'element especificat es troba o no contingut en la tupla:

```
In [39]: print( "Tupla:\n\t", tuple_two )
         print( "\nL'element 'ham' existeix?", 'ham' in (tuple_two) )
```

```
Tupla:
         ('cookies', 10, 'ham', 45, 'eggs', 5, 'apples', 8)

L'element 'ham' existeix? True
```

Sempre podem comprovar si el valor està o no contingut en una tupla, i en funció del resultat fer una acció o una altra:

```
In [40]: tuple_two = ('cookies', 10, 'ham', 45, 'eggs', 5, 'apples', 8)

         if 'eggs' not in tuple_two:
             print('No valor "eggs"')
         else:
             print('Existeix "eggs"')
```

```
Existeix "eggs"
```

2.9.2.3.5 Accedint al contingut d'una tupla

Com en el cas de les llistes, s'accedeix a un valor mitjançant l'operador claudàtor []:

```
In [41]: # Diccionari en què el camp valor és de tipus simple
         tuple_three = ('qui', 'quae', 'quod', 'quorum', 'quarum')

         print ('Primer element: ', tuple_three[0])
         print ('Segon element: ', tuple_three[1])
         print ('Tercer element: ', tuple_three[2])
         print ('Quart element: ', tuple_three[3])
         print ('Cinquè element: ',tuple_three[4])
```

```
Primer element:  qui
Segon element:  quae
Tercer element:  quod
Quart element:  quorum
Cinquè element:  quarum
```

També és possible accedir a un conjunt de valors mitjançant *slicing*. Adrecem el lector a l'apartat corresponent de les llistes per veure el funcionament de l'*slicing*.

2.9.2.4 Modificar elements d'una tupla

Recordem que una tupla és un tipus de dades **immutable**. Per tant, el contingut no es pot modificar una vegada aquesta hagi estat inicialitzada. Si s'intenta alterar part del seu contingut, obtindrem un error similar al següent:

```
In [42]: t_cheese = ('manchego', 'roncal', 'idiazabal', 'garrotxa', 'tupí', 'cabrales')
         t_cheese[1] = 'cuajada'


        ---------------------------------------------------------------------------

        TypeError                                 Traceback (most recent call last)

        <ipython-input-42-f53f9c420a54> in <module>()
          1 t_cheese = ('manchego', 'roncal', 'idiazabal', 'garrotxa', 'tupí', 'cabrales')
    ----> 2 t_cheese[1] = 'cuajada'


        TypeError: 'tuple' object does not support item assignment
```

2.9.2.5 Conversions

Una tupla es pot convertir en una llista i utilitzar qualsevol mètode d'aquest tipus:

```
In [43]: t_cheese = ('manchego', 'roncal', 'idiazabal', 'garrotxa', 'tupí', 'cabrales')

         l_cheese = list(t_cheese)
         l_cheese.append('cuajada')
         l_cheese.sort()
         print ( l_cheese )
```

```
['cabrales', 'cuajada', 'garrotxa', 'idiazabal', 'manchego', 'roncal', 'tupí']
```

2.9.2.6 Arguments d'un `print`

Una tupla es pot passar com un argument d'un `print` i imprimir les seves parts separadament. Els elements de la tupla usaran els especificadors de format de forma successiva, com es veu en l'exemple següent:

```
In [44]: tuple_person_00 = ("Percy", "Strait", 45, 1.70)
         print ("Nom: %s  Cognom: %s  Edat: %d anys  Alçada: %f m." % tuple_person_00)

         tuple_person_01 = ("John", "Hopkins", 24, 1.98)
         print ("Nom: %s  Cognom: %s  Edat: %d anys  Alçada: %f m." % tuple_person_01)
```

```
Nom: Percy  Cognom: Strait  Edat: 45 anys  Alçada: 1.700000 m.
Nom: John  Cognom: Hopkins  Edat: 24 anys  Alçada: 1.980000 m.
```

2.9.2.7 Desempaquetament d'elements d'una tupla

Amb una única sentència podem assignar directament cadascun dels seus elements a variables diferents i després operar amb elles:

```
In [45]: point_3D_1 = (0.98, 0.45, 34.2)
         point_3D_2 = (15, 89.45, 2)

         x1, y1, z1 =  point_3D_1
         x2, y2, z2 =  point_3D_2

         print( 'Suma de components: ({}, {}, {})'.format(x1+x2, y1+y2, z1+z2) )
```

```
Suma de components: (15.98, 89.9, 36.2)
```

2.9.2.8 Retorn de múltiples valors des d'una funció

A vegades és necessari que una funció retorni més d'un valor. Això es pot aconseguir retornant una llista, però és més pràctic fer-ho com una tupla.

En aquest segon cas es pot rebre com una tupla o com els seus valors separadament, com es veu en l'exemple següent.

```
In [46]: import random

         def return_2_values():
             """ Aquesta funció és un exemple de retorn de dos valors en forma de tupla
             """
             x = random.randint(0, 100)
             y = random.randint(500, 1e3)

             # Retornem una tupla amb els dos valors
             return (x, y)

         # Podem cridar la funció i obtenir els dos valors a la
         # vegada, en variables separades
         val_x, val_y = return_2_values()
         print ( "[0] val_x = {}, val_y = {}".format(val_x, val_y) )

         # També es pot rebre com una tupla, si cal
         tuple_val = return_2_values()
         print ( "[1] val_x = {}, val_y = {}".format(tuple_val[0], tuple_val[1]) )
```

```
[0] val_x = 78, val_y = 909
[1] val_x = 29, val_y = 632
```

2.9.3 Tuples amb nom (*named tuples*)

Python permet etiquetar amb un nom els elements constituents d'una tupla mitjançant les **named tuples**. Això facilita l'ús d'aquests components ja que es poden referenciar amb els noms assignats.

2.9.3.1 Creació de *named tuples*

Les *named tuples* es creen mitjançant el constructor **namedtuple()**, present al mòdul **collections** (nota que cal importar-lo per usar-les). El primer argument és `namedtuple` i el segon és un *string* que conté els noms de cadascun dels atributs, separats per una coma o un espai en

blanc. Un cop definida una `namedtuple` es pot usar per crear instàncies individuals amb la seva estructura, com es veu en l'exemple següent:

```
In [47]: # Cal fer aquesta importació per usar les tuples amb nom
         from collections import namedtuple

         # Definim una tupla amb nom per representar punts en el pla
         # Les seves dues components són les coordenades x i y
         Point = namedtuple('point2D', 'x y')

         # Creem un parell de punts usant aquesta tupla amb nom
         p1 = Point(1.0, 5.0)
         p2 = Point(y=2.5, x=1.5)

         print (p1)
         print (p2)
```

```
point2D(x=1.0, y=5.0)
point2D(x=1.5, y=2.5)
```

Un altre exemple:

```
In [48]: from collections import namedtuple

         person = namedtuple('record', 'name, age, jobs')
         johnny = person(name='Johnny Guitar', age=25, jobs=['musician', 'painter'])

         print(johnny)
```

```
record(name='Johnny Guitar', age=25, jobs=['musician', 'painter'])
```

2.9.3.2 Accedint al contingut d'una *named tuple*

Igual que en una tupla convencional, es pot accedir a un valor mitjançant l'operador claudàtor [] però en aquest cas també s'hi pot accedir a través de la seva etiqueta:

```
In [49]: # Usem la tupla Point definida anteriorment
         # Podem accedir a les coordenades a partir del seu nom o de la forma usual
         scalar_prod_1 = p1[0]*p2[0] + p1[1]*p2[1]  # Accés per índex
         scalar_prod_2 = p1.x*p2.x + p1.y*p2.y  # Accés per nom

         print (scalar_prod_1)
         print (scalar_prod_2)
```

```
14.0
14.0
```

Un altre exemple més:

```
In [50]: from math import sqrt

         length_1 = sqrt((p1.x-p2.x)**2 + (p1.y-p2.y)**2)
         length_2 = sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)
```

```
        print (length_1)
        print (length_2)
```

```
2.5495097567963922
2.5495097567963922
```

També podem fer el desempaquetament dels diferents elements, tal com hem vist en les tuples convencionals:

```
In [51]: from collections import namedtuple

        person = namedtuple('record', 'name, age, jobs')
        johnny = person(name='Johnny Guitar', age=25, jobs=['musician', 'painter'])

        name, age, jobs = johnny

        print(name, age, jobs, sep=' - ')
```

```
Johnny Guitar - 25 - ['musician', 'painter']
```

Exercici 2.9.12 *Donada la tupla x = ('a', 2, 0.5, 'm', 7, 'abd'), efectua:*

a) `print(x[0])`

b) `print(x[2])`

c) `print(x[:2])`

d) `print(x[2:])`

e) substitueix l'element x[1] per la lletra b.

Comenta els resultats.

Exercici 2.9.13 *Una tupla pot contenir components de tipus llista i podem canviar elements dintre de la llista, però no fora d'ella. Donada la tupla x = (5, 'a', [3,[2,7]], 6, (4, 1)), comprova i comenta els resultats de les operacions següents:*

a) `print(x[0])`

b) `print(x[2])`

c) `print(x[2][0])`

d) `print(x[2][1][1])`

e) canvia l'element x[2][1][1] per 49

f) l'element x[4][1] per 20.

Exercici 2.9.14 *Els parèntesis són opcionals per definir tuples. Donada la tupla x = ('p', 4, 7, 0.34, 'ads', 207, 'g'), fes els* `print(x)` *escrivint-la amb parèntesis i sense. Comenta el resultat.*

Exercici 2.9.15 *Donades la tupla x = (1, 2, 3, 4, 5) i la tupla y = (3, 6, 9, 12, 15), calcula la tupla que tingui per components el producte component a component de les dues tuples anteriors.*

2.9.4 Diccionaris

Així com les llistes són una col · lecció ordenada o una seqüència de valors, els diccionaris contenen un conjunt de dades **no ordenat**.

Es tracta d'una espècie de conjunt de parelles, en què el primer element de la parella és la clau (**key**) i el segon es correspon al valor associat a aquesta clau. En els diccionaris no s'accedeix a les dades mitjançant un `índex` numèric o `slicing`, tal com fèiem amb les llistes o les tuples, sinó que s'hi accedeix mitjançant la clau. És a dir, quan accedim a un diccionari, donada una clau determinada, obtenim el seu valor associat.

S'anomenen *diccionaris* precisament per la seva similitud amb els diccionaris de paper tradicionals. En aquests últims, s'accedeix a un significat determinat a través del valor d'una clau, en aquest cas una paraula.

Important:

- Les claus han de ser úniques (no es poden repetir en un mateix diccionari).
- Les claus han de ser immutables; per tant, usarem *strings*, nombres o tuples.

2.9.4.1 Creació de diccionaris

Els diccionaris poden ser creats mitjançant el seu constructor (**dict**):

```
In [52]: telephones = dict( [('Steve', 99090), ('Nick', 1234), ('Wayne', 78666)] )
         print(telephones)
```

```
{'Wayne': 78666, 'Nick': 1234, 'Steve': 99090}
```

En l'exemple anterior el diccionari s'especifica mitjançant parelles **(clau, valor)** i és assignat a la variable `telephones`. Concretament, la clau del primer element del diccionari és la cadena de text **'Steve'** i el valor associat és **99090**, la segona clau és **'Nick'** i el valor **1234**, etc.

També podem crear-ne un de manera simplificada si les claus són *strings* simples:

```
In [53]: movies_dicc = dict(Life_of_Brian=1979, Forrest_Gump=1994, Casablanca=1942, Star_Tre
         print(movies_dicc)
```

```
{'Life_of_Brian': 1979, 'Forrest_Gump': 1994, 'Star_Trek': 1966, 'Casablanca': 1942}
```

Tal com es pot apreciar en fer un *print*, l'ordre en què es mostren els elements continguts en un diccionari pot no correspondre amb l'ordre de creació.

De forma similar a les llistes, un diccionari pot crear-se mitjançant l'operador claus (**{ }**):

```
In [54]: inventory_dicc = {'cookies': 10, 'ham': 45, 'eggs': 5, 'apples': 8}
         print(inventory_dicc)
```

```
{'apples': 8, 'cookies': 10, 'eggs': 5, 'ham': 45}
```

El valor associat a una clau pot ser de qualsevol tipus, incloent-hi tipus de dades estructurades com ara llistes o altres diccionaris:

```
In [55]: person_dicc={'Patty': [22, 58], 'Ricky': [44 , 82.5], 'Johnny': [18, 75]}
         print(person_dicc)

         heter_dicc={"hello": ["h","e","l","l","o"], "numbers": [5,4,3,2,1]}
```

```
        print(heter_dicc)

        basket_dicc={'Jordan': {'name': 'Michael', 'age': 40}, 'Bird': {'name': 'Larry', 'a
        print(basket_dicc)
```

```
{'Johnny': [18, 75], 'Ricky': [44, 82.5], 'Patty': [22, 58]}
{'numbers': [5, 4, 3, 2, 1], 'hello': ['h', 'e', 'l', 'l', 'o']}
{'Jordan': {'name': 'Michael', 'age': 40}, 'Bird': {'name': 'Larry', 'age': 55}}
```

Les claus no han de ser necessàriament *strings*, també poden ser nombres enters:

```
In [56]: int_keys_dicc = {2: 'Firebird', 1: 'Mustang',  4: 'Camaro', 100: 'Corolla', -1: 'Pa
         print( int_keys_dicc )
```

```
{1: 'Mustang', 2: 'Firebird', 4: 'Camaro', -1: 'Patrol', 100: 'Corolla'}
```

Podem també crear diccionaris buits:

```
In [57]: empty_dicc = {}
         empty_plus = dict()
         print(empty_dicc)
         print(empty_plus)
```

```
{}
{}
```

2.9.4.2 Operacions bàsiques

Tot seguit mostrem les operacions bàsiques comunes entre llistes i diccionaris:

Operació	Descripció
len(<dictionary>)	Nombre d'elements que conté el diccionari
<key> in <dictionary>	Indica si la clau és al diccionari

2.9.4.2.1 Nombre d'elements o mida d'un diccionari (`len`)

```
In [58]: print( "El nombre d'elements de {} és {}".format(inventory_dicc, len(inventory_dic
         print( "El nombre d'elements de\n\t{}\n\tés {}".format(basket_dicc, len(basket_dic
```

```
El nombre d'elements de {'apples': 8, 'cookies': 10, 'eggs': 5, 'ham': 45} és 4
El nombre d'elements de
        {'Jordan': {'name': 'Michael', 'age': 40}, 'Bird': {'name': 'Larry', 'age': 55}}
        és 2
```

2.9.4.2.2 in

Retorna *True* o *False* en funció de si l'element especificat es troba o no contingut en el diccionari:

```
In [59]: print( "Diccionari:\n\t", basket_dicc )
         print( "\nLa clau 'Bird' existeix?", 'Bird' in (basket_dicc) )
```

```
Diccionari:
```

```
        {'Jordan': {'name': 'Michael', 'age': 40}, 'Bird': {'name': 'Larry', 'age': 55}}

La clau 'Bird' existeix? True
```

Sempre podem comprovar si una clau existeix o no en un diccionari, i en funció del resultat fer una acció o una altra:

```
In [60]: inventory_dicc = {'eggs':1, 'ham':2, 'bacon':5}

         if 'eggs' not in inventory_dicc:
             print('No existeix la clau')
         else:
             print('El valor associat a "eggs" és ', inventory_dicc['eggs'])

Els valor associat a "eggs" és  1
```

2.9.4.3 Accedint al contingut d'un diccionari

Com ja hem comentat, s'accedeix a un valor mitjançant la seva clau i l'operador claudàtor []:

```
In [61]: inventory_dicc = {'eggs':1, 'ham':2, 'bacon':5}
         person_dicc = {'Ricky':[44, 82.5], 'John':[22,70.6]}
         basket_dicc = {'Shaquile':{'age': 35, 'name': 'ONeal'}, 'Jordan':{'age': 40, 'name


         # Diccionari el camp valor del qual és de tipus simple
         print("El valor de la clau 'ham' és: ", inventory_dicc['ham'])
         print('---------------\n')

         # Diccionari el camp valor del qual és una llista
         print("El valor de la clau 'Ricky' és: ", person_dicc['Ricky'] )
         print("El valor de la clau 'Ricky'/segon valor de la llista és: ", person_dicc['Ri
         print('---------------\n')

         # Diccionari el camp valor del qual és un altre diccionari
         print("El valor de la clau 'Jordan' és: ", basket_dicc['Jordan'] )
         print("El valor de la clau 'Jordan'/'name' és: ", basket_dicc['Jordan']['name'] )
         print('---------------\n')

El valor de la clau 'ham' és:  2
---------------

El valor de la clau 'Ricky' és:  [44, 82.5]
El valor de la clau 'Ricky'/segon valor de la llista és:  82.5
---------------

El valor de la clau 'Jordan' és:  {'name': 'Michael', 'age': 40}
El valor de la clau 'Jordan'/'name' és:  Michael
---------------
```

2.9.4.4 Afegir i eliminar elements d'un diccionari

Es poden afegir nous elements al diccionari senzillament amb una assignació:

```
In [62]: # Inserint valors simples
         inventory_dicc = {'cookies': 10, 'ham': 45, 'eggs': 5, 'apples': 8}
         inventory_dicc['bananas'] = 16
         print( inventory_dicc )

         # Inserint valors estructurats (diccionaris)
         soccer_dicc = {'van Basten': {'name': 'Marco', 'age': 50}}
         soccer_dicc['Gullit'] = {'name': 'Ruud', 'age': 49}
         print( soccer_dicc )
```

```
{'apples': 8, 'bananas': 16, 'cookies': 10, 'eggs': 5, 'ham': 45}
{'van Basten': {'name': 'Marco', 'age': 50}, 'Gullit': {'name': 'Ruud', 'age': 49}}
```

I es poden esborrar amb la sentència **del**:

```
In [63]: del inventory_dicc['eggs']
         print( inventory_dicc )
```

```
{'apples': 8, 'bananas': 16, 'cookies': 10, 'ham': 45}
```

Els diccionaris també disposen de mètodes addicionals de manipulació, alguns de similars als que hem vist prèviament en l'apartat de llistes:

Mètode	Descripció
`get(clau)`	Retorna el valor associat a la clau donada
`update(clau_valor_parelles)`	Afegeix les parelles corresponents al diccionari
`pop(clau)`	Elimina i retorna el valor associat a la clau
`popitem()`	Esborra/retorna una parella (clau, valor) arbitrària
`clear()`	Esborra tots els elements

Tots aquests mètodes cal aplicar-los sobre un diccionari determinat: **<nom_diccionari>.mètode()**

```
In [64]: # Mostrem el contingut del diccionari
         inventory_dicc = {'cookies': 10, 'ham': 45, 'eggs': 5, 'apples': 8}
         print('Valors del dicc.: ', inventory_dicc )

         # Valor associat a una clau existent
         print("El valor de 'apples' és {}".format(inventory_dicc.get('apples')) )

         # Valor associat a una clau inexistent
         print("El valor de 'lemon' és {}".format(inventory_dicc.get('lemon')) )

         # Traiem la parella determinada per la clau 'apples'
```

```
        ap_val = inventory_dicc.pop('apples')
        print("El valor de 'apples' és {} i el cont. del dicc. és {}\n".format(ap_val, inve

        # Hi afegim noves parelles
        inventory_dicc.update( {'strawberry': 90, 'artichoke': 7} )
        print('Valors del dicc.:', inventory_dicc )

        # Traiem qualsevol parella
        rand_val = inventory_dicc.popitem()
        text = "El valor de la parella és {}\n\t i el cont. del dicc. és {}"
        print(text.format(rand_val, inventory_dicc))
        inventory_dicc.clear()
        print("Contingut després d'un clear: {}".format(inventory_dicc))
```

```
Valors del dicc.:  {'apples': 8, 'cookies': 10, 'eggs': 5, 'ham': 45}
El valor de 'apples' és 8
El valor de 'lemon' és None
El valor de 'apples' és 8 i el cont. del dicc. és {'cookies': 10, 'eggs': 5, 'ham': 45}

Valors del dicc.: {'strawberry': 90, 'cookies': 10, 'artichoke': 7, 'eggs': 5, 'ham': 45}
El valor de la parella és ('strawberry', 90)
        i el cont. del dicc. és {'cookies': 10, 'artichoke': 7, 'eggs': 5, 'ham': 45}
Contingut després d'un clear: {}
```

2.9.4.5 Modificar elements d'un diccionari

En tractar-se d'un tipus de dada mutable podem modificar-ne el contingut directament amb l'operador []:

```
In [65]: inventory_dicc = {'cookies': 10, 'ham': 45, 'eggs': 5, 'apples': 8}
         inventory_dicc['ham'] = 220
         print( inventory_dicc )

         basket_dicc['Bird'] = {'name': 'Larry', 'age': 66}
         print( basket_dicc )

         inventory_dicc['ham'] += 220
         print( inventory_dicc )
```

```
{'apples': 8, 'cookies': 10, 'eggs': 5, 'ham': 220}
{'Shaquile': {'name': 'ONeal', 'age': 35}, 'Jordan': {'name': 'Michael', 'age': 40}, 'Bird'
{'apples': 8, 'cookies': 10, 'eggs': 5, 'ham': 440}
```

2.9.4.6 Separació de claus i valors

En cas que sigui necessari, les claus i els valors d'un diccionari es poden extreure com a llistes:

```
In [66]: # Obtenim les claus i valors separadament com a llistes
         print ( list(inventory_dicc.keys()) )
         print ( list(inventory_dicc.values()) )
```

```
['apples', 'cookies', 'eggs', 'ham']
```

```
[8, 10, 5, 440]
```

Exercici 2.9.16 *Donat el diccionari: x = {1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f'}, cerca fent els prints corresponents:*

a) El primer element.

b) El tercer element.

c) Canvia el tercer element per bcn.

Comenta els resultats.

Exercici 2.9.17 *Els diccionaris poden tenir components tipus llista. Donat el diccionari: x = {1:[3, 'a'], 11: '4', 2: 'r', 22: [2, ['#', '%'],7]}, efectua:*

a) `print(x[22])`

b) `print(x[22][0])`

c) `print(x[22][1][0])`

d) substitueix l'element 22 per la paraula Barcelona

Comenta els resultats.

Exercici 2.9.18 *Donat el diccionari x = {1:[3, 'a'], 11: '4', 2: 'r', 22: [2, ['#', '%'],7]}:*

a) Elimina l'element 22.

b) Fes un llistat de les claus.

c) Comprova si en el diccionari hi ha la clau 11.

d) Comprova si hi ha la clau 499.

Fes els prints corresponents i comenta els resultats.

Exercici 2.9.19 *Donat el diccionari x = {1:[3, 'a'], 11: '4', 2: 'r', 22: [2, ['#', '%'],7]}: a/ afegeix-hi l'element m amb clau 3 i l'element ['e',33] amb clau 33 i b/ recorre, i mostra amb un print, tots els elements del diccionari donant la clau i l'element corresponent.*

2.9.5 `Set` (conjunt)

És similar a una llista, però només pot emmagatzemar elements únics (no hi pot haver repeticions). Els elements no tenen cap ordenació associada.

```
In [67]: animals_set = {'lion', 'elephant', 'tiger', 'lion'}

         print(animals_set)
{'elephant', 'tiger', 'lion'}
```

Els conjunts s'utilitzen de forma similar als altres conjunts de dades que hem estudiat en aquest tema.

2.9.5.1 Operacions amb conjunts

`add(element)`

Afegeix un element, que ha de ser immutable, al conjunt.

```
In [68]: colours = {"red","green"}
         colours.add("yellow")
         colours
```

```
Out[68]: {'green', 'red', 'yellow'}
```

clear()

Esborra tots els elements d'un conjunt.

```
In [69]: colours.clear()
         colours
```

```
Out[69]: set()
```

copy()

Crea un còpia completa del conjunt i la retorna:

```
In [70]: cities = {"Stuttgart", "Konstanz", "Freiburg"}
         cities_backup = cities.copy()
         cities.clear()
         cities_backup
```

```
Out[70]: {'Freiburg', 'Konstanz', 'Stuttgart'}
```

Si la mateixa operació es fa només amb una assignació, les dades es perden.

```
In [71]: cities = {"Stuttgart", "Konstanz", "Freiburg"}
         cities_backup = cities  # Just an assigment
         cities.clear()
         cities_backup
```

```
Out[71]: set()
```

difference(set)

Retorna la diferència entre dos conjunts o més:

```
In [72]: x = {"a","b","c","d","e"}
         y = {"b","c"}
         z = {"c","d"}
         x.difference(y)
```

```
Out[72]: {'a', 'd', 'e'}
```

```
In [73]: x.difference(y).difference(z)
```

```
Out[73]: {'a', 'e'}
```

També es pot usar l'operador menys (-):

```
In [74]: x - y
```

```
Out[74]: {'a', 'd', 'e'}
```

```
In [75]: x - y - z
```

```
Out[75]: {'a', 'e'}
```

difference_update(set)

Esborra tots els elements d'un altre conjunt del conjunt inicial:

```
In [76]: x = {"a","b","c","d","e"}
         y = {"b","c"}
         x.difference_update(y)
         x

Out[76]: {'a', 'd', 'e'}
```

És la mateixa operació que amb els operadors "x = x - y":

```
In [77]: x = {"a","b","c","d","e"}
         y = {"b","c"}
         x = x - y
         x

Out[77]: {'a', 'd', 'e'}
```

discard(element)

Esborra un element del conjunt. Si l'element no pertany al conjunt no fa res.

```
In [78]: x = {"a","b","c","d","e"}
         x.discard("a")
         x

Out[78]: {'b', 'c', 'd', 'e'}
```

remove(element)

Fa el mateix que `discard`, però en aquest cas es llença un *KeyError* si l'element no hi és present:

```
In [79]: x = {"a","b","c","d","e"}
         x.remove("a")
         x

Out[79]: {'b', 'c', 'd', 'e'}

In [80]: x.remove("z")
         x

         ---------------------------------------------------------------------------

         KeyError                                  Traceback (most recent call last)

         <ipython-input-80-0bde869d614a> in <module>()
         ----> 1 x.remove("z")
               2 x


         KeyError: 'z'
```

intersection(set)

Retorna la intersecció de dos conjunts en un de nou, si els dos conjunts tenen elements en comú.

```
In [81]: x = {"a","b","c","d","e"}
         y = {"c","d","e","f","g"}
         x.intersection(y)
```

```
Out[81]: {'c', 'd', 'e'}
```

L'operador (&) també es pot fer servir:

```
In [82]: x = {"a","b","c","d","e"}
         y = {"c","d","e","f","g"}
         x & y
```

```
Out[82]: {'c', 'd', 'e'}
```

pop()

Es treu un element arbitrari del conjunt. Si el conjunt és buit, es llença un error *KeyError*.

```
In [83]: x = {"a","b","c","d","e"}
         x.pop()
```

```
Out[83]: 'c'
```

isdisjoint(set)

Retorna cert quan la intersecció entre dos conjunts és nul · la:

```
In [84]: x = {"a","b","c","d","e"}
         y = {"c","d","e","f","g"}
         x & y
```

```
Out[84]: {'c', 'd', 'e'}
```

```
In [85]: x.isdisjoint(y)
```

```
Out[85]: False
```

```
In [86]: z = x - y
         z.isdisjoint(y)
```

```
Out[86]: True
```

issubset(set) and **issuperset(set)**

Comproven si els conjunts són un subconjunt (*subset*) o un superconjunt (*superset*) d'un altre. També es poden fer servir els operadors relacionals:

```
In [87]: x = {"a","b","c","d","e"}
         y = {"c","d"}
         print("x subset of y: ", x.issubset(y))
         print("y subset of x: ", y.issubset(x))
         print("x > y: ", x > y)
         print("x >= y: ", x >= y)
         print("x > x: ", x > x)
         print("x >= x: ", x >= x)
         print("-----------------")
         print("x superset of y: ", x.issuperset(y))
         print("y superset of x: ", y.issuperset(x))
         print("x < y: ", x < y)
         print("x <= y: ", x <= y)
         print("x < x: ", x < x)
         print("x <= x: ", x <= x)
```

```
x subset of y:  False
y subset of x:  True
```

```
x > y:  True
x >= y:  True
x > x:  False
x >= x:  True
---------------
x superset of y:  True
y superset of x:  False
x < y:  False
x <= y:  False
x < x:  False
x <= x:  True
```

2.9.6 *Python Collections*

Com a resum del tema, tot seguit es mostra en forma de taula les característiques principals dels diferents tipus de dades estructurades que suporta Python. S'hi ha inclòs també el tipus **string** ja vist en temes anteriors:

Name	Type	Empty	Iterable	Mutable	Indexed	Unique	Homogeneous	Named
string	`str`	”	Yes	No	Yes	No	Yes	No
list	`list`	[]	Yes	Yes	Yes	No	No	No
tuple	`tuple`	()	Yes	No	Yes	No	No	No/Yes
dictionary	`dict`	{}	*Yes*	Yes	No	No	No	Yes
set	`set`	set()	Yes	Yes	No	Yes	No	No

Tot lector que vulgui aprofundir en algun dels tipus anteriors, pot adreçar-se a la documentació oficial de Python 3.4.

Exercici 2.9.20 *Recordatori: per definició, les components d'un conjunt no tenen per què estar ordenades ni contenir elements repetits ni escriure's entre claus, però també es poden definir per la instrucció* `set(c)` *a on c és qualsevol element que es pot indexar ja siguin tuples, llistes o cadenes de caràcters.*

a) Donat el conjunt u = {3, 3, 4, 7, 9, 5} efectua `print(u)`*.*

b) Donat x = set([4,5, 3,5]), fes `print(x)`*.*

Comenta els resultats.

Exercici 2.9.21 *Donats els conjunts: a = {1,2,3,4,5,6}, b = {2,4,6,8,10,12}, calcula:*

a) a intersecció b

b) a unió b

c) les diferències a-b i b-a

d) la diferència simètrica de a i b

Comenta els resultats.

Exercici 2.9.22 *Donat el conjunt x={1,2,3,4,5,6,7,8,9}:*

a) Converteix-lo en llista.

b) Afegeix al conjunt x l'element 10.

c) Comprova si l'element 6 pertany a x, i el mateix per als elements 10 i 300.

Comenta els resultats.

Exercici 2.9.23 *Sigui una cadena de caràcters: 'exemple':*

a) Converteix-la en conjunt.

b) Converteix-la en llista.

Comenta els resultats.

Exercici 2.9.24 *Donat el conjunt x = {0,1,2,3,4,5,6,7,8,9,}:*

a) Esborra l'element 7.

b) Afegeix-hi els elements 10 i 11.

c) Calcula el nombre d'elements.

d) Esborra tots els elements.

Fes els prints corresponents i comenta els resultats.

2.10 Control de flux (II). La sentència `for`

Les sentències **`for`** permeten executar repetidament un bloc de sentències un nombre de vegades controlat pel contingut d'una llista. La sintaxi bàsica és:

```
for variable_de_for in llista:
    sentència 1
    sentència 2
    ...
```

Les sentències del bloc de codi associat s'executen una vegada per a cada valor en la llista, i en cada execució la variable associada al bucle `variable_de_for` pren successivament cadascun dels valors de la llista.

Exemple bàsic de sentència *for*:

```
In [1]: # Definim una llista, en aquest cas amb valors enters
        llista = [5,2,3,4,9,6]

        # Fem un bucle for que recorri la llista; en cada iteració
        # la variable "valor" pren consecutivament cadascun dels
        # valors de la llista
        for valor in llista:
            print("Valor: " + str(valor))
```

```
Valor: 5
Valor: 2
Valor: 3
Valor: 4
Valor: 9
Valor: 6
```

```
In [2]: # La llista pot ser de qualsevol tipus, no només numèrica
        llista = ["gat","gos","ratolí"]
        for valor in llista:
            print ("Valor: " + valor)
```

```
Valor: gat
Valor: gos
Valor: ratolí
```

2.10.1 La funció `range()`

Molt freqüentment es necessita usar un bucle **`for`** per iterar sobre una seqüència numèrica regular, per exemple:

```
0,1,2,3,4,5,6,7,8,9
0,2,4,6,8,10
```

```
In [3]: # La usem en un for
        for valor in range(0,10):
            print(valor, valor**2, valor**3)
```

```
0 0 0
1 1 1
```

```
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
```

Tanmateix, podem crear una llista i fer iteracions amb la llista:

```
In [4]: lista = list(range(10))
        print(lista)
        print(range(10))
        # Exemple amb range()
        for valor in lista:
            print(valor, valor**2, valor**3)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
range(0, 10)
0 0 0
1 1 1
2 4 8
3 9 27
4 16 64
5 25 125
6 36 216
7 49 343
8 64 512
9 81 729
```

2.10.2 Índex i valors: la funció `enumerate()`

Usant la funció **`enumerate()`** es pot aconseguir que un bucle **`for`** ens doni simultàniament el valor de cada element d'una llista i l'índex corresponent. Això és útil per recórrer llistes controlant al mateix temps l'índex en curs.

```
In [5]: llista = ["gat","gos","ratolí"]

        # Usem enumerate per fer un bucle for dual
        for index, valor in enumerate(llista):
            print(index,valor)

        # Usem enumerate per a fer un bucle for dual
        for enum in enumerate(llista):
            print(enum)
```

```
0 gat
1 gos
2 ratolí
(0, 'gat')
(1, 'gos')
```

```
(2, 'ratolí')
```

```
In [6]: tupla = (1, 3, 18)
        a, b, c = tupla
        c
```

```
Out[6]: 18
```

Això es pot aconseguir de forma alternativa usant el bucle per recórrer el rang d'índexs:

```
In [7]: llista = ["gat","gos","ratolí"]

        # Usem el for per recórrer tots els índexs
        for index in range(len(llista)):
            print(index,llista[index])
```

```
0 gat
1 gos
2 ratolí
```

Exercici 2.10.1 *Donada la llista [1, 2, 3, 5, 3, 7], eleva els seus elements a la quinta potència, expressa els resultats indicant l'element i el resultat corresponent.*

Exercici 2.10.2 *Suma els múltiples de 3 compresos entre 2 i 1000.*

Exercici 2.10.3 *Introdueix per teclat un nombre enter n. Cerca aleatòriament un valor x pertanyent al rang (1,n), eleva el seu valor al quadrat i fes que s'aturi el càlcul quan aquest valor sigui superior al quàdruple de n.*

Exercici 2.10.4 *Donada la llista: x = ['a', 4, 'am', 1, $,],*

a) Indexa'n els valors.

b) Converteix-la en conjunt.

c) Calcula la longitud de la llista i la del conjunt resultant.

2.10.3 Recorrent múltiples llistes: `zip()`

A vegades és necessari recórrer dues llistes simultàniament. Això es pot aconseguir amb un bucle **`for`** que recorri l'índex de la llista més curta o bé amb la funció **`zip()`**.

```
In [8]: llista1 = ["A","B","C"]
        llista2 = ["1","2","3","4","5","6"]

        # Usem zip() per recórrer diverses llistes a la vegada. La funció
        # zip() retorna tuples de valors de les llistes i acaba quan una
        # d'elles no té més elements
        for valor1, valor2 in zip(llista1,llista2):
            print(valor1, valor2)

        print()
        # Podem fer-ho també recorrent els índexs de la llista més curta
        for index in range(len(llista1)):
            print(llista1[index], llista2[index])
```

```
A 1
B 2
C 3

A 1
B 2
C 3
```

2.10.4 Sentència for amb diccionaris, tuples i cadenes

Les sentències `for` també funcionen amb diccionaris, tuples i amb una cadena. En aquest últim cas, s'iteren els caràcters de la cadena.

```
In [9]: # Exemple amb diccionari
        preu = {'cafè':1, 'dònut':2, 'refresc':1.5, 'entrepà':3}
        # element recorre els valors de les claus del diccionari
        for element in preu:
            print("Element de diccionari: " + element + " " + str(preu[element]))

        # Exemple amb cadena
        text= "abcdef"
        for caracter in text:
            print("Caràcter: " + caracter)

        print(preu.keys())
        print(preu.values())
        print(preu.items())
```

```
Element de diccionari: cafè 1
Element de diccionari: dònut 2
Element de diccionari: refresc 1.5
Element de diccionari: entrepà 3
Caràcter: a
Caràcter: b
Caràcter: c
Caràcter: d
Caràcter: e
Caràcter: f
dict_keys(['cafè', 'dònut', 'refresc', 'entrepà'])
dict_values([1, 2, 1.5, 3])
dict_items([('cafè', 1), ('dònut', 2), ('refresc', 1.5), ('entrepà', 3)])
```

Nota que un diccionari no té un ordre definit. Quan utilitzem un diccionari en un bucle **`for`**, aquest itera sobre totes les **`keys`**.

També podem iterar sobre els valors:

```
In [10]: #  Definim el diccionari
         preus = {'cafè':1, 'dònut':2, 'xocolata':1.5, 'entrepà':3}

         for value in preus.values():
             print("El preu és: {}".format(value))
```

```
El preu és: 1
El preu és: 2
El preu és: 1.5
El preu és: 3
```

O podem iterar explícitament sobre els parells (`key`, `value`) que formen el diccionari:

```
In [11]: #  Definim el diccionari
         preus = {'cafè':1, 'dònut':2, 'xocolata':1.5, 'entrepà':3}

         for clau, value in preus.items():
             print("El preu del {} és: {}".format(clau, value))
```

```
El preu del cafè és: 1
El preu del dònut és: 2
El preu del xocolata és: 1.5
El preu del entrepà és: 3
```

Nota important: quan operem sobre seqüències mutables com llistes o diccionaris que poden ser modificades dintre d'un *loop*, es recomana fer-ne primer una còpia, ja que una modificació de la llista dintre del bucle implica una modificació de la llista inicial.

```
In [12]: #  Tenim la llista...
         animals = ['gat', 'gos', 'gos']

         for animal in animals:
             if animal =='gos':
                 animals.remove(animal)
             print('animal de la llista: ', animal)
         print (animals)
```

```
animal de la llista:  gat
animal de la llista:  gos
['gat', 'gos']
```

En el cas de les llistes utilitzades en bucles **`for`** es pot fer servir *slicing*. Fent servir aquesta notació, `list[:]` genera una nova llista amb els ítems seleccionats, tots en aquest cas concret.

```
In [13]: #  Tenim la llista...
         animals = ['gat', 'gos', 'gos']
         print(animals)
         for animal in animals[:]:
             if animal =='gos':
                 animals.remove(animal)
             print('animal de la llista: ', animal)
         print (animals)
```

```
['gat', 'gos', 'gos']
animal de la llista:  gat
animal de la llista:  gos
animal de la llista:  gos
['gat']
```

En el cas dels diccionaris, les funcions `nom.keys()`, `nom.values()` o `nom.items()` proporcionen objectes que ens permeten iterar sobre les `keys`, `values` o tuples del tipus (`key,value`).

```
In [14]: diccionari = {'cafè':1, 'dònuts':4, 'xocolatines':3}
         llista_claus = diccionari.keys()
         llista_valors = diccionari.values()
         tupla_diccionari = diccionari.items()
         print(llista_claus, type(llista_claus))
         print(llista_valors, type(llista_valors))
         print(tupla_diccionari, type(tupla_diccionari))
```

```
dict_keys(['cafè', 'dònuts', 'xocolatines']) <class 'dict_keys'>
dict_values([1, 4, 3]) <class 'dict_values'>
dict_items([('cafè', 1), ('dònuts', 4), ('xocolatines', 3)]) <class 'dict_items'>
```

Exercici 2.10.5 *Donades les llistes a = [1, 2, 5, 4, 3, 6] i b = ['m','n','ab','a1','d','t'], forma la llista de parelles entre els seus elements.*

Exercici 2.10.6 *Donades les llistes a = [1,2,3,4,5,6], b = ['m','n','ab','a1','d','t'] i c = [7,8,9,10,11,12], forma la llista de ternes entre els seus elements.*

Exercici 2.10.7 *Donades les llistes a = [1, 2, 5, 4, 3, 6] i b = ['m','n','ab','a1','d','t'], construeix una taula, mitjançant un sol print, en la qual la primera columna siguin els elements de a i la segona els de b. Primer forma parelles amb un* ***zip****.*

Exercici 2.10.8 *Donats el diccionari x = {'a': 1, 'b': 3, 'c': 5, 'd': 7, 'e': 9} i la cadena y = "Catalunya":*

a) Calcula les longituds de x i de y.

b) Explicita, utilitzant la sentència ***for****, cada clau i l'element corresponent del diccionari x.*

c) Cada un dels caràcters de y.

2.10.5 Seqüències de valors `float`

Per generar seqüències de valors **`float`** caldrà usar la biblioteca **`NumPy`** que estudiarem més endavant. En particular la funció

```
numpy.arange([start], stop[, step], dtype=None)
```

Exemple:

```
In [15]: # Usem arange per generar valors float entre 0 i 1 amb un pas de 0.1
         import numpy
         for valor in numpy.arange(0.,1.,0.1):
             print(valor)
```

```
0.0
0.1
0.2
0.30000000000000004
0.4
0.5
0.6000000000000001
```

```
0.7000000000000001
0.8
0.9
```

2.10.6 `break`: interrupció de l'execució d'un bloc

La comanda **`break`** permet interrompre l'execució d'un bloc **`for`**. Quan s'executa aquesta comanda l'execució salta el que quedi del bloc `for` i segueix amb el programa.

Exemple:

```
In [16]: valors = {"a": [1, 4, 3],
                   "b": [2, 7, 8],
                   "c": [9, 5, 4]}

         for lletra in valors:
             llista = valors[lletra]
             print(llista)
             print(lletra)
             for enter in llista:
                 print(enter)
```

```
[1, 4, 3]
a
1
4
3
[2, 7, 8]
b
2
7
8
[9, 5, 4]
c
9
5
4
```

```
In [17]: # Interrompem aquest bucle a la meitat amb la comanda break
         print("El programa arriba al bucle for")
         for valor in range(10):
             if valor < 5 :
                 print("Valor: " + str(valor))
             else:
                 print("Trobat un valor més gran que 5: interrompent")
                 break

         print("Seguim amb el programa")
```

```
El programa arriba al bucle for
Valor: 0
Valor: 1
```

```
Valor: 2
Valor: 3
Valor: 4
Trobat un valor més gran que 5: interrompent
Seguim amb el programa
```

2.10.7 break-else

La comanda **break** es pot combinar amb la sentència **else**. En un bucle **for** el bloc de codi associat a la sentència **else** s'executa al final del bucle si el bucle no s'ha interromput mitjançant un **break**.

Exemple:

```
In [18]: # Exemple 1: el bucle no s'interromp amb break
         llista = [1,1,2,3,3,4]
         for valor in llista:
             print(valor)
             if valor > 5 :
                 break
         else:
             print("No s'ha interromput el bucle for")

         # Exemple 2: el bucle s'interromp amb break
         llista = [1,1,6,3,3,4]
         for valor in llista:
             print(valor)
             if valor > 5 :
                 break
         else:
             print("No s'ha interromput el bucle for")
```

```
1
1
2
3
3
4
No s'ha interromput el bucle for
1
1
6
```

2.10.8 continue: saltant una iteració del bloc for

La sentència **continue** permet interrompre la iteració del bloc **for** en curs i saltar directament a la iteració següent.

Exemple:

```
In [19]: # En aquest exemple usem continue per saltar els valors parells del bucle
         for valor in range(10):
```

```
            if valor%2==0 :
                continue
            print("Saltant valors parells: "+str(valor))
```

```
Saltant valors parells: 1
Saltant valors parells: 3
Saltant valors parells: 5
Saltant valors parells: 7
Saltant valors parells: 9
```

Exemple: Escriurem el codi necessari per aconseguir que l'ordinador:

1. Imprimeixi per pantalla el quadrat de tots els enters entre -5 i +5.
2. Demani a l'usuari una cadena i la imprimeixi per pantalla, però cada lletra en majúscules i en una fila diferent.
3. Demani a l'usuari una cadena i la imprimeixi per pantalla, en ordre invers i en una fila diferent.
4. Demani a l'usuari 3 nombres enters i els guardi directament en una llista.

```
In [20]: for x in range(-5,6):
             print (x**2)
```

```
25
16
9
4
1
0
1
4
9
16
25
```

```
In [21]: cadena = input("cadena: ")
         for lletra in cadena:
             print (lletra.upper())

         print(cadena)
```

```
cadena: Bon dia
B
O
N

D
I
A
Bon dia
```

```
In [22]: cadena = input("cadena: ")
```

```
        for lletra in cadena[::-1]:
            print (lletra," ")

        print (cadena)
```

```
cadena: Bon dia
a
i
d

n
o
B
Bon dia
```

```
In [23]: llista_enters = []
         for i in range(3):
             llista_enters.append(int(input("Nombre enter? ")))
         print (llista_enters)
```

```
Nombre enter? 2
Nombre enter? 4
Nombre enter? 2
[2, 4, 2]
```

Exercici 2.10.9 *Defineix una funció val(a,b,c,n), que calculi els valors y d'una equació de segon grau per a x en el rang(0,n,0.3). En un sol* **`print`** *has de mostrar els coeficients i el valor de n, el valor de x i el corresponent de y. Comprova-ho per a val(1,-3,5,2).*

Exercici 2.10.10 *Donada la llista en el rang (7, 160), suma'n els elements ignorant els que són múltiples de cinc.*

Exercici 2.10.11 *Per a tot element a, en el rang (2,3,0.5), i b, en el rang (1,2,0.2), fes un petit programa que calculi p = a*b però que s'aturi quan p superi el valor de la suma a+b. La sortida ha de donar a, b, a+b i p quan s'hagi superat aquest valor.*

Exercici 2.10.12 *Defineix una funció s(n,m) que permeti definir un rang (0,n,m), n i m arbitraris, però amb m<1, i que calculi el quadrat dels seus elements sempre que el resultat sigui menor que 5. Has de mostrar solament l'element i el seu quadrat quan se superi aquest valor a partir del qual el programa s'ha d'interrompre.*

Exercici 2.10.13 *Donats els n primers nombres naturals, introduint n pel teclat, efectua el producte de cada nombre pel seu consecutiu. Si aquest producte supera n, el programa s'ha d'interrompre donant com a sortida n, el nombre i el producte; si no, ha de donar la diferència entre n i el producte corresponent. Comprova-ho per a n = 25.*

2.11 Comprensió de llistes

La comprensió de llistes consisteix a optimitzar la utilització de llistes i bucles **`for`** i clàusules **`if`**. El resultat dona una menor quantitat de línies de codi i una optimització en l'ús de la memòria. Consisteix a definir després del primer *bracket* l'expresió que volem assolir seguida de sentències **`for`** i **`if`** i, tot seguit, la llista sobre la qual operem. El resultat dona una nova llista. Vegem-ne un exemple:

```
In [24]: # A partir de la llista A definida com:

         A = list(range(10))

         #  Ens demanen una nova llista que faci el quadrat de cada valor de A

         quadrat_de_A = [x**2 for x in A]
         A, quadrat_de_A
```

```
Out[24]: ([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 4, 9, 16, 25, 36, 49, 64, 81])
```

```
In [25]: #  Altres exemples...
         [(x,y) for x in [1,2,3] for y in [3,1,4] if x!=y]
```

```
Out[25]: [(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Podem també operar amb llistes multidimensionals:

```
In [26]: matrix = [[1,2,3],
                   [4,5,6],
                   [9,8,7]]

         matriu_quadrada = [[dada**2 for dada in row] for row in matrix]

         print(matrix)
         print(matriu_quadrada)
```

```
[[1, 2, 3], [4, 5, 6], [9, 8, 7]]
[[1, 4, 9], [16, 25, 36], [81, 64, 49]]
```

Exemple: escriurem un programa que calculi el nombre π amb les fórmules següents:

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}$$

$$\prod_{n=1}^{\infty} \left(\frac{2n}{2n-1} \cdot \frac{2n}{2n+1} \right) = \frac{2}{1} \cdot \frac{2}{3} \cdot \frac{4}{3} \cdot \frac{4}{5} \cdot \frac{6}{5} \cdot \frac{6}{7} \cdots = \frac{\pi}{2}$$

```
In [27]: M = 25
         suma = 0.
         for n in range(M):
             numerador = (-1.)**n
             denominador = ((2.*n)+1)
             suma = suma + (numerador/denominador)
             print (suma)

         print("PI = {}".format(suma*4))

         # Amb comprensió de llistes
         pi = sum([(-1.)**n/((2.*n)+1) for n in range(M)])*4

         print( "\nPI = {}".format(pi))
```

```
1.0
0.6666666666666667
```

```
0.8666666666666667
0.7238095238095239
0.8349206349206351
0.7440115440115441
0.8209346209346211
0.7542679542679545
0.8130914836797192
0.7604599047323508
0.8080789523513985
0.7646006914818333
0.8046006914818333
0.7675636544447964
0.802046413065486
0.769788348549357
0.8000913788523872
0.7715199502809587
0.7985469773079856
0.77290595166696
0.797296195569399
0.7740403816159106
0.7962626038381329
0.774986008093452
0.7953941713587581
PI = 3.1815766854350325

PI = 3.1815766854350325
```

```
In [28]: M = 25
         prod = 1.
         for n in range(1,M):
             n1 = 2.*n
             d1 = (2.*n-1)
             n2 = 2.*n
             d2 = (2.*n+1)
             prod = prod * (n1/d1) * (n2/d2)
             print (prod)

         print("PI = {}".format(prod*2.))

         # Amb comprensió de llistes

         pi = 2.
         prod = [(2*n/(2.*n+1))*(2*n/(2.*n-1)) for n in range(1,M)]
         for n in prod:
             pi *= n

         print( "\nPI = {}".format(pi))
```

```
1.3333333333333333
1.4222222222222223
1.4628571428571429
```

```
1.4860770975056687
1.5010879772784531
1.51158509600068
1.519336814441709
1.5252949980277548
1.5300172735634443
1.5338519033217486
1.5370275801402202
1.5397006715839423
1.5419817096159185
1.5439510349155556
1.5456684442981092
1.5471793616434641
1.5485189108743243
1.5497146783730686
1.5507886317191344
1.5517584807696152
1.552638661416677
1.5534410586577192
1.5541755461559024
1.554850394417368
PI = 3.109700788834736

PI = 3.1097007888347363
```

Exercici 2.11.1 *Per a la comprensió de llistes, escriu un programa que calculi ln(2)=1-1/2+1/3-1/4+1/5-1/6 · · ·. Comprova el resultat amb la funció* `log1p`.

Exercici 2.11.2 *Per a la comprensió de llistes, escriu un programa que calculi la suma d'una progressió geomètrica de primer terme x = 3 i raó -1 < r <1, valor que cal introduir pel teclat. Comprova el resultat amb la fórmula S =x/(1-r).*

Exercici 2.11.3 *Demostra que la suma dels n primers nombres senars tendeix a n^2, quan n és al suficientment gran. Fes la comprovació corresponent.*

2.12 Fitxers

Moltes vegades és necessari emmagatzemnar i demanar dades per no perdre-les. Python permet guardar la informació en un fitxer i llegir-la quan sigui necessari.

2.12.1 Funcions màgiques al Notebook

IPython Notebook té un conjunt de *magic functions* que pots cridar des del Notebook **cell of code**. N'hi ha de dos tipus: *line-oriented* (%) i *cell-oriented* (%%).

La funció `%lsmagic` s'utilitza per llistar totes les funcions màgiques, i mostrarà els dos tipus correctament definits:

```
In [1]: %lsmagic

Out[1]: Available line magics:
        %alias  %alias_magic  %autoawait  %autocall  %automagic  %autosave  %bookmark  %cat

        Available cell magics:
        %%!  %%HTML  %%SVG  %%bash  %%capture  %%debug  %%file  %%html  %%javascript  %%js

        Automagic is ON, % prefix IS NOT needed for line magics.
```

Per exemple, podem utilitzar en Unix/Linux l'ordre `pwd` per mostrar el directori de treball actual:

```
In [2]: %pwd

Out[2]: '/Users/chema/Documents/OneDrive - Universitat de Barcelona/Clases/Informatica/Repos
```

A les properes seccions, utilitzarem més funcions màgiques, especialment les relacionades amb fitxers. Pots trobar més exemples aquí.

2.12.2 Treballant amb dades de text bàsiques

El primer pas és llegir o escriure dades desestructurades en un fitxer.

2.12.2.1 Llegint un fitxer de text simple

Primer creem un fitxer simple, utilitzant la funció màgica `cell-oriented %%file`:

```
In [3]: %%file test.txt
        This is a text file created to check
        Python file read and write.

Overwriting test.txt
```

Per llegir el fitxer, en primer lloc cal obrir-lo:

```
In [4]: inputFile = open('test.txt', 'r')

        print(inputFile)

<_io.TextIOWrapper name='test.txt' mode='r' encoding='UTF-8'>
```

Els paràmetres són:

- name of the file.
- read mode (`r`), we can also use binary mode (`b`) to avoid problems with text conversion.
- the optional universal line-end mode (`U`) to be able to interchange documents between operating systems.

A continuació llegim les línies del fitxer:

```
In [5]: file_in = inputFile.readlines()
        print(file_in)

        for line in file_in:
            print(line, end='')
```

```
['This is a text file created to check\n', 'Python file read and write.\n']
This is a text file created to check
Python file read and write.
```

A continuació el fitxer s'ha de tancar:

```
In [6]: inputFile.close()
```

2.12.2.2 Escrivint un fitxer de text simple

Ara seguint un procediment similar escriurem un fitxer desestructurat.

```
In [7]: outputFile = open('secondTest.txt', 'w')
```

```
In [8]: outputFile.write('Another file\n')
        outputFile.write('A second line\n')
```

```
Out[8]: 14
```

```
In [9]: outputFile.close()
```

```
In [10]: %less secondTest.txt
```

2.12.2.3 La sentència `with`

De vegades els fitxers no es poden escriure o llegir apropiadament i apareixen errors. I si apareix un error, el fitxer s'ha de tancar sempre. Per evitar problemes, Python té la sentència **`with`**.

```
In [11]: with open('thirdTest.txt', 'w') as outputFile:
             for i in range(10):
                 outputFile.write('New test. ' +
                                  str(i) + '\n')
```

```
In [12]: %less thirdTest.txt
```

```
In [13]: outputFile.closed
```

```
Out[13]: True
```

El fitxer s'ha escrit i tancat.

```
In [14]: with open('thirdTest.txt', 'r') as inputFile:
             print(inputFile.read(), end = '')
```

```
New test. 0
New test. 1
```

```
New test. 2
New test. 3
New test. 4
New test. 5
New test. 6
New test. 7
New test. 8
New test. 9
```

Exercici 2.12.1 *Donat el text: "Els nostres avantpassats van comprovar que la longitud de l'ombra era més curta a migdia, i que la longitud de l'ombra al migdia variava amb les estacions". a/ Escriu-lo en un fitxer de manera que ocupi quatre línies, i b/ Una vegada guardat obre'l i fes un* ***print*** *que en mostri el contingut.*

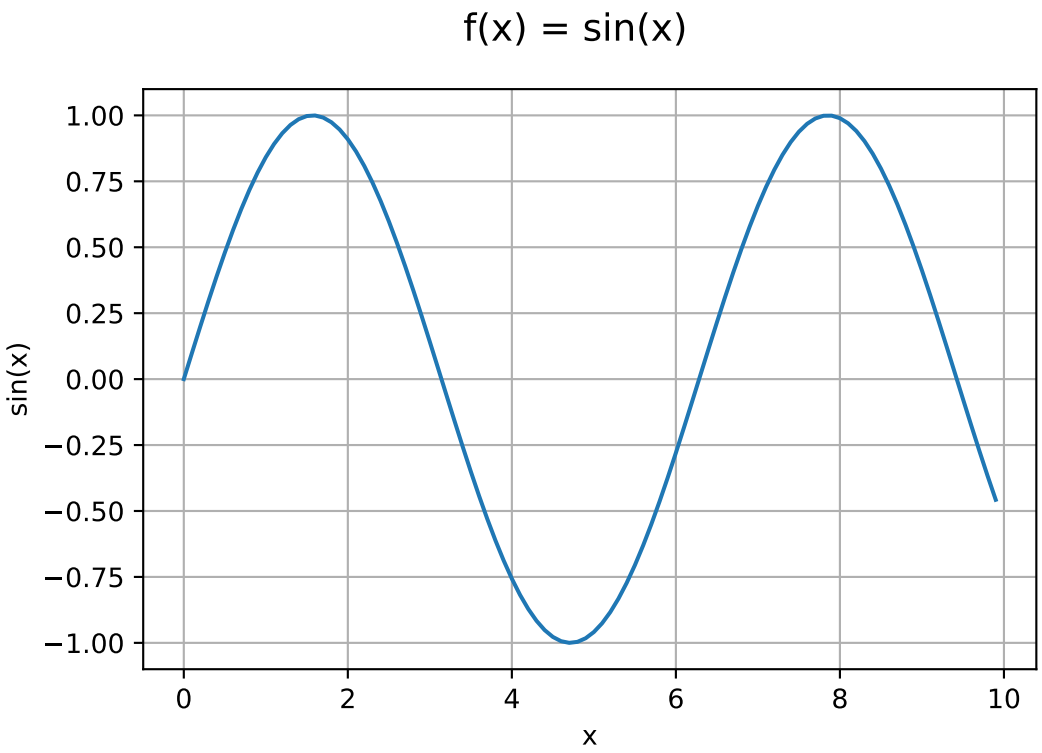
2.12.3 Treballant amb mòduls d'ajuda

Ara intentarem escriure dades. Per exemple, podem crear dues fileres de dades:

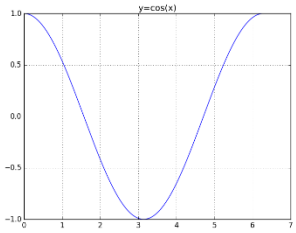
```
In [15]: import math

         x = [point/10. for point in range(0, 100)]
         y = [math.sin(x_point) for x_point in x]
```

```
In [16]: %pylab inline
         %config InlineBackend.figure_format = 'pdf'

         import matplotlib.pyplot as plt

         str_title = 'f(x) = sin(x)'
         fig1 = plt.figure()
         fig1.suptitle(str_title, fontsize=14)
         fig1_ax = fig1.add_subplot(1, 1, 1)
         fig1_ax.set_xlabel('x')
         fig1_ax.set_ylabel('sin(x)')
         fig1_ax.grid(True, which='both')

         fig1_ax.plot(x, y)
         plt.show()
```

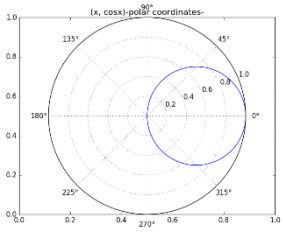
```
Populating the interactive namespace from numpy and matplotlib
```



Exercici 2.12.2 *Repeteix l'exercici anterior per a cos(x), per a un rang de x entre (0, 63) recorregut per tots els seus valors enters dividits entre 10.*



Exercici 2.12.3 *Repeteix l'exercici anterior expressant x,y en coordenades polars. Quina figura obtens? Per què?*



2.12.3.1 Escrivint un fitxer amb `json`

La manera més simple d'emmagatzemar dades des de Python és utilitzant el mòdul `json`. En aquest cas, només cal utilitzar la funció **`dump`** amb la variable que s'ha d'emmagatzemar com a primer argument i el fitxer com a segon:

```
In [17]: import json
```

```
        with open('sin.json', 'w') as outputFile:
            json.dump({'x': x, 'y': y}, outputFile, sort_keys=True)
```

Amb això, s'ha creat un fitxer que conté la informació de x a y.

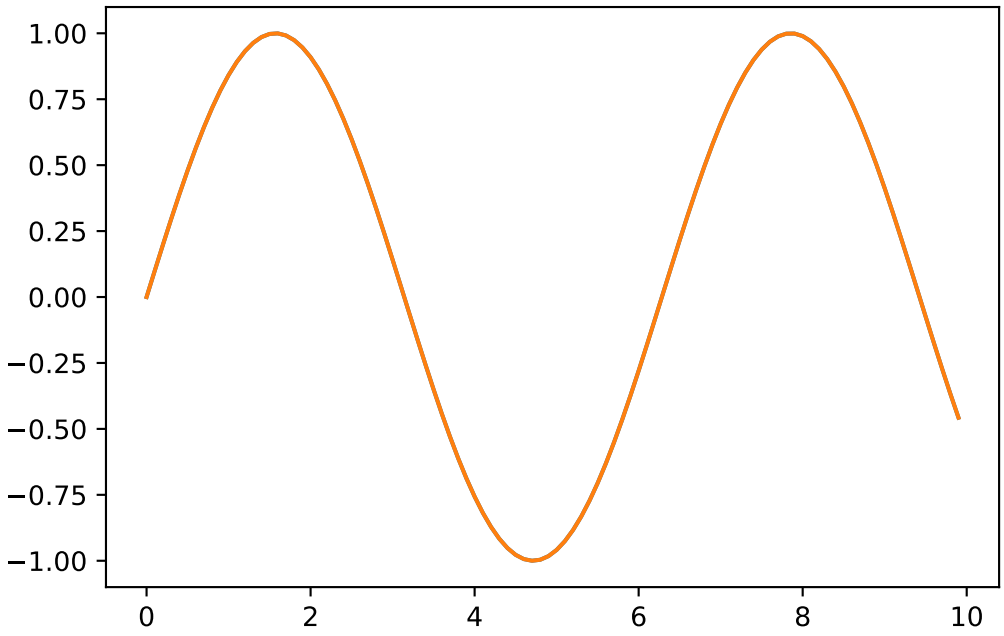
```
In [18]: %less sin.json
```

Però no és fàcil entendre el fitxer.

2.12.3.2 Llegint el fitxer `json`

Ara es pot demanar amb la funció **`load`**, la qual retorna els continguts de la variable. És important tenir en compte que l'ordre de llegir les dades ha de ser el mateix en què s'han emmagatzemat:

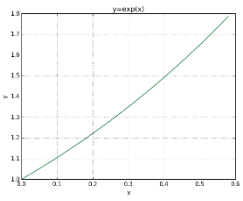
```
In [19]: with open('sin.json', 'r') as inputFile:
             data = json.load(inputFile)
             xf = data['x']
             yf = data['y']

In [20]: import matplotlib.pyplot as plt

         plt.plot(xf, yf, x, y)
         plt.show()
```



La dificultat més gran d'aquest format és que només Python l'entén. Per aquesta raó, si desitgem intercanviar les dades, serà necessari utilitzar un format estructurat.

Exercici 2.12.4 *Repeteix l'exercici anterior per a la funció $y = cos(x)$.*

Exercici 2.12.5 *Repeteix l'exercici per a la funció $y = exp(x)$, per als punts $a/50$, i valors enters de a en el rang (0,30).*



2.12.4 Dades estructurades

La manera bàsica serà treballant amb fitxers **txt**.

2.12.4.1 Treballant amb l'estructura txt

Primer cal definir el format. Per simplificar com llegir-lo, l'escriurem de forma similar a com ho faríem en una taula. Les columnes estaran separades per tabuladors:

```
In [21]: with open('sin.txt', 'w') as outputFile:
             for dataX, dataY in zip(x, y):
                 outputFile.write(str(dataX) +
                                  "\t" + str(dataY) +
                                  "\n")
```

Les dades s'emmagatzemaran entre línies, la x a la primera columna i la y a la segona, i separades per un tabulador.

```
In [22]: %less sin.txt
```

La dificultat més gran és llegir de nou les dades, ja que s'han de descodificar. Primer s'obre el fitxer i totes les línies de text són demanades:

```
In [23]: with open('sin.txt', 'r') as inputFile:
             lines = []

             # The lines are read from the file
             for line in inputFile.readlines():

                 # The line is stored in the list
                 lines.append(line)
```

Cada línia té dos valors, la x i la y, encara que barrejades amb la tabulació i la nova línia:

```
In [24]: lines[1]
```

```
Out[24]: '0.1\t0.09983341664682815\n'
```

Primer, és necessari separar els dos valors. Per això cal dos passos: en primer lloc, cal treure els innecessaris caràcters espais (**strip**):

```
In [25]: lineStripped = lines[0].strip()

         lineStripped
```

```
Out[25]: '0.0\t0.0'
```

El segon pas serà **split**, la línia en dues cel · les **\t**.

```
In [26]: lineSplitted = lineStripped.split('\t')

         lineSplitted
```

```
Out[26]: ['0.0', '0.0']
```

Finalment, serà necessari convertir les cadenes en valors flotants, utilitzant la funció **float**:

```
In [27]: float(lineSplitted[0])
```

```
Out[27]: 0.0
```

Amb aquest procés, les dades són demanades:

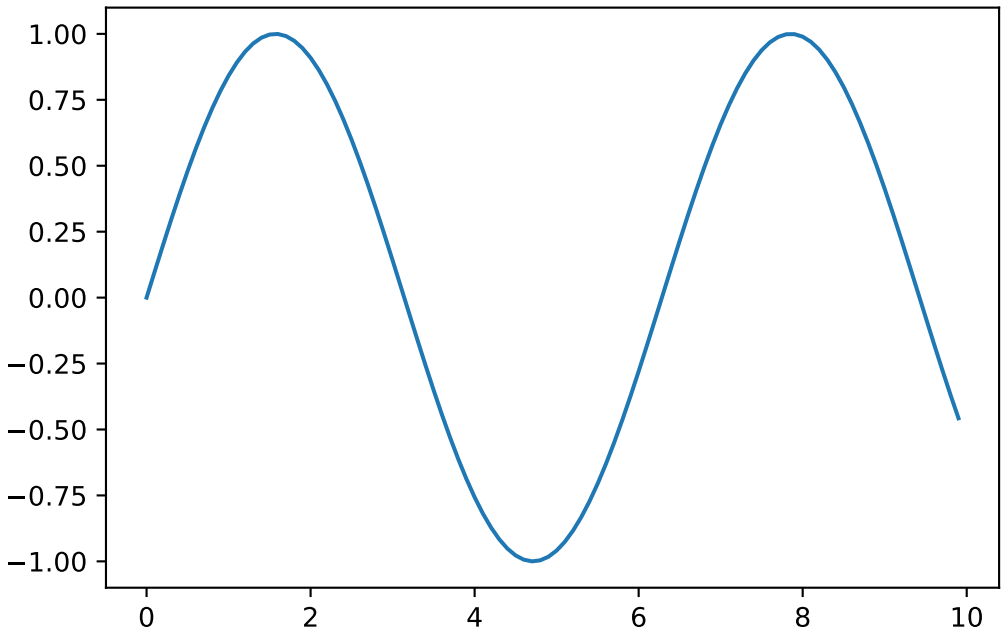
```
In [28]: xf = []
         yf = []

         for line in lines:
             # The line is stripped taking out all the unnecesary characters
             lineStripped = line.strip()

             # The line is splitted using the tab character
             lineSplitted = lineStripped.split('\t')

             xf.append(float(lineSplitted[0]))
             yf.append(float(lineSplitted[1]))
```

El procés total es pot agrupar:

```
In [29]: xf = []
         yf = []

         with open('sin.txt', 'r') as inputFile:
             for line in inputFile.readlines():
                 lineStripped = line.strip()
                 lineSplitted = lineStripped.split('\t')

                 xf.append(float(lineSplitted[0]))
                 yf.append(float(lineSplitted[1]))
```

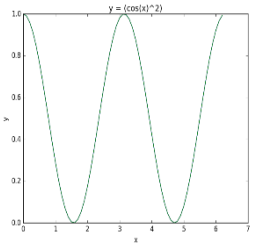
Ara les dades es poden mostrar:

```
In [30]: import matplotlib.pyplot as plt

         plt.plot(xf, yf)
         plt.show()
```

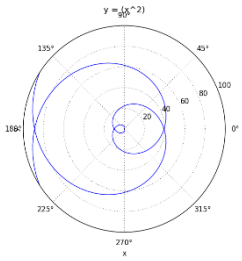


Això implica que per a cada fitxer, és necessari escriure el codi `strip` i `split`, i processar la informació. Python dona una alternativa.

Exercici 2.12.6 *Donada la funció $y = cos^2(x)$ efectua treballant amb l'estructura* `txt` *la seva representació gràfica, per als punts a/10, i valors enters de a en el rang(0, 63).*



Exercici 2.12.7 *Repeteix l'exercici anterior per a la funció $y = x^2$, representant-lo amb coordenades polars, per als punts a/10, i valors enters de a en el rang(-100, 101).*



2.12.4.2 Utilitzant el mòdul `csv`

Les mateixes operacions es poden seguir, però amb el mòdul `csv`, que fa fitxers compatibles amb Excel o Calc. En aquest cas, necessitem utilitzar l'ordinador local per estar segurs que els dos programes entenen correctament els continguts dels fitxers. Vegem-ne un exemple:

```
In [31]: import locale

         locale.setlocale(locale.LC_ALL, '')

         with open('sinCSV.csv', 'w') as outputFile:
             for dataX, dataY in zip(x, y):
                 outputFile.write('{:n};{:n}\n'.format(dataX, dataY))
```

```
In [32]: %less sinCSV.csv
```

El fitxer separa les columnes amb semicomes. Ara es pot llegir amb Excel o Calc.

Podem fer el mateix procés, però utilitzant el mòdul `csv`. En aquest cas, per estar segurs que tenim el format natiu desitjat, hem de modificar lleugerament el procediment:

```
In [33]: import locale
         import csv

         locale.setlocale(locale.LC_ALL, '')

         with open('sinCSV.csv', 'w') as outputFile:
             writer = csv.writer(outputFile, delimiter=';')
             dataX = [locale.str(value) for value in x]
             dataY = [locale.str(value) for value in y]
             writer.writerows(zip(dataX, dataY))
```

El resultat és equivalent. També podem llegir el fitxer:

```
In [34]: %less sinCSV.csv
```

Això és fàcil de fer amb el mòdul **csv**:

```
In [35]: import csv

         sinList = []

         with open('sinCSV.csv', 'r') as csvfile:
             sinReader = csv.reader(csvfile,
                                    delimiter=';',
                                    quotechar='"')
             for row in sinReader:
                 sinList.append(row)
```

```
In [36]: len(sinList[0])
```

```
Out[36]: 2
```

En aquest cas, les cel · les estan completament separades:

```
In [37]: type(sinList[0][0])
```

```
Out[37]: str
```

Ara els valors es poden convertir a flotants com en el cas previ. Per simplificar podem utilitzar:

```
In [38]: data = [[float(cell) for cell in row] for row in sinList]
```

```
ValueError                                Traceback (most recent call last)

<ipython-input-38-3f966bddeb0f> in <module>
----> 1 data = [[float(cell) for cell in row] for row in sinList]


<ipython-input-38-3f966bddeb0f> in <listcomp>(.0)
----> 1 data = [[float(cell) for cell in row] for row in sinList]


<ipython-input-38-3f966bddeb0f> in <listcomp>(.0)
----> 1 data = [[float(cell) for cell in row] for row in sinList]


ValueError: could not convert string to float: '0,1'
```

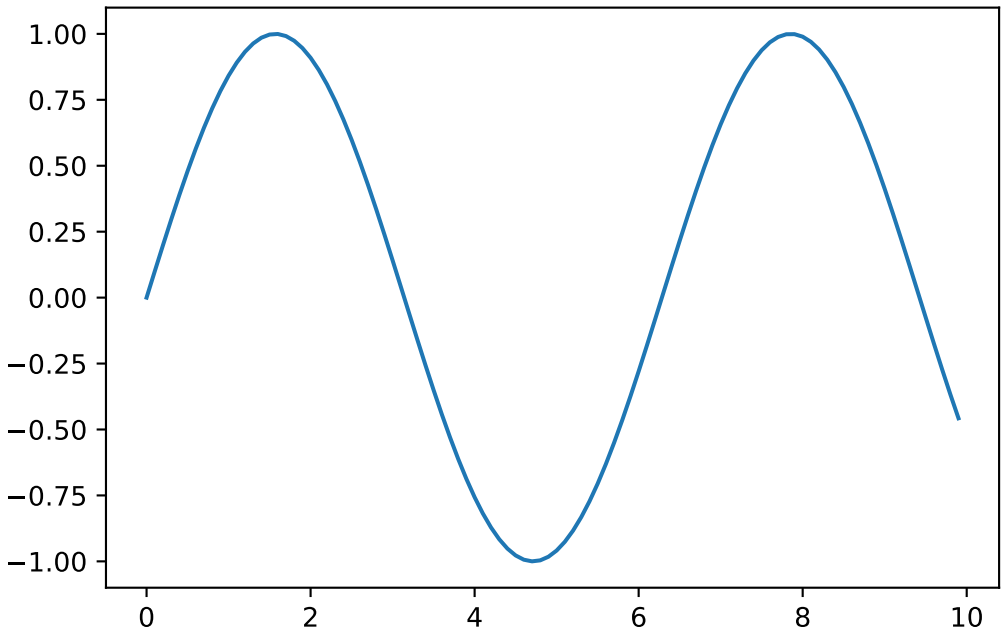
Apareix un problema nou: l'ús de la coma en lloc del punt no permet convertir la cadena a flotant. Això es pot solucionar utilitzant el mòdul **`locale`** altre cop. Hem convertit les cadenes a flotants utilitzant la funció **`atof function`** de `locale`:

```
In [39]: data = [[locale.atof(cell) for cell in row] for row in sinList]
```

Ara aquestes dades es poden transferir a dues llistes:

```
In [40]: xf = []
         yf = []

         with open('sinCSV.csv', 'r') as inputFile:
             sinReader = csv.reader(inputFile, delimiter=';', quotechar='"')
             for row in sinReader:
                 xf.append(locale.atof(row[0]))
                 yf.append(locale.atof(row[1]))
```

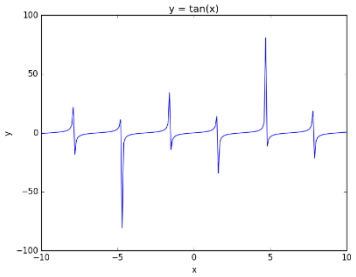
```
In [41]: %config InlineBackend.figure_format = 'pdf'

         import matplotlib.pyplot as plt

         plt.plot(xf, yf)
         plt.show()
```



Ara és possible llegir els fitxers escrits.

Exercici 2.12.8 *Sense utilitzar el mòdul* `csv`*, crea un fitxer de dades x, tan(x), per als punts a/10, i valors enters de a en el rang (-100, 101) i comprova'n l'estructura.*

Exercici 2.12.9 *Repeteix l'exercici anterior utilitzant el mòdul* `csv`*, i fes la representació gràfica de les dades.*



2.13 Biblioteca NumPy

NumPy és una biblioteca de Python per al càlcul científic que conté una gran varietat d'eines per a la **creació, manipulació i càlculs de vectors i matrius**. L'avantatge respecte de l'ús de llistes normals de Python per fer aquest tipus de càlculs és que les eines de NumPy són més senzilles, completes i **molt més ràpides**.

Pots veure una introducció a NumPy aquí: What is NumPy? i diversos exemples d'ús aquí: NumPy examples

Nota. Per usar la biblioteca NumPy cal fer la importació:
`import numpy`

2.13.1 `ndarray`

L'objecte bàsic de NumPy és `ndarray`. Aquest tipus d'objecte és una extensió de les llistes normals de Python amb adaptacions pensades per al càlcul numèric. En aquest sentit ja inclou conceptes com *dimensions* i *eixos*.

- `ndarray.ndim` dona la dimensió de la llista.
- `ndarray.shape` dona la "forma" de la llista (n1 x n2 x n3 x ...).
- `ndarray.size` nombre d'elements de la llista.

Exemple:

```
In [1]: import numpy as np

        # Generem un ndarray a partir de la funció de NumPy arange(),
        # similar a range() però que genera ndarrays
        # reshape() converteix la llista en una matriu 3 x 5
        un_array = np.arange(15).reshape(3, 5)
        print(un_array)

        # Comprovem la dimensió
        print("Dimensió: ",un_array.ndim)

        # Comprovem la forma
        print("Forma: ",un_array.shape)

        # Comprovem el nombre total d'elements
        print("Nombre d'elements:", un_array.size)
```

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
Dimensió:  2
Forma:  (3, 5)
Nombre d'elements: 15
```

2.13.2 Creació de `ndarrays`

NumPy ofereix diverses formes per crear objectes `ndarray` (ja hem vist en l'exemple anterior una forma simple de crear-ne un).

2.13.2.1 A partir d'una llista: `array()`

La manera més simple de crear un objecte `ndarray` és a partir d'una llista normal, utilitzant la funció `array()`. Aquesta funció convertirà una llista en un objecte `ndarray` de les mateixes dimensions.

```
In [2]: import numpy as np

        # Creem una llista 1D amb nou elements i la convertim en un ndarray 1D
        llista_1D = range(9)
        print(llista_1D)
        print(type(llista_1D))

        array_1D = np.array(llista_1D)
        print(array_1D)
        print("Forma: ", array_1D.shape)
        print("Mida: ", len(array_1D),array_1D.size)
        print(type(array_1D))

        # Fem el mateix partint d'una llista 2D
        llista_2D = [ [1,2,3], [4,5,6], [7,8,9] ]
        array_2D = np.array(llista_2D)
        print(llista_2D)
        print(array_2D)
        print("Forma: ", array_2D.shape)
        print("Mida: ", len(array_2D),array_2D.size)
        print(type(array_2D))

        # Exemple d'operació amb ndarray. L'analitzarem més tard
        array_vell= array_2D
        array_2D=array_2D+1.9
        print(array_vell[1][1], type(array_vell[1][1]))
        print(array_2D[1][1], type(array_2D[1][1]))
```

```
range(0, 9)
<class 'range'>
[0 1 2 3 4 5 6 7 8]
Forma:  (9,)
Mida:  9 9
<class 'numpy.ndarray'>
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
Forma:  (3, 3)
Mida:  3 9
<class 'numpy.ndarray'>
5 <class 'numpy.int64'>
6.9 <class 'numpy.float64'>
```

Cal tenir en compte que els objectes `ndarray` són homogenis, tots els seus elements són del mateix tipus bàsic. Si és necessari, es pot especificar aquest tipus quan es crea l'objecte amb la

funció `array()`.

```
In [3]: import numpy as np

        un_array = np.array( [ [1,2], [3,4] ], dtype=complex )
        print(un_array) # Nota que tots els enters s'han convertit en complexos
```

```
[[1.+0.j 2.+0.j]
 [3.+0.j 4.+0.j]]
```

2.13.2.2 Usant la funció `zeros()`

Aquesta funció crea un `ndarray` de les mides donades amb totes les components a zero:

```
In [4]: import numpy as np

        array_Z = np.zeros( (5,5) )
        print(array_Z)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

2.13.2.3 Usant la funció `ones()`

De forma similar, la funció `ones()` crea un `ndarray` de la mida donada ple de uns.

```
In [5]: import numpy as np

        array_U = np.ones( (5,5) )
        print(array_U)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

2.13.2.4 Amb la funció `empty()`

La funció `empty(shape, dtype=float, order='C')` retorna un `ndarray` amb la forma indicada. Els valors de les seves components no estan inicialitzats, i poden tenir qualsevol valor.

```
In [6]: import numpy as np
        empty_floats = np.empty([2, 2])
        empty_ints = np.empty([2, 2], dtype=int)
        print('2x2 float empty ndarray\n', empty_floats)
        print('2x2 int empty ndarray\n', empty_ints)
```

```
2x2 float empty ndarray
 [[0. 0.]
```

```
 [0. 0.]]
2x2 int empty ndarray
 [[4607182418800017408 4611686018427387904]
 [4613937818241073152 4616189618054758400]]
```

2.13.2.5 Usant la funció `eye()`

Aquesta funció retorna una matriu identitat (usualment s'escriu I, que es pronuncia 'eye' en anglès):

```
In [7]: import numpy as np

        # Generem la identitat 3 x 3
        I = np.eye(3)

        print("Matriu identitat 3 x 3")
        print(I)
```

```
Matriu identitat 3 x 3
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

2.13.2.6 Usant les funcions `arange()` o `linspace()`

La funció `arange()` permet crear seqüències numèriques regulars (de forma similar a `range()` però més general). La seva sintaxi és

```
arange(min,max,pas)   *màxim no inclòs*
```

i el resultat de la funció és un objecte `ndarray`:

```
In [8]: import numpy as np

        llista = np.arange(0.,4.5,0.5)
        print(llista)
        print(type(llista))
```

```
[0.  0.5 1.  1.5 2.  2.5 3.  3.5 4. ]
<class 'numpy.ndarray'>
```

Com que el resultat és un objecte `ndarray` podem usar directament `reshape` per modificar-ne la forma:

```
In [9]: import numpy as np

        un_array = np.arange(0.,4.5,0.5).reshape( (3,3) )
        print(un_array)
```

```
[[0.  0.5 1. ]
 [1.5 2.  2.5]
 [3.  3.5 4. ]]
```

La funció `arange()` té el problema que, atesa la precisió finita dels càlculs amb *float*, a vegades no es pot predir exactament el nombre d'elements que generarà. Quan això sigui important, es pot usar la funció `linspace()` que permet especificar exactament el nombre d'elements:

```
linspace(min, max, num_elements)
```

```
In [10]: import numpy as np
         import math

         vector = np.linspace(0.,math.pi,10)
         print(vector)
```

```
[0.         0.34906585 0.6981317  1.04719755 1.3962634  1.74532925
 2.0943951  2.44346095 2.7925268  3.14159265]
```

2.13.2.7 A partir d'una funció

Es pot crear un objecte `ndarray` usant una funció. La funció ha de rebre com a paràmetres els índexs d'una component i retornar el valor de la component que correspongui a aquests índexs.

```
In [11]: import numpy as np

         # Definim la funció
         def funcio(i,j):
             """Rep com a paràmetres els dos índexs i1,i2 i assigna com
                a valor de la component la suma dels dos"""
             return i+j

         # Creem una matriu de floats a partir de la funció
         A = np.fromfunction(funcio,(5,5),dtype=float)
         print("Matriu i1 + i2:")
         print(A)
```

```
Matriu i1 + i2:
[[0. 1. 2. 3. 4.]
 [1. 2. 3. 4. 5.]
 [2. 3. 4. 5. 6.]
 [3. 4. 5. 6. 7.]
 [4. 5. 6. 7. 8.]]
```

2.13.3 Còpia de `ndarray`

Important: tingues en compte que, de forma similar al que succeeix amb altres tipus mutables, quan es fa un assignament a una nova variable d'un `ndarray` no es copia, sinó que les dues variables corresponen al mateix objecte:

```
In [12]: import numpy as np

         A = np.arange(10)
         print("Array A")
         print(A,id(A))

         B = A
```

```
print("Array B")
print(B,id(B))

# Modifiquem A
A[0]=10
print("Array A modificat")
print(A,id(A))

# Comprovem que B també s'ha modificat
print("Array B modificat?")
print(B,id(B))

# Tanmateix, si reassignes A aleshores B no es modifica!
A = np.arange(2)
print("A reassignat")
print(A,id(A))
print("B com queda?")
print(B,id(B))
```

```
Array A
[0 1 2 3 4 5 6 7 8 9] 4564157584
Array B
[0 1 2 3 4 5 6 7 8 9] 4564157584
Array A modificat
[10  1  2  3  4  5  6  7  8  9] 4564157584
Array B modificat?
[10  1  2  3  4  5  6  7  8  9] 4564157584
A reassignat
[0 1] 4564158944
B com queda?
[10  1  2  3  4  5  6  7  8  9] 4564157584
```

Si es vol realment fer una còpia d'un `array` i assignar-lo a una nova variable s'ha d'usar la funció `copy()`:

```
In [13]: import numpy as np

A = np.arange(10)
print("Array A")
print(A, id(A))

B = A.copy()
print("Array B")
print(B, id(B))

# Modifiquem A
A[0]=10
print("Array A modificat")
print(A, id(A))

# Comprovem que B també s'ha modificat
print("Array B modificat?")
```

```
        print(B, id(B))
```

```
Array A
[0 1 2 3 4 5 6 7 8 9] 4564157904
Array B
[0 1 2 3 4 5 6 7 8 9] 4564226256
Array A modificat
[10  1  2  3  4  5  6  7  8  9] 4564157904
Array B modificat?
[0 1 2 3 4 5 6 7 8 9] 4564226256
```

2.13.4 Canvi de forma d'un objecte `ndarray`

La "forma" d'un objecte `ndarray` es pot canviar amb la funció `reshape()`:

```
In [14]: import numpy as np

         # Creem una matriu 1D
         llista_1D = range(9)
         array_1D = np.array(llista_1D)

         # Convertim la matriu_1D de 9 elements en una matriu 3 x 3
         array_2D = np.reshape(llista_1D,(3,3))
         print("Array 2D")
         print(array_2D)
         print()

         # Fem el mateix però creant una matriu 3D a partir d'una llista
         llista_1D = range(27)
         array_3D = np.reshape(llista_1D,(3,3,3))
         print("Array 3D")
         print(array_3D)
```

```
Array 2D
[[0 1 2]
 [3 4 5]
 [6 7 8]]

Array 3D
[[[ 0  1  2]
  [ 3  4  5]
  [ 6  7  8]]

 [[ 9 10 11]
  [12 13 14]
  [15 16 17]]

 [[18 19 20]
  [21 22 23]
  [24 25 26]]]
```

2.13.5 Accés als elements

Als elements d'un objecte `ndarray` s'hi pot accedir com als de les llistes normals de Python, donant els índexs corresponents:

```
In [15]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print(un_array)
         print("Element [0][0]: ",un_array[0][0])
         print("Element [2][2]: ",un_array[2][2])
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Element [0][0]:  1.0
Element [2][2]:  9.0
```

Però en el cas de `ndarray` també es pot indicar un element donant els dos índexs en forma de llista:

```
In [16]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print(un_array)
         print("Element [0][0]: ",un_array[0][0])
         print("Element [0,0]: ",un_array[0,0])
         print("Element [2][2]: ",un_array[2][2])
         print("Element [2,2]: ",un_array[2,2])
```

```
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Element [0][0]:  1.0
Element [0,0]:  1.0
Element [2][2]:  9.0
Element [2,2]:  9.0
```

També, com en el cas de les llistes, es poden seleccionar subconjunts indicant un rang d'índexs i el pas:

```
In [17]: import numpy as np

         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print("Array")
         print(un_array)
         print("Elements [0:2,0]: ")
         print(un_array[0:2,0])
         print("Elements [0:3:2,0]: ")
         print(un_array[0:3:2,0])
         print("Elements [0:2,0:2]: ")
         print(un_array[0:2,0:2])
         print("Elements [0:3,1:2]: ")
```

```
        print(un_array[0:3,1:2])
```

```
Array
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Elements [0:2,0]:
[1. 4.]
Elements [0:3:2,0]:
[1. 7.]
Elements [0:2,0:2]:
[[1. 2.]
 [4. 5.]]
Elements [0:3,1:2]:
[[2.]
 [5.]
 [8.]]
```

2.13.6 Inserció i supressió d'elements

Per afegir o suprimir elements cal usar les funcions `insert()`, `append()` i `delete()`. Nota que aquestes operacions **no modifiquen l'array original i retornen un nou array amb els canvis**.

Cal anar amb compte amb `arrays` de dimensió més gran que 1. Si el paràmetre **`axis`** no apareix, els `arrays` es transformen en unidimensionals abans d'executar la funció. En aquest cas, possiblement caldrà especificar el paràmetre **`axis`** perquè el resultat sigui el desitjat.

```
In [18]: import numpy as np

         # En una dimensió
         un_array = np.linspace(1.,5.,5)
         print("Array original 1D: ", un_array)

         print("append():", np.append(un_array,un_array))
         print("insert(pos 1):", np.insert(un_array,1,0))
         print("delete(pos 1):", np.delete(un_array,1))

         # En dues dimensions
         un_array = np.linspace(1.,9.,9).reshape( (3,3) )
         print("Array original 2D: ", un_array)

         print("append() 1D:", np.append(un_array,un_array))
         print("append(axis=0) 2D:")
         print(np.append(un_array,un_array,axis=0))
         print("append(axis=1) 2D:")
         print(np.append(un_array,un_array,axis=1))
```

```
Array original 1D:  [1. 2. 3. 4. 5.]
append(): [1. 2. 3. 4. 5. 1. 2. 3. 4. 5.]
insert(pos 1): [1. 0. 2. 3. 4. 5.]
delete(pos 1): [1. 3. 4. 5.]
Array original 2D:  [[1. 2. 3.]
```

```
 [4. 5. 6.]
 [7. 8. 9.]]
append() 1D: [1. 2. 3. 4. 5. 6. 7. 8. 9. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
append(axis=0) 2D:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]
 [1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
append(axis=1) 2D:
[[1. 2. 3. 1. 2. 3.]
 [4. 5. 6. 4. 5. 6.]
 [7. 8. 9. 7. 8. 9.]]
```

Nota que si `axis` no es defineix, els `arrays` es fan unidimensionals abans d'efectuar l'operació.

Exercici 2.13.1 *Donades a = [1,2,3,4,5], b = [[1],[2],[3],[4],[5]], c = [[1,2,3],[4,5,6],[7,8,9]] i d = [[1,2,3],[4,5,6],[7,8,9]]; efectua: a/ Els prints de a, b, c, i d expressant-los com a matrius i a més els elements de d com a elements float. b/ Per a un print indica quin és l'element* d_{11} d_{11}. *c/ Escriu l'última fila i l'última columna de d. d/ Dona la longitud de les files i de les columnes de d i calcula si és verdader o fals que el 10 sigui un element de d.*

Exercici 2.13.2 *Efectua: a/ escriu una matriu a d'una fila i quatre columnes que tingui com a elements els quatre primers nombres senars, b/ escriu una matriu de quatre files i dues columnes els elements de la primera columna de la qual siguin 3 i els de la segona estiguin en el rang (5,9), c/ escriu una matriu de dues columnes i cinc files els elements de la primera columna de la qual siguin iguals a 3x, i els de la segona x + 2, per a valors de x en el rang (0,5).*

Exercici 2.13.3 *Efectua: a/ escriu una matriu 1D que tingui com a elements de zero a onze, fes el print corresponent, b/ converteix-la en una matriu 3 x 4, c/ converteix la matriu resultant a vector, d/ converteix el vector una altra vegada a matriu i canvia tots els elements per 3.*

Exercici 2.13.4 *Efectua: a/ escriu una matriu 1D que tingui com a elements d'1 a 30, b/ converteix-la a matriu 3D, de manera que sigui 5 x 3 x 2, c/ repeteix l'exercici però ara converteix-la en matriu 3 x 5 x 2. Comenta els resultats.*

Exercici 2.13.5 *Efectua: a/ escriu una matriu 1D formada pels múltiples de 4 menors que 100, b/ forma amb els seus elements una matriu 5 x 5, i c/ canvia el quart element de la primera fila per 999.*

2.13.7 Operacions amb `ndarray`

Un dels grans avantatges dels objectes `ndarray` és que s'hi poden fer directament operacions amb valors numèrics o amb altres `arrays`, senzillament usant els operadors habituals +, -, *, /, **, %.

Nota que dur a terme les operacions d'aquesta manera és molt més ràpid que fer-ho usant bucles *for*.

2.13.7.1 Operacions amb valors numèrics

Si usem aquests operadors entre un `ndarray` i un nombre, la mateixa operació s'aplica a cadascun dels elements.

```
In [19]: import numpy as np

         A = np.linspace(1.,9.,9).reshape( (3,3) )
         print("Array A:")
         print(A)

         # Suma i resta d'un valor a tots els elements
         C = A+2
         print("Array C=A+2:")
         print(C)
         D = A-2
         print("Array D=A-2:")
         print(D)

         # Producte i divisió de tots els elements per un valor
         E = A*2
         print("Array E=A*2:")
         print(E)
         F = A/2
         print("Array F=A/2:")
         print(F)

         # Exponenciació
         G = A**2
         print("Array G=A**2:")
         print(G)

         # Mòdul
         H = A%2
         print("Array H=A%2:")
         print(H)

         # Operadors booleans
         I = A>5
         print("Array I=A>5:")
         print(I)

Array A:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Array C=A+2:
[[ 3.  4.  5.]
 [ 6.  7.  8.]
 [ 9. 10. 11.]]
Array D=A-2:
[[-1.  0.  1.]
 [ 2.  3.  4.]
 [ 5.  6.  7.]]
Array E=A*2:
[[ 2.  4.  6.]
 [ 8. 10. 12.]
```

```
 [14. 16. 18.]]
Array F=A/2:
[[0.5 1.  1.5]
 [2.  2.5 3. ]
 [3.5 4.  4.5]]
Array G=A**2:
[[ 1.  4.  9.]
 [16. 25. 36.]
 [49. 64. 81.]]
Array H=A%2:
[[1. 0. 1.]
 [0. 1. 0.]
 [1. 0. 1.]]
Array I=A>5:
[[False False False]
 [False False  True]
 [ True  True  True]]
```

2.13.7.2 Operacions element a element

Si usem aquests operadors entre dos `ndarray`, l'operació s'aplica a cada parell d'elements.

```
In [20]: import numpy as np

         A = np.linspace(1.,9.,9).reshape( (3,3) )
         B = np.linspace(10.,18.,9).reshape( (3,3) )
         print("Array A:")
         print(A)
         print("Array B:")
         print(B)

         # Suma i resta de matrius (element a element)
         C = A+B
         print("Array C=A+B:")
         print(C)
         D = A-B
         print("Array D=A-B:")
         print(D)

         # Producte i divisió (element a element)
         E = A*B
         print("Array E=A*B:")
         print(E)
         F = A/B
         print("Array F=A/B:")
         print(F)

         # Exponenciació (element a element)
         G = A**B
         print("Array G=A**B:")
         print(G)
```

```
        # Operacions booleanes (element a element)
        H = A>B
        print("Array H=A>B:")
        print(H)
```

```
Array A:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Array B:
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]
Array C=A+B:
[[11. 13. 15.]
 [17. 19. 21.]
 [23. 25. 27.]]
Array D=A-B:
[[-9. -9. -9.]
 [-9. -9. -9.]
 [-9. -9. -9.]]
Array E=A*B:
[[ 10.  22.  36.]
 [ 52.  70.  90.]
 [112. 136. 162.]]
Array F=A/B:
[[0.1        0.18181818 0.25      ]
 [0.30769231 0.35714286 0.4       ]
 [0.4375     0.47058824 0.5       ]]
Array G=A**B:
[[1.00000000e+00 2.04800000e+03 5.31441000e+05]
 [6.71088640e+07 6.10351562e+09 4.70184985e+11]
 [3.32329306e+13 2.25179981e+15 1.50094635e+17]]
Array H=A>B:
[[False False False]
 [False False False]
 [False False False]]
```

2.13.7.3 Operacions unitàries

NumPy inclou algunes operacions unitàries (que s'apliquen a un únic objecte `ndarray`):

- `sum()` suma de totes les components (o de les d'un eix).
- `min()` retorna la component amb el mínim valor (o de les d'un eix).
- `max()` retorna la component amb el màxim valor (o de les d'un eix).

```
In [21]: import numpy as np

        A = np.linspace(1.,9.,9).reshape( (3,3) )
        print("Array A: ")
        print(A)
```

```
# Suma de totes les components
print("Suma global: ", A.sum())
print("Suma de les columnes: ", A.sum(axis=0))
print("Suma de les files: ", A.sum(axis=1))

# Màxim
print("Màxim global: ",A.max())
print("Màxim de cada columna: ",A.max(axis=0))
print("Màxim de cada fila: ",A.max(axis=1))

# Mínim
print("Mínim global: ",A.min())
print("Mínim de cada columna: ",A.min(axis=0))
print("Mínim de cada fila: ",A.min(axis=1))
```

```
Array A:
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Suma global:  45.0
Suma de les columnes:  [12. 15. 18.]
Suma de les files:  [ 6. 15. 24.]
Màxim global:  9.0
Màxim de cada columna:  [7. 8. 9.]
Màxim de cada fila:  [3. 6. 9.]
Mínim global:  1.0
Mínim de cada columna:  [1. 2. 3.]
Mínim de cada fila:  [1. 4. 7.]
```

Exercici 2.13.6 *Efectua: a/ Donades les llistes x = [1,2,3] i y = [2,4,3], converteix-les en* ***ndarrays*** *i verifica, element a element, si x és menor que y, si x és més gran que y o si són diferents. b/ Donades les llistes n = [1,3,5] i m = [0,2,7], converteix-les en* ***ndarrays*** *i comprova, element a element, si n és més gran que m, si n és més gran o igual que m, si n és igual a m, si n és menor que m, o si n és menor o igual a m.*

Exercici 2.13.7 *Donada la llista x= [1,2,3,4,5,6,7,8,9,4,6]: a/ Converteix-la en un* ***ndarray*** *1D i calcula el producte dels seus elements. b/ Els índexs dels valors màxim i mínim. c/ El valor mitjà, i la desviació típica.*

Exercici 2.13.8 *Donada la matriu 1D dels múltiples de 3 en el rang (0,103): a/ Converteix-la en matriu 5 x 7. b/ Transforma-la de manera que si un nombre no és múltiple de dos se sostregui una unitat.*

Exercici 2.13.9 *Donades les llistes x = [[1,2],[3,4]] i y = [[8,0],[5,6]]. a/ Converteix-les en ndarrays a, b respectivament. b/ Efectua a + b, a - b, el producte, element per element, i el producte ab.*

Exercici 2.13.10 *Donada la matriu 1D dels múltiples de 5 en el rang (5,50). a/ Converteix-la en un ndarray 3 x 3. b/ Calcula el seu logaritme. c/ Extreu l'arrel quadrada dels seus elements.*

2.13.7.4 Funcions

NumPy també estén les funcions matemàtiques habituals de manera que es poden aplicar directament als `ndarray`. En altres paraules, es pot aplicar una funció a cadascuna de les components de `ndarray` amb una sola crida.

```
In [22]: import numpy as np

         A = np.linspace(1,np.pi,9).reshape( (3,3) )
         print("Array A: ")
         print(A)

         # sin()
         print("sin(A): ")
         print(np.sin(A))

         # exp()
         print("exp(A): ")
         print(np.exp(A))

         # log()
         print("log(A): ")
         print(np.log(A+0.0001)) # Sumem 0.0001 per evitar que el zero doni errors

         # sqrt()
         print("sqrt(A): ")
         print(np.sqrt(A))
```

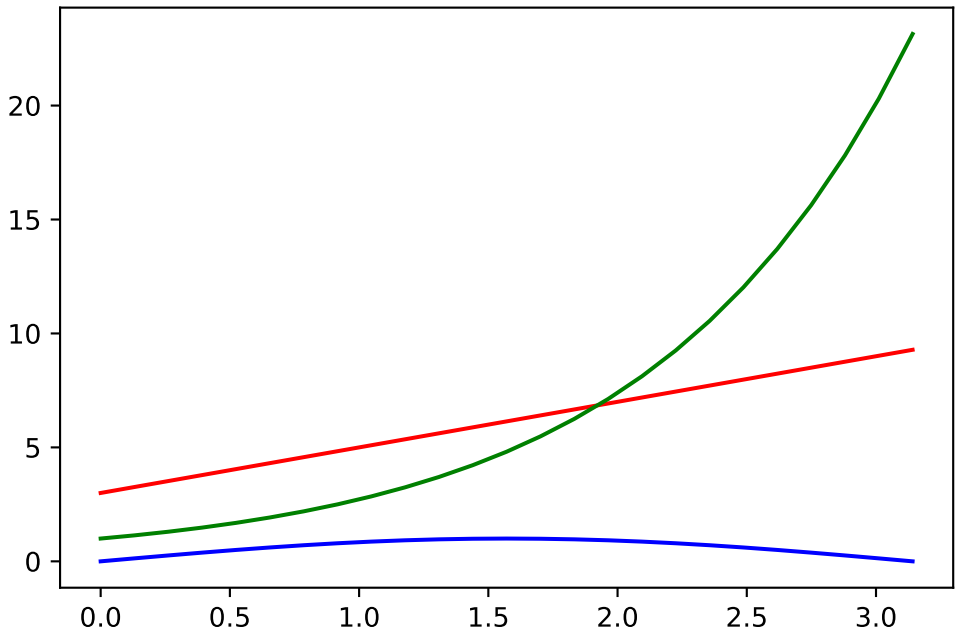
```
Array A:
[[1.         1.26769908 1.53539816]
 [1.80309725 2.07079633 2.33849541]
 [2.60619449 2.87389357 3.14159265]]
sin(A):
[[8.41470985e-01 9.54416610e-01 9.99373550e-01]
 [9.73139260e-01 8.77582562e-01 7.19510518e-01]
 [5.10183526e-01 2.64513174e-01 1.22464680e-16]]
exp(A):
[[ 2.71828183  3.55266875  4.6431739 ]
 [ 6.06841375  7.93113638 10.36562879]
 [13.54739788 17.70582306 23.14069263]]
log(A):
[[9.99950003e-05 2.37282391e-01 4.28854865e-01]
 [5.89561337e-01 7.27981522e-01 8.49550496e-01]
 [9.57929477e-01 1.05570255e+00 1.14476172e+00]]
sqrt(A):
[[1.         1.12592144 1.23911184]
 [1.34279457 1.43902617 1.52921398]
 [1.61437124 1.6952562  1.77245385]]
```

Nota que en aquest cas és important distingir la funció sinus del mòdul `math` de la funció amb el mateix nom del mòdul `numpy`. D'aquí la importància del prefix i la recomanació d'evitar fer *wild imports*. El mateix ocorre amb `sum`, `max` i `min`, que a més tenen un nombre d'arguments diferent.

Nota també que amb la funcionalitat de les operacions de NumPy el càlcul de valors de funcions i la seva representació és molt simple. Només cal definir un `ndarray` amb els valors de x i calcular l'`ndarray` amb els valors corresponents de la funció, com podem veure en l'exemple següent:

```
In [23]: %pylab inline
         %config InlineBackend.figure_format = 'pdf'


         import numpy as np
         import matplotlib.pyplot as plt

         # Calculem els valors de x entre 0 i pi
         x = np.linspace(0., np.pi, 25, endpoint=True)

         # Calculem les funcions d'aquests valors
         lineal = 2*x+3
         sinusoidal = np.sin(x)
         exponencial = np.exp(x)

         print ("x = ", x)
         print ("lineal = ", lineal)

         # Els representem (usem mathplotlib, que explicarem en les sessions següents)
         plt.plot(x, lineal, color = 'red')
         plt.plot(x, sinusoidal, color = 'blue')
         plt.plot(x, exponencial, color = 'green')
         plt.show()
```

```
Populating the interactive namespace from numpy and matplotlib
x =  [0.         0.13089969 0.26179939 0.39269908 0.52359878 0.65449847
 0.78539816 0.91629786 1.04719755 1.17809725 1.30899694 1.43989663
 1.57079633 1.70169602 1.83259571 1.96349541 2.0943951  2.2252948
 2.35619449 2.48709418 2.61799388 2.74889357 2.87979327 3.01069296
 3.14159265]
lineal =  [3.         3.26179939 3.52359878 3.78539816 4.04719755 4.30899694
 4.57079633 4.83259571 5.0943951  5.35619449 5.61799388 5.87979327
 6.14159265 6.40339204 6.66519143 6.92699082 7.1887902  7.45058959
 7.71238898 7.97418837 8.23598776 8.49778714 8.75958653 9.02138592
 9.28318531]
```



2.13.8 Acumulació i separació d'arrays

NumPy també ofereix funcions per ajuntar dos `ndarray`, verticalment o horitzontalment:

```
In [24]: import numpy as np

         A = np.linspace(1.,9.,9).reshape( (3,3) )
         B = np.linspace(10.,18.,9).reshape( (3,3) )
         C = np.linspace(19.,27.,9).reshape( (3,3) )
         print ("Array A:")
         print (A)
         print ("Array B:")
         print (B)
         print ("Array C:")
         print (C)

         # Els ajuntem horitzontalment
         print ("hstack((A,B))")
         print (np.hstack( (A,B) )) # Nota que passem com a argument una tupla
         print ("hstack((A,B,C))")
         print (np.hstack( (A,B,C) ))

         # Els ajuntem verticalment
         print ("vstack((A,B))")
         print (np.vstack( (A,B) ))
         print ("vstack((A,B,C))")
         print (np.vstack( (A,B,C) ))
```

```
Array A:
[[1. 2. 3.]
```

```
 [4. 5. 6.]
 [7. 8. 9.]]
Array B:
[[10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]
Array C:
[[19. 20. 21.]
 [22. 23. 24.]
 [25. 26. 27.]]
hstack((A,B))
[[ 1.  2.  3. 10. 11. 12.]
 [ 4.  5.  6. 13. 14. 15.]
 [ 7.  8.  9. 16. 17. 18.]]
hstack((A,B,C))
[[ 1.  2.  3. 10. 11. 12. 19. 20. 21.]
 [ 4.  5.  6. 13. 14. 15. 22. 23. 24.]
 [ 7.  8.  9. 16. 17. 18. 25. 26. 27.]]
vstack((A,B))
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]
 [10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]]
vstack((A,B,C))
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]
 [10. 11. 12.]
 [13. 14. 15.]
 [16. 17. 18.]
 [19. 20. 21.]
 [22. 23. 24.]
 [25. 26. 27.]]
```

Nota. També es pot separar un *array* en diverses parts amb les funcions `vsplit()` i `hsplit()`.

```
In [25]: import numpy as np
         help(np.vsplit)
```

```
Help on function vsplit in module numpy:

vsplit(ary, indices_or_sections)
    Split an array into multiple sub-arrays vertically (row-wise).

    Please refer to the ``split`` documentation.  ``vsplit`` is equivalent
    to ``split`` with `axis=0` (default), the array is always split along the
    first axis regardless of the array dimension.

    See Also
    --------
```

```
    split : Split an array into multiple sub-arrays of equal size.

    Examples
    --------
    >>> x = np.arange(16.0).reshape(4, 4)
    >>> x
    array([[  0.,   1.,   2.,   3.],
           [  4.,   5.,   6.,   7.],
           [  8.,   9.,  10.,  11.],
           [ 12.,  13.,  14.,  15.]])
    >>> np.vsplit(x, 2)
    [array([[ 0.,  1.,  2.,  3.],
           [ 4.,  5.,  6.,  7.]]),
     array([[  8.,   9.,  10.,  11.],
           [ 12.,  13.,  14.,  15.]])]
    >>> np.vsplit(x, np.array([3, 6]))
    [array([[  0.,   1.,   2.,   3.],
           [  4.,   5.,   6.,   7.],
           [  8.,   9.,  10.,  11.]]),
     array([[ 12.,  13.,  14.,  15.]]),
     array([], dtype=float64)]

    With a higher dimensional array the split is still along the first axis.

    >>> x = np.arange(8.0).reshape(2, 2, 2)
    >>> x
    array([[[ 0.,  1.],
            [ 2.,  3.]],
           [[ 4.,  5.],
            [ 6.,  7.]]])
    >>> np.vsplit(x, 2)
    [array([[[ 0.,  1.],
            [ 2.,  3.]]]),
     array([[[ 4.,  5.],
            [ 6.,  7.]]])]
```

2.13.9 Bucles `for`

Ja hem vist que els `ndarray` es poden usar en els bucles *for*. Quan són d'una dimensió, s'itera sobre els elements, i quan són multidimensionals, s'itera sobre els elements del primer eix, que són `ndarrays`:

```
In [26]: import numpy as np

         print ("Array 1D")
         array_1D = np.arange(3)
         for element in array_1D:
             print (element)
             print ("-------")
```

```
        print ("Array 2D")
        array_2D = np.arange(9).reshape(3,3)
        for element in array_2D:
            print (element)
            print ("-------")

        print ("Array 3D")
        array_3D = np.arange(27).reshape(3,3,3)
        for element in array_3D:
            print (element)
            print ("-------")
```

```
Array 1D
0
-------
1
-------
2
-------
Array 2D
[0 1 2]
-------
[3 4 5]
-------
[6 7 8]
-------
Array 3D
[[0 1 2]
 [3 4 5]
 [6 7 8]]
-------
[[ 9 10 11]
 [12 13 14]
 [15 16 17]]
-------
[[18 19 20]
 [21 22 23]
 [24 25 26]]
-------
```

Si es vol iterar sobre tots els elements independentment de les dimensions de l'`ndarray` es pot usar l'iterador `flat` per "desplegar" l'`ndarray` en el bucle `for`:

```
In [27]: import numpy as np
```

```
        array_2D = np.arange(9).reshape(3,3)
        print (array_2D)
        print

        for element in array_2D.flat:
```

```
            print (element)
```

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
0
1
2
3
4
5
6
7
8
```

Exercici 2.13.11 *Construeix un* **ndarray** *3 x 3, a, de nombres aleatoris en el rang (2, 9). Calcula:*

a) Els **ndarray** *sin(a) i cos(a)*

b) **ndarray** $sin^2(a) + cos^2(a)$

Recordes aquesta propietat?

Exercici 2.13.12 *Donat un ndarray 1D generat en el rang (1,36), efectua les operacions i prints corresponents:*

a) Resta 2 a tots els seus elements.

b) Insereix a la posició 3 el nombre 40.

c) Calcula la seva exponencial.

d) Converteix en un ndarray 6 x 6 l'ndarray 1D obtingut a l'apartat b).

e) Divideix la matriu resultant en dues parts verticalment.

f) Ajunta l'ndarray obtingut a l'apartat d) horitzontalment amb si mateix.

Exercici 2.13.13 *Calcula:*

a) Utilitzant la sentència **for**, *escriu un array n d'una columna i cinc files tots els elements del qual siguin 6.*

b) Repeteix l'exercici però ara per a un ndarray 5 x 5 de manera que els elements de la primera columna siguin 2x per a x en un rang (0,5) i els elements restants de cada fila siguin x-y per a y en el rang (0,4).

2.13.10 Àlgebra lineal

NumPy implementa les operacions d'àlgebra lineal, és a dir, les operacions amb vectors i matrius. Les eines d'àlgebra lineal es troben en el mòdul `numpy.linalg`:

```
from numpy.linalg import *
```

2.13.10.1 Eines d'àlgebra lineal

2.13.10.1.1 Transposició

Es pot fer amb el mètode `transpose()` dels `ndarrays`:

```
In [28]: import numpy as np

         # Generem un array 2D
         A = np.linspace(1.,9.,9).reshape(3,3)
         print ("Array A")
         print (A)

         # El transposem
         print ("Array A transposat")
         print (A.transpose())
```

```
Array A
[[1. 2. 3.]
 [4. 5. 6.]
 [7. 8. 9.]]
Array A transposat
[[1. 4. 7.]
 [2. 5. 8.]
 [3. 6. 9.]]
```

2.13.10.2 Determinants i matriu inversa

Els determinants es calculen amb la funció `linalg.det()` i la matriu inversa amb la funció `linalg.inv()`

```
In [29]: import numpy as np

         # Generem un array 2D
         A = np.array( [[1,1,2], [2,1,1], [1,1,1]] )
         print ("Array A")
         print (A)

         # Calculem el determinant
         print ("Determinant de A: ", np.linalg.det(A))

         # L'invertim
         print ("Inversa de A")
         print (np.linalg.inv(A))
```

```
Array A
[[1 1 2]
 [2 1 1]
 [1 1 1]]
Determinant de A:  1.0
Inversa de A
[[ 0.  1. -1.]
 [-1. -1.  3.]
 [ 1. -0. -1.]]
```

2.13.10.3 Rang i traça d'una matriu

El rang d'una matriu es pot calcular usant la funció `numpy.linalg.matrix_rank(A)`.

La traça d'una matriu es calcula amb la funció `numpy.trace()`:

```
In [30]: import numpy as np

         # Generem un array 2D
         A = np.array( [[1,1,2], [2,4,1], [1,1,1]] )
         print ("Array A")
         print (A)

         # Calculem el rang
         print("Rang de A: ", np.linalg.matrix_rank(A))
         # Calculem la traça
         print ("Traça de A: ", np.trace(A))
```

```
Array A
[[1 1 2]
 [2 4 1]
 [1 1 1]]
Rang de A:  3
Traça de A:  6
```

2.13.10.4 Producte de matrius

El producte de matrius s'implementa amb la funció `numpy.dot()`:

```
In [31]: import numpy as np

         # Generem un array 2D
         A = np.array( [[1,1,2], [2,1,1], [1,1,1]] )
         print ("Array A")
         print (A)

         # El multipliquem per ell mateix
         print ("A*A")
         print (np.dot(A,A))

         # Multipliquem la matriu per un vector
         v = array( [3,1,3] )
         print ("A*v")
         print (np.dot(A,v))
```

```
Array A
[[1 1 2]
 [2 1 1]
 [1 1 1]]
A*A
[[5 4 5]
 [5 4 6]
 [4 3 4]]
A*v
```

```
[10 10  7]
```

Exercici 2.13.14 *Donada la llista x = [[4,6,7],[1,2,3],[5,8,9]], efectua:*

a) Converteix-la en ***ndarray****.*

b) Calcula la suma dels seus elements, el seu producte, el valor mitjà i la desviació típica.

c) Ordena-la per files.

d) Mostra, fent el ***print*** *corresponent, els elements de la diagonal, i continua l'exercici calculant el determinant.*

e) Calcula la matriu inversa, la matriu transposada, la matriu triangular superior i la inferior.

f) Genera un ***ndarray*** *3 x 3 de zeros.*

g) Genera un ***ndarray*** *identitat 3 x 3 float i un* ***ndarray*** *identitat 3 x 3 enter.*

Exercici 2.13.15 *Donada la llista a = [[1,2,3],[4,5,6],[0,8,9]], efectua les operacions i prints corresponents:*

a) Converteix-la en ***ndarray****.*

b) Calcula'n el rang i la traça.

c) Indica quin lloc ocupa l'element 8.

2.13.10.5 Solució de sistemes d'equacions

Un sistema d'equacions

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

es pot representar en forma matricial

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$A\vec{x} = \vec{b}$$

Donada la matriu A del sistema i el vector de termes independents $\vec{b}$ es pot obtenir la solució $\vec{x}$ amb la funció `numpy.linalg.solve()`:

```
In [32]: import numpy as np

         # Matriu del sistema
         A = np.array( [[1,1,2], [2,1,1], [1,1,1]] )

         print (A)

         # Vector de termes independents
         b = np.array( [1,1,1] )
```

```
print ("Vector de termes independents")
print (b)

# Calculem la solució
x = np.linalg.solve(A,b)
print ("Solució")
print (x)
```

```
[[1 1 2]
 [2 1 1]
 [1 1 1]]
Vector de termes independents
[1 1 1]
Solució
[ 0.  1. -0.]
```

2.13.11 Classe `matrix`

En les operacions d'àlgebra lineal que hem descrit fins ara hem usat objectes de tipus `ndarray`. Tanmateix, si hem de treballar amb matrius "clàssiques", és a dir, 2D i regulars, podem usar una classe de NumPy dissenyada específicament per a aquest cas, la classe `matrix`. Podeu trobar una descripció completa de la classe aquí.

Aquesta classe es pot usar com un `ndarray` normal en les operacions anteriors.

Exercici 2.13.16 *Usant la classe* ***`matrix`*** *resol el sistema:*

$$\begin{cases} 2x + 4y + z = 24 \\ x - 2y + z = -9 \\ 3z + y - z = 16 \end{cases}$$

Exercici 2.13.17 *Donada la matriu del sistema anterior, calcula:*

a) La matriu transposada

b) El determinant

c) La matriu inversa.

2.14 Introducció a Matplotlib

Matplotlib és una biblioteca de Python per crear gràfics 2D que inclou una gran varietat de funcions i eines. Veurem aquí només una petita introducció a les seves funcionalitats i et pots remetre a la pàgina oficial de la biblioteca, http://matplotlib.org/ i en particular al tutorial de **pyplot**, una de les seves components que discutirem en aquest document:

http://matplotlib.org/users/pyplot_tutorial.html

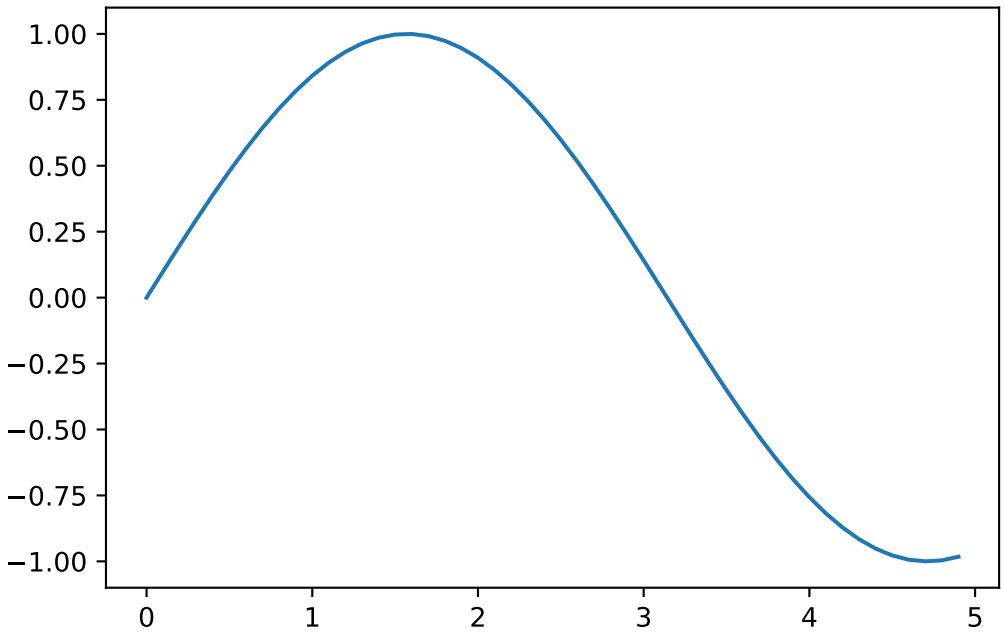
matplotlib.pyplot proporciona funcions que faciliten la creació de gràfics del tipus habitualment usat en física i funciona de forma similar a MATLAB. Per usar-la haurem de fer la importació següent:

```
import matplotlib.pyplot
```

2.14.1 Creació d'un gràfic bàsic: comanda `plot()`

La funció `plot()` permet fer una gràfica d'un conjunt de punts. L'ús més senzill és a partir de dues llistes que donin els valors de x i els valors de y dels punts que es passen com a arguments, com es veu en l'exemple següent. Per defecte, els punts s'uneixen seqüencialment amb línies, de manera que s'obté una representació en forma de línia contínua.

```
In [1]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,5) i a partir d'ells, valors de sin(x)
        x = np.arange(0, 5, 0.1);
        y = np.sin(x)

        # Representem els punts
        plt.plot(x, y)
        plt.show()
```



Nota. Si només es dona una llista com a argument `plot()` la interpreta com una llista de valors de y i crea la gràfica assumint valors de x equiespaiats $x = [0, 1, 2, ...]$.

2.14.2 Opcions de `plot()`

La funció `plot()` permet configurar la forma com es representen les llistes de valors mitjançant arguments suplementaris.

La forma abreviada de fer-ho és una cadena de dos caràcters:

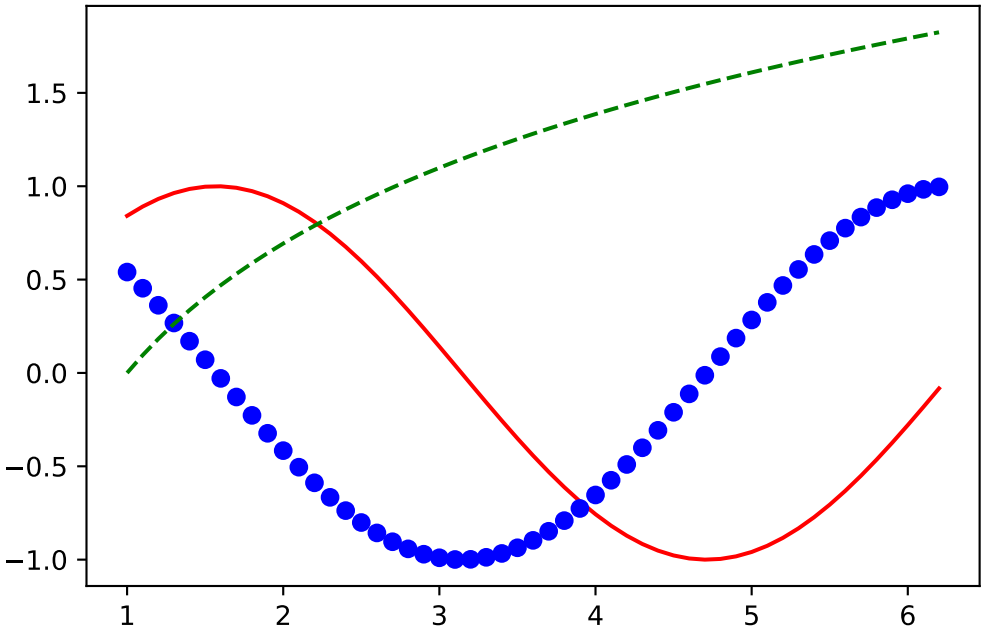
- El primer caràcter és una lletra que indica el color de traçat (nota la correspondència entre el caràcter i el nom del color en anglès):

caràcter	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

- Després es poden afegir un o dos caràcters que indiquen el tipus de traçat. Per exemple `'-'` indica una línia contínua, `'--'` una línia discontínua, `'o'` no s'uneixen els punts amb línies sinó que es marquen amb un símbol rodó, etc. Es pot trobar la llista completa d'opcions a la documentació de la funció `plot`.

En l'exemple següent usem aquesta opció per representar tres conjunts de punts en una sola gràfica. Nota que es poden traçar diverses llistes en una única crida a la funció `plot()`.

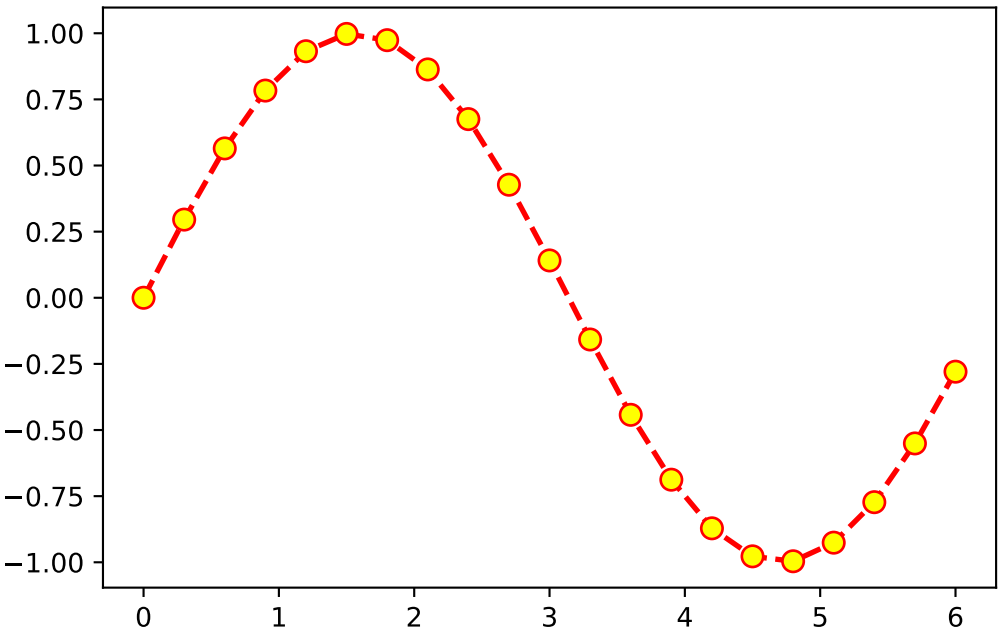
```
In [2]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (1,2*pi) i a partir d'ells, tres funcions
        x = np.arange(1., 2*np.pi, 0.1);
        y1 = np.sin(x)
        y2 = np.cos(x)
        y3 = np.log(x)

        # Representem els punts
        plt.plot(x, y1, "r-")
        plt.plot(x, y2, "bo", x, y3, "g--")
        plt.show()
```



La forma expandida d'especificar l'aparença del traçat és especificar cada característica per separat. En l'exemple següent es poden veure algunes opcions i es pot trobar la llista completa d'opcions a la documentació de la funció `plot`.

```
In [3]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,2*pi) i a partir d'ells, valors de sin(x)
        x = np.arange(0., 2*np.pi, 0.3);
        y1 = np.sin(x)
```

```
# Representem els punts
plt.plot(x, y1, color='red', linestyle='dashed', marker='o',
         markerfacecolor='yellow', markersize=8, linewidth=2)
plt.show()
```



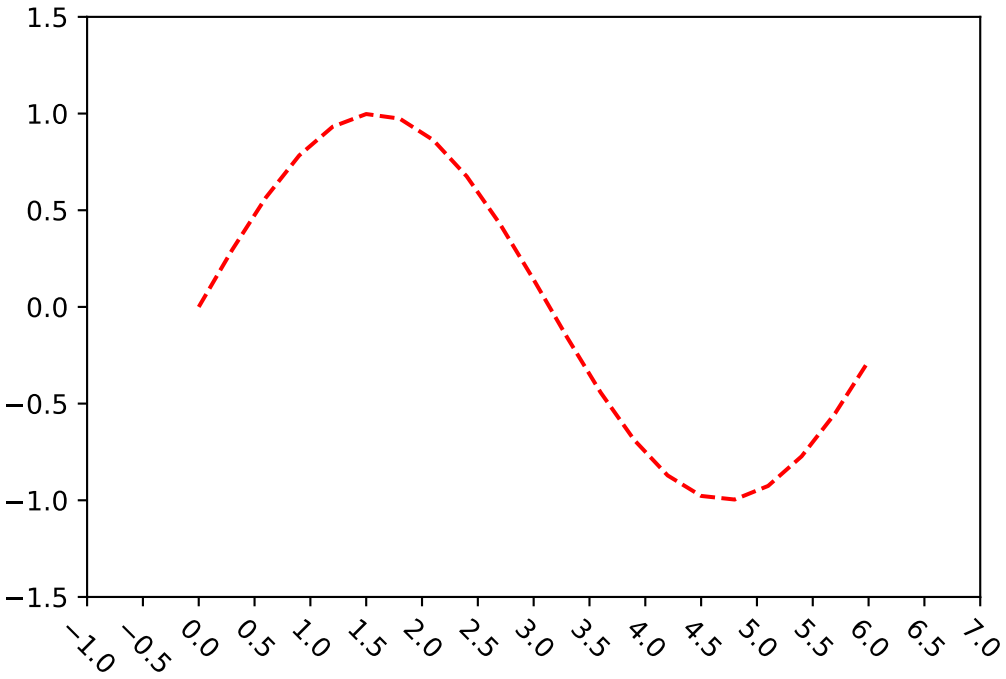
2.14.3 Control dels eixos

Es pot controlar l'aparença dels eixos amb les funcions:

- `xlim(a,b)` límits de l'eix x
- `ylim(c,d)` límits de l'eix y
- `axis(a,b,c,d)` requadre amb els límits dels eixos
- `xticks()` marcadors de l'eix x
- `yticks()` marcadors de l'eix y
- `xscale()` escala de l'eix x (lineal, logarítmica)
- `yscale()` escala de l'eix y (lineal, logarítmica)

```
In [4]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,2*pi) i a partir d'ells, valors de sin(x)
        x = np.arange(0., 2*np.pi, 0.3);
        y1 = np.sin(x)

        # Representem els punts
        plt.plot(x, y1, color='red', linestyle='dashed')
        plt.xlim(-1,7)
```

```
plt.ylim(-1.5,1.5)
plt.xticks(np.arange(-1, 7.5, 0.5), rotation = -45)
plt.show()
```



2.14.4 Gràfics múltiples

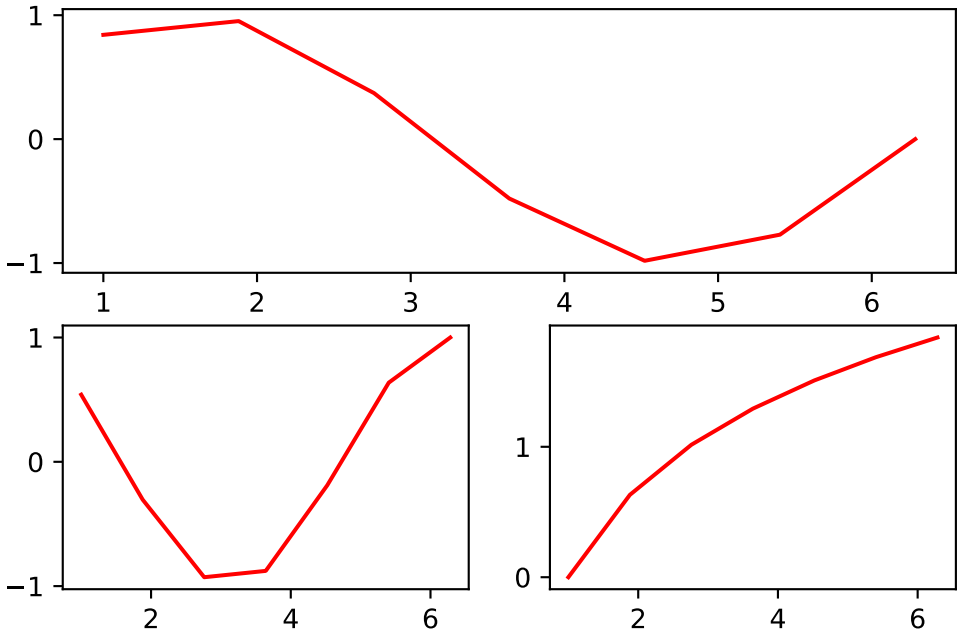
Es poden crear figures amb diversos gràfics usant la funció `subplot()`. Aquesta funció rep tres arguments: `subplot(nFiles,nCols,numPlot)`:

- La figura es divideix conceptualment en una quadrícula de `nFiles` × `nCols`.
- `numPlot` indica en quina posició de la quadrícula ens situem per treballar.
- A partir de la crida les funcions gràfiques s'aplicaran a la posició de la quadrícula indicada.

En l'exemple següent fem una figura amb tres gràfiques:

```
In [5]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (1,2*pi) i a partir d'ells, tres funcions
        x = np.linspace(1., 2*np.pi, 7);
        y1 = np.sin(x)
        y2 = np.cos(x)
        y3 = np.log(x)

        # Representem els arrays anteriors en tres gràfiques
        plt.subplot(2,1,1)
        plt.plot(x, y1, "r-")
```

```
plt.subplot(2,2,3)
plt.plot(x, y2, "r-")
plt.subplot(2,2,4)
plt.plot(x, y3, "r-")
plt.show()
```



Les crides a `subplot()` es poden fer de forma abreviada:

`subplot(2,2,1) = subplot(221)`

Si fas crides successives amb valors diferents dels arguments `nFiles`,`nCols`, les gràfiques es posicionaran a la figura de la forma corresponent i poden sobreescriure gràfiques anteriors.

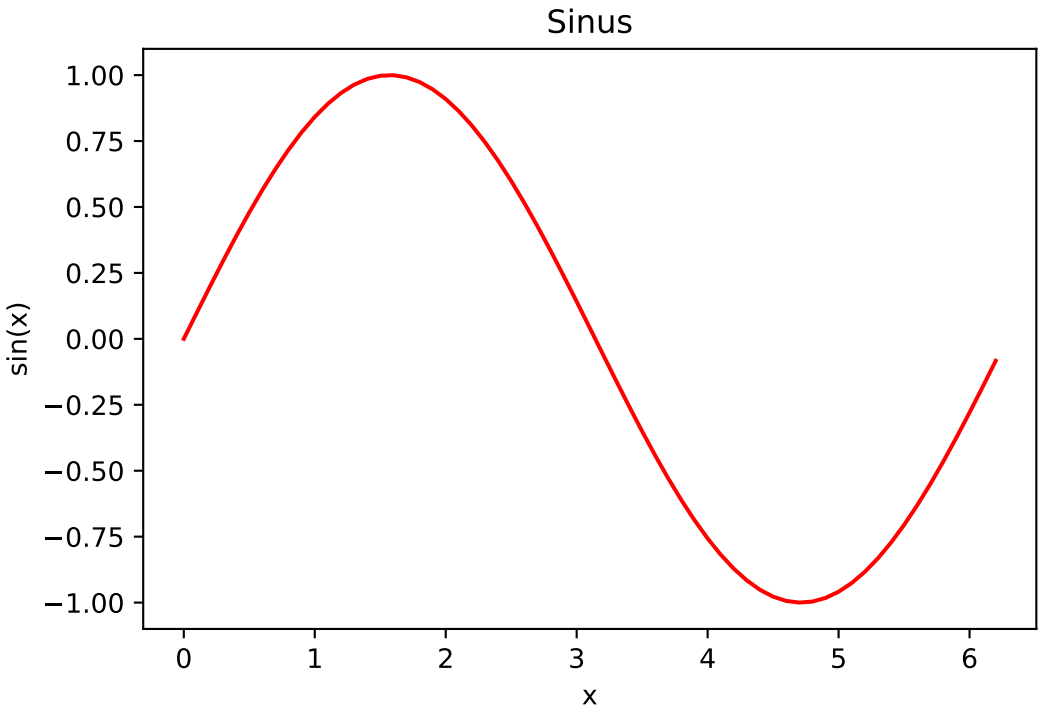
2.14.5 Text

Les funcions següents permeten afegir text a les gràfiques:

- `xlabel()` etiqueta de l'eix x
- `ylabel()` etiqueta de l'eix y
- `title()` títol del gràfic

```
In [6]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,2*pi) i a partir d'ells, valors de sin(x)
        x = np.arange(0., 2*np.pi, 0.1);
        y1 = np.sin(x)

        # Representem els punts
```

```
plt.plot(x, y1, color='red')

# Hi afegim text
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.title("Sinus")
plt.show()
```



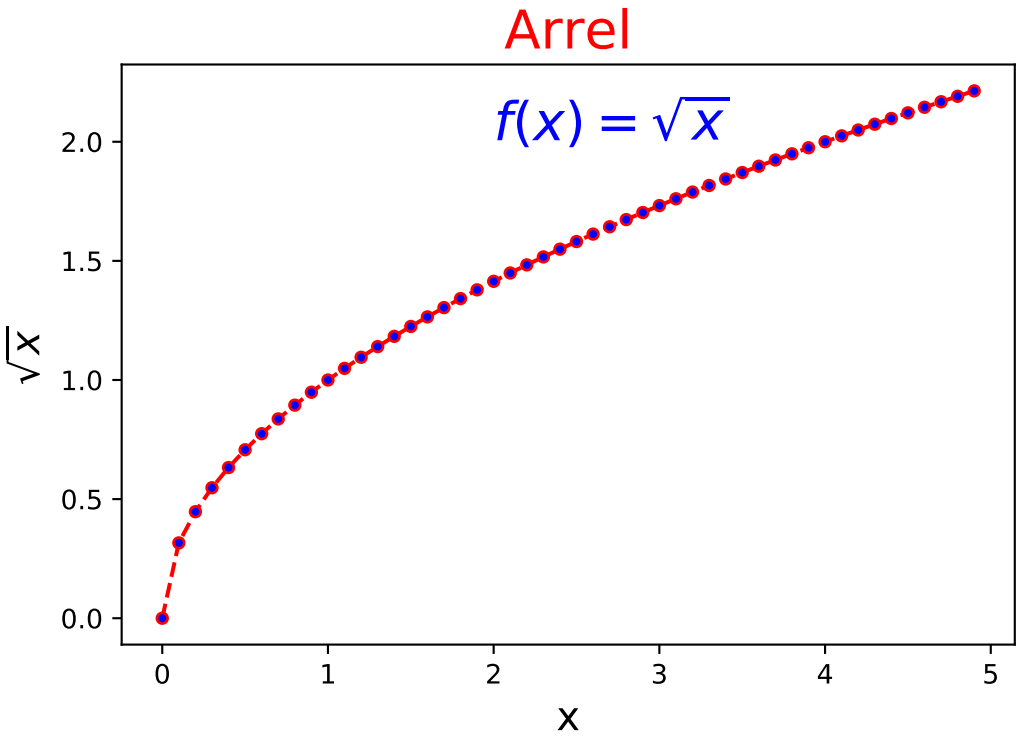
Més en general, la funció `text(x,y,str)` permet escriure un text en qualsevol posició del gràfic, on x, y són les coordenades en què posicionar-lo a la gràfica i `str` la cadena amb el text a incloure.

Nota. Per a les quatre funcions de text

- Si dones text entre "$"s'interpretarà com una fórmula de LaTeX.
- Pots canviar la mida i color del text usant com a arguments `, fontsize=NN, color='xxxx'`.

```
In [7]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,5) i a partir d'ells, valors de sqrt(x)
        x = np.arange(0., 5, 0.1);
        y1 = np.sqrt(x)

        # Representem els punts
```

```
plt.plot(x, y1, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=4)

# Hi afegim text
plt.xlabel("x",fontsize=15)
plt.ylabel("$\sqrt{x}$",fontsize=15)
plt.title("Arrel", fontsize=20, color='red')
plt.text(2,2,"$f(x)=\sqrt{x}$", fontsize=20, color='blue')
plt.show()
```



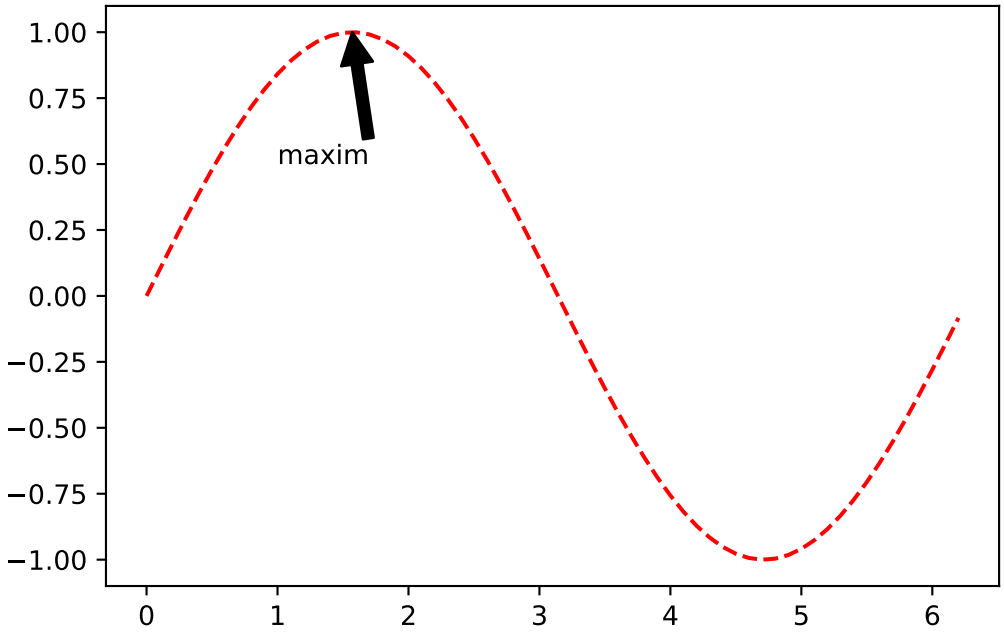
Finalment, la funció `annotate()` crea un text amb una fletxa per inserir una anotació que assenyali un punt de la gràfica. Rep com a arguments:

- El text a mostrar
- `xy=(a,b)` les coordenades de la punta de la fletxa
- `xytext=(c,d)` la posició del text
- `arrowprops=dict(facecolor='xxx', ...)` les característiques de la fletxa

Pots veure l'ajuda d'`annotate()` per a més detalls.

```
In [8]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import math
        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,2*pi) i a partir d'ells valors, de sin(x)
        x = np.arange(0., 2*np.pi, 0.1);
```

```
y1 = np.sin(x)

# Representem els punts
plt.plot(x, y1, color='red', linestyle='dashed')

# Marquem el màxim
plt.annotate("màxim", xy=(math.pi/2.,1), xytext=(1,0.5),
             arrowprops=dict(facecolor='black'))
plt.show()
```



2.14.6 Guardant les figures en un fitxer

Pots usar la comanda `savefig()` per guardar la figura en un fitxer. Pren com a arguments:

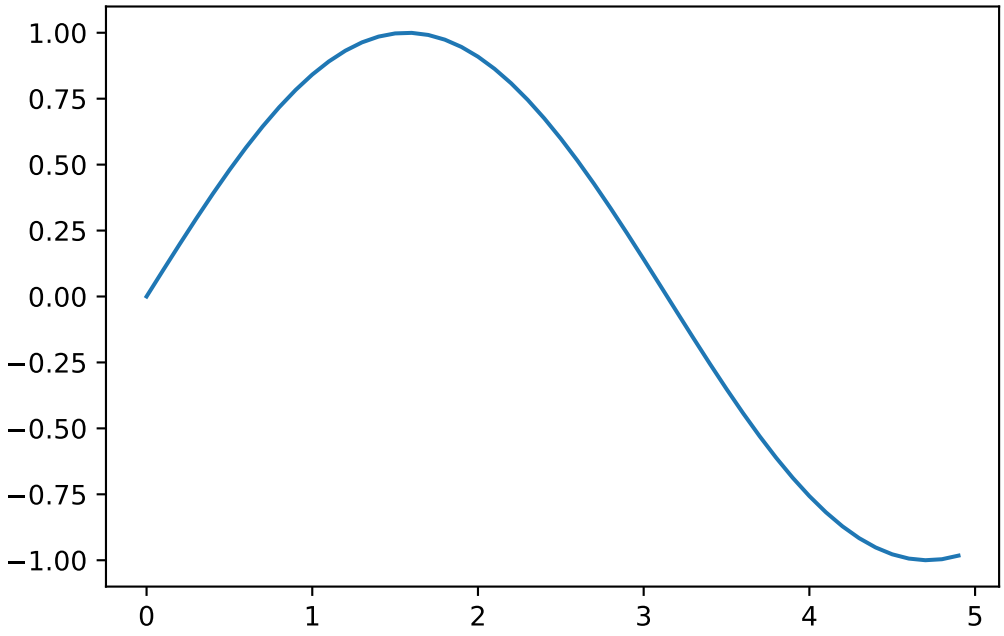
- Nom del fitxer
- Format del fitxer (PDF, JPG, PS...)
- Resolució, expressada en punts per polzada (*dots per inch* = *dpi*)

Per exemple:

```
plt.savefig('nom_fitxer', format='jpg', dpi=900)
```

```
In [9]: %matplotlib inline
        %config InlineBackend.figure_format = 'pdf'

        import matplotlib.pyplot as plt
        import numpy as np

        # Calculem valors de x en (0,5) i a partir d'ells valors de sin(x)
        x = np.arange(0, 5, 0.1);
        y = np.sin(x)
```

```
# Representem els punts
plt.plot(x, y)
plt.savefig('sinus.png', format='png', dpi=300)
plt.show()
```



La gràfica resultant es mostra a la figura 2.16.

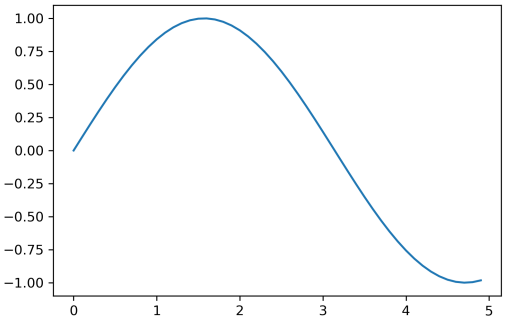
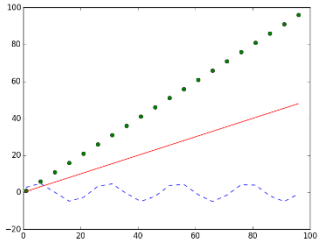
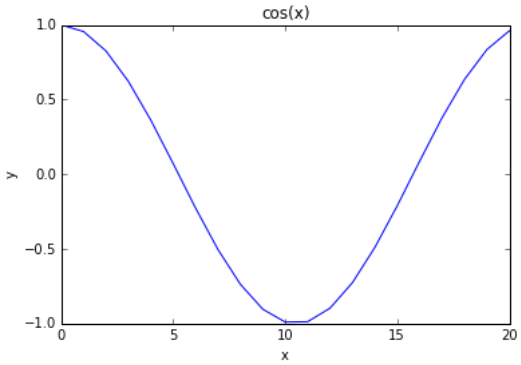


Figura 2.16: Imatge resultant en format PNG.

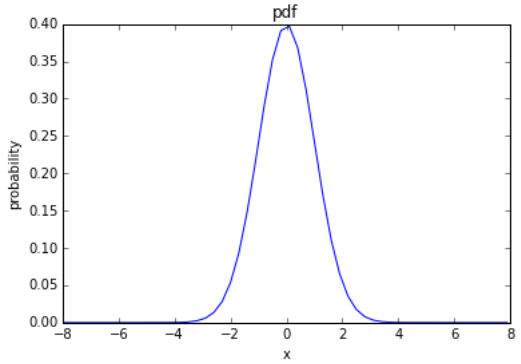
Exercici 2.14.1 *Genera un array 1D de valors en el rang (1,100,5) i representa en un gràfic conjunt: els seus valors amb punts de color verd, el triple del cosinus amb una línia a traços de color blau i la meitat del valor dels seus elements amb una línia contínua de color vermell.*



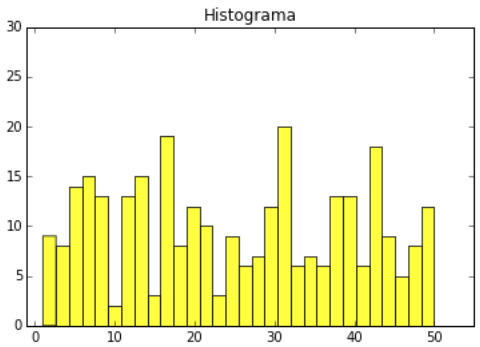
Exercici 2.14.2 *Dibuixa la funció cos(x) per a x en el rang (0,2pi,0.3). Indica quins són els eixos x i y i posa-hi un títol.*



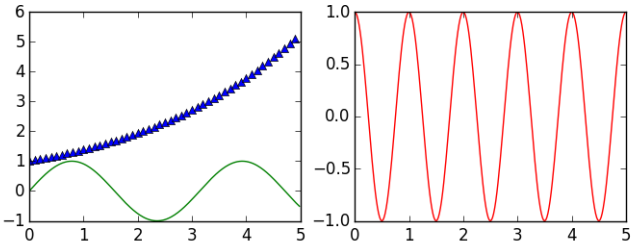
Exercici 2.14.3 *Dibuixa una distribució normal* $(\mu = 0, \sigma = 1)$ *per a x en el rang (-8,8,0.3). Indica quins són els eixos x i y, i posa-hi un títol.*



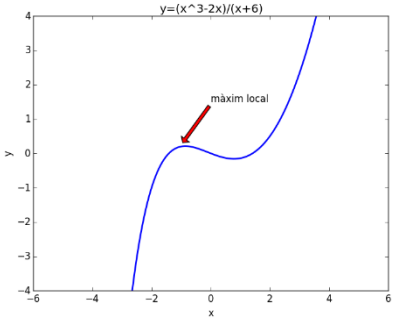
Exercici 2.14.4 *Dibuixa l'histograma de 300 nombres de valors elegits aleatòriament en el rang (1,50); l'histograma ha de tenir: a/ 30 intervals (bins), b/ ha de ser de color groc, c/ l'eix x ha d'anar de -1 a 55 i l'eix y de 0 a 30 i d/ ha de portar el títol: Histograma.*



Exercici 2.14.5 *Dibuixa dos gràfics un al costat de l'altre. El primer ha de contenir superposades les gràfiques de les funcions* $y = exp(x/3)$ *de color blau traçada a punts i* $y = sin(2x)$ *de color verd amb línia contínua; el segon, la funció* $y = cos(2 * pi * x)$ *de color vermell i línia contínua.*

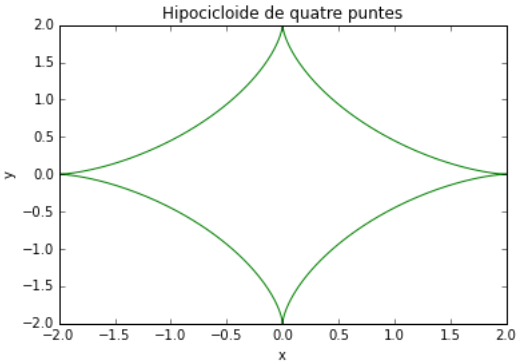


Exercici 2.14.6 *Representa la funció* $y = (x^3 - 2x)/(x + 6)$*, posa-la com a títol, assenyala quins són el eixos x i y i mitjançant una fletxa indica on és el màxim local. El rang de x és (-5, +5, 0.02) i els límits de y en el gràfic han de ser (-4,4).*

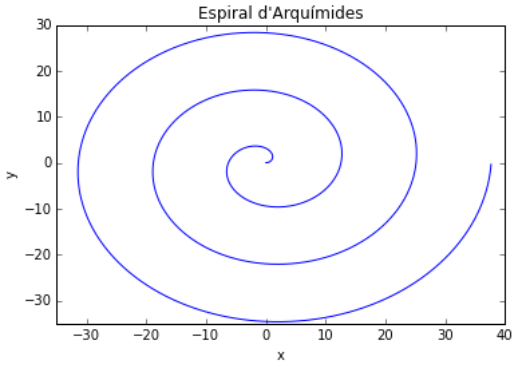


Exercici 2.14.7 *Representa un hipocicloide de quatre puntes de color verd en una circumferència de radi* $r = 2$*. Has de posar-hi un títol i indicar el nom dels eixos. Equació paramètrica:*

$$x = a(cos(b))^3$$
$$y = a(sin(b))^3$$



Exercici 2.14.8 *Representa una espiral d'Arquimedes indicant els eixos, amb títol i el rang de valors x, y en el gràfic; tipus de línia i el color. Equació polar r = a.theta, pren a = 2.*



2.15 Biblioteca SymPy

La biblioteca **SymPy** proporciona eines per al **càlcul simbòlic**. En altres paraules, permet fer amb l'ordinador manipulacions de símbols algebraics (fórmules, equacions) de forma similar a com ho fan els humans. Amb **SymPy** podrem:

- resoldre equacions,
- derivar funcions,
- integrar funcions,
- ...

Amb **SymPy**, les variables de Python no fan referència a valors numèrics sinó a funcions i ens permeten operar amb elles. Pot trobar-se una descripció completa de SymPy i les seves capacitats al tutorial de Sympy.

SymPy dona a Python una funcionalitat similar a

Per usar la biblioteca cal fer la importació següent:

```
import sympy
```

Tot i que és preferible fer-ho amb l'àlies **sp**:

```
import sympy as sp
```

2.15.1 Funcionalitat bàsica: símbols i expressions

El primer pas per utilitzar SymPy és definir els símbols que volem manipular. Cal identificar algunes variables de Python com a objectes que representen símbols (o variables en el sentit matemàtic).

Per fer-ho es poden fer servir les funcions de **SymPy**:

- `var("x")` defineix un únic símbol.
- `symbols("x y z")` defineix com a símbols les paraules contingudes a la cadena.

Vegem-ho en un exemple:

```
In [1]: import sympy as sp
        sp.init_printing()

        # Definició individual. Fem que x sigui un símbol
        sp.var("x")

        # Definició múltiple. Fem que a, b i c siguin símbols
        a,b,c = sp.symbols("a b c")


        # A partir d'aquests símbols definim una equació de segon grau
        # Nota que assignem la combinació de símbols a una nova variable
        equacio = a*x**2 + b*x + c

        # La resolem simbòlicament amb SymPy
        equacio, sp.solve(equacio,x)
```

```
Out[1]:
```

$$\left(ax^2 + bx + c, \quad \left[\frac{-b + \sqrt{-4ac + b^2}}{2a}, \quad -\frac{b + \sqrt{-4ac + b^2}}{2a}\right]\right)$$

En els dos casos, el que Phython està fent és:

1. Crear un objecte de tipus **Symbol** associat al nom que li donem.
2. Associar l'objecte creat a una variable.

Hi ha una petita diferència entre **var()** i **symbols()**. **var()** crea d'una manera automàtica una variable de Python amb el mateix nom i **symbols()** només crea els objectes que hem d'assignar necessàriament a una variable o més per poder utilitzar-les.

Podem fer que el nom de la variable no coincideixi amb el nom del símbol, però cal anar amb compte perquè això pot portar a confusions.

```
In [2]: y, z = sp.symbols('z y')
        (z, y)
```

Out[2]:

$$(y, \quad z)$$

Nota com en imprimir `(x, y)` Python ens mostra els noms dels símbols que representen. En aquest cas amb `y, x = sp.symbols('x y')` hem fet que la variable `x` representi el símbol de nom **y** (o variable matemàtica **y**) i la variable `y` el símbol de nom **x** (o variable matemàtica **x**).

Una vegada que hem definit els nostres símbols els podem combinar mitjançant els operadors matemàtics habituals per definir expressions algebraiques:

```
equacio = a*x**2 + b*x + c
```

Finalment, podem fer que **SymPy** resolgui l'equació que resulta en igualar aquesta expressió a zero.

```
sp.solve(equacio)
```

Sense més arguments, la funció **solve()** considera l'equació resultant d'igualar a zero el seu argument i tractarà de trobar-hi la solució. Si **SymPy** és capaç de trobar la solució (o solucions) ens tornarà el resultat en forma de llista.

```
In [3]: import sympy as sp
        sp.init_printing()

        # Associem a la variable "incognita" el símbol "x"
        incognita = sp.symbols("x")

        # Creem una equació usant la variable.
        # Veurem que es genera amb el símbol "x"
        equacio = incognita**2 - 1
        equacio
```

Out[3]:

$$x^2 - 1$$

```
In [4]: sp.solve(equacio)
```

Out[4]:

$$[-1, \quad 1]$$

Nota. La funció **sp.init_printing()** configura **SymPy** perquè la impressió dels símbols i de les expressions es faci de la forma més eficient possible en l'entorn on s'executa Python. En l'entorn Notebook, **SymPy** utilitza el format LaTeXper a un millor resultat.

2.15.1.1 Creant expressions a partir de cadenes de text

Una altra possibilitat que ofereix **SymPy** és la creació d'expressions algebraiques directament a partir d'una cadena de text. En aquest cas es creen els símbols necessaris i es construeix l'expressió en un sol pas, de manera semblant al que fem amb **var()**.

Per fer-ho, cal usar la funció **sympify()**.

```
In [5]: import sympy as sp
        sp.init_printing()

        sp.var("x")

        # Creem una expressió, una equació de segon grau
        equacio = sp.sympify("A*x**2+B*x+C")

        # Exemple 1
        equacio, sp.solve(equacio,x)
```

Out[5]:

$$\left(Ax^2 + Bx + C, \quad \left[\frac{-B + \sqrt{-4AC + B^2}}{2A}, \quad -\frac{B + \sqrt{-4AC + B^2}}{2A}\right]\right)$$

En aquest cas la nostra equació $Ax^2 + Bx + C = 0$ té diverses variables (els símbols A, B, C i x). Per tant, hem d'indicar a **SymPy** quina d'aquestes variables s'ha de considerar com a constant. Per això hem utilitzat la funció **solve()** amb un segon argument, **x**. És possible no fer-ho i deixar que **SymPy** en seleccioni alguna.

```
In [6]: # Exemple 2
        sp.solve(equacio)
```

Out[6]:

$$\left[\left\{A : -\frac{Bx + C}{x^2}\right\}\right]$$

En aquest cas, el símbol s'indica fent servir la clau d'un diccionari.

Una altra opció és indicar-ho directament.

```
In [7]: # Exemple 3

        sp.var('A')
        sp.var('B')
        sp.var('C')
```

```
solucioA = sp.solve(equacio, A)
solucioB = sp.solve(equacio, B)
solucioC = sp.solve(equacio, C)
{ A: solucioA, B: solucioB, C: solucioC }
```

Out[7]:

$$\left\{A : \left[-\frac{Bx+C}{x^2}\right], \quad B : \left[-Ax - \frac{C}{x}\right], \quad C : \left[-x\left(Ax+B\right)\right]\right\}$$

Cal destacar que **SymPy** no necessita que les variables `A`, `B` i `C` estiguin definides per poder resoldre l'equació (exemples 1 i 2). Només ens fa falta definir-les de forma explícita quan volem indicar-li que les utilitzi com a variables independents per resoldre la nostra equació (exemple 3). Per això necessitem variables de Python que facin referència als símbols corresponents:

```
sp.var('A')
sp.var('B')
sp.var('C')
```

Si volem fer servir lletres gregues a les nostres expressions, cal que creem els símbols utilitzant els seus noms en anglès. De la mateixa manera **Sympy** i `sympify()` reconeixen la major part de les funcions matemàtiques habituals.

```
In [8]: sp.var('theta')        # Crea un símbol i l'assigna a una variable
                               # amb el mateix nom
        sp.sin( theta )
```

Out[8]:

$$\sin\left(\theta\right)$$

```
In [9]: expressio = sp.sympify('A*cos(phi)')
        expressio
```

Out[9]:

$$A\cos\left(\phi\right)$$

SymPy representa internament les expressions de la manera següent:

```
In [10]: expressio.args
```

Out[10]:

$$\left(A, \quad \cos\left(\phi\right)\right)$$

```
In [11]: expressio.args[1].args
```

Out[11]:

$$\left(\phi\right)$$

```
In [12]: print( expressio, type(expressio) )
         print( expressio.args[0], type(expressio.args[0]))
         print( expressio.args[1], type(expressio.args[1]))
         print( expressio.args[1].args[0], type(expressio.args[1].args[0]))
```

```
A*cos(phi) <class 'sympy.core.mul.Mul'>
A <class 'sympy.core.symbol.Symbol'>
cos(phi) cos
phi <class 'sympy.core.symbol.Symbol'>
```

Nota. Quan fem servir la funció **print()** per mostrar un símbol o una expressió, Python ens mostra la seva representació com a cadena de text, sense fer servir la notació LaTeX.

2.15.1.2 Com representar gràficament una expressió

Considerem l'equació d'una recta:

```
In [13]: import sympy as sp

         sp.var('x')
         sp.var('a')
         sp.var('b')

         y = a*x + b
         y
```

Out[13]:

$$ax + b$$

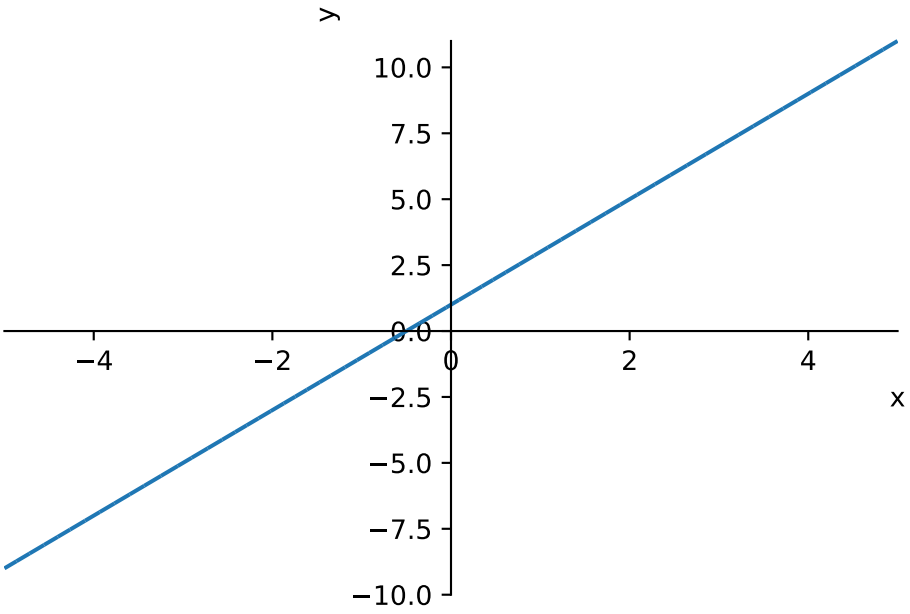
Podem assignar valors als símbols a i b i representar gràficament el resultat

```
In [14]: y_plot = y.subs(a, 2).subs(b, 1)
         y_plot
```

Out[14]:

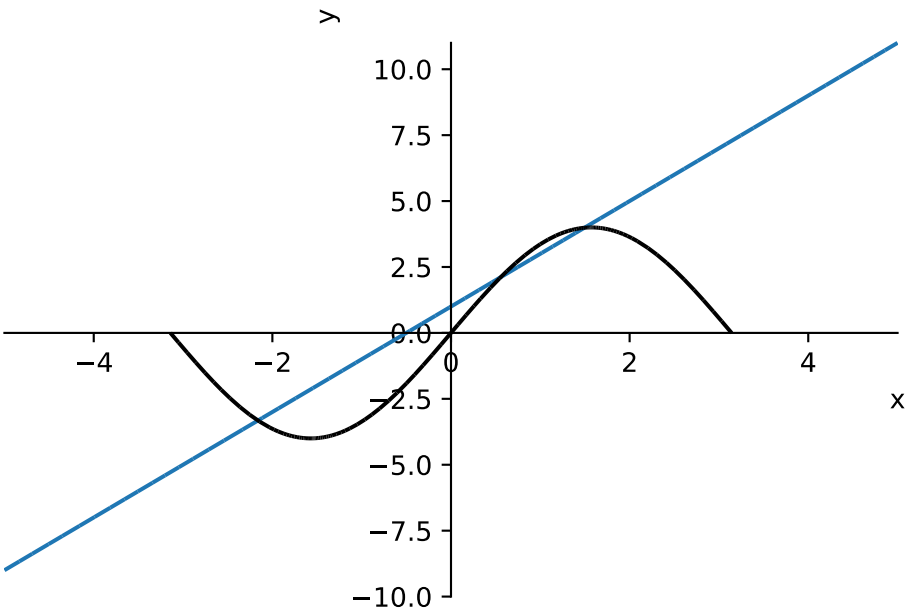
$$2x + 1$$

```
In [15]: %matplotlib inline
         %config InlineBackend.figure_format = 'pdf'

         import sympy.plotting as symplot
         # Per obtenir la gràfica en el mateix Notebook
         drawing = symplot.plot(y_plot, (x, -5, 5), xlabel='x', ylabel='y')
         drawing.show()
```

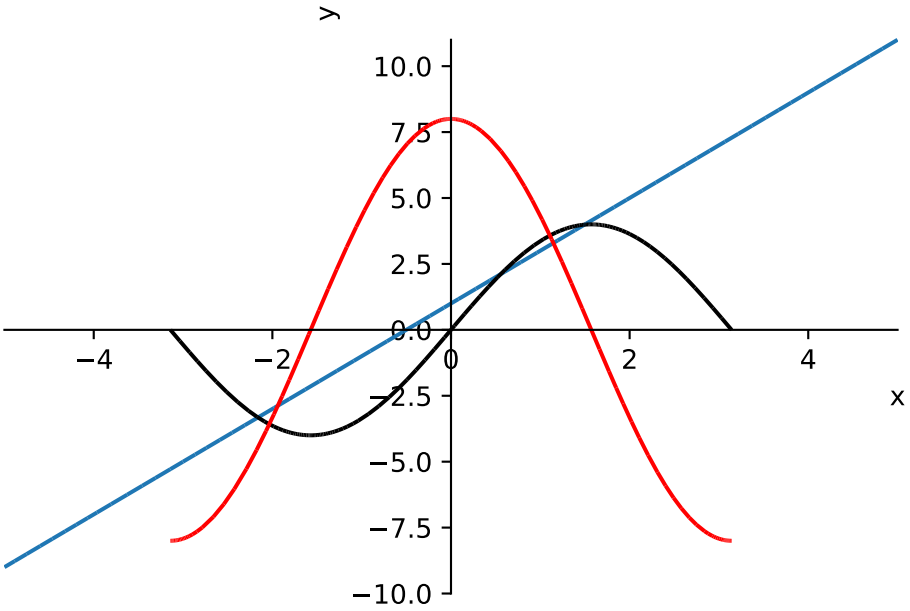


Ara podem afegir més funcions a la mateixa gràfica.

```
In [16]: %matplotlib inline
         %config InlineBackend.figure_format = 'pdf'

         y_sin = 4*sp.sin(x)
         y_cos = 8*sp.cos(x)
         drawing.extend( symplot.plot( y_sin, (x,-sp.pi,sp.pi),
                                       line_color='black') )
         drawing.show()
```



```
In [17]: %matplotlib inline
         %config InlineBackend.figure_format = 'pdf'

         drawing.extend( symplot.plot( y_cos, (x,-sp.pi,sp.pi), line_color='red', show=False
         drawing.show()
```



2.15.2 Manipulant expressions simbòliques

En els exemples anteriors ja hem vist alguna manipulació simbòlica amb **SymPy**. Hem definit quatre símbols i amb ells hem construït una equació de segon grau, que després hem resolt algebraicament.

Usant **SymPy** podem realitzar una gran varietat de manipulacions simbòliques. Aquí en veurem alguns exemples, però remet-te al tutorial de **SymPy** per a una descripció més completa.

2.15.2.1 Simplificant expressions

La funció `simplify()` permet la simplificació d'expressions algebraiques. Aplica diverses tècniques per intentar reduir l'expressió donada a una forma més senzilla. Pots trobar més detalls sobre la simplificació a Sympy tutorial: simplification.

Podem veure algunes aplicacions en els exemples següents.

```
In [18]: import sympy as sp

         # Simplificació usant igualtats trigonomètriques
         expr = sp.sympify("sin(x)**2 - cos(x)**2")
         print("Expressió: ", expr)
         print("Simplificació: ", sp.trigsimp(expr))
         print("Simplificació: ", sp.simplify(expr))
         print(10*'-')
```

```
        # Simplificació de quocients de polinomis
        expr = sp.sympify("(x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)")
        print("Expressió: ", expr)
        print("Simplificació: ", sp.ratsimp(expr))
        print("Simplificació: ", sp.simplify(expr))
        print(10*'-')

        # Simplificació usant propietats de funcions
        expr = sp.sympify("gamma(x)/gamma(x - 2)")
        print("Expressió: ", expr)
        print("Simplificació: ", sp.combsimp(expr))
        print("Simplificació: ", sp.simplify(expr))
```

```
Expressió:  sin(x)**2 - cos(x)**2
Simplificació:  -cos(2*x)
Simplificació:  -cos(2*x)
----------
Expressió:  (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
Simplificació:  x - 1
Simplificació:  x - 1
----------
Expressió:  gamma(x)/gamma(x - 2)
Simplificació:  (x - 2)*(x - 1)
Simplificació:  (x - 2)*(x - 1)
```

Nota que a més de **simplify()** hem fet servir **trigsimp()**, **ratsimp()** o **combsimp()**. Quan utilitzem directament **simplify()** Python tractarà d'utilitzar el mètode més apropiat per a l'expressió.

A continuació veurem algunes aplicacions de les tècniques de simplificació en el cas de polinomis.

2.15.2.2 Factorització de polinomis

La factorització de polinomis es pot fer usant la funció **factor()**:

```
In [19]: import sympy as sp

        # Factorització d'un polinomi
        p = sp.sympify("x**2 + 2*x + 1")
        print("Polinomi: ", p)
        print("Factorització: ", sp.factor(p))

        print(10*'-')
        # També funciona amb polinomis de més d'una variable
        p = sp.sympify("x**2*z + 4*x*y*z + 4*y**2*z")
        print("Polinomi: ", p)
        print("Factorització: ", sp.factor(p))
```

```
Polinomi:  x**2 + 2*x + 1
Factorització:  (x + 1)**2
----------
Polinomi:  x**2*z + 4*x*y*z + 4*y**2*z
Factorització:  z*(x + 2*y)**2
```

Per a polinomis més complexos, amb diverses variables, és possible indicar l'ordre de precedència dels símbols per a la factorització:

```
In [20]: sp.var('x,y,z,t')
         p = sp.sympify("x**2*z + 4*x*y*z + 4*y**2*z + x*y + t*z*x")
         print( "Polinomi: ", p )
         print( "Factorització: ", sp.factor(p) )
         print( "Factorització (x, t): ", sp.factor(p, [x, t]) )
         print( "Factorització (y, t): ", sp.factor(p, [y, t]) )
         print( "Factorització (z, t): ", sp.factor(p, [z, t]) )
```

```
Polinomi:  t*x*z + x**2*z + 4*x*y*z + x*y + 4*y**2*z
Factorització:  t*x*z + x**2*z + 4*x*y*z + x*y + 4*y**2*z
Factorització (x, t):  t*x*z + x**2*z + x*(4*y*z + y) + 4*y**2*z
Factorització (y, t):  t*x*z + x**2*z + 4*y**2*z + y*(4*x*z + x)
Factorització (z, t):  t*x*z + x*y + z*(x**2 + 4*x*y + 4*y**2)
```

2.15.2.3 Expansió de polinomis

També podem fer el procés invers a la factorització. Podem convertir un producte de polinomis en el polinomi equivalent usant la funció **expand()**:

```
In [21]: import sympy as sp

         sp.var("x")

         # Expansió d'un producte de polinomis
         p = sp.sympify("(x + 1)**2")
         print("Producte: ", p)
         print("Polinomi: ", sp.expand(p))

         print(10*'-')
         # En alguns casos expand pot simplificar l'expressió si es cancel·len termes
         p = sp.sympify("(x + 1)*(x - 2) - (x - 1)*x")
         print("Producte: ", p)
         print("Polinomi: ", sp.expand(p))
```

```
Producte:  (x + 1)**2
Polinomi:  x**2 + 2*x + 1
----------
Producte:  -x*(x - 1) + (x - 2)*(x + 1)
Polinomi:  -2
```

2.15.3 Substitució de símbols: subs()

Quan tenim una expressió algebraica podem substituir els símbols que la componen per altres símbols o per valors numèrics usant la funció **subs()**.

```
In [22]: import sympy as sp

         # Definim un polinomi i substituïm la x per cos(x)
```

```
        p = sp.sympify("(x + 1)**2")
        print("p = ", p)
        print("p subs. = ", p.subs(x,sp.cos(x)))

        print(10*'-')
        # També podem fer una substitució per un valor numèric
        q = sp.sympify("y*(x + 1)**2")
        print("q = ", q)
        print("q subs. = ", q.subs(x,3.))
```

```
p =  (x + 1)**2
p subs. =  (cos(x) + 1)**2
----------
q =  y*(x + 1)**2
q subs. =  16.0*y
```

2.15.4 Avaluació numèrica d'expressions: `evalf()`

Donada una expressió algebraica podem avaluar-la donant valors numèrics als seus símbols usant la funció **`evalf()`**:

```
In [23]: import sympy as sp

         # Definim una expressió i l'avaluem per a un valor concret de x
         sp.var("x")
         expr = 2*x**2+2*x-1
         print("Expressió: ", expr)
         print("Valor per a "x" = 2: ", expr.evalf(subs={x:2}))
```

```
Expressió:  2*x**2 + 2*x - 1
Valor per a x = 2:  11.0000000000000
```

```
In [24]: # També es pot fer si hi ha més d'un símbol
         sp.var("y")
         expr = y + 2.*x**2
         print("Expressió: ", expr)
         print("Valor per a x = 2 y = 5: ",expr.evalf(subs={x:2,y:5}))
```

```
Expressió:  2.0*x**2 + y
Valor per a x = 2 y = 5:  13.0000000000000
```

La funció **`evalf()`** permet a més que s'especifiqui el nombre de xifres significatives del càlcul, ja que **SymPy** admet càlculs en coma flotant de precisió arbitrària.

```
In [25]: import sympy as sp

         # Avaluació d'arrels quadrades
         sp.var("x")
         expr = sp.sqrt(x)
         print("Expressió: ", expr)
         print(10*'-')
         print("Avaluació sqrt(2) amb 100 xifres: ", expr.evalf(100,subs={x:2}))
```

```
        print(10*'-')
        print("Avaluació sqrt(3) amb 100 xifres: ", expr.evalf(100,subs={x:3}))
```

```
Expressió:  sqrt(x)
----------
Avaluació sqrt(2) amb 100 xifres:  1.414213562373095048801688724209698078569671
    875376948073176679737990732478462107038850387534327641573
----------
Avaluació sqrt(3) amb 100 xifres:  1.732050807568877293527446341505872366942801
    525381038062805580697945193301690880003708114618675724857 6
```

```
In [26]: print(type(expr.evalf(100,subs={x:3})))
```

```
<class 'sympy.core.numbers.Float'>
```

Nota que podem usar **evalf()** per avaluar constants matemàtiques com π o e usant els objectes **SymPy** que representen aquests valors.

```
In [27]: import sympy as sp

        # Podem usar l'objecte SymPy que representa pi per obtenir
        # un nombre arbitrari de decimals de pi
        print("pi =", sp.pi.evalf(1000))

        print(10*'-')
        # El mateix per a e
        print("e =", sp.exp(1).evalf(1000))
```

```
pi = 3.141592653589793238462643383279502884197169399375105820974944592307816
    406286208998628034825342117067982148086513282306647093844609550582231725
    359408128481117450284102701938521105559644622948954930381964428810975665
    933446128475648233786783165271201909145648566923460348610454326648213393
    607260249141273724587006606315588174881520920962829254091715364367892590
    360011330530548820466521384146951941511609433057270365759591953092186117
    381932611793105118548074462379962749567351885752724891227938183011949129
    833673362440656643086021394946395224737190702179860943702770539217176293
    176752384674818467669405132000568127145263560827785771342757789609173637
    178721468440901224953430146549585371050792279689258923542019956112129021
    960864034418159813629774771309960518707211349999983729780499510597317328
    160963185950244594553469083026425223082533446850352619311881710100031378
    387528865875332083814206171776691473035982534904287554687311595628638823
    537875937519577818577805321712268066130019278766111959092164201 99
```

```
----------
e = 2.718281828459045235360287471352662497757247093699959574966967627724076
    630353547594571382178525166427427466391932003059921817413596629043572900
    334295260595630738132328627943490763233829880753195251019011573834187930
    702154089149934884167509244761460668082264800168477411853742345442437107
    539077744992069551702761838606261331384583000752044933826560297606737113
    200709328709127443747047230696977209310141692836819025515108657463772111
    252389784425056953696770785449969967946864454905987931636889230098793127
    736178215424999229576351482208269895193668033182528869398496465 10
```

```
582093923982948879332036250944311730123819706841614039701983767932068328237646480429531180232878250981945581530175671736133206981125099618188159304169035159888851934580727386673858942287922849989208680582574927961048419844436346324496848756023362482704197862320900216099023530436994184914631409343173814364054625315209618369088870701676839642437814059271456354906130310720851038375051011574770417189861068739696552126715468895703503 5
```

```
In [28]: help(sp.evalf)
```

Ens dona tota la informació de la funció.

Exercici 2.15.1 *Amb SymPy podem utilitzar el llenguatge simbòlic. Per tant, utilitzant la biblioteca SymPy:*

a) Imprimeix la lletra x.

b) Imprimeix (a, b).

c) Imprimeix el valor de $2^3 + 5$.

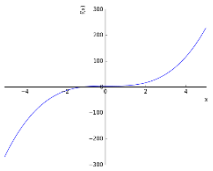
d) Calcula el cos(a) per a a = 1,1.

Exercici 2.15.2 *Donada la funció* $f = x^2 - 3exp(y)$*:*

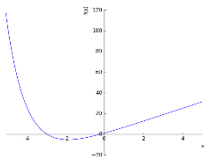
a) Imprimex-la tal com està escrita substituint la x per 3 i la y per 2.

b) Dona el seu valor amb set dígits.

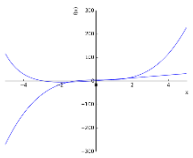
Exercici 2.15.3 *Representa gràficament la funció* $f = 2x^3 - x^2 + 4$ *a l'interval (-5,5). Indica fent un* `print` *la funció a representar.*



Exercici 2.15.4 *Repeteix l'exercici anterior per a la funció* $f = e^{-x} + 2\pi x$.



Exercici 2.15.5 *Superposa els dos gràfics anteriors.*



Exercici 2.15.6 *Expandeix la funció* $sin(a + b)$.

Exercici 2.15.7 *Repeteix l'exercici per a la funció* $(x - 1)^3$.

Exercici 2.15.8 *Factoritza la funció a=* $x^4 + 4(x^3)y + 6(x^2)(y^2) + 4x(y^3) + y^4$.

Exercici 2.15.9 *Desenvolupa en sèrie sin(n), per a n = 1 donant 9 termes.*

Exercici 2.15.10 *Donada la sèrie 1/n, n nombre natural, suma els seus primers 8 termes.*

2.15.5 Resolució d'equacions: `solve()`

SymPy implementa la resolució algebraica d'equacions mitjançant la funció **`solve()`**.

Nota. La funció **`solve()`** assumeix per defecte que l'expressió que rep s'iguala a zero per plantejar l'equació.

Exemple 1: equació polinòmica

$$x^3 + 2x - 2 = 0$$

```
In [29]: import sympy as sp
         sp.init_printing()

         # Definim el polinomi
         p = sp.sympify("x**3 + 2*x - 2")

         # En resolem
         solucions = sp.solve(p, x)
         print("Polinomi: ", p)
         print("Solucions:  ", solucions)
         solucions
```

```
Polinomi:  x**3 + 2*x - 2
Solucions:   [(-1/2 - sqrt(3)*I/2)*(1 + sqrt(105)/9)**(1/3) - 2/(3*(-1/2 - sqrt(3)*I/2)*(1 +
```

Out[29]:

$$\left[\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{1+\frac{\sqrt{105}}{9}}-\frac{2}{3\left(-\frac{1}{2}-\frac{\sqrt{3}i}{2}\right)\sqrt[3]{1+\frac{\sqrt{105}}{9}}},\right.$$
$$-\frac{2}{3\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{1+\frac{\sqrt{105}}{9}}}+\left(-\frac{1}{2}+\frac{\sqrt{3}i}{2}\right)\sqrt[3]{1+\frac{\sqrt{105}}{9}},$$
$$\left.-\frac{2}{3\sqrt[3]{1+\frac{\sqrt{105}}{9}}}+\sqrt[3]{1+\frac{\sqrt{105}}{9}}\right]$$

A vegades pot ser útil fer servir la funció **`simplify()`**:

```
In [30]: [solucio.simplify() for solucio in solucions]
```

Out[30]:

$$\left[\frac{\sqrt[3]{3}\left(8\sqrt[3]{3}-\left(1+\sqrt{3}i\right)^2\left(9+\sqrt{105}\right)^{\frac{2}{3}}\right)}{6\left(1+\sqrt{3}i\right)\sqrt[3]{9+\sqrt{105}}},\right.$$
$$\frac{\sqrt[3]{3}\left(8\sqrt[3]{3}-\left(1-\sqrt{3}i\right)^2\left(9+\sqrt{105}\right)^{\frac{2}{3}}\right)}{6\left(1-\sqrt{3}i\right)\sqrt[3]{9+\sqrt{105}}},$$
$$\left.\frac{\sqrt[3]{3}\left(-2\sqrt[3]{3}+\left(9+\sqrt{105}\right)^{\frac{2}{3}}\right)}{3\sqrt[3]{9+\sqrt{105}}}\right]$$

També podem fer servir **evalf()** per obtenir un resultat numèric.

In [31]:
```
[solucio.evalf() for solucio in solucions]
```

Out[31]:

$$[-0.385458498529624 - 1.56388451052696i,$$
$$-0.385458498529624 + 1.56388451052696i,$$
$$0.770916997059248]$$

Exemple 2: equació trigonomètrica

$$\cos(x) + \sin(x) = 0$$

In [32]:
```
import sympy as sp
sp.init_printing()

# Definim l'equació trigonomètrica
e = sp.sympify("cos(x)+sin(x)")

# El resolem
solucio = sp.solve(e, x)
print("Expressió: ",e )
print("Solució: ", solucio)
solucio
```

```
Expressió:  sin(x) + cos(x)
Solució:  [-pi/4, 3*pi/4]
```

Out[32]:

$$\left[-\frac{\pi}{4}, \quad \frac{3\pi}{4}\right]$$

Exemple 3: sistema d'equacions

Si es vol resoldre un sistema d'equacions (lineal o no), les diverses equacions s'han de passar com a arguments a la funció **solve()** en forma de llista, tant les equacions com els símbols a resoldre.

```
In [33]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         sp.var("y")

         # Resolem dos sistemes: un de lineal i l'altre no lineal
         sol1 = sp.solve([x + y - 3, x - y - 1], [x, y])
         sol2 = sp.solve([sp.sin(x) + y , sp.cos(x) - y - 1], [x, y])
         sol1,sol2
```

Out[33]:

$$\left(\{x : 2, \quad y : 1\}, \quad \left[(0, \quad 0), \quad \left(\frac{\pi}{2}, \quad -1\right), \quad \left(\frac{\pi}{2}, \quad -1\right)\right]\right)$$

Les equacions suportades actualment per **SymPy** són:

- polinomis d'una variable,
- transcendentals,
- combinacions a trossos de les anteriors,
- sistemes d'equacions polinòmiques lineals i
- sistemes que continguin expressions relacionals.

Nota. Quan `solve()` retorna `[]` o un `NotImplementedError` no vol dir que l'equació no tingui solució. Solament vol dir que no ha pogut trobar-ne cap. Freqüentment això vol dir que la solució no es pot representar d'una manera simbòlica. Per exemple, l'equació $x = cos(x)$ té solució, però no es pot representar simbòlicament mitjançant funcions estàndards.

```
In [34]: import sympy as sp
         sp.init_printing()

         sp.var('x')

         try:
             sp.solve(x - sp.cos(x), 'x')
         except Exception as excepcio:
             print(type(excepcio))
             print(excepcio)
```

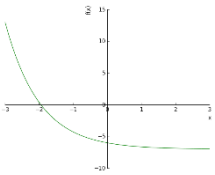
```
<class 'NotImplementedError'>
multiple generators [x, cos(x)]
No algorithms are implemented to solve equation x - cos(x)
```

Exercici 2.15.11 *Resol:*

a) L'equació polinòmica $a = 2x^3 - x^2 + 4$.

b) L'equació no lineal $b = e^{-x} + 2\pi x$.

Exercici 2.15.12 *Repeteix l'exercici anterior per a la funció* $f = e^{-x} - 7$. *Representa-la gràficament a l'interval (-3, 3) i dibuixa'n la línia de color verd.*



Exercici 2.15.13 *Resol el sistema* $x + y + z = 2$, $3x - y + z = 0$, $x + y - z = 4$.

Exercici 2.15.14 *Resol el sistema* $sin(x) + y = 2$, $cos(x) + y^2 = 1$.

Exercici 2.15.15 *Simplifica* $a = x^3 + 5xy^2 - 5x^3 + y - 3xy^2 + y^3 + 6x^3$.

2.15.6 Àlgebra lineal amb SymPy

SymPy implementa operacions d'àlgebra lineal, és a dir, operacions amb vectors i matrius. De forma similar a NumPy inclou una classe específica per representar una matriu, la classe `matrix`, que pot incloure valors simbòlics.

2.15.7 Creació de matrius

2.15.7.1 A partir d'una llista

La funció `matrix()` pot crear una matriu a partir d'una llista, o una llista de llistes:

```
In [35]: import sympy as sp
         sp.init_printing()

         matriu = sp.Matrix([[1, -1], [3, 4], [0, 2]])
         print( 'Matriu:', matriu )
         print( 'Matriu shape:', matriu.shape )
         matriu
```

```
Matriu: Matrix([[1, -1], [3, 4], [0, 2]])
Matriu shape: (3, 2)
```

```
Out[35]:
```

$$\begin{bmatrix} 1 & -1 \\ 3 & 4 \\ 0 & 2 \end{bmatrix}$$

Si es crea un objecte `matrix` a partir d'una llista unidimensional **SymPy** ho interpreta com un vector i el representa en forma de matriu columna, per facilitar les operacions amb matrius. Com els `ndarray` de **NumPy**, els objectes de tipus `matrix` de **SymPy** es podem cambiar de forma.

```
In [36]: import sympy as sp
         sp.init_printing()

         matriu = sp.Matrix([1,2,3])
         matriu
```

Out[36]:

$$\begin{bmatrix}1\\2\\3\end{bmatrix}$$

In [37]:
```
matriu = sp.Matrix( range(9)).reshape(3,3)
matriu
```

Out[37]:

$$\begin{bmatrix}0 & 1 & 2\\3 & 4 & 5\\6 & 7 & 8\end{bmatrix}$$

Com és natural en **SymPy**, les components d'un objecte **`matrix`** poden ser símbols.

In [38]:
```
import sympy as sp
sp.init_printing()

matriu = sp.Matrix( sp.var('x y z') )
matriu
```

Out[38]:

$$\begin{bmatrix}x\\y\\z\end{bmatrix}$$

2.15.7.2 Les funcions `zeros()`, `ones()`, `eye()` i `diag()`

Com en NumPy, es poden crear matrius de les mides donades amb totes les components iguals a zero o a u, unitàries o diagonals.

In [39]:
```
import sympy as sp
sp.init_printing()

matriu0 = sp.zeros(2, 3)
matriu1 = sp.ones(3, 2)
matriuI = sp.eye(3)
matriuD = sp.diag(range(5))

matriu0, matriu1, matriuI, matriuD
```

Out[39]:

$$\left(\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\end{bmatrix}, \quad \begin{bmatrix}1 & 1\\1 & 1\\1 & 1\end{bmatrix}, \quad \begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}, \quad \begin{bmatrix}0\\1\\2\\3\\4\end{bmatrix}\right)$$

La funció **`diag()`** admet arguments de tipus **`matrix`** i les organitza de forma diagonal:

```
In [40]: import sympy as sp
         sp.init_printing()

         matriu = sp.diag(-1, sp.ones(2, 2), sp.Matrix([5, 7, 5]))
         matriu
```

Out[40]:

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 5 \\ 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 5 \end{bmatrix}$$

2.15.7.3 A partir d'una funció

Es pot crear un objecte **matrix** usant una funció. La funció ha de rebre com a paràmetres els índexs d'una component i retornar el valor de la component que correspongui a aquests índexs. Per crear la funció se n'especifiquen les mides i la funció a usar.

```
In [41]: import sympy as sp

         # Definim la funció
         def funcio(i1,i2):
             """Rep com a paràmetres els dos índexs i1 i i2 i assigna com
                a valor de la component la suma dels dos"""
             return i1+i2

         # Creem una matriu a partir de la funció
         A = sp.Matrix(4,4,funcio)
         print("Matriu i1+i2:")
         A
```

```
Matriu i1+i2:
```

Out[41]:

$$\begin{bmatrix} 0 & 1 & 2 & 3 \\ 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \end{bmatrix}$$

2.15.8 Accés als elements d'una matriu

2.15.8.1 Elements

Es pot accedir als elements d'una matriu de la forma habitual donant [fila,columna] o rangs de [files,columnes] similar a l'*slicing* de les llistes.

```
In [42]: import sympy as sp
         sp.init_printing()
```

```
matriu = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])

matriu
```

Out[42]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}$$

```
In [43]: print( 'Element 1,2:', matriu[1,2])
         matriu[0:2,0:3]
```

```
Element 1,2: 4
```

Out[43]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\end{bmatrix}$$

2.15.8.2 Files i columnes

Es pot accedir a les files i columnes d'una matriu usant els mètodes **`row()`** i **`col()`** o *slicing*. Aquests mètodes retornen un objecte **`matrix`** que conté la fila o columna requerida.

```
In [44]: import sympy as sp
         sp.init_printing()

         matriu = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])
         matriu
```

Out[44]:

$$\begin{bmatrix}1 & -1 & 0\\2 & 3 & 4\\0 & 2 & 7\end{bmatrix}$$

```
In [45]: matriu.row(1), matriu[1, :]
```

Out[45]:

$$\left(\begin{bmatrix}2 & 3 & 4\end{bmatrix}, \quad \begin{bmatrix}2 & 3 & 4\end{bmatrix}\right)$$

```
In [46]: matriu.col(2), matriu[:, 2]
```

Out[46]:

$$\left(\begin{bmatrix}0\\4\\7\end{bmatrix}, \quad \begin{bmatrix}0\\4\\7\end{bmatrix}\right)$$

Nota que retornen una matriu de la forma apropiada.

La matriu transposada també és accessible fàcilment usant el membre **`T`** de l'objecte **`matrix`**:

In [47]:
```
matriu, matriu.T
```

Out[47]:

$$\left(\begin{bmatrix}1 & -1 & 0\\ 2 & 3 & 4\\ 0 & 2 & 7\end{bmatrix}, \quad \begin{bmatrix}1 & 2 & 0\\ -1 & 3 & 2\\ 0 & 4 & 7\end{bmatrix}\right)$$

2.15.9 Operacions bàsiques

Els objectes de tipus **matrix** es poden operar usant els operadors habituals **+, -, *, /, ****.

In [48]:
```
import sympy as sp
sp.init_printing()

# Definim les matrius
M = sp.Matrix([[1, -1, 0], [2, 3, 4], [0, 2, 7]])
N = sp.Matrix([[5, 6, 2], [8, 7, 7], [5, 1, 1]])

print('M, N')
M,N
```

```
M, N
```

Out[48]:

$$\left(\begin{bmatrix}1 & -1 & 0\\ 2 & 3 & 4\\ 0 & 2 & 7\end{bmatrix}, \quad \begin{bmatrix}5 & 6 & 2\\ 8 & 7 & 7\\ 5 & 1 & 1\end{bmatrix}\right)$$

In [49]:
```
# Suma, resta
print('M + N, M - N')
M + N, M - N
```

```
M + N, M - N
```

Out[49]:

$$\left(\begin{bmatrix}6 & 5 & 2\\ 10 & 10 & 11\\ 5 & 3 & 8\end{bmatrix}, \quad \begin{bmatrix}-4 & -7 & -2\\ -6 & -4 & -3\\ -5 & 1 & 6\end{bmatrix}\right)$$

In [50]:
```
# Producte de matrius
print('M * N')
M * N
```

```
M * N
```

Out[50]:

$$\begin{bmatrix} -3 & -1 & -5 \\ 54 & 37 & 29 \\ 51 & 21 & 21 \end{bmatrix}$$

In [51]:
```
# Divisió o producte per la inversa
print('M / N, M * N ** -1')
M / N, M * N ** -1
```

```
M / N, M * N ** -1
```

Out[51]:

$$\left(\begin{bmatrix} -\frac{1}{4} & \frac{1}{108} & \frac{47}{108} \\ -\frac{1}{4} & \frac{77}{108} & -\frac{53}{108} \\ -\frac{5}{4} & \frac{55}{36} & -\frac{43}{36} \end{bmatrix}, \quad \begin{bmatrix} -\frac{1}{4} & \frac{1}{108} & \frac{47}{108} \\ -\frac{1}{4} & \frac{77}{108} & -\frac{53}{108} \\ -\frac{5}{4} & \frac{55}{36} & -\frac{43}{36} \end{bmatrix}\right)$$

In [52]:
```
# Exponenciació
print('M, M ** 2, M ** 3')
M, M ** 2, M ** 3
```

```
M, M ** 2, M ** 3
```

Out[52]:

$$\left(\begin{bmatrix} 1 & -1 & 0 \\ 2 & 3 & 4 \\ 0 & 2 & 7 \end{bmatrix}, \quad \begin{bmatrix} -1 & -4 & -4 \\ 8 & 15 & 40 \\ 4 & 20 & 57 \end{bmatrix}, \quad \begin{bmatrix} -9 & -19 & -44 \\ 38 & 117 & 340 \\ 44 & 170 & 479 \end{bmatrix}\right)$$

In [53]:
```
# Operacions amb escalars
print('M, M * 2, M / 2')
M, M * 2, M / 2
```

```
M, M * 2, M / 2
```

Out[53]:

$$\left(\begin{bmatrix} 1 & -1 & 0 \\ 2 & 3 & 4 \\ 0 & 2 & 7 \end{bmatrix}, \quad \begin{bmatrix} 2 & -2 & 0 \\ 4 & 6 & 8 \\ 0 & 4 & 14 \end{bmatrix}, \quad \begin{bmatrix} \frac{1}{2} & -\frac{1}{2} & 0 \\ 1 & \frac{3}{2} & 2 \\ 0 & 1 & \frac{7}{2} \end{bmatrix}\right)$$

Nota que **matrix** no admet operacions de suma o resta amb valors numèrics (escalars). Cal fer-ho usant la matriu unitària auxiliar **ones()**:

In [54]:
```
# Aquestes expressions donen error
# M + 2
# M - 2
print('M, M + 2, M - 2')
# Cal fer
M, M + 2* sp.ones(M.rows, M.cols), M - 2 * sp.ones(M.rows, M.cols)
```

```
M, M + 2, M - 2
```

Out[54]:

$$\left(\begin{bmatrix}1 & -1 & 0\\ 2 & 3 & 4\\ 0 & 2 & 7\end{bmatrix}, \quad \begin{bmatrix}3 & 1 & 2\\ 4 & 5 & 6\\ 2 & 4 & 9\end{bmatrix}, \quad \begin{bmatrix}-1 & -3 & -2\\ 0 & 1 & 2\\ -2 & 0 & 5\end{bmatrix}\right)$$

I, òbviament, amb **SymPy** totes aquestes operacions es poden fer amb matrius simbòliques:

In [55]:
```
import sympy as sp
sp.init_printing()

# Definim una matriu simbòlica
sp.var("x")
M = sp.Matrix([[x, 2 * x, x],
               [x * x, 3, 4],
               [x - 1, x, x ** 3]])

print("Matriu M")
M
```

```
Matriu M
```

Out[55]:

$$\begin{bmatrix}x & 2x & x\\ x^2 & 3 & 4\\ x-1 & x & x^3\end{bmatrix}$$

In [56]:
```
M ** 2
```

Out[56]:

$$\begin{bmatrix}2x^3 + x^2 + x\,(x-1) & 3x^2 + 6x & x^4 + x^2 + 8x\\ x^3 + 3x^2 + 4x - 4 & 2x^3 + 4x + 9 & 5x^3 + 12\\ x^3\,(x-1) + x^3 + x\,(x-1) & x^4 + 2x\,(x-1) + 3x & x^6 + x\,(x-1) + 4x\end{bmatrix}$$

Per defecte **SymPy** no intenta simplificar l'expressió resultant, però pot ser instruït per fer-ho utilitzant el mètode **`simplify( )`**:

In [57]:
```
N = M ** 2
N.simplify()
N
```

Out[57]:

$$\begin{bmatrix}x\,(2x^2 + 2x - 1) & 3x\,(x+2) & x\,(x^3 + x + 8)\\ x^3 + 3x^2 + 4x - 4 & 2x^3 + 4x + 9 & 5x^3 + 12\\ x\,(x^3 + x - 1) & x\,(x^3 + 2x + 1) & x\,(x^5 + x + 3)\end{bmatrix}$$

In [58]:
```
M ** -1
```

Out[58]:

No pot ser representat.

```
In [59]: N = M ** -1
         N.simplify()
         N
```

Out[59]:

$$\begin{bmatrix} \frac{-3x^2+4}{2x^5-4x^3-x+5} & \frac{2x^3-x}{2x^5-4x^3-x+5} & -\frac{5}{2x^5-4x^3-x+5} \\ \frac{x^5-4x+4}{x(2x^5-4x^3-x+5)} & \frac{-x^3+x-1}{2x^5-4x^3-x+5} & \frac{-x^2+4}{2x^5-4x^3-x+5} \\ \frac{-x^3+3x-3}{x(2x^5-4x^3-x+5)} & \frac{-x+2}{2x^5-4x^3-x+5} & \frac{2x^2-3}{2x^5-4x^3-x+5} \end{bmatrix}$$

Exercici 2.15.16 *Imprimeix:*

a) La matriu a = [[a b] [c d]].

b) Una matriu de zeros 3×3.

c) Una matriu d'uns 2×3.

d) Una matriu identitat 2×2.

e) Una matriu diagonal 3×3 amb valors en el rang (2,5).

Exercici 2.15.17 *Donada a = [[1, 3, 7], [0, 2, 5], [8, 4, 6]]:*

1. *Imprimeix la matriu corresponent.*
2. *Indica quin és el segon element de la primera fila.*

Exercici 2.15.18 *Donades a = [[1, 4, 6], [-2, 6, 1], [0, 1, 5]] i b = [[1, 3, 7], [0, 2, 5], [8, 4, 6]], expressa-les com a matrius i calcula:*

a) $a + b$,

b) $a - b$,

c) $3a$,

d) ab,

e) $\frac{a}{3}$.

Exercici 2.15.19 *Donada: a = [[x, 2, x-1],[3 * x, 0, x**2],[1, x+1, 3]]:*

a) Expressa-la com a matriu i eleva-la al quadrat.

b) Simplifica-la.

2.15.10 Operacions avançades

A més de les operacions anteriors **SymPy** proporciona també diverses operacions avançades d'àlgebra lineal.

2.15.10.1 Transposició

Es fa amb el membre `T` de qualsevol objecte de tipus **`matrix`**.

```
In [60]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         M = sp.Matrix([[x, 2 * x, 0],
```

```
                        [2, x - 1, 4],
                        [x + 1, 2, 7]])

        print("M, transposada de M")
        M, M.T
```

```
M, transposada de M
```

Out[60]:

$$\left(\begin{bmatrix} x & 2x & 0 \\ 2 & x-1 & 4 \\ x+1 & 2 & 7 \end{bmatrix}, \quad \begin{bmatrix} x & 2 & x+1 \\ 2x & x-1 & 2 \\ 0 & 4 & 7 \end{bmatrix}\right)$$

2.15.10.2 Determinants

Es calculen amb el mètode **det()**:

In [61]:
```
        import sympy as sp
        sp.init_printing()

        sp.var("x")
        M = sp.Matrix([[x, 2 * x, 0],
                       [2, x - 1, 4],
                       [x + 1, 2, 7]])

        print("M, determinant de M")
        M, M.det()
```

```
M, determinant de M
```

Out[61]:

$$\left(\begin{bmatrix} x & 2x & 0 \\ 2 & x-1 & 4 \\ x+1 & 2 & 7 \end{bmatrix}, \quad 7x\left(x-1\right) + 8x\left(x+1\right) - 36x\right)$$

2.15.10.3 Vectors generadors del nucli (kernel)

El mètode **nullspace()** permet obtenir una base (vectors generadors) del nucli (subespai vectorial que té per imatge el vector zero) de l'aplicació lineal representada per la matriu.

$$M \cdot v = 0$$

In [62]:
```
        import sympy as sp
        sp.init_printing()

        # Definim la matriu
        M = sp.Matrix([[1, 2, 3, 0, 0], [4, 10, 0, 0, 1]])

        print('M, kernel de M')
        M,M.nullspace()
```

```
M, kernel de M
```

Out[62]:

$$\left(\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 4 & 10 & 0 & 0 & 1 \end{bmatrix}, \quad \left[\begin{bmatrix} -15 \\ 6 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ -\frac{1}{2} \\ 0 \\ 0 \\ 1 \end{bmatrix} \right] \right)$$

```
In [63]: for v in M.nullspace():
             print( M * v )
```

```
Matrix([[0], [0]])
Matrix([[0], [0]])
Matrix([[0], [0]])
```

2.15.10.4 Valors i vectors propis

El mètode **eigenvals()** permet trobar els valors propis d'una matriu. Retorna un diccionari que conté els valors propis com a claus i la seva multiplicitat com a elements.

```
In [64]: import sympy as sp
         sp.init_printing()

         # Definim la matriu
         M = sp.Matrix([[3, -2, 4, -2],
                        [5, 3, -3, -2],
                        [5, -2, 2, -2],
                        [5, -2, -3, 3]])
         print('M, valors propis de M')
         M, M.eigenvals()
```

```
M, valors propis de M
```

Out[64]:

$$\left(\begin{bmatrix} 3 & -2 & 4 & -2 \\ 5 & 3 & -3 & -2 \\ 5 & -2 & 2 & -2 \\ 5 & -2 & -3 & 3 \end{bmatrix}, \quad \{-2 : 1, \quad 3 : 1, \quad 5 : 2\} \right)$$

De forma similar, el mètode **eigenvects()** permet calcular els vectors propis de la matriu. La funció retorna els valors propis, la seva multiplicitat i els vectors propis.

```
In [65]: M.eigenvects()
```

Out[65]:

$$\left[\left(-2,\quad 1,\quad \left[\begin{bmatrix}0\\1\\1\\1\end{bmatrix}\right]\right),\quad \left(3,\quad 1,\quad \left[\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\right]\right),\quad \left(5,\quad 2,\quad \left[\begin{bmatrix}1\\1\\1\\0\end{bmatrix},\quad \begin{bmatrix}0\\-1\\0\\1\end{bmatrix}\right]\right)\right]$$

2.15.10.5 Diagonalització

La funció **diagonalize()** permet diagonalitzar una matriu. Donada una matriu M, retorna una tupla (P, D) de manera que $M = PDP^{-1}$.

```
In [66]: print("M, (P,D)     Nota que la diagonal de D conté els valors propis")
         M, M.diagonalize()
```

```
M, (P,D)     Nota que la diagonal de D conté els valors propis
```

Out[66]:

$$\left(\begin{bmatrix}3 & -2 & 4 & -2\\5 & 3 & -3 & -2\\5 & -2 & 2 & -2\\5 & -2 & -3 & 3\end{bmatrix},\quad \left(\begin{bmatrix}0 & 1 & 1 & 0\\1 & 1 & 1 & -1\\1 & 1 & 1 & 0\\1 & 1 & 0 & 1\end{bmatrix},\quad \begin{bmatrix}-2 & 0 & 0 & 0\\0 & 3 & 0 & 0\\0 & 0 & 5 & 0\\0 & 0 & 0 & 5\end{bmatrix}\right)\right)$$

2.15.10.6 Resolució de sistemes d'equacions lineals

Es poden resoldre a partir d'una matriu mitjançant el mètode de descomposició LU inclòs a la classe **matrix** o bé simplement a partir de la matriu inversa del sistema.

Si considerem el sistema d'equacions (el mateix que amb **NumPy**):

$$\begin{array}{rcrcrcl} x & - & y & + & z & = & 4\\ 2x & + & y & - & 3z & = & 1\\ 7x & - & y & - & 3z & = & 14 \end{array}$$

Hi ha quatre possibilitats per resoldre-ho:

```
In [67]: import sympy as sp
         sp.init_printing()

         # Definim la matriu del sistema
         A = sp.Matrix([[1, -1, 1],
                        [2, 1, -3],
                        [7, -1, -3]])

         # Definim el vector de termes independents
         b = sp.Matrix([4, 1, 14])

         print("Sistema d'equacions")
         try:
             A,b,A.LUsolve(b),A.inv()*b
         except Exception as x:
             print(type(x))
             print(x)
```

```
Sistema d'equacions
<class 'ValueError'>
Matrix det == 0; not invertible.
```

In [68]:
```
import sympy as sp
sp.init_printing()

sp.var("x, y, z")
expre1 = x - y + z - 4
expre2 = 2*x + y - 3*z - 1
expre3 = 7*x - y - 3*z - 14

sp.solve([expre1, expre2, expre3])
```

Out[68]:

$$\left\{x : \frac{2z}{3} + \frac{5}{3}, \quad y : \frac{5z}{3} - \frac{7}{3}\right\}$$

In [69]:
```
eq1 = sp.Eq( x - y + z, 4 )
eq2 = sp.Eq( 2*x + y - 3*z, 1 )
eq3 = sp.Eq( 7*x - y - 3*z, 14 )

sp.solve([eq1, eq2, eq3])
```

Out[69]:

$$\left\{x : \frac{2z}{3} + \frac{5}{3}, \quad y : \frac{5z}{3} - \frac{7}{3}\right\}$$

In [70]:
```
M = sp.Matrix( [[1, -1, 1],
                [2, 1, -3],
                [7, -1, -3]] )
eq = sp.Eq(M * sp.Matrix([x,y,z]), sp.Matrix([4, 1, 14]) )
eq, sp.solve(eq)
```

Out[70]:

$$\left(\begin{bmatrix} x - y + z \\ 2x + y - 3z \\ 7x - y - 3z \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 14 \end{bmatrix}, \quad \left\{x : \frac{2z}{3} + \frac{5}{3}, \quad y : \frac{5z}{3} - \frac{7}{3}\right\}\right)$$

Ara una equació amb determinant no nul.

In [71]:
```
import sympy as sp
sp.init_printing()

# Definim la matriu del sistema
A = sp.Matrix([[1, -1, 0],
               [2, 3, 4],
               [0, 2, 7]])

# Definim el vector de termes independents
b = sp.Matrix([1, 1, 1])
```

```
        print("Sistema d'equacions")
        A,b,A.LUsolve(b),A.inv()*b
```

```
Sistema d'equacions
```

Out[71]:

$$\left(\begin{bmatrix}1 & -1 & 0\\ 2 & 3 & 4\\ 0 & 2 & 7\end{bmatrix}, \quad \begin{bmatrix}1\\ 1\\ 1\end{bmatrix}, \quad \begin{bmatrix}\frac{16}{27}\\ -\frac{11}{27}\\ \frac{7}{27}\end{bmatrix}, \quad \begin{bmatrix}\frac{16}{27}\\ -\frac{11}{27}\\ \frac{7}{27}\end{bmatrix}\right)$$

In [72]:
```
import sympy as sp
sp.init_printing()

sp.var("x, y, z")
expre1 =   x -   y       - 1
expre2 = 2*x + 3*y + 4*z - 1
expre3 =       2*y + 7*z - 1

sp.solve([expre1, expre2, expre3])
```

Out[72]:

$$\left\{x : \frac{16}{27}, \quad y : -\frac{11}{27}, \quad z : \frac{7}{27}\right\}$$

In [73]:
```
eq1 = sp.Eq(  x -   y      ,1)
eq2 = sp.Eq(2*x + 3*y + 4*z,1)
eq3 = sp.Eq(    + 2*y + 7*z,1)

sp.solve([eq1, eq2, eq3])
```

Out[73]:

$$\left\{x : \frac{16}{27}, \quad y : -\frac{11}{27}, \quad z : \frac{7}{27}\right\}$$

In [74]:
```
M = sp.Matrix([[1, -1, 0],
               [2, 3, 4],
               [0, 2, 7]])
eq = sp.Eq(M * sp.Matrix([x,y,z]), sp.Matrix([1, 1, 1]))
eq, sp.solve(eq )
```

Out[74]:

$$\left(\begin{bmatrix}x - y\\ 2x + 3y + 4z\\ 2y + 7z\end{bmatrix} = \begin{bmatrix}1\\ 1\\ 1\end{bmatrix}, \quad \left\{x : \frac{16}{27}, \quad y : -\frac{11}{27}, \quad z : \frac{7}{27}\right\}\right)$$

Exercici 2.15.20 *Sigui la matriu A = [[x 1 -x], [3 2x 4], [1-x 0 3]], imprimeix-la donant el valor del seu determinant i escriu la seva transposada.*

Exercici 2.15.21 *Determina: els vectors i valors propis de la matriu A = [[1 -3 3], [3 -5 3], [6 -6 4]] i dona'n la matriu diagonal corresponent.*

Exercici 2.15.22 *Escriu el polinomi característic, $|tI - A|$, de la matriu anterior i fes la factorització corresponent. Analitza el resultat.*

Exercici 2.15.23 *Resol el sistema: $x + 2y + z = 5$, $x - y + z = 3$, $2x + y - z = 0$.*

2.15.11 Integració amb SymPy

2.15.11.1 Intregrals indefinides

SymPy permet fer integració simbòlica de funcions (integrals indefinides) mitjançant la funció **integrate()**, que rep com a argument una expressió simbòlica.

```
In [75]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         print( "Integrant sin(x)*exp(x)" )
         sp.integrate(sp.sin(x)*sp.exp(x),x)
```

```
Integrant sin(x)*exp(x)
```

Out[75]:

$$\frac{e^{x}\sin\left(x\right)}{2} - \frac{e^{x}\cos\left(x\right)}{2}$$

Aquesta mateixa funció permet fer integrals múltiples:

```
In [76]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         sp.var("y")

         print( "Integrant exp(-x**2-y**2)" )
         sp.integrate(sp.exp(-x**2 - y**2),x,y)
```

```
Integrant exp(-x**2-y**2)
```

Out[76]:

$$\frac{\pi\,\mathrm{erf}\left(x\right)\mathrm{erf}\left(y\right)}{4}$$

2.15.11.2 Integrals definides

La mateixa funció **integrate()** permet també calcular integrals definides indicant els límits d'integració. Per exemple, la integral

$$\int_0^{\pi} sin(x)dx = -cos(x)|_0^{\pi} = 2$$

s'implementa com:

```
In [77]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         sp.integrate(sp.sin(x), (x, 0, sp.pi))
```

Out[77]:

$$2$$

Es pot indicar un límit d'integració infinit amb el símbol **simpy.oo**. L'exemple següent implementa la integral:

$$\int_0^\infty e^{-x}dx = -e^{-x}|_0^\infty = 1$$

```
In [78]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         sp.integrate(sp.exp(-x), (x, 0, sp.oo))
```

Out[78]:

$$1$$

De forma similar es poden calcular integrals dobles definides. Per exemple, la integral

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-x^2-y^2}dxdy = \pi$$

```
In [79]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         sp.var("y")

         sp.integrate(sp.exp(-x**2 - y**2), (x, -sp.oo, sp.oo), (y, -sp.oo, sp.oo))
```

Out[79]:

$$\pi$$

Els límits d'integració poden ser símbols, i el resultat s'expressa en funció d'ells. Per exemple

$$\int_0^a sin(x)dx = -cos(a) + 1$$

```
In [80]: import sympy as sp
         sp.init_printing()
```

```
sp.var("x")
sp.var("a")

sp.integrate(sp.sin(x), (x, 0, a))
```

Out[80]:

$$-\cos(a) + 1$$

Nota que cal definir el símbol que utilitzem com a límit d'integració.

En l'exemple següent el resultat indica dues possibilitats perquè la integral no convergeix si la part real de a no és > 1:

```
In [81]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         sp.var("a")

         sp.integrate(x**a*sp.exp(-x), (x, 0, sp.oo))
```

Out[81]:

$$\begin{cases} \Gamma(a+1) & \text{for } -\Re(a) < 1 \\ \int_0^\infty x^a e^{-x}\, dx & \text{otherwise} \end{cases}$$

2.15.11.3 Integrals com a símbols

En cas que la integral no es vulgui avaluar, i només es vulgui obtenir la funció o el valor numèric corresponent, es pot usar **`integral()`**, que retorna la integral com una expressió simbòlica de **SymPy**. Posteriorment, la integral es pot avaluar usant **`doit()`**:

Per exemple, en el cas d'integració definida:

```
In [82]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         resultat = sp.Integral(sp.exp(-x**2 - y**2),
                                (x, -sp.oo, sp.oo), (y, -sp.oo, sp.oo))
         resultat, resultat.doit()
```

Out[82]:

$$\left(\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{-x^2-y^2}\, dx\, dy, \quad \pi\right)$$

I un exemple en el cas d'integració simbòlica:

```
In [83]: import sympy as sp
         sp.init_printing()

         sp.var("x")
```

```
resultat = sp.Integral((x**4 + x**2*sp.exp(x) - x**2 -
                        2*x*sp.exp(x) - 2*x - sp.exp(x))*sp.exp(x) /
                       ((x - 1)**2*(x + 1)**2*(sp.exp(x) + 1)), x)
resultat, resultat.doit()
```

Out[83]:

$$\left(\int \frac{\left(x^4 + x^2e^x - x^2 - 2xe^x - 2x - e^x\right)e^x}{\left(x-1\right)^2\left(x+1\right)^2\left(e^x+1\right)}\,dx, \quad \log\left(e^x+1\right) + \frac{e^x}{x^2-1}\right)$$

També podem usar la funció **evalf()** per obtenir un resultat numèric amb precisió arbitrària:

```
In [84]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         resultat = sp.Integral(sp.exp(-x**2), (x, -sp.oo, sp.oo))
         print( resultat.doit().evalf(100)**2 )
         print( sp.pi.evalf(100) )
         resultat = sp.Integral(sp.exp(-x**2), (x, -1, 1))
         resultat, resultat.doit(), resultat.doit().evalf(100)
```

```
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253
3.1415926535897932384626433832795028841971693993751058209749445923078164062862089986280348253
```

Out[84]:

$$\left(\int_{-1}^{1} e^{-x^2}\,dx, \quad \sqrt{\pi}\,\mathrm{erf}\left(1\right), \quad 1.4936482656248540507989348722637060107089993736252126580553089979172107\right.$$

2.15.12 Derivació amb SymPy

2.15.12.1 Funció `diff()`

SymPy permet obtenir la derivada d'una expressió simbòlica mitjançant la funció **diff()**. La forma més senzilla d'aplicar-la és per calcular una derivada simple respecte d'una única variable:

```
In [85]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         print( "Derivant sin(x)*exp(x)" )
         sp.diff(sp.sin(x)*sp.exp(x), x)
```

```
Derivant sin(x)*exp(x)
```

Out[85]:

$$e^x \sin\left(x\right) + e^x \cos\left(x\right)$$

La funció **diff()** permet també fer derivades de graus superiors; només cal indicar el grau en cridar la funció.

```
In [86]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         print( "Derivada tercera de x**4" )
         sp.diff(x**4, x, 3)
```

```
Derivada tercera de x**4
```

Out[86]:

$$24x$$

De la mateixa manera, també permet fer derivades parcials respecte de diverses variables. Per exemple:

$$\frac{\partial}{\partial_x \partial_y \partial_z} e^{x,y,x}$$

```
In [87]: import sympy as sp
         sp.init_printing()

         sp.var("x")
         sp.var("y")
         sp.var("z")

         print( "Derivada parcial de exp(x*y*z) respecte de x, y, z" )
         sp.diff(sp.exp(x * y * z), x, y, z)
```

```
Derivada parcial de exp(x*y*z) respecte de x, y, z
```

Out[87]:

$$\left(x^2y^2z^2 + 3xyz + 1\right) e^{xyz}$$

2.15.12.2 Derivades com a símbols

En cas que la derivada no es vulgui calcular explícitament es pot usar **derivative()** que retorna la derivada com una expressió simbòlica de **SymPy**. Posteriorment, la derivada es pot calcular usant **doit()**.

```
In [88]: import sympy as sp
         sp.init_printing()

         sp.var("x")

         print( "Derivant sin(x)*exp(x)" )
```

```
D = sp.Derivative(sp.sin(x) * sp.exp(x), x, 3)
D, D.doit()
```

```
Derivant sin(x)*exp(x)
```

Out[88]:

$$\left(\frac{d^3}{dx^3}e^x \sin(x), \quad 2\left(-\sin(x)+\cos(x)\right)e^x\right)$$

Exercici 2.15.24 *Donada la funció $f = x^3 + xy - 2e^y$:*

a) Calcula la derivada respecte de y.

b) Calcula la derivada respecte de x.

Exercici 2.15.25 *Donada la funció $f = 3x^2 + e^{4x}$:*

a) Integra-la.

b) Expressa'n de forma simbòlica la integral quan hi posem límits -1, 2.

c) Dona'n el seu valor amb 12 dígits.

Exercici 2.15.26 *Donada la funció $f = 4x^7 - e^{5y} + xy$:*

a) Integra-la respecte de x.

b) Integra-la respecte de y.

c) Integra-la respecte de x a l'interval (0, 2) i de y a l'interval (2, 5). Expressa'n simbòlicament el resultat.

d) Calcula el valor d'aquesta última integral amb 8 dígits.

Exercici 2.15.27 *Calcula el límit* `(limit(fun, var, val))` *de la funció (cos(x))/x quan x tendeix a zero.*

Exercici 2.15.28 *Repeteix l'exercici per a (sin(x))/x.*

Exercici 2.15.29 *Calcula el límit de $tan(x)$ quan x tendeix a $\frac{\pi}{2}$:*

a) Per la branca dreta (dir='+').

b) Per la branca esquerra (dir='-').

Exercici 2.15.30 *Calcula la integral de tan(x) a l'interval $(0, \frac{\pi}{6})$ i la derivada primera i segona de la funció $f = \left(\frac{x+1}{x-1}\right)^{\frac{1}{2}}$.*

2.16 Gestió d'errors

Python inclou un mecanisme per gestionar errors en temps d'execució o, més acuradament, gestionar excepcions. Això permet gestionar casos en què un programa troba una situació excepcional (d'aquí el nom *excepció*) que li impediria continuar l'execució normal. Mitjançant els mecanismes de gestió d'errors es poden "gestionar" aquestes situacions i prendre mesures que permetin continuar o acabar el programa de forma ordenada.

2.16.1 Sentència `try` i `except`

La gestió s'implementa amb la sentència *try* - *except*:

```
try:
    "sentències on es poden produir errors"
except TipusError1:
    "sentències a executar en cas d'error tipus 1"
except TipusError2:
    "sentències a executar en cas d'error tipus 2"
else:
    "sentències a executar si no hi ha cap error"
```

Exemple 1. Error en l'entrada de dades per consola

Sense gestió d'errors. Dona un text no interpretable com un `float` i comprova com el programa s'interromp amb un error.

```
In [1]: valor = float(input("Dona un nombre real "))
        print("El valor donat és: ", valor)
```

```
Dona un nombre real a


        ---------------------------------------------------------------------------

        ValueError                                Traceback (most recent call last)

        <ipython-input-1-9e13625e5caf> in <module>
    ----> 1 valor = float(input("Dona un nombre real "))
          2 print("El valor donat és: ", valor)


        ValueError: could not convert string to float: 'a'
```

Amb gestió d'errors. Comprova que podem gestionar els casos en què l'usuari dona valors erronis.

```
In [2]: try:
            valor = float(input("Dona un nombre real "))
            print("El valor donat és: ", valor)
        except ValueError:
            print("Has donat un valor no interpretable com un de real!")
```

```
Dona un nombre real a
Has donat un valor no interpretable com un de real!
```

Podem millorar el codi usant la gestió d'errors per repetir l'entrada de dades fins que es doni un valor acceptable:

```
In [3]: valid = False

        while not valid:
            try:
                valor = float(input("Dona un nombre real "))
                valid= True
            except ValueError:
                print("Has donat un valor no interpretable com un de real! Torna-ho a provar

        print("El valor donat és: ", valor)
```

```
Dona un nombre real a
Has donat un valor no interpretable com un de real! Torna-ho a provar
Dona un nombre real 2.3
El valor donat és:  2.3
```

Exemple 2. Gestió d'errors múltiples

El mecanisme `try-except` permet ser específic en la gestió dels diversos tipus d'error. Es poden usar diversos blocs *except* per gestionar cada tipus d'error que es pugui produir:

```
In [4]: import math

        try:
            angle = float(input("Dona un angle en graus "))
            print("L'angle donat és: ", angle)

            invers = 1./angle

        except ValueError:
            print("Has donat un valor de l'angle invàlid")
        except ZeroDivisionError:
            print("Error: el valor de l'angle ha provocat una divisió per zero")
        else:
            print("No s'ha produït cap error")
```

```
Dona un angle en graus 0
L'angle donat és:  0.0
Error: el valor de l'angle ha provocat una divisió per zero
```

Exemple 3. Gestió d'errors de fitxer

Un ús molt comú de `try-except` és el de gestionar els errors relacionats amb l'ús de fitxers. Per exemple, quan es vol obrir un fitxer que no existeix:

```
In [5]: NomFitxer = input("Dona el nom del fitxer a obrir ")
        try:
            fitxer = open(NomFitxer,"r")
```

```
            linia1 = fitxer.read()
            print("Primera línia del fitxer: ",linia1)

            fitxer.close()

        except IOError:
            print("El fitxer", NomFitxer, "no existeix")
```

```
Dona el nom del fitxer a obrir hola.txt
El fitxer hola.txt no existeix
```

2.16.2 Sentència `finally`

Una opció alternativa per usar la sentència *try* és combinar-la amb la sentència `finally`. Serveix per assegurar-se que un cert bloc de codi s'executi encara que es produeixi un error, **fins i tot els no capturats amb `except`**.

```
In [6]: import math

        try:
            angle = float(input("Dona un angle en graus "))
            print("L'angle donat és: ", angle)

            invers = 1./angle

        except ValueError:
            print("Has donat un valor de l'angle invàlid")
        finally:
            print("Càlcul acabat")
```

```
Dona un angle en graus 0
L'angle donat és:  0.0
Càlcul acabat
```

```
        ---------------------------------------------------------------------------

        ZeroDivisionError                         Traceback (most recent call last)

        <ipython-input-6-bc8f319676e0> in <module>
          5     print("L'angle donat és: ", angle)
          6
    ----> 7     invers = 1./angle
          8
          9 except ValueError:


        ZeroDivisionError: float division by zero
```

Exercici 2.16.1 *Introdueix el nombre 3.25 per pantalla com a nombre enter i comprova que dona error.*

Exercici 2.16.2 *Repeteix l'exercici anterior fent gestió d'errors.*

Exercici 2.16.3 *Demana consecutivament dos nombres enters per pantalla i, per gestió d'errors, indica quan en un dels casos o en els dos hi pot haver error.*

Exercici 2.16.4 *Demana per pantalla un nombre real però escriu un nombre complex, utilitza la gestió d'errors, i la sentència* `finally`*, i imprimeix tot seguit: 'continuem'.*

2.17 Una aplicació per resoldre problemes

Volem fer una aplicació per resoldre equacions. Establim que farà servir fitxer per.

2.17.1 Definim el format del fitxer

La funció **solve()** de **SymPy** fa servir un conjunt d'equacions i variables. Per aquesta raó definirem un fitxer amb un primer marcardor que es dirà **incognites**, on es podran trobar les variables. El següent marcador seria **equacions** que inclourà a sota les diferents equacions. S'ha de tenir cura per assegurar que el nombre d'equacions i variables és el mateix:

```
In [1]: %%file equacions.eq
        incognites
        x
        y
        equacions
        a*x + b*y - c
        d*x - b*y - d
```

```
Overwriting equacions.eq
```

Es defineix una funció per obrir el fitxer anterior:

```
In [2]: def decodificar_eq(fitxer_eq):
            '''La funció decodificar_eq llegeix l'objecte de tipus fitxer fitxer_eq
            i n'extreu el contingut.
            El marcador incognites indica on es guarden les diferents variables.
            Cada variable ha de ser indicada a continuació.
            Seguint les incògnites, s'ha d'afegir el camp equacions. Les diferents
            equacions s'hauran d'escriure després del marcador.
            La funció retorna una tupla que inclou una llista amb les equacions i
            una segona amb les variables.
            '''
            import sympy as sp

            # Llista de les equacions i les variables
            equacions = []
            variables = []

            # Es llegeix el primer marcador i es neteja
            marcador = fitxer_eq.readline().strip()

            # Es comprova que el marcador és incognites.
            if marcador == 'incognites':
                # Si és així es fa un bucle.
                while True:
                    # El nou valor es llegeix i neteja
                    linia = fitxer_eq.readline().strip()

                    # Si el contingut de la línia no és el marcador
                    # equacions, es considera que és una variable
                    if linia != 'equacions':
                        # Es crea la variable i es guarda a la llista
```

```
                    # corresponent.
                    variables.append(sp.symbols(linia.strip()))
                else:
                    # En cas que la línia contingui el marcador
                    # 'equacions', es trenca el bucle.
                    break

            # Ara continuem llegint línia a línia fent servir un
            # bucle for. És important indicar que aquesta part
            # del fitxer es llegeix just després de la línia
            # 'equacions'.
            for linia in fitxer_eq.readlines():
                # La línia és també netejada
                linia = linia.strip()
                # La línia és simplificada i guardada a equacions.
                equacions.append(sp.sympify(linia))
        else:
            print('Marcador no vàlid:', token)

        # Es retornen les equacions i variables.
        return (equacions, variables)
```

In [3]:
```
with open('equacions.eq', 'r') as fitxer_eq:
    equacions, variables = decodificar_eq(fitxer_eq)
```

In [4]:
```
import sympy as sp

resultats = sp.solve(equacions, variables)
```

In [5]:
```
sp.init_printing()

resultats
```

Out[5]:

$$\left\{x : \frac{c+d}{a+d}, \quad y : \frac{d\,(-a+c)}{b\,(a+d)}\right\}$$

In [6]:
```
list(resultats.keys())
```

Out[6]:

$$[x, \quad y]$$

In [7]:
```
list(resultats.values())
```

Out[7]:

$$\left[\frac{c+d}{a+d}, \quad \frac{d\,(-a+c)}{b\,(a+d)}\right]$$

Ara el resultat es pot guardar de la mateixa manera. Fem servir els marcadors **`variables`** i **`resultats`**, el fitxer l'anomenarem *eqr*, i indicarem que proporciona els resultats:

```
In [8]: def codificar_eqr(fitxer_eqr, resultats):
            '''La funció codificar_eqr escriu a l'objecte de tipus fitxer
            fitxer_eqr per guardar els diferents elements inclosos al
            diccionari de resultats.
            El marcador variables indica on es guarden les variables.
            Cada clau de resultats s'ha de guardar just després.
            Seguint les variables, hi ha el marcador resultats.
            Els diferents valors del diccionari es guarden a
            continuació.
            '''

            # El marcador variables s'escriu primer
            fitxer_eqr.write('variables\n')

            # La llista de variables es llegeix del diccionari
            variables = resultats.keys()

            # Les variables es guarden al fitxer
            for nom_var in variables:
                fitxer_eqr.write(str(nom_var) + '\n')

            # El marcador resultats s'escriu després
            fitxer_eqr.write('resultats\n')

            # Els resultats es guarden al fitxer
            for nom_var in variables:
                fitxer_eqr.write(str(resultats[nom_var]) + '\n')

In [9]: with open('resultatsEquacions.eqr', 'w') as fitxer_eqr:
            codificar_eqr(fitxer_eqr, resultats)

In [10]: %less resultatsEquacions.eqr

In [11]: def llegir_eq(nom_fitxer):
             with open(nom_fitxer, 'r') as fitxer_eq:
                 entrada_eq = decodificar_eq(fitxer_eq)

             return entrada_eq

         def resoldre_equacio(dades_entrada):
             import sympy as sp

             return sp.solve(dades_entrada[0], dades_entrada[1])

         def escriure_eqr(nom_fitxer, resultats):
             with open(nom_fitxer, 'w') as fitxer_eqr:
                 codificar_eqr(fitxer_eqr, resultats)

         def programa():
             nom_fitxer = input("Nom del fitxer d'entrada: ")

             dades_entrada = llegir_eq(nom_fitxer)
```

```
            resultats = resoldre_equacio(dades_entrada)

            nom_fitxer = input("Nom del fitxer de sortida: ")

            escriure_eqr(nom_fitxer, resultats)
```

```
In [12]: programa()
```

```
Nom del fitxer d'entrada: equacions.eq
Nom del fitxer de sortida: resultatsEquacions.eqr
```

```
In [13]: %less resultatsEquacions.eqr
```

Fem una prova amb un sistema que inclogui una equació de segon grau:

```
In [14]: %%file equacionsSegonGrau.eq
         incognites
         x
         y
         equacions
         a*x**2 + b*x + c
         d*x - b*y - d
```

```
Overwriting equacionsSegonGrau.eq
```

```
In [15]: programa()
```

```
Nom del fitxer d'entrada: equacionsSegonGrau.eq
Nom del fitxer de sortida: resultatsEquacionsSegonGrau.eqr
```

```
        ---------------------------------------------------------------------------
        AttributeError                            Traceback (most recent call last)

        <ipython-input-15-3553f2b98f09> in <module>
    ----> 1 programa()


        <ipython-input-11-3a822dba0351> in programa()
         23     nom_fitxer = input("Nom del fitxer de sortida: ")
         24
    ---> 25     escriure_eqr(nom_fitxer, resultats)


        <ipython-input-11-3a822dba0351> in escriure_eqr(nom_fitxer, resultats)
         12 def escriure_eqr(nom_fitxer, resultats):
         13     with open(nom_fitxer, 'w') as fitxer_eqr:
    ---> 14         codificar_eqr(fitxer_eqr, resultats)
         15
         16 def programa():
```

```
<ipython-input-8-3e196fa2e079> in codificar_eqr(fitxer_eqr, resultats)
 14
 15     # La llista de variables es llegeix del diccionari
---> 16     variables = resultats.keys()
 17
 18     # Les variables es guarden al fitxer


AttributeError: 'list' object has no attribute 'keys'
```

Salta un error. El motiu és que en el resultat, en aquest cas, la funció **solve()** retorna una llista en lloc d'un diccionari. Ho podem comprovar amb l'exemple següent:

```
In [16]: sp.var('x, y')
         sp.solve([sp.sympify("a*x**2 + b*x + c"), sp.sympify("d*x - b*y - d")], [x, y])
```

Out[16]:

$$\left[\left(\frac{b\left(-\frac{d(2a+b)}{2ab}-\frac{d\sqrt{-4ac+b^2}}{2ab}\right)+d}{d},\quad -\frac{d\,(2a+b)}{2ab}-\frac{d\sqrt{-4ac+b^2}}{2ab}\right),\right.$$
$$\left.\left(\frac{b\left(-\frac{d(2a+b)}{2ab}+\frac{d\sqrt{-4ac+b^2}}{2ab}\right)+d}{d},\quad -\frac{d\,(2a+b)}{2ab}+\frac{d\sqrt{-4ac+b^2}}{2ab}\right)\right]$$

2.17.2 Exercici

Quan resolem un problema amb equacions de segon grau, el resultat es retorna en una llista en lloc d'un diccionari. Com podem modificar el problema i el fitxer de resultat per tenir en compte aquest problema?

> La funció **isinstance(object, type)** es pot fer servir per determinar el tipus i actuar de forma adequada.

```
In [17]: with open('equacionsSegonGrau.eq', 'r') as fitxer_eq:
             equacions, variables = decodificar_eq(fitxer_eq)
```

```
In [18]: import sympy as sp
         resultats = sp.solve(equacions, variables)
```

```
In [19]: resultats
```

Out[19]:

$$\left[\left(\frac{b\left(-\frac{d(2a+b)}{2ab}-\frac{d\sqrt{-4ac+b^2}}{2ab}\right)+d}{d},\quad -\frac{d\,(2a+b)}{2ab}-\frac{d\sqrt{-4ac+b^2}}{2ab}\right),\right.$$
$$\left.\left(\frac{b\left(-\frac{d(2a+b)}{2ab}+\frac{d\sqrt{-4ac+b^2}}{2ab}\right)+d}{d},\quad -\frac{d\,(2a+b)}{2ab}+\frac{d\sqrt{-4ac+b^2}}{2ab}\right)\right]$$

```
In [20]: def codificar_eqr_dicc(fitxer_eqr, variables, resultats):
             # El marcador variables s'escriu primer
```

```
        fitxer_eqr.write('variables\n')

        # Les variables es guarden al fitxer
        for nom_var in variables:
            fitxer_eqr.write(str(nom_var) + '\n')

        # El marcador ordre s'escriu al fitxer
        fitxer_eqr.write('ordre\n')

        # En aquest cas, l'ordre és sempre 1
        fitxer_eqr.write('1\n')

        # El marcador resultats s'escriu al fitxer
        fitxer_eqr.write('resultats\n')

        # Els resultats també s'escriuen
        for nom_var in variables:
            fitxer_eqr.write(str(resultats[nom_var]) + '\n')

    def codificar_eqr_llista(fitxer_eqr, variables, results):
        # El marcador variables s'escriu primer
        fitxer_eqr.write('variables\n')

        # Les variables es guarden al fitxer
        for nom_var in variables:
            fitxer_eqr.write(str(nom_var) + '\n')

        # El marcador ordre s'escriu al fitxer
        fitxer_eqr.write('ordre\n')

        # En aquest cas, l'ordre és la longitud de resultats
        fitxer_eqr.write(str(len(resultats)) + '\n')

        # El marcador resultats s'escriu al fitxer
        fitxer_eqr.write('resultats\n')

        # Els resultats també s'escriuen
        for resultat in resultats:
            for valors in resultat:
                fitxer_eqr.write(str(valors) + '\n')

    def codificar_eqr_multiple(fitxer_eqr, variables, resultats):
        '''La funció codificar_eqr_multiples escriu un objecte fitxer
        fitxer_eqr i guarda els diferents elements incloent-hi variables
        i resultat. Pot rebre els resultats en forma de diccionari o
        de llista.
        El marcador variables indica on es guarden les variables.
        Cada variable es guarda seguidament.
        Seguint aquestes variables s'escriu el marcador ordre i l'ordre
        de la solució resultant.
        Finalment, s'escriu el marcador resultats. Les diferents
```

```
            solucions es guarden a continuació.
            '''

            if isinstance(resultats, dict):
                codificar_eqr_dicc(fitxer_eqr, variables, resultats)
            elif isinstance(resultats, list):
                codificar_eqr_llista(fitxer_eqr, variables, resultats)
```

```
In [21]: with open('resultatsEquacionsSegonGrau.eqr', 'w') as fitxer_eqr:
             codificar_eqr_multiple(fitxer_eqr, variables, resultats)
```

```
In [22]: %less resultatsEquacionsSegonGrau.eqr
```

El programa es modifica per tenir en compte els canvis:

```
In [23]: def escriure_eqr(nom_fitxer, variables, resultats):
             with open(nom_fitxer, 'w') as eqr_file:
                 codificar_eqr_multiple(eqr_file, variables, resultats)

         def programa_multiple():
             nom_fitxer = input("Nom del fitxer d'entrada: ")

             dades_entrada = llegir_eq(nom_fitxer)

             resultats = resoldre_equacio(dades_entrada)

             nom_fitxer = input("Nom del fitxer de sortida: ")

             escriure_eqr(nom_fitxer, dades_entrada[1], resultats)
```

```
In [24]: programa_multiple()
```

```
Nom del fitxer d'entrada: equacionsSegonGrau.eq
Nom del fitxer de sortida: resultatsEquacionsSegonGrau.eqr
```

```
In [25]: %less resultatsEquacionsSegonGrau.eqr
```

```
In [26]: programa_multiple()
```

```
Nom del fitxer d'entrada: equacions.eq
Nom del fitxer de sortida: resultatsEquacions.eqr
```

```
In [27]: %less resultatsEquacions.eqr
```

Exercici 2.17.1 *Repeteix el mateix problema fent servir CSV Reader i Writer. En lloc d'escriure cada conjunt de resultats en files, guarda un conjunt en una fila i cada variable en una columna.*

Exercici 2.17.2 *Repeteix l'exercici anterior per al sistema:* $ax^2 + by = -c$, $bx^2 - y^2 = 0$.

Exercici 2.17.3 *Repeteix el primer exemple del capítol per al sistema:* $ax + by + z = c$, $dx - by - -z = d$, $ax + dy + cz = a$.

2.18 Bibliografia

2.18.1 Llibres

http://www.onlineprogrammingbooks.com/python/

http://pythonbooks.revolunet.com/

https://wiki.python.org/moin/PythonBooks

http://readwrite.com/2011/03/25/python-is-an-increasingly-popu

http://readwrite.com/2011/03/25/python-is-an-increasingly-popu

http://shop.oreilly.com/category/browse-subjects/programming/python.do

http://pythontips.com/2014/02/04/free-python-books/

http://www.onlineprogrammingbooks.com/python/

http://www.amazon.com/Python-Languages-Tools-Programming-Books/b?ie=UTF8&node=285856

http://learnpythonthehardway.org/book/

http://codepancake.com/5-python-books-for-beginners/

http://www.goodreads.com/list/show/32685.Best_Python_programming_books

http://www.fullstackpython.com/best-python-resources.html

http://www.pixelmonkey.org/2015/06/06/pybooks

2.18.2 Articles

http://www.nature.com/news/programming-pick-up-python-1.16833

http://www.informit.com/articles/index.aspx?st=86026

https://srodriguezv.wordpress.com/2013/07/14/de-matlab-a-python-python-como-lenguaje-de-programacion-cientifico-y-tecnico/

2.18.3 Exemples

http://www.programiz.com/python-programming/examples

http://www.hlevkin.com/Shell_progr/hellopython.htm

http://www.secnetix.de/olli/Python/

http://sthurlow.com/python/lesson02/

http://www.comoprogramar.org/ejemplos-de-python-codigo/

Llistat d'exercicis

Índex de figures

Índex de taules