Introduction to python language

Juan Carlos Paniagua
jpaniagua@ub.edu

Departament de Ciència de Materials i Química Física
Institut de Química Teòrica i Computacional
Universitat de Barcelona

Barcelona, September 2017 (revised on January 22th, 2021)

Acknowledgements: I am indebted to my colleague Albert Solé*, who carefully read this presentation, made many valuable comments and suggestions to improve it, and pointed out several errors.

Index

1. Concepts of informatics. Codification of the information. GNU/Linux in a GUI and in text mode.
2. My first program. Comments. Entering numbers. Arithmetic operations. Executing in a notebook.
3. Modules. Conditional construct. Comparison operators. Combining conditions. Stopping the program before the end.
4. Repetitive construct: for. Accumulators. Nested for's. Strategies for debugging.
5. Repetitive construct: while. Iterative equation resolution. Iteration counters.
6. Strings. Functions and methods.
7. Lists. Creating lists within a program. Creating lists from input. Entering a list as a string. Operating with lists. Copying lists.
8. Creating functions. Creating modules. Lists as arguments and/or results.
9. Module NumPy. Array attributes. Array slices. Copying arrays. Creating vectors. Creating matrices. Operating with vectors.
10. Graphics with Matplotlib.
11. Output and input files in NumPy. Output and input files in standard python.

My first program

- Write this **source code** in a file with any simple text editor:

variable

```
print("What's your name?")
name = input()
print('Hello',name,'!')
```

input instruction

→ *in a new file*

assignment instruction

output instruction

- Save it in your user folder with a file name extension `.py`
  For example, you could name it `greeting.py`

- Open a text terminal, set your user folder as the working directory (if it is not) and execute the program:

```
python3 greeting.py
```

→ *in a (linux) text terminal*

(the interpreter `python3` interprets each instruction of the source code and executes it)

A simpler version

- Edit the source file and leave it this way:

```
name = input("What’s your name? ")
print('Hello',name,'!')
```

- Save it
- Execute it: `python3 greeting.py`

Comments

```
"""This is my first program, it has only two instructions
or sentences"""
name = input("What’s your name? ")
# "name" is a variable in which the input datum is stored
print('Hello',name,'!') # this is an i/o instruction
# (input too)
```

Entering numbers

```
n = input('Enter an integer number ')
# n contains a string-type datum
n = int(n) # n contains now an integer-type numerical datum
# the variables acquire their type in the assignments (n_int = int(n))
print(n)
# The first 2 sentences can be written in a single line:
n = int(input('Enter an integer number '))
print(n)
n = float(n) # the datum contained in n is recoded to real-type format
print(n) # the value of n is now printed with .0 at the end

# A body mass index calculator
# ****************************
# This program needs 2 real numbers as input data
# (a dot must be used as the decimal separator; e.g.: 70.5)
height = float(input('Enter your height in meters '))
weight = float(input('Enter your weight in kg '))
print('Your height and weight are {} m and {} kg'.format(height,weight))
BMI = weight / height**2
print('Your body mass index is {:.1f} kg/m2'.format(BMI)) # {:5.1f}
# the result is printed with 1 decimal; e.g. 21.5 kg/m2
# Alternatively:
print('Your body mass index is {:.2e} kg/m2'.format(BMI)) # {:10.2e}
# the result is now printed in scientific notation: 2.15e+01 kg/m2
# Units should be specified when needed in i/o instructions
```

Arithmetic operations

Arithmetic operators: `+, -, *, /, //, %, **`

integer division (`//`)

exponentiation (`**`)

modulo (remainder of the division) (`%`)

Priority:

what does `a+b/c**2` mean? $a+\frac{b}{c^2}$, $\frac{a+b}{c^2}$, $a+\left(\frac{b}{c}\right)^2$, $\left(a+\frac{b}{c}\right)^2$ or $\left(\frac{a+b}{c}\right)^2$

higher priority ↑ — order of execution ↓

- `( )` — `(a+b)/(c+d)`
- `**` — from right to left if there are more than one: `a**2**3=a**8`
- `*, /, //, %` — from left to right: `a/2*b = (a/2)*b`
- `+, -` — `a+b/c+d = a+(b/c)+d; -a**2 = -(a**2)`

Basic exercises

1. Write a program that asks the user to chose one of these 3 letters: r, p or s. The program must print in the text terminal the chosen letter.
2. Write a program that asks the user to enter the numbers of 5€ bills, 10€ bills, 20€ bills and 50€ bills he has in his wallet (one at a time). From these data, the program should calculate and print the total amount of € contained in the user's wallet.
3. a) Write a program to calculate and print the molecular mass in m_u units of a hydrocarbon with molecular formula C_nH_m. The program should ask the user for the values of n and m. Take the carbon and hydrogen masses as 12 m_u and 1 m_u respectively.
   b) Modify the program so that it prints for, say n=7 and m=9, a text like this:
   ```
   The molecular mass of C7H9 is 93 mu
   ```
4. Write a program to calculate the volume in L of an ideal gas from its temperature in K, pressure in bar and mole number. Recall that $R = 0.08314472$ bar L K^{-1} mol^{-1}. The volume should be printed with 3 decimals, and units should be specified.
5. Write a program that asks the user to enter 3 real numbers, store them in the variables a, b and c, calculates $a + \frac{b}{c^2}$, $\frac{a+b}{c^2}$, $a + \left(\frac{b}{c}\right)^2$, $\left(a + \frac{b}{c}\right)^2$, $\left(\frac{a+b}{c}\right)^2$ and $a^{b/c}$ and prints the results of these operations. *Hint*: for a=2, b=3 and c=4 you should obtain (rounding off to two decimal places): 2.19, 0.31, 2.56, 7.56, 1.56, 1.68

Executing in a notebook

- Download a notebook from the Virtual Campus.
- Open a text terminal and move to the download folder (`cd Downloads`)
- Write jupyter notebook in the text terminal and press Enter
  - or select Applications > RIIA > jupyter notebook if you are using QUBuntu 4.3 or later
- Select the downloaded file.
- Read its contents and execute each cell by clicking on it and pressing Shift+Enter
- To save a program from a notebook cell to a python file copy the program source code, open a new file with a text editor, and paste the code in it.
- To close the notebook select File (below "jupyter") > Close and Halt
- To finish working with notebooks click at Logout.
  - If you have launched jupyter from a text terminal you should clic in it and type Ctrl+c, then type y and press Enter

Modules (libraries)

- constants
```
import math
ang_degrees = 60
ang_radians = ang_degrees*math.pi/180
# since one of the operands is real (float) the result is real
x = math.sin(ang_radians)
print('The sinus of',ang_degrees,'º is',x)
print('4! =',math.factorial(4))
```
-
```
import cmath
x = cmath.sqrt(-3)
print(x) # note that √-1 is noted "j"
print(dir(cmath)) # list the functions in module cmath
```
-
```
import random as rd # rd will be used as a short name for random
rand_char = rd.choice('abcd') # or rd.choice(any_list)
rand_float = rd.uniform(1,4) # a real number ∈ [1,4)
rand_int = rd.randint(1,4) # chooses 1, 2, 3 or 4
print(rand_char,rand_float,rand_int)
```
- To list the functions of a module in a notebook cell:
```
import math
dir(math)
```
Information about any particular function of the imported module: `math.sin?`

Conditional construct

```
print('This program calculates the volume of an ideal gas
from its temperature, pressure and mole number')
P = float(input('Enter the pressure in bars: '))
if P>0:
    T = float(input('Enter the temperature in kelvins: '))
    n = float(input('Enter the number of moles: '))
    R = 0.08314472 # bar L K-1 mol-1
    V = n*R*T/P
    print('Volume = {:.3f} L'.format(V))
else:
    print('Pressure must be >0')
```

Comparison operators

mathematic symbol	python
$=$	`==`
$\neq$	`!=`
$>$	`>`
$\geq$	`>=`
$<$	`<`
$\leq$	`<=`

Combining conditions

The *logical operators* not, and and or can be used to combine conditions:

```
P = float(input('Enter the pressure in bars: '))
T = float(input('Enter the temperature in kelvins: '))
n = float(input('Enter the number of moles: '))
R = 0.08314472
if P>0 and T>0 and n>0: # equivalent to (P>0)and(T>0)and(n>0)
    V = n*R*T/P
    print('Volume = {:.3f} L'.format(V))
else:
    print('Pressure, temperature and mole number must be >0')
```

or, equivalently,

```
...
if P<=0 or T<=0 or n<=0:
    print('Pressure, temperature and mole number must be >0')
else:
    V = n*R*T/P
    print('Volume = {:.3f} L'.format(V))
```

Priority

higher priority

operator
comparison operators
not
and
or

order of execution

- *Example*:

```
if a>0 and b>0 or a<0 and not b>=0:
    print('they have the same sign')
```

is equivalent to

```
if (a>0 and b>0) or (a<0 and (not b>=0)):
    print('they have the same sign')
```

In case of doubt use parenthesis

else: may not be present

```
T = float(input('Enter the temperature: '))
units = input('Type 'C' for Celsius and 'K' for kelvins ')
if units=='C':
    T = T + 273.15
...
```

elif

```
if condition1:
    sentence_1
else:
    if condition2:
        sentences_2
    else:
        if condition3:
            sentences_3
        else:
            sentences_4
```

is equivalent to:

```
if condition1:
    sentences_1
elif condition2:
    sentences_2
elif condition3:
    sentences_3
else:
    sentences_4
```

Stopping the program before the end

```
P = float(input('Enter the pressure in bars: '))
T = float(input('Enter the temperature in kelvins: '))
n = float(input('Enter the number of moles: '))
R = 0.08314472
if P<=0 or T<=0 or n<=0:
    print('Pressure, temperature and mole number must be >0')
    exit() # quit() can also de used
else:
    V = n*R*T/P
    print('Volume = {:.3f} L'.format(V))
...
# (the program could be longer)
```

In a notebook module 'sys' has to be imported:

```
import sys
...
sys.exit()
```

and a warning is emitted while running the program: An exception has occurred,...

Basic exercises

1. Write a program that asks for the scores (out of 10) that an student has obtained in an exam and writes 'failed' if the score is under 5, 'passed' if it is ≥ 5.
2. Write a program that asks for the scores (out of 10) that an student has obtained in the theory and problem parts of an exam, writes the total score (out of 10) considering that theory and problems weigh 40% and 60% respectively, and writes 'failed' if the score is under 5, 'passed' if it is ≥ 5 and < 7.5 and 'excellent' if it is ≥ 7.5.
3. Write a program that asks for an integer number and tells whether it is divisible by 7 or not and, if it is not, gives the integer quotient (without decimals) and the real quotient.
4. Write a program that asks the user to enter one of these 3 letters: 'r' for 'rock', 'p' for 'paper' or 's' for 'scissors'. We will assume that the computer had chosen the word "paper", so that it must write "You won", "Tie" or "I won" depending on the user choice being "scissors", "paper" or "rock" (the scissors cut the paper, but the rock is wrapped by the paper).

Basic exercises

5. Write a program that
   a) asks the user to enter one of these 3 letters: 'r' for 'rock', 'p' for 'paper' or 's' for 'scissors';
   b) chooses randomly one of those 3 words and write it;
   c) writes "Tie" if both words coincide, "You won" if the user chosen word is cyclically after the one chosen by the computer ("rock" is after "scissors"), "I won" if the user chosen word is cyclically before the one chosen by the computer ("scissors" are before "rock").

$$\begin{tikzcd}
\text{rock} \arrow[rr] & & \text{paper} \arrow[dl] \\
& \text{scissors} \arrow[ul] &
\end{tikzcd}$$

Repetitive construct: for

- When we know the number of repetitions (*iterations*).

```
print('This program writes the squares of the first n natural
numbers')
n = int(input('Enter n: '))
print('The squares of 1 ...',n,'are:')
for i in range(n): # i=0,1,2...n-1
    square = (i+1)**2
    print(square)
```

Note that each time `print` is executed the writing starts in a new line.

Alternatively:

```
for i in range(1,n+1): # i=1,2...n
    square = i**2
    print(square)
```

```
print('This program writes the multiples of 13 less than n')
n = int(input('Enter n: '))
print('The multiples of 13 less than',n,'are:')
for i in range(13,n,13): # i=13,26,39...k*13<n (k integer)
    print(i)          # (if n=30 the last printed number is 26)
```

Accumulators

```
print('This program writes the sum of the first n odd numbers')
n = int(input('Enter n: '))
sum = 0
for i in range(n): # i=0,1,2,...n-1
    odd = 2*i+1
    sum = sum + odd
print('The sum of the first',n,'odd numbers is ',sum)
```

i	odd	sum
		0
0	1	0+1 = 1
1	3	0+1+3 = 4
2	5	0+1+3+5 = 9
…	…	…
n–1	2(n–1)+1 = 2n–1	0+1+…2n–1

Accumulators

```
print('This program writes the sum of the first n odd numbers')
n = int(input('Enter n: '))
sum = 0
for i in range(n): # i=0,1,2,...n-1
    odd = 2*i+1
    sum = sum + odd
print('The sum of the first',n,'odd numbers is ',sum)
```

i	odd	sum
		0
0	1	0+1 = 1
1	3	0+1+3 = 4
2	5	0+1+3+5 = 9
…	…	…
n–1	2(n–1)+1 = 2n–1	0+1+...2n–1

or, alternatively,

```
sum = 0
for i in range(1,2*n,2): # i=1,3,...2n-1
    sum = sum + i
print('The sum of the first',n,'odd numbers is',sum)
```

Nested for's

Example: variations with repetition of the digits 1 to 9 taken two by two $(11, 12, \ldots, 19, 21, 22, \ldots, 29, \ldots, 99)$

```
print('This program writes the variations with
repetition of the digits 1 to 9 taken two by two')
for i in range(1,10):
    for j in range(1,10):
        print(i,j)
```

i	1				2			…	9		
j	1	2	…	9	1	…	9	…	1	…	9
`printing`	11	12	…	19	21	…	29	…	91	…	99

Strategies for debugging

- Start with the simplest version of the program –or with the first stage of an algorithm that can be divided in several steps– and test it. Once it works properly, add a single new feature to a copy of the working source file and test the program again, and so on until all the features or steps work correctly. Always keep the latest working version!
- It the program aborts try to understand the error message. If you don't find the error in the line to which the message refers, look at at the previous line (a missing parenthesis at the end of a sentence produces an error in the following sentence).
- If you still can't find the cause of an abortion or an erroneous result, write (in a sheet of paper) a table with the values that should take the relevant variables at every step of the program (obtained with a calculator), make the program print those variables, and check them to find the point in which a discrepancy appears.
- If the error resists to be identified split the program in 2 parts than can be executed separately to see which of them is not working properly. If necessary, repeat this with the non-working part until the error is located.
- If you get an 'IndentationError' or 'TabError' look for inconsistent use of tabs and spaces for indentation (use an editor allowing to visualize the invisible characters).

Basic exercises

1. Write a program that asks for a natural number n and writes the sequence of the squares: 1, 4, 9, ... n^2.
2. Write a program that asks for a natural number n and writes the sum of the squares: 1+4+9+ ... n^2.
3. Write a program that asks for a natural number n and writes its factorial: $n! = 1 \times 2 \times 3 \times ... n$.
4. Write a program that writes the variations without repetition of the digits 1 to 9 taken two by two: 12, 13, . . . , 19, 21, 23, . . . , 29, . . . , 98.
5. Write a program that asks for a natural number n and writes the sequence of "triangular numbers" $T_1, T_2, ... T_n$, where $T_i = i(i+1)/2$. *Result for n=6*: 1, 3, 6, 10, 15, 21.
6. Write a program that asks for a natural number n and calculates the "tetrahedral number" D_n , which is the sum of the first n triangular numbers: $D_n = \sum_{i=1}^{n} T_i = T_1 + T_2 + ... T_n$, where $T_i = i(i+1)/2$. *Test*: $D_5 = 35$.
7. Write a program that asks for a natural number n and writes the sequence of tetrahedral numbers $D_1, D_2, ... D_n$, where $D_j = \sum_{i=1}^{j} T_i$. *Result for n=5*: 1, 4, 10, 20, 35.
8. Write a program that asks for a natural number $n > 2$ and tells whether it is prime (non-divisible by any integer between 2 and n–1) or not.

Additional exercises

1. a) Write a program that asks the user for a natural number larger than 1 and tells him whether it is perfect or not. A natural number larger than 1 is perfect if it is equal to the sum of its positive divisors (including 1 and excluding the number itself). *Test*: 6, 28, 496, ... are perfect numbers.
   b) Modify the program so that it asks if the list of divisors should be written (no matter whether the number is perfect or not).

2. (Adapted from an exercise by Gerard Alonso) You have received three € 50 notes from your family for your birthday, and you decide to place them in 3 different pockets in order to minimize the loss in case of robbery. Suppose you have *p* pockets in your pants and jacket, and you want to know how many ways you can place *n* notes ($n<p$).
   a) Write a program that asks the user the numbers of notes and pockets he has and tells him how many ways he can place those notes in his pockets. Note that the number of ways of choosing *n* objects among a set of *p* is the combinatorial number:

$$\binom{p}{n} = \frac{p(p-1)(p-2)\cdots(p-n+1)}{n!}$$

Hint: note that the numerator is similar to $p!=1\text{x}2...p$ but starts by $p-n+1$ instead of 1.
   b) Write a new version of the program that, if $n > p$, writes an error message and stops the execution.

Repetitive construct: while

- When we don't know the number of iterations but can state a condition for stopping the iterations.

```
P = float(input('Enter the pressure in bars: '))
while P<=0:
    print('Pressure must be >0')
    P = float(input('Enter the pressure in bars: '))
...
```

Prime numbers: 2 versions

- If the program does nothing more:

```
print('This program tells whether a natural number is prime or not')
n = int(input('Enter a natural number > 2: '))
for div in range(2,n):
    res = n % div
    if res==0:
        print(n,'is not a prime number')
        exit() # the program is stopped
print(n,'is a prime number') # executed only if no divisor was found
```

- If the program continues:

```
print('This program tells whether a natural number is prime or not')
n = int(input('Enter a natural number > 2: '))
div = 2
prime = True # prime is a boolean type variable
while prime and div<n:
    res = n % div
    if res==0:
        prime = False
    div = div + 1
if prime:
    print(n,'is a prime number')
else:
    print(n,'is not a prime number')
# the program could continue
```

Basic exercises

1. A teacher has finished correcting a lot of exams and wants to make a list of the students and their marks. Write a program that asks for the surname and the mark (out of 10) of each student and writes these two data (each student in a new line). The program must stop asking data when the teacher enters a negative mark.

2. A teacher has finished correcting a lot of exams and wants to make a list of the students *having passed* and their marks. Write a program that asks for the surname and the mark (out of 10) of each student and writes these two data (each student in a new line). The program must stop asking data when the teacher enters a negative mark.

3. A teacher has finished correcting a lot of exams and wants to know the average mark of all of them. Write a program that asks for the student marks (out of 10) and calculates their average. The program should stop asking data when a negative mark is entered.

4. Write a program that asks for a real number *x* and a small number ε, calculates e^x = exp(*x*) as a taylor expansion: $e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ including terms larger than ε, and write this result. For comparison, write also the value of exp(*x*) calculated with the exp() function of module math.

Iterative eqn. resolution

- Let's calculate the volume occupied by a given number of mols of a gas at given pressure and temperature using the van der Waals equation (*a* and *b* are parameters that depend on the gas; use those of N_2: $a = 1.370$ L^2 bar mol^{-2}, $b = 0.0387$ L mol^{-1}):

$$\left(P + \frac{an^2}{V^2}\right)(V - nb) = nRT \qquad V = \frac{nRT}{P + \frac{an^2}{V^2}} + nb \qquad V_1 = \frac{nRT}{P + \frac{an^2}{V_0^2}} + nb$$

```
n = float(input('Enter the number of mols: '))
P = float(input('Enter the pressure in bars: '))
T = float(input('Enter the temperature in kelvins: '))
R = 0.08314472 # in bar L / (K mol)
a = 1.370 # in L2 bar / mol2
b = 0.0387 # in L / mol
V0 = n*R*T/P
V1 = n*R*T/(P+a*n**2/V0**2) + n*b
while V1 != V0:
      V0 = V1
      V1 = n*R*T/(P+a*n**2/V0**2) + n*b
print('The volume is',V1,'L')
```

Iterative eqn. resolution

- A version with a non-vanishing tolerance: $\boxed{|V_1 - V_0| \le \varepsilon}$

```
n = float(input('Enter the number of mols: '))
P = float(input('Enter the pressure in bars: '))
T = float(input('Enter the temperature in kelvins: '))
eps = float(input('Enter the desired tolerance of the result: '))
R = 0.08314472 # in bar L / (K mol)
a = 1.370 # in L2 bar / mol2
b = 0.0387 # in L / mol
V0 = n*R*T/P
V1 = n*R*T/(P+a*n**2/V0**2) + n*b
while abs(V1-V0)>eps:
    V0 = V1
    V1 = n*R*T/(P+a*n**2/V0**2) + n*b
print('The volume is',V1,'L')
```

Iteration counters

- An infinite loop can be stopped by pressing ctrl + c (when executing in a text terminal)
- Use an *iteration counter* to prevent infinite loops:

```
n = float(input('Enter the number of mols: '))
P = float(input('Enter the pressure in bars: '))
T = float(input('Enter the temperature in kelvins: '))
eps = float(input('Enter the desired tolerance of the result: '))
R = 0.08314472 # in bar L / (K mol)
a = 1.370 # in L2 bar / mol2
b = 0.0387 # in L / mol
V0 = n*R*T/P
V1 = n*R*T/(P+a*n**2/V0**2) + n*b
iter = 0 # set the counter to 0
while abs(V1-V0)>eps:
    iter = iter+1 # increase the counter by 1 (accumulator of 1's)
    if iter>999:
        print('Convergence not attained after',iter,'iterations')
        exit()
    V0 = V1
    V1 = n*R*T/(P+a*n**2/V0**2) + n*b
print('The volume is',V1,'L')
print('I have done',iter,'iterations')
```

Basic exercises

5. Write a program to solve iteratively the equation $x = 1/\sqrt{1+\tan^2(x)}$ with a tolerance ε given by the user. The program should ask the user for a seed between 0 and 1 to start the iterations.

   Note: That equation appears in the calculation of the energies of the bound quantum states of a particle in a potential well.

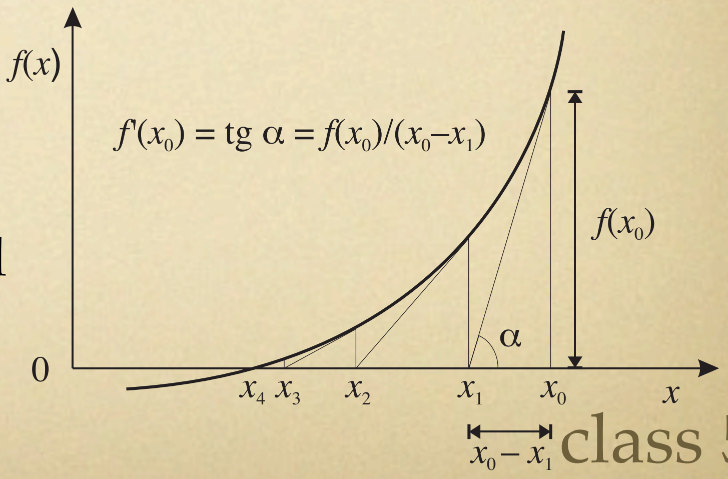
   Result: $x = 0.739085$

6. The iterative Newton-Raphson algorithm for finding real solutions of an equation of the type $f(x)$=0 works as follows:

   a) choose an initial estimate x_0 of a solution,

   b) calculate a new value of x_1 by using the expression $x_1 = x_0 - f(x_0)/f'(x_0)$. If $|f(x_1)| > \varepsilon$ (a small number) then change x_0 for x_1 and recalculate x_1. This is iteratively repeated until $|f(x_1)| \le \varepsilon$.

   Use this method to find a solution of the equation $ax^3+bx^2+cx+d=0$. The values of a, b, c and d and the initial estimate of the solution (x_0) should be asked to the user.



$f(x)$
$f'(x_0) = \text{tg}\ \alpha = f(x_0)/(x_0-x_1)$
$f(x_0)$
α
0
x_4 x_3 x_2 x_1 x_0 x
$x_0 - x_1$

Additional exercise

1. Write a program that plays repeatedly the "rock", "paper" or "scissors" game described in this slide until the user enters a letter different from r, p or s. The program must show the result of each game, keep track of the number of plays won by the user, the number of wins of the computer and the number of ties, and write these data.[1]

[1] I acknowledge Rosendo Valero for the idea of this multi-step exercise.

Strings

```
name = input('What is your name? ')
print('Hello',name,'!')
numletters = len(name)
print('your name has',numletters,'letters')
print('the first letter of your name is',name[0])
print('the first 3 letters are',name[0:3])
print('or, again', name[:3])
print('the last letter is', name[numletters-1])
print('or, again', name[-1])
print('the last 2 letters are',name[-2:])
print('every second letter:',name[::2])
surname = input('What is your family name? ')
full_name = name + ' ' + surname # concatenation
print('Nice to meet you',full_name)
bye = 'Bye '
bye2 = 2*bye # replication
print(bye2,name)
```

Strings

```
name = input('What is your name? ')
#
if 'a' in name:
    print("There is an 'a' in your name.")
else:
    print("There is not an 'a' in your name.")
#
print("Let's write your name vertically:")
for char in name:
    print(char)
"""Each time print is executed the writing starts in
a new line"""
```

Functions and methods

- result = function_name(arguments)

```
import math
x = float(input('Enter a real number: '))
absx = abs(x)
y = math.sin(x)
xsqr = math.sqrt(absx)
lnx = math.log(x)
logx = math.log10(x)
expx = math.exp(x) # better than math.e**x
n = int(2/3) # n takes the value 0
n = round(2/3) # n takes the value 1
str25 = str(25) # converts a number into a string
name = input('What's your name? ')
numletters = len(name)
```

- result = variable_name.method_name(arguments)

```
name = 'Antonieta'
name_cap = name.upper()
name_low = name.lower()
num_of_ts = name.count('t') # number of occurrences of substring
position_t = name.find('t') # the first occurrence: 2
name_masc = name.replace('eta','') # every occurrence
print('You have {:.2f} €'.format(total)) # also for constants
```

Basic exercises

1. A DNA strand is composed of monomer units called nucleotides. Each nucleotide has of one of four bases (adenine, guanine, thymine or cytosine) and it is identified with the first letter of its base: 'a' for adenine, 'g' for guanine, 't' for thymine and 'c' cytosine. The nucleotides are arranged in grups of 3 –*codons*– that encode the amino acids that make up proteins; e.g. 'aaa' encodes lysine, 'cat' histidine, 'cta' leucine, 'tgt' cysteine and 'caa' glutamine. Write a program to

   a) assign each of these 5 codons to a literal variable with the name of the corresponding amino acid (lysine='aaa', etc.),

   b) concatenate the 5 codons in the indicated order and save the result into a new variable,

   c) replicate 10 times the concatenated string and assign the result to a new variable,

   d) write this result.

   Result:

   aaacatctatgtcaaaaacatctatgtcaaaaacatctatgtcaaaaacatctatgtcaaaaacatctatgtcaaaaacatct
   atgtcaaaaacatctatgtcaaaaacatctatgtcaaaaacatctatgtcaaaaacatctatgtcaa

   (adapted from an exercise by Fermín Huarte Larrañaga)

Basic exercises

2. Write a program to

a) assign these two codon sequences to two variables
'agcgccttgaattcggcaccaggcaaatctcaaggagaagttccggggagaaggtgaaga'
'cggggagtggggagttgagtcgcaagatgagcgagcggatgtccactatgagcgataata';

b) concatenate the two variables into a new variable `dna` and print it;

c) write the first and the last codons of the concatenated sequence;

d) write the number of nucleotids of the concatenated sequence;

e) use the method `dna.count()` to obtain the percentage of each nucleotid (a, g, t and c).

Result: a: 29.17%, g: 36.67%, t: 16.67%, c: 17.50%)

(adapted from an exercise by Fermín Huarte Larrañaga)

Basic exercises

3. a) Write a program that asks for the developed formula of an organic molecule, that is, a formula with the symbols C, H, O, N, S and no numbers (e.g., HHHCCHHCONHH), writes an empirical molecular formula indicating the number of occurrences of each atom (e.g., C3H7O1N1S0) The empirical formula should be written with no spaces between every two characters.

   b) Write a new version of the program that gives an error message if the molecule entered by the user contains atoms different from C, H, O, N or S.

   c) Write a new version of the program that calculates the molecular mass in m_u units (you can take m(C)=12, m(H)=1, m(O)=16, m(N)=14, m(S)=32). *Hint*: For the above example you should obtain 73.

   d) Write a new version of the program in which the atoms that are not present in the molecule do not appear in the empirical formula, and those appearing only once have no number (e.g., C3H7ON).

Lists

```
my_list = [2, 2.5, 3, 'end', False, [10,20]]
number_of_elements = len(my_list) # 6 elements
first_element = my_list[0] # first_element takes the value 2
x = my_list[5][1] # x takes the value 20
y = my_list[0:4:2] # y takes the value [2, 3]
z = my_list[:2] # z takes the value [2, 2.5]
u = my_list[3:] # u takes the value ['end', False, [10,20]]
v = my_list[-1] # v takes the value [10, 20]
w = my_list[-2:] # w takes the value [False, [10, 20]]
my_list.append(25) # adds the element 25 at the end of my_list
my_list.insert(3,'new') # insert 'new' in position [3] of my_list
my_list.pop(1) # removes the element my_list[1] from my_list
b = my_list.pop() # removes the last element and assigns it to b
```

Creating lists within a program

```
my_list = [2, 2.5, 3, 'end', False, [10,20]]


num = list(range(7)) # creates the list [0, 1, 2, 3, 4, 5, 6]


line = 'The water boiling point is 373.15 K at 1 atm'
words = line.split() # creates a list with 10 strings:
['The','water','boiling','point','is','373.15','K','at','1','atm']
temp_Celsius = float(words[5]) - 273.15
print(temp_Celsius) # prints 100.0
...
# use .split('x') if the word separator is 'x'; e.g.: '20x35x5'
```

Creating lists from input

```
new_list = [] # creates an empty list

# If the user knows the length of the list:
length = int(input('Enter the length of the list: '))
for i in range(length):
    element = input('enter an element of the list: ')
    new_list.append(element)

# If the user doesn't know the length of the list:
element = input('Enter the first element of the list: ')
print('(press the return key to finish)')
while element != '':
    new_list.append(element) # or ...append(float(element))
    element = input('enter another element: ')
```

In the 2nd example you can't use `element = float(input(...` Why?

Entering a list as a string

```
temp_str = input('Enter the ºC temperatures separated by spaces ')
temp_list_str = temp_str.split()
temp_list_num = []
for temp in temp_list_str:
    temp_list_num.append(float(temp)) # one by one
print('The Celsius temperatures are: {}'.format(temp_list_num))
```

If the user enters 100 200 300 then:
- `temp_str` takes the value `'100 200 300'`
- `temp_list_str` takes the value `['100','200','300']`
- `temp_list_num` takes the value `[100.0,200.0,300.0]`

Formats such as `:.2f` apply only to individual data; so this does not work:

```
print('The Celsius temperatures are: {:.2f}'.format(temp_list_num))
```

Operating with lists

```
list_a = list(range(7)) # [0, 1, 2, 3, 4, 5, 6]
list_b = [7, 8, 9]
list_ab = list_a + list_b # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
print(2*list_b) # same as list_b + list_b: [7, 8, 9, 7, 8, 9]
list_b[1] = list_b[1] + 2 # list_b[1] is modified: [7, 10, 9]
if 10 in list_b:
    print('10 is in list_b')
list_ab2 = [] # to operate with each element of list_ab:
for ele in list_ab:
    list_ab2.append(ele**2)
print(list_ab2) # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
list_ab2i = []
for i in range(len(list_ab2)): # if we need the index
    list_ab2i.append(list_ab2[i] - i)
print(list_ab2i) # [0, 0, 2, 6, 12, 20, 30, 42, 56, 72]
```

Copying lists

- If `a` and `b` are scalars (variables holding a single value) `b=a` copies the content of `a` into `b` (if we change `a` then `b` is not affected and *vice versa*).
- If `a` is a list `b=a` gives a 2nd name `b` to list `a`, so that `a` and `b` both refer to the same memory positions (if we change `a` then `b` is also changed and *vice versa*).
- To copy a list `a` into a new list `b`: `b=list(a)`

Basic exercises

1. Write a program that
   a) asks the user to enter the mole number, Celsius temperature and pressure in bars of an ideal gas, the 3 data in a single line separated by spaces.
   b) Calculates and writes the volume in liters occupied by the gas.

2. Write a program that
   a) asks the user to enter the number of data in a list of Celsius temperatures;
   b) asks for the values of these temperatures, pressing Enter after each value, and store them in a list;
   c) creates a new list containing the corresponding absolute temperatures in kelvins and shows it.

3. Write a program that asks for the surnames of the students who have attended an exam and store them in a list. After entering the name of the last student the user should press the return key and the program should write the list as a column.
   a) Write a second version of the program in which, after having entered the list of names, the user enters the scores obtained by the students. Then the program should write a 2-column list with the surname and the mark of a student in each row.
   b) Write a third version of the program that writes a 2-column list including only the surnames and the marks of the students having passed the exam.

Basic exercises

4. Write a program that
   a) asks the user to enter a text paragraph, pressing the return key to end;
   b) writes the number of words in the paragraph;
   c) writes the total number of characters (spaces excluded);
   d) writes the percentage of words having the letter 'a'.

5. Write a program that
   a) asks the user to enter a list containing the coefficients a_0, a_1,... a_n of a polynomial: $P(x) = a_0 + a_1x + \ldots\ a_nx^n = \sum_{i=0}^{n} a_ix^i$;
   b) asks the user to enter a value for x;
   c) calculates and writes the value of $P(x)$;
   d) calculates and writes the value of $P'(x) = a_1 + 2a_2x + \ldots\ na_nx^{n-1} = \sum_{i=1}^{n} ia_ix^{i-1}$
   Test: For x=2 the polynomial $1 + x^2 + 3x^4$ takes the value 53 and its derivative takes the value 100.

Additional exercise

1. Write a program that asks the user for the surname and the mark (out of 10) of the students who have attended an exam. Then the program should write a 2-column list with the surname and the mark of one student in each line. Assume that the user does not know the total number of students, so that the program should:
   a) create an empty exam_list;
   b) ask for the surname of the first student;
   c) repeat the following actions until the user presses Return without entering any text:
      i) ask for the mark of the student and create a 2-element list with the surname and the corresponding mark: [surname, mark];
      ii) append this 2-element list to the exam_list;
      iii) ask for a new surname;
   d) write a 2-column list with the surname and the mark of one student in each line.

Creating functions

```
def Cel2K(tempC): # function declaration
    """This function converts a Celsius temperature into an absolute one"""
    tempK = tempC + 273.15 # tempK and tempC are local variables
    return tempK
tCi = float(input('Enter the initial Celsius temperature: '))
tCf = float(input('Enter the final Celsius temperature: '))
tKi = Cel2K(tCi)
tKf = Cel2K(tCf)
print('The initial and final absolute temperatures are',tKi,'and',tKf,'K')

def cart2pol(x,y):
    """This function converts x, y cartesian coordinates into polar"""
    import math # makes math functions available within function cart2pol
    r = math.sqrt(x**2+y**2)
    phi = math.atan2(y,x) # better than math.atan(y/x)
    return r,phi
# importing math here would make them available anywhere in the program
x = float(input('Enter the cartesian x coordinate: '))
y = float(input('Enter the cartesian y coordinate: '))
r,phi = cart2pol(x,y) # same order as in the function definition
print('The polar coordinates are',r,'and',phi,'radians')
```

- Function definitions must be before the programs that use it (preferably before the main program).

Creating modules

- We can create a module by saving the 2 functions in a file *conversions.py* The main program must import this file to use them:

```
import conversions
tCi = float(input('Enter the initial Celsius temperature: '))
tCf = float(input('Enter the final Celsius temperature: '))
tKi = conversions.Cel2K(tCi)
tKf = conversions.Cel2K(tCf)
print('The initial and final absolute temperatures are',tKi,'and',tKf,'K')
x = float(input('Enter the cartesian x coordinate: '))
y = float(input('Enter the cartesian y coordinate: '))
r,phi = conversions.cart2pol(x,y)
print('The polar coordinates are',r,'and',phi,'radians')
```

- Content of *conversions.py*:

```
def Cel2K(tempC):
    """This function converts a Celsius temperature into an absolute one"""
    tempK = tempC + 273.15
    return tempK
def cart2pol(x,y):
    """This function converts x, y cartesian coordinates into polar"""
    import math
    r = math.sqrt(x**2+y**2)
    phi = math.atan2(y,x)
    return r,phi
```

Lists as arguments and/or results

```
def Cel2K_list(tC_list): # an argument may be a list
    tK_list = []
    for t in tC_list:
        tK_list.append(t+273.15)
    return tK_list # the result may also be a list

tC = [0, 25, 30, 40, 45]
tK = Cel2K_list(tC)
print('Celsius temperatures:',tC)
print('Absolute temperatures in kelvins:',tK)
```

Basic exercises

1. a) Write a function that receives as variables the 3 cartesian components (*x*, *y*, *z*) of a vector and returns its modulus (or norm): $\sqrt{x^2+y^2+z^2}$. Then write a main program that asks the user to enter the cartesian components of a vector and uses that function to calculate its modulus.

   b) Write a new program that asks the user to enter the cartesian components of 2 vectors and uses the function made in *a*) to calculate their moduli and the distance between them (which is the modulus of their difference).

2. The electromotive force (in V) of the battery Zn | $ZnCl_2$(aq) | | AgCl(s) | Ag working at an *absolute* temperature *T* is: $E(T,c) = E^\circ_T - \frac{RT}{nF}\ln(4c^3)$

   where $R = 8.3145$ JK^{-1}mol^{-1}, $F = 96458$ Cmol^{-1}, *c* is the concentration of $ZnCl_2$ in mole/L, *n* in the number of electrons interchanged in the global redox reaction (Zn + 2AgCl(s) → $ZnCl_2$(aq) + Ag) and $E_T{}^o$ is the standard electromotive force of the battery in V, that takes the value 0.984 V at 25ºC.

   Write a program that asks the user for *c*, uses a function to calculate *E* at 25ºC, and writes the result. The values of *R*, *F*, *n*, *T* and $E_T{}^o$ should be assigned within the function. *Test*: for $c = 1.00 \times 10^{-4}$ mol/L, $E = 1.32$ V.

Basic exercises

3. For the battery described in exercise 2, write a program that asks the user for a list of $ZnCl_2$ concentrations in mole/L and calls a single time a function to calculate and return the corresponding list of electromotive forces.

4. a) Write a function that receives a natural number >2 as an argument and returns the boolean result *True* if the number is prime and *False* if it is not prime (see this slide). To verify the proper operation of the function write a main program that asks for a natural number >2 and tells whether it is prime or not.

   b) Write a program that asks the user to enter a natural number >2 (say, n) and uses that function to find all the prime numbers that are >2 and $\leq n$.

   c) Write an alternative program for section b) without boolean variables that uses the command `exit()` to abandon the function if a divisor is found.

Additional exercises

1. Write a program that asks the user to enter the coefficients a_0, a_1,... a_n of a polynomial (see ex. 5 of this slide) and uses the Newton-Raphson method (see ex. 2 of this slide) to obtain a zero of the polynomial; that is, a solution of the equation $a_0+a_1x+\ldots a_nx^n = 0$. To calculate the value of the polynomial and its derivative at a point x the program should use a function (to which it should pass the value of x and the list of coefficients).

2. Write a program that plays repeatedly the "rock", "paper" or "scissors" game described in this slide until the user enters a letter different from r, p or s. The main program should
   a) ask the user to enter one letter;
   b) if the letter is r, p or s call a function that receives this letter, chooses randomly one of the words "rock", "paper" or "scissors", writes it, tells the result of the play, and returns 1, 0 or –1 depending on the user having won, tie or lost;
   c) repeat the steps a) and b) until the user enters a different letter, and then write the net number of plays won by the user (wins minus losses).

Module NumPy

$$\begin{pmatrix} 0 & \pi/4 & \pi/2 & 3\pi/4 & \pi \end{pmatrix}$$

```
import numpy
angles = [0,numpy.pi/4,numpy.pi/2,3*numpy.pi/4,numpy.pi]
angles = numpy.array(angles) # type ndarray: same type elements
cos_angles = numpy.cos(angles) # a 1D-array is a vector
print(cos_angles)
```

$$\begin{pmatrix} 1 & 1/\sqrt{2} & 0 & -1/\sqrt{2} & -1 \end{pmatrix}$$

```
print(cos_angles[2]) # cos_angles[2] takes the value 0
print(len(angles)) # number of elements of vector angles
angles[:2] = 5.0 # the first 2 elements of angles are set to 5.0
matrix = numpy.array([[1,2,3],[4,5,6]]) # a 2D-array is a matrix (a table)
print(matrix)
```

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

```
print(matrix[0,2]) # matrix[0,2] takes the value 3
print(len(matrix)) # number of rows of matrix: 2
```

Array attributes

```
print(matrix.ndim) # dimensionality of the array(2 for matrices, 1 for vectors)
print(angles.size) # total number of elements of angles: 5
print(matrix.size) # total number of elements of matrix: 6
nfil,ncol = matrix.shape # number of elements in each dimension: 2, 3
print(nfil,ncol)
```

Array slices

```
x = numpy.linspace(-1,1,21) # the content of vector x is:
# [-1. -0.9 -0.8 ... -0.2 -0.1 0.0 0.1 ... 0.6 0.7 0.8 0.9 1.]
#   0    1    2  ...   8    9  10  11  ... 16  17  18  19  20

x_slice = x[8:18:2] # the content of x_slice is [-0.2 0.0 0.2 0.4 0.6]
x_slice[0] = 5.0 # x[8] is also modified!!! (in contrast with list slices!)
print(x) # [-1. -0.9 ... 5.0 -0.1 0.0 0.1 ... 0.9 1.]

# slices of matrices:
a3x3matrix = numpy.array([[1,2,3],[4,5,6],[7,8,9]])
a3x3matrix_slice = a3x3matrix[:2,:2] # [[1,2],[4,5]]
```

$$\begin{pmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 7 & 8 & 9\end{pmatrix}$$

$$\begin{pmatrix}1 & 2\\ 4 & 5\end{pmatrix}$$

Copying arrays

```
# Like with lists:
y = x # both refer (point) to the same memory positions

# to create a new array with the same content (a copy):
x_copy = numpy.array(x)
```

Creating vectors

```
v = numpy.linspace(start, stop, num) # stop is included
# num is optional (default = 50)
v = numpy.zeros(num)
v = numpy.ones(num)

# Entering the elements from the keyboard in a single line:
v_line = input('Enter the components of the vector and press Enter: ')
v_list = v_line.split()
v_array = numpy.float_(v_list) # numpy.array would generate an array of strings

# Entering the elements from the keyboard in separate lines:
v_comp = input('Enter the first component and press Enter: ')
v_list=[]
while v_comp != '':
    v_list.append(v_comp)
    v_comp = input('Enter another component (press Enter to end): ')
v_array = numpy.float_(v_list)

# or, if the number of components is known:
v = numpy.zeros(num)
for i in range(num):
    v[i] = float(input('enter the {}th component of the vector: '.format(i)))
#   This does NOT work (input() only accepts one string):
#   v[i] = float(input('enter the',i,'th component of the vector: '))
```

Creating matrices

```
mat = numpy.zeros((nfil,ncol))
mat = numpy.ones((nfil,ncol))


# Entering the elements from the keyboard one by one in rows:
mat = numpy.zeros((nfil,ncol))
for i in range(nfil):
    for j in range(ncol):
        mat[i,j] = float(input('enter the element {},{}: '.format(i,j)))
        print('(press Enter after each element)')
```

$$\begin{pmatrix} mat[0,0] & \cdots & mat[0,j] & \cdots & mat[0,ncol-1] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ mat[i,0] & \cdots & mat[i,j] & \cdots & mat[i,ncol-1] \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ mat[nfil-1,0] & \cdots & mat[nfil-1,j] & \cdots & mat[nfil-1,ncol-1] \end{pmatrix}$$

Operating with vectors

```
import numpy
v1 = numpy.linspace(1,5,5) # [1. 2. 3. 4. 5.]
v2 = numpy.linspace(6,10,5) # [6. 7. 8. 9. 10.]
v3 = v1 + v2 # [7. 9. 11. 13. 15.]
v4 = v2 - v1 # [5. 5. 5. 5. 5.]
v5 = v1 * v2 # [6. 14. 24. 36. 50.] NOT scalar product
v6 = v2 / v1 # [6. 3.5 2.66666667 2.25 2.]
v7 = v1 * 3 # [3. 6. 9. 12. 15.] Similar for +, -, /, //,% and **
v8 = numpy.sqrt(v1) # [ 1. 1.41421356 1.73205081 2. 2.23606798] Universal function
# other ufuncs: cos, sin, tan, arcsin, arccos, arctan, log (neperian), log10 (decimal)
# complex numbers: real, imag, conj, angle, abs (math functions do not work with arrays)
# see https://docs.scipy.org/doc/numpy/reference/ufuncs.html
s1 = numpy.min(v1) # 1.0 Also as a methode: s1=v1.min()
s2 = numpy.max(v1) # 5.0
s3 = numpy.argmin(v1) # 0
s4 = numpy.argmax(v1) # 4
s5 = numpy.sum(v1) # 15.0
s6 = numpy.prod(v1) # 120.0
s7 = numpy.mean(v1) # 3.0
s8 = numpy.var(v1) # variance: 2.0
s9 = numpy.std(v1) # standard deviation: 1.41421356237
```

Basic exercises

1. Create a vector with 21 Celsius temperatures uniformly distributed between –50 and 50, convert them to absolute temperatures in kelvins, and write the resulting vector of absolute temperatures.

2. Create a vector with 11 angles in radians uniformly distributed between 0 and π, calculate their sines, their cosines, and the angle values in degrees, and
   a) write the resulting 3 vectors, each one in a new line;
   b) write the resulting 3 vectors as a 3-column table, with each angle, its sine and its cosine in a new line (you don't need to convert them into a matrix).

3. Write a program to create 100 random real numbers between 20 and 80 and calculate their mean and their standard deviation.
   b) In a first version of the program use the function random.uniform(20,80) to generate each random number. Write the array, the mean and the standard deviation. Run several times the program to see that the results differ from one execution to the other.
   c) In a second version of the program use the function numpy.random.uniform(20,80,100) to generate directly a vector with the 100 random numbers.

Basic exercises

4. Write a program that asks for the *number of students* that have attended an exam and the marks (out of 10) they have obtained in the theory and problem parts, that weight 40% and 60% respectively. Save this data in 2 vectors. Then the program should:
   a) write a table with 3 columns containing the theory marks, the problem marks and the total marks;
   b) write the maximum, minimum and mean total marks;
   c) write the position in the list (starting with 1) of the exam with the maximum total mark.

5. A teacher has a pile of exams and wants to do the list of marks. The exam has two parts: theory (40%) and problems (60%). Write a program that asks for the surname of each student and his marks (out of 10) in the two parts. After the last one the user should press Enter to finish the data input. Then the program should write a table with 3 columns containing the theory, problems and total marks, and the name of the student with the highest mark.

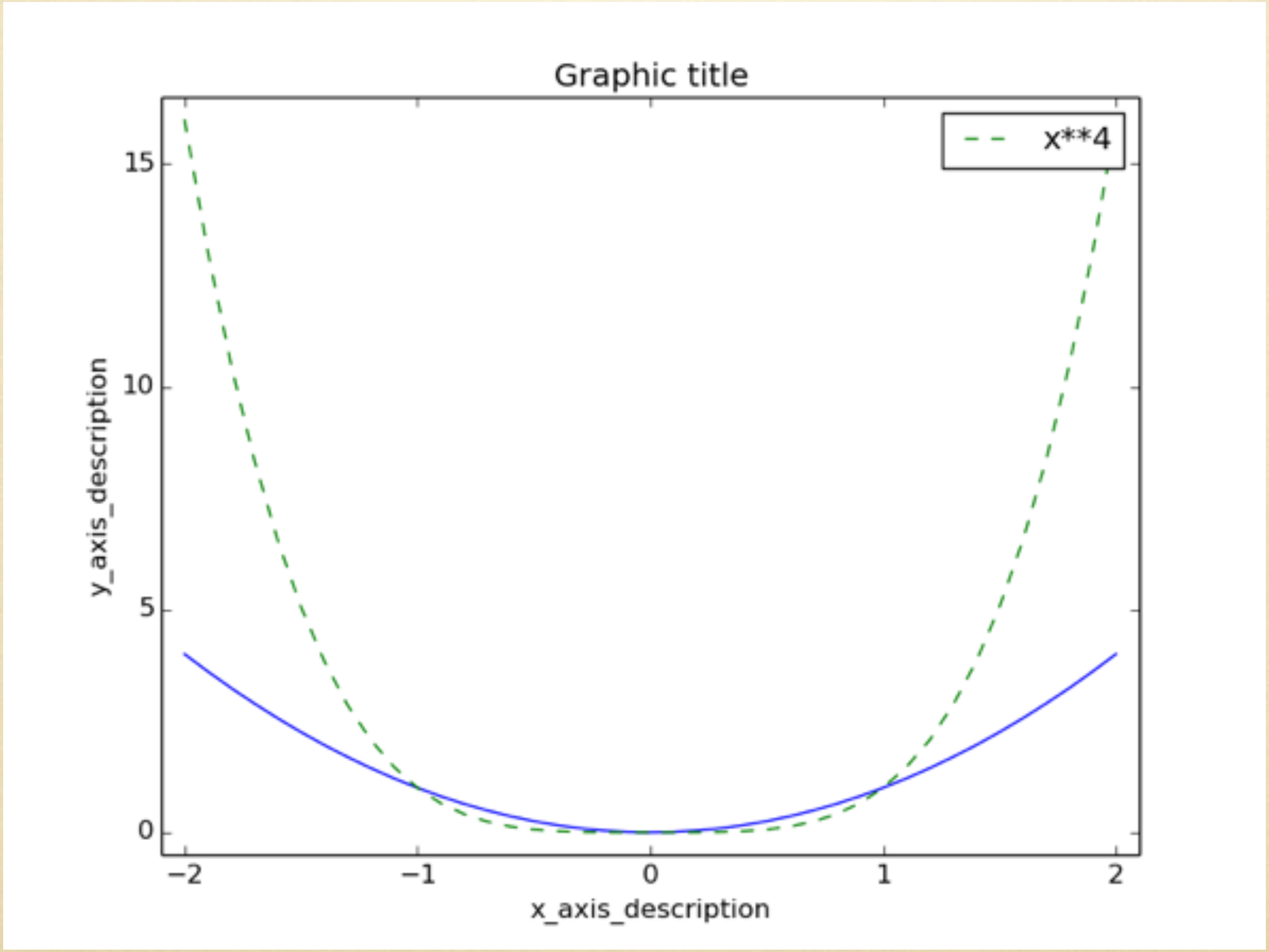
6. Modify the program of exercise 3 of class 8 (this slide) so that vectors are used instead of lists for the concentrations and the electromotive forces.

Graphics with Matplotlib

- http://matplotlib.org/api/pyplot_api.html#matplotlib.pyplot.plot

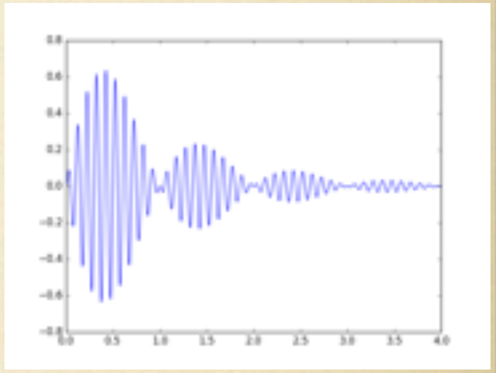
```
import numpy, matplotlib.pyplot as plt
x_data = numpy.linspace(-2,2,41)
y_data = x_data**2 # lists of numbers can also be plotted
plt.plot(x_data, y_data) # if x_data is omitted 0,1,...41 is taken
plt.xlabel('x_axis_description')
plt.ylabel('y_axis_description',size='x-large')
plt.title('Graphic title')
plt.xlim(-2.1, 2.1) # redefine the x axis limits (xmin and xmax)
plt.ylim(-0.5, 16.5) # redefine the y axis limits (ymin and ymax)
y2_data = y_data**2 # we calculate the data for a 2nd curve to be added
plt.plot(x_data, y2_data,'g--', label='x**4') # green dashed line; x,o…
plt.legend() # add a framed legend with the label x**4
# plt.legend(frameon=False): no-framed legend
plt.savefig('filename') # if we want to save the graphic to filename.png
# plt.savefig('filename', format='fmt') with 'fmt'='jpg','pdf','eps'...
# plt.savefig('filename.pdf') also works
plt.show() # to make the graphic appear (in a separated window)
# Close the graphic window to continue (it may be behind the terminal).
# If present, plt.savefig must be placed before plt.show()
```

Graphics with Matplotlib

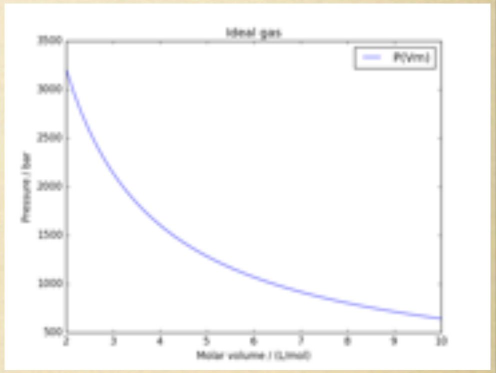


Basic exercises

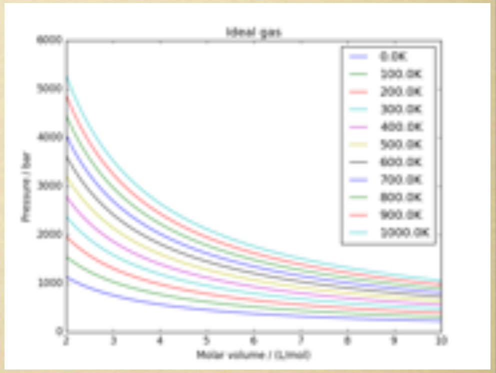
1. Write a program to create a graphic representation of the function $f(x) = \sin(\pi x)\sin(20\pi x)e^{-x}$ for x values between 0 and 4. The number of points should be a datum to be entered by the user. Test it with 50 points (the linspace default) and with 400 points. ¿Why do they look so different?



2. Write a program to represent the pressure in pascals of an ideal gas in terms of the molar volume ($V_m = V/n$) for V_m between 2 and 10 L/mol at 500ºC. Label the axes, show the legend, add the title "Ideal gas", and save the graphic to disk.



3. Modify the previous program so that it represents, in the same graphic, the P *vs* V_m curves corresponding to 11 equi-spaced temperatures between 0 and 1000 ºC. The legend should show the Celsius temperature of each curve. *Hint*: use `for t in numpy.linspace(0,1000,11):` to generate the 11 curves.



Basic exercises

4. A plane curve can be defined by expressing the coordinates (x, y) of each point of the curve in terms of a parameter t. For example, the equations $x=\cos(t)$ and $y=\sin(t)$ define a circumference of unit radius. To obtain the points (x, y) of the circumference, you have to give values to the parameter t between 0 and 2π.

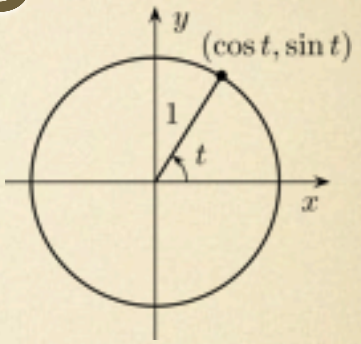
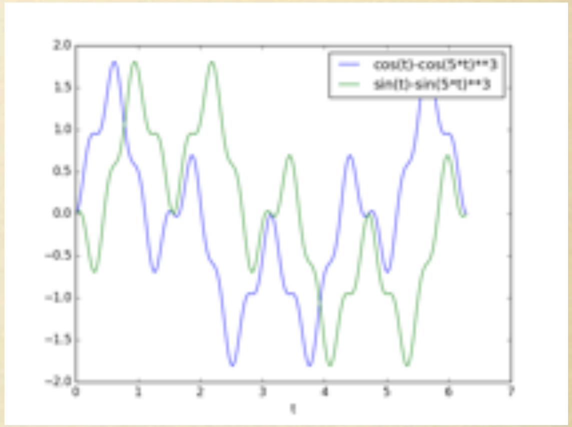
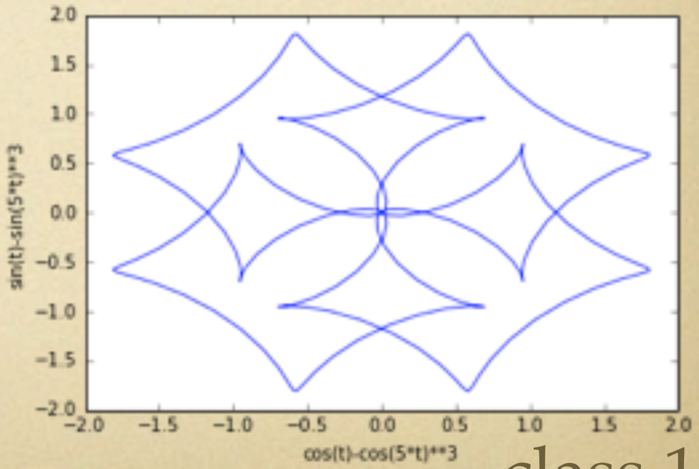


Figure by Gustavb under license CC BY-SA 3.0

a) Write a program that calculates the values of $x = \cos(t)-\cos^3(5t)$ and $y = \sin(t)-\sin^3(5t)$ corresponding to 200 values of t equally-spaced between 0 and 2π, and represents the curves $x(t)$ and $y(t)$. The axes should be labelled accordingly. *Hint*: $\cos^3(5t)=(\cos(5t))^3$



b) Modify the program made in a) so that it represents the curve formed by the points (x, y) and saved the graph to a file named figure.png

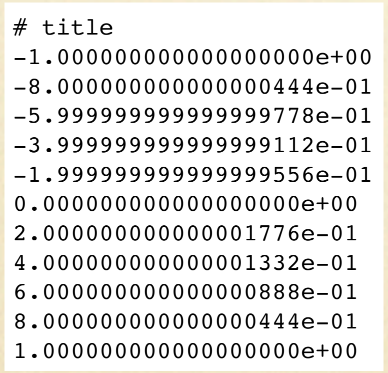


Output files (NumPy)

- To save a 1D-array of numbers (or a list of numbers) to a disk file:

```
array = numpy.linspace(-1,1,11)
numpy.savetxt('file.out',array,header='title') # file.out is created
# header='title' (optional) to write '# title' at the first line
# If a list of numbers is saved it is converted to an array before saving it
```

  - Content of file.out:



```
# title
-1.000000000000000000e+00
-8.000000000000000444e-01
-5.999999999999999778e-01
-3.999999999999999112e-01
-1.999999999999999556e-01
0.000000000000000000e+00
2.000000000000001776e-01
4.000000000000001332e-01
6.000000000000000888e-01
8.000000000000000444e-01
1.000000000000000000e+00
```

- For a 2D-array; e.g., a 2-column table with 100 (x, $ln(x)$) data pairs:

```
import numpy
xy_array = numpy.zeros((100,2)) # an array with 100 x 2 zeros is created
xy_array[:,0] = numpy.linspace(1,100,100) # 1st column of xy_array
xy_array[:,1] = numpy.log(xy_array[:,0]) # 2nd column of xy_array
numpy.savetxt('file.out',xy_array)
```

Input files (NumPy)

- To input a file with one real (or complex) number in each line and generate a 1D-array with those data:

a numpy array of real (or complex) numbers is created

```
inp_arr = numpy.loadtxt('file.inp') # file.inp must exist
# Lines with '#' as the first character are skipped (e.g., a title line)
print(inp_arr)
```

- If the file contains *m* lines with *n* numbers in each one `numpy.loadtxt` generates an *m* x *n* 2D-array.

Basic exercises

1. Write a program to create a 1D-array with the values of the function $f(x) = \sin(\pi x)\sin(20\pi x)e^{-x}$ for 400 values of x between 0 and 4 (see this slide), and save that array to a file named 'FID'.
2. Write a new program to input the data saved in the file FID and plot them by using the sequence 0, ... 399 as horizontal axis.
3. Write a program to create a 2D-array with 400 rows and 2 columns. Each row should contain a couple of values $(x, f(x))$ as defined in exercise 1. Save this array to a file named 'FID2'
4. Write a new program to input the data saved in file FID2 and plot them. The program should also calculate the mean of the $f(x)$ values and write it. What approximate value should take this mean?

Output files (std. python)

- Writing any string (that could contain text, numbers, ...) in a file:

```
out = open('file_out', mode='w') # file_out is created; write mode
out.write('This is the text for the 1st line') # the argument must be 1 string
out.write('\n' + str(273.15)) # '\n' to start writing in a new line
out.write(' ' + str(373.15)) # ' ' to separate from the previous string
out.write(' ' + str(473.15) + '\n') # '\n' to end the line
for temp in range(-50,60,10): # to write a 2-column list:
    out.write('{:6d}{:6d}\n'.format(temp,temp**2)) # 6 positions for each integer
out.close()
```

- Content of file.out:

```
This is the text for the 1st line
273.15 373.15 473.15
   -50  2500
   -40  1600
   -30   900
   -20   400
   -10   100
     0     0
    10   100
    20   400
    30   900
    40  1600
    50  2500
```

Input files (std. python)

- Reading files with mixed information (text, numbers, ...):

```
# Reading single lines:
inp = open('file_inp') # file_inp must exist; read mode
text_line = inp.readline() # the 1st line is read as a string
text_line = text_line.strip() # to delete \n at the end of the line
text_line_list = text_line.split() # no .strip() is needed is .split() is used
...
inp.close()


# Reading all the lines of the file:
inp1 = open('file1_inp')
for text_line in inp1:
    word_list = text_line.split() # no .strip() is needed
#   also word1,word2 = text_line.split() if each line contains, say, 2 words
    print(word_list)
inp1.close()
```

Basic exercises

5. a) Write a program that:

i) asks for the number of atoms of a molecule and save it in the first line of a file named `geometry.cart`,

ii) leaves a blank line in the file,

iii) asks for the atomic symbol and the *x*,*y*,*z* cartesian coordinates in Å of each atom and save them in consecutive lines of the file.

Hint: For the molecule HCN the file could be:

```
3

H  0  0 -1
C  0  0  0
N  0  0  1.5
```

b) Write a program that opens the file `geometry.cart` and

i) reads the first line and assign the number of atoms to an integer variable named 'numat',

ii) reads the lines containing the information of each atom and generates a list named 'symbols' with the atomic symbols (check this point before continuing),

iii) creates a nat x 3 matrix 2D-array of real numbers named 'coordinates' and saves the cartesian coordinates of each atoms into each line of that array,

iv) asks for the indexes of 2 atoms and calculates the distance between them. *Hint*: the distance between 2 points of cartesian coordinates (x_1,y_1,z_1) and (x_2,y_2,z_2) is the norm of the vector connecting them: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$

Basic exercises

6. a) Write a program that asks the user for the number of students having attended an exam; then it should ask for the surname and the mark (out of 10) of each student and write these data in a file named 'marks' as a two-column table with the header '`Surname     Mark`'.

   b) Write a second program that reads the data contained in the file 'marks' and writes in a file named 'passed' a two-column list with the names and marks of the students having passed the exam with the same header.

Additional exercises

1. (Have a look to the additional exercise 1 of class 7 in this slide) Write a program that asks the user for the surname and the mark (out of 10) of the students who have attended an exam and writes these data in a file as a two-column table. Assume that the user does not know the total number of students, so that the program should:
   a) open a file in write mode;
   b) ask for the surname of the first student;
   c) repeat the following actions until the user press enter without entering any text:
      i) ask for the mark of the student;
      ii) write in the file a line with the student's name and its mark;
      iii) ask for a new surname;
   d) close the file.
2. Write a program that reads the surnames and marks saved in the file created in the preceding exercise and writes in the text terminal a 2-column list including only the surname and the mark of the students having passed the exam. You can add the heading 'Surname   Mark'.

Additional exercises

3. Write a program to extract from the file 'CH3CN.log' –an output of a quantum-chemistry calculation– the lines starting with the text '` FINAL RHF ENERGY IS`' and save to a 1D-array named `energies` the numbers located to the right of that text. The program should:
a) Write in a file named 'energies' a one-column list with those energies headed by a line with the text '`# Energies/hatree`'.
b) Make a plot to show the evolution of the energy along the calculation. Label the *y*-axis with 'Energies/hatree'. Note how are the *y*-data expressed.