Introduction to python language (solutions of some exercises)

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 March 4th, 2020)

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

Basic exercises

1. Write a program that asks the user to enter one of this 3 words: "rock", "paper" or "scissors", and prints in the text terminal the chosen word.

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 € 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

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, gives the integer quotient without decimals.
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).

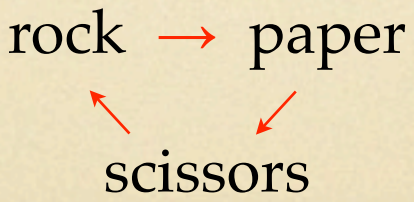
Basic exercises

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").



5.

```
import random

# The user enter his choice:
user_choice = input('Enter r for rock, p for paper or s for scissors: ')
if user_choice != 'r' and user_choice != 'p' and user_choice != 's':
    print('This is not a valid option. Bye!')
    exit()

# The computer chooses:
computer_choice = random.choice('rps')
# The computer choice is shown:
if computer_choice == 'r':
    print('Computer: I chose rock')
elif computer_choice == 'p':
    print('Computer: I chose paper')
elif computer_choice == 's':
    print('Computer: I chose scissors')

# The user and computer choices are compared:
if user_choice == computer_choice:
    print('Tie')
elif user_choice=='r' and computer_choice=='p' or user_choice=='p' and
computer_choice=='s' or user_choice=='s' and computer_choice=='r':
    print('Computer: I won')
else:
    print('Computer: You won')
```

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).

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).

```
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 we want the program to continue after determining whether the number is prime or not:

```
prime = True # prime is a boolean type variable
for div in range(2,n):
    res = n % div
    if res==0:
        prime = False
        break # the iterations are interrupted
if prime:
    print(n,'is a prime number')
else:
    print(n,'is not a prime number')
# the program could continue
```

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).

```
num = int(input('Enter a natural number > 1: '))
writediv = input('Do you want the list of divisors to be written? (y/n) ')
sum = 0
for div in range(1,num):
    if num % div == 0 :
        sum = sum + div
        if writediv=='y':
            print (div,'is a divisor')
if num == sum:
    print(num,'is perfect.')
else:
    print(num,'is not perfect.')
```

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 $p!$=1x2...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.

```
p = int(input("Enter the number of pockets: "))
n = int(input("Enter the number of notes: "))
if n>p:
    print ("The number of notes cannot be be larger than that of pockets")
else:
    numerator=1
    for i in range(p-n+1,p+1): # or (p,p-n,-1)
        numerator = numerator*i
    denominator=1
    for i in range(1,n+1):
        denominator = denominator*i
    combinations = numerator//denominator
    print ("There are",combinations,"ways of keeping the notes.")
```

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 $\mathcal{E}$, 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 $\mathcal{E}$, and write this result. For comparison, write also the value of $\exp(x)$ calculated with the exp() function of module math.

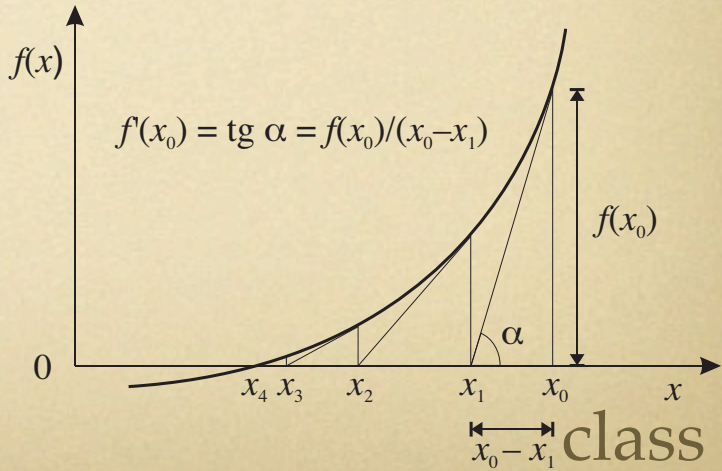
Basic exercises

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 $|f(x_1)| \leq \varepsilon$.
   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.



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.

2.

```
import random
# We initialize the counters
n_win_user = 0
n_win_computer = 0
n_tie = 0

# The user enter his first choice:
user_choice = input('Enter r for rock, p for paper, s for scissors (press Enter to finish)')

# Start of repetitions:
while user_choice=='r' or user_choice=='p' or user_choice=='s':
#   The computer chooses:
    computer_choice = random.choice('rps')
    if computer_choice=='r':
        print('Computer: I chose rock')
    elif computer_choice=='p':
        print('Computer: I chose paper')
    elif computer_choice=='s':
        print('Computer: I chose scissors')
#   Let's compare the choices:
    if user_choice==computer_choice:
        print('Tie')
        n_tie = n_tie + 1
    elif user_choice=='r' and computer_choice=='p' or user_choice=='p' and computer_choice=='s'
or user_choice=='s' and computer_choice=='r':
        print('Computer: I won')
        n_win_computer = n_win_computer + 1
    else:
        print('Computer: You won')
        n_win_user = n_win_user + 1
#   The user enters a new choice and we repeat the game:
    user_choice = input('Enter a new choice: ')

# The final scores are writen:
print('The final scores are:')
print('  We tied',n_tie,'times')
print('  I won',n_win_computer,'times')
print('  You won',n_win_user,'times')
```

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). *Hint*: `dna.count(subs)` returns the number of occurrences of substring `subs` in the string `dna`.

   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).

3. d)

```
develop_form = input("Enter the developed formula of an organic molecule (only C,
H, O, N and S atoms with no numbers) ")
# We verify that only CHONS atoms have been entered
for char in develop_form:
    if not(char in 'CHONS'):
        print("Only C, H, O, N and S atoms with no numbers are accepted. Bye!")
        exit()
# We write the empirical formula
empir_form = ''
for char in 'CHONS':
    char_num = develop_form.count(char)
    empir_form = empir_form + char + str(char_num)
print('Empirical formula:',empir_form)
# We calculate the molecular mass
c = develop_form.count('C')
h = develop_form.count('H')
o = develop_form.count('O')
n = develop_form.count('N')
s = develop_form.count('S')
molec_mass = c*12 + h*1 + o*16 + n*14 + s*32
print('Molecular mass =',molec_mass)
# We write the simplified empirical formula
simpl_empir_form = ''
for char in 'CHONS':
    char_num = develop_form.count(char)
    if char_num > 1:
        simpl_empir_form = simpl_empir_form + char + str(char_num)
    elif char_num == 1:
        simpl_empir_form = simpl_empir_form + char
print('Simplified empirical formula:',simpl_empir_form)
```

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, tapping 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'.

```
text_str = input("Write a text and press return: ")
text_list = text_str.split()
numWords = len(text_list)
print("The number of words of the text is",numWords)
numCharacters = 0
for word in text_list:
    numCharacters = numCharacters + len(word)
print("The number of characters of the text is",numCharacters)
numWords_a = 0
for word in text_list:
    if 'a' in word:
        numWords_a = numWords_a + 1
percWords_a = numWords_a/numWords*100
print("The percentage of words containing letter a is",percWords_a)
```

5. Write a program that

a) asks the user to enter the coefficients a_0, a_1,... a_n of a polynomial:
$P(x) = a_0+a_1x+... a_nx^n = \sum_{i=0}^{n} a_ix^i$;

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 $P'(x) = a_1+2a_2x+... na_nx^{n-1} = \sum_{i=1}^{n} ia_ix^{i-1}$.

Test: For x=2 the polinomial $1 + x^2 + 3x^4$ takes the value 53 and its derivative takes the value 100.

```
coef_str = input("Enter the coefficients separated by blank spaces: ")
coef_list = coef_str.split()
n = len(coef_list)-1
x=float(input("Enter the value of x: "))
pol=0
for i in range(n+1):
    coefi = float(coef_list[i])
    pol = pol + coefi*x**i
print("P({}) = {}".format(x,pol))
dpol=0
for i in range(1,n+1):
    coefi = float(coef_list[i])
    dpol = dpol + i*coefi*x**(i-1)
print("P'({}) = {}".format(x,dpol))
```

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 press enter 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.

Solution

1.
```
exam_list = []
name = input('Enter the name of the first student (press enter to finish): ')
while name != '':
    mark = float(input('enter the mark of the student: '))
    name_mark = [name,mark]
    exam_list.append(name_mark)
    name = input('enter the name of another student: ')

print()
print('Full list')
print('{:12}{}'.format('Surname','Mark'))
for name_mark in exam_list:
    print('{:12}{}'.format(name_mark[0],name_mark[1]))
# a fixed length of 12 spaces is used for each surname
print()
print('Passed list')
print('{:12}{}'.format('Surname','Mark'))
for name_mark in exam_list:
    if name_mark[1] >= 5:
        print('{:12}{}'.format(name_mark[0],name_mark[1]))
```

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° 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° should be assigned within the function. *Test*: for c = 1.00 x 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 instruction `exit()` to abandon the function if a divisor is found.

Solutions

2.
```
def emf(c):
    import math
    R = 8.314 # in J*K-1*mol-1
    F = 96485 # in C/mol
    n = 2
    Eo = 0.984
    T = 25 + 273.15
    E = Eo - R*T/(n*F) * math.log(4*c**3)
    return E
c = float(input('Enter the concentration of ZnCl2 in mole/L: '))
E = emf(c)
print('FEM at 25ºC = {:.2f} V'.format(E))
```

3.
```
def emf_list(c_list):
    import math
    R = 8.314 # in J*K-1*mol-1
    F = 96485 # in C/mol
    n = 2
    Eo = 0.984
    T = 25 + 273.15
    E_list = []
    for c in c_list:
        E = Eo - R*T/(n*F) * math.log(4*c**3)
        E_list.append(E)
    return E_list
c_list = []
c = input('Enter the first concentration of ZnCl2 in mole/L (press Enter to finish): ')
while c != '':
    c_list.append(float(c))
    c = input('Enter another concentration of ZnCl2 in mole/L: ')
E_list = emf_list(c_list)
print('FEMs at 25ºC in V:{}'.format(E_list)) # ...{:.2f}'.format(E_list)) does not work
```

Solutions

4. b)

```
def fprime(num):
    i = 2
    prime = True
    while prime and i<n:
        if n%i == 0:
            prime = False
        i = i+1
    return prime
nmax = int(input('Enter a natural number >2: '))
print('Prime list from 3 to {}:'.format(nmax))
for n in range(3,nmax+1):
    if fprime(n):
        print(n)
```

4. c)

```
def fprime(num):
    for d in range(2,num):
        res = num%d
        if res==0:
            return 0
            exit() # exit from the function
    return num
n=int(input("Enter a natural number >2: "))
for i in range (2,n+1):
    prime = fprime(i)
    if prime!=0:
        print(prime)
```

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 $a_0+a_1x+... a_nx^n = 0$. 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 that 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 word, and then write the net number of plays won by the user (wins minus loss).

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 one after the other;
   b) write the resulting 3 vectors with each angle, its sine and its cosine in a different line.

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 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 to generate directly the array of 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.

Solutions

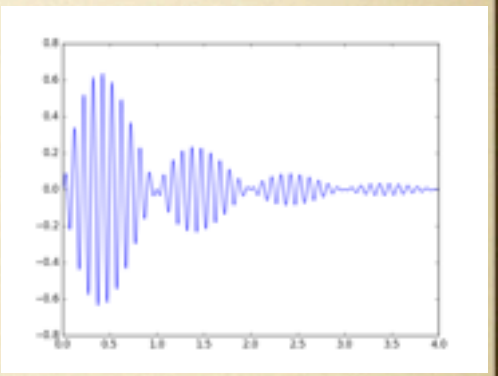
4.
```
import numpy as np

num = int(input('Enter the number of exams: '))
t_vector = np.zeros(num)
p_vector = np.zeros(num)
for i in range(num):
    t_vector[i] = float(input('enter the theory mark of one exam: '))
    p_vector[i] = float(input('enter the problem mark of the exam: '))

tot_vector = 0.4*t_vector + 0.6*p_vector

print('Theor Probl Total')
for i in range(num):
    print('',t_vector[i], ' ', p_vector[i], ' ', tot_vector[i])
print('Maximum mark: {:.1f}'.format(tot_vector.max()))
print('Minimum mark: {:.1f}'.format(tot_vector.min()))
print('Mean mark: {:.1f}'.format(tot_vector.mean()))
maxMarkSt = tot_vector.argmax() + 1
print('The student with the highest mark is the {}th'.format(maxMarkSt))
```

Solutions

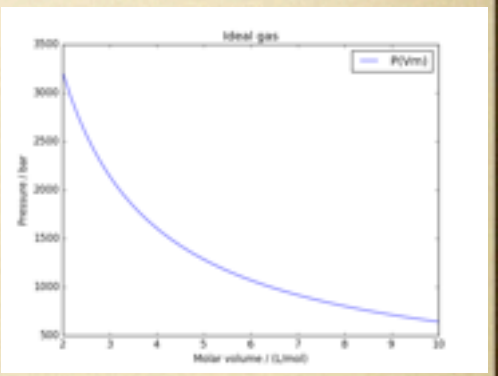
5.
```
import numpy as np

name_list = []
t_list = []
p_list = []
name = input('Enter the surname of the first student: ')
while name != '':
    name_list.append(name)
    t = input('enter his theory mark: ')
    t_list.append(float(t))
    p = input('enter his problem mark: ')
    p_list.append(float(p))
    name = input('Enter the name of another student: ')
    print('(press Enter to finish)')

t_vector = np.array(t_list)
p_vector = np.array(p_list)
tot_vector = 0.4*t_vector + 0.6*p_vector

print('Surname    Theor Probl Total')
num_exams = tot_vector.size
for i in range(num_exams):
    print('{:12}{:.1f}   {:.1f}   {:.1f}'.format
(name_list[i],t_vector[i], p_vector[i],tot_vector[i]))
maxMarkSt = tot_vector.argmax()
print('The student with the highest mark is',name_list[maxMarkSt])
```

Basic exercises

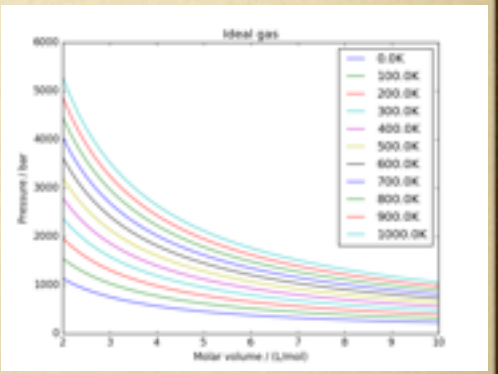
1. Write a program to create a graphic representation of the function $f(x) = \sin(\pi x)\sin(20\pi x)e^{-x}$ for x values between 0 and 4. The number of points should be a datum to be entered by the user. Test it with 50 points (the linspace default) and with 400 points. ¿Why do they look so different?



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 $P(V_m)$ curves corresponding to 11 temperatures equispaced 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.

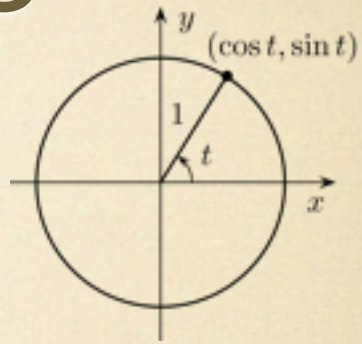
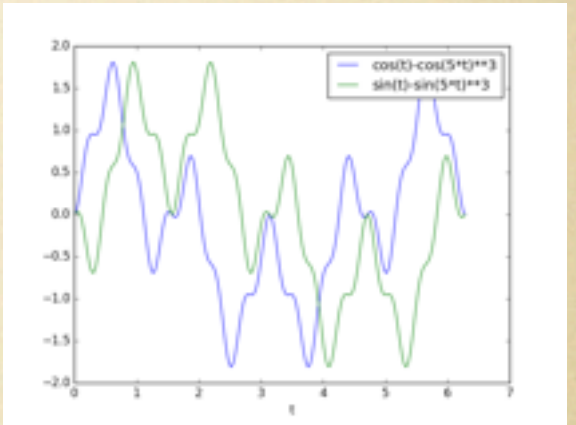
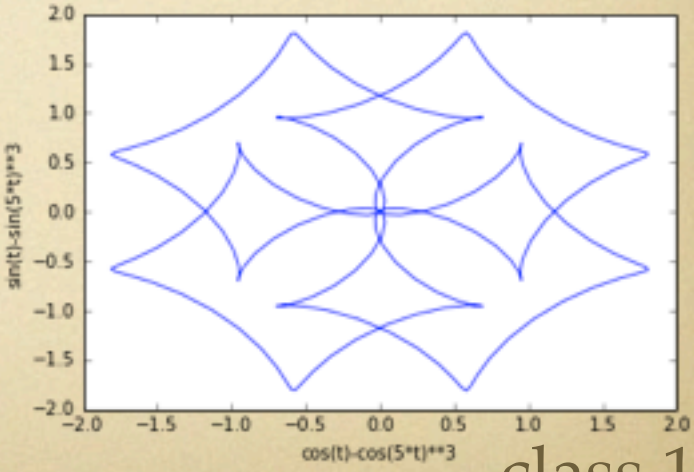


Figure by Gustavb under license CC BY-SA 3.0

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



Solutions

1.
```
import numpy as np, matplotlib.pyplot as plt
npoints = int(input('Enter the number of points :'))
x_array = np.linspace(0,4,npoints)
y_array = np.sin(np.pi*x_array)*np.sin(20*np.pi*x_array)*np.exp(-x_array)
plt.plot(x_array,y_array)
plt.show()
```

2.
```
import numpy as np, matplotlib.pyplot as plt
Vm_array = np.linspace(2,10,50)
R = 8.3145
T = 273.15 + 500
P_array = R*T/Vm_array
plt.plot(Vm_array,P_array,label='P(Vm)')
plt.xlabel('Molar volume / (L/mol)')
plt.ylabel('Pressure / bar')
plt.legend()
plt.title('Ideal gas')
plt.savefig('plotPVt500')
plt.show()
```

Solutions

3.

```
import numpy as np, matplotlib.pyplot as plt
Vm_array = np.linspace(2,10,50)
R = 8.3145
for t in np.linspace(0,1000,11):
  T = 273.15 + t
  P_array = R*T/Vm_array
  plt.plot(Vm_array,P_array,label='{} K'.format(t))
# or: plt.plot(Vm_array,P_array,label=str(t)+' K')
plt.xlabel('Molar volume / (L/mol)')
plt.ylabel('Pressure / bar')
plt.title('Ideal gas')
plt.legend()
plt.show()
```

4. a)

```
import numpy as np, matplotlib.pyplot as plt
t_vec = np.linspace(0,2*np.pi,200)
x_vec = np.cos(t_vec)-np.cos(5*t_vec)**3
y_vec = np.sin(t_vec)-np.sin(5*t_vec)**3
plt.plot(t_vec,x_vec,label="cos(t)-cos(5*t)**3")
plt.plot(t_vec,y_vec,label="sin(t)-sin(5*t)**3")
plt.xlabel("t")
plt.legend()
plt.show()
```

b)

```
import numpy as np, matplotlib.pyplot as plt
t_vec = np.linspace(0,2*np.pi,200)
x_vec = np.cos(t_vec)-np.cos(5*t_vec)**3
y_vec = np.sin(t_vec)-np.sin(5*t_vec)**3
plt.plot(x_vec,y_vec)
plt.xlabel("cos(t)-cos(5*t)**3")
plt.ylabel("sin(t)-sin(5*t)**3")
plt.savefig("figure.png")
plt.show()
```

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, 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 value should take this mean?

Solutions

1.
```
import numpy as np
x_array = np.linspace(0,4,400)
y_array = np.sin(np.pi*x_array)*np.sin(20*np.pi*x_array)*np.exp(-x_array)
np.savetxt('FID',y_array)
```

2.
```
import numpy as np, matplotlib.pyplot as plt
y_array = np.loadtxt('FID')
plt.plot(y_array)
plt.show() # the graphic window must be closed for the program to continue
```

3.
```
import numpy as np
x_array = np.linspace(0,4,400)
xy_array = np.zeros((400,2))
xy_array[:,0] = x_array
xy_array[:,1] = np.sin(np.pi*x_array)*np.sin(20*np.pi*x_array)*np.exp(-x_array)
np.savetxt('FID2',xy_array)
```

4.
```
import numpy as np, matplotlib.pyplot as plt
xy_array = np.loadtxt('FID2')
plt.plot(xy_array[:,0], xy_array[:,1])
plt.show()
mean = xy_array[:1].mean()
print(mean)
```

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.

Solutions

5. a)

```
out = open('geometry.cart',mode='w')
nat_str = input("Enter the number of atoms of the molecule: ")
out.write(nat_str+'\n'+'\n')
nat = int(nat_str)
for i in range(nat):
    symbol = input("enter the symbol of atom {}: ".format(i))
    coords = input("enter its x,y,x coordinates in Å separated by spaces: ")
    out.write(symbol+' '+coords+'\n')
out.close()
```

b)

```
import numpy
inp = open('geometry.cart')
nat = int(inp.readline())
inp.readline() # we skip the blank line
symbols = []
coords = numpy.zeros((nat,3))
for i in range(nat):
    line = inp.readline()
    line_list = line.split()
    symbols.append(line_list[0])
    for j in range(3):
        coords[i,j] = float(line_list[j+1])
inp.close()
atom1 = int(input('Enter the index of one atom: '))
atom2 = int(input('Enter the index of another atom: '))
difcoords = coords[atom1]-coords[atom2]
distance = numpy.sqrt(numpy.sum(difcoords**2))
print('The distance between atoms',atom1,'and',atom2,'is',distance,'Å')
```

Solutions

6. a)

```
numstd = int(input('Enter the number of students: '))
out = open('marks',mode='w')
out.write('Surname   Mark')
for i in range(numstd):
    name_mark_line = input('Enter the surname and the mark of a student: ')
    name,mark = name_mark_line.split()
    out.write('\n{:10}{:5}'.format(name,mark))
out.close()
```

b)

```
inp = open('marks')
out = open('passed',mode='w')
out.write('Surname   Mark')
inp.readline()
for line in inp:
    name,mark = line.split()
    if float(mark) >= 5:
        out.write('\n{:10}{:5}'.format(name,mark))
```

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'.

Solutions

1.
```
out = open('exam_list',mode='w')
name = input('Enter the surname of the first student (enter space to finish): ')
while name != ' ':
    mark = input('enter the mark of the student: ')
    out.write('{:15}{:6.2f}\n'.format(name,mark))
    name = input('enter the surname of another student: ')
out.close()
```

2.
```
inp = open('exam_list')
print('{:14}{:8}'.format('Surname','Mark')) # we write a table header
for line in inp:
    name,mark = line.split()
    if float(mark) >= 5:
        print('{:15}{:7}'.format(name,mark))
inp.close()
```

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.

```
import numpy, matplotlib.pyplot as plt
inp = open('CH3CN.log')
energies_list = []
for text_line in inp:
    if "FINAL RHF ENERGY IS" in text_line:
        words = text_line.split()
        energies_list.append(float(words[4]))
inp.close()
energies_arr = numpy.array(energies_list)
numpy.savetxt('energies',energies_arr,header='Energies/hartree')
plt.plot(energies_arr)
plt.ylabel('Energies/hartree')
plt.show()
```