EFS Informatique et Société Numérique 1
SCAN - January 2023

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Duration: 1h30
Documents and calculator forbidden

Warning : A program that is **badly indented**, **badly commented** or with the **wrong choice of variable names** will be **penalized** *(up to -1 point)*.

« Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live. » *John F. Woods*

The exercises are independent and can be done in any order.

You are **not allowed** to use any of the built-in python functions on lists or strings (`sum`, `min`, `max`, `index`...) neither slicing. You can use `append`, `len`, and the concatenation operator.

Exercice 1 Code reading (3 pts)

(Q1.1) What is displayed when the following code is executed ?

```
1  def f(x):
2      return x*x
3
4  def fonction(a,b):
5      z = 5
6      r = f(2*a) + f(b) + z
7      return r
8
9  z = 2
10 print(fonction(1,f(z)))
11 print(f(z))
```

Correction:
25
4

(Q1.2) What is displayed when the following code is executed ?

```
1 l = [1,2,5,8]
2 for i in range(1,len(l)):
3     if i % 2 == 0:
4         l[i] = l[i] + l[i-1]
5     else:
6         l[i] = l[i] - l[i-1]
7 print(l)
```

Correction:
[1,1,6,2]

(Q1.3) What is displayed when the following code is executed ?

```
1  def f(a,b):
2      return a+b
3
4  for i in range(4):
5      s = ""
6      for j in range(i+1):
7          s += str(f(i,j))
8      print(s)
```

Correction:
0
12
234
3456

Exercice 2 Code Correction (3 pts)

(Q2.1) The following program contains **3 execution errors** (the code will not run and raise an error) and **3 coding conventions issues** (the code run but does not respect the coding conventions, we do not consider spacing). Identify each error or issue by indicating : the line number, the type of error and the problem. *(e.g. : line 5 (error) the symbol '=' has been used in a comparison, when a '==' should be used.)*

```
1  def compute_average(my_list):
2      ssum = 0
3      for i in range(len(my_list)):
4          ssum += my_list(i)
5      return ssum / len(my_list)
6
7  from math import sqrt
8
9  l = [2,1,3,-1]
10 res = 0
11 def std_dev(my_list):
12     avg = compute_average(my_list)
13     for value in my_list:
14         res = res + (value - avg) ** 2
15     return sqrt(res / len(my_list))
16
17 avg = compute_average(l)
18 std = std_dev(l, avg)
19 diff = []
20 for i in range(len(l)):
21     diff.append(l[i]-l[i+1])
```

Correction:

Ligne 4 : Error : prenthesis instead of brackets for indexing
Ligne 7 : Convention : import in the middle of the code

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

(a) Before the player starts

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

(b) The user has completed the grid

FIGURE 1 – The sudoku puzzle

Ligne 9 : Convention : code in middle of functions
Ligne 14 (or 10) : Convention : res is a global variable (ok si donne comme une erreur)
Ligne 18 : Error : 2 parameters to the function instead of one
Ligne 20 : Error : the range should go up til len(l)-1 to be able to do l[i+1]

Exercice 3 Functional decomposition : sudoku (4pts)

The sudoku is a puzzle which consists of a grid usually made up of nine rows, nine columns, and nine boxes that are separated by thicker, darker lines (as seen in Figure 1). Some of these boxes contain numerals from 1 to 9. To solve the puzzle, a person must fill in all the empty squares without using the same numeral twice in each column, row, or box, and without changing the numerals that are already in the grid. There is a unique answer to a given sudoku and the game is usually created by creating a grid with all the numerals in it and removing a certain number of them.

The goal of this exercise is to propose a functional decomposition of the sudoku game.

In this version of the game, we will ask the user to enter a numeral in one of the square at each turn. If the user enter a numeral that is incorrect, he loses.

We provide you with the following code skeleton that you will use as a basis for you main program :

```
1 grid_to_play, grid_solution = init_grid()
2 # to complete
3
4 while # to complete:
5     # to complete
6
7 # to complete
```

The function `init_grid()` creates the sudoku grid that is shown to the user (`grid_to_play`) and the one containing all the numbers (the solution, `grid_solution`).

Your functional decomposition should respect the following rules :

- Use between 4 and 6 functions in addition to the given function `init_grid()`.
- Only propose the functions that you will use in your main program, we do not ask for intermediate functions that will be used within these functions
- The program should display, depending on the case :
  - "Congratulation, you correctly filled the sudoku grid!"
  - "Your last input was incorrect, you lose the game!"

(Q3.1) Suggest between 4 and 6 functions that you will use in your functional decomposition. For each function, write a short description of what it does and its inputs and outputs. An example is shown below for the provided `init_grid()` function.

```
    init_grid() : initialize the sudoku grid (9x9) and its solution. It
       does not take any parameters and returns the playable grid (with
       only some numbers) and the grid of the solution.
```

(Q3.2) Write the main program of the sudoku game by using the functions you suggested in previous question. Your program should respect the guidelines given above.

```
grid_to_play, grid_solution = init_grid()
lose = False
ended = False
while not lose and not ended:
    x, y, val = choose_number()
    grid_to_play = update_grid(grid_to_play, x, y, val)
    display_grid()
    if grid_full(grid):
        ended = True
    lose = has_error(grid_to_play,grid_solution)

if lose:
    print(f"Your last input was incorrect, you lose the game!")
else:
    print("Congratulation, you correctly filled the sudoku grid!")
```

Exercice 4 Code writing : integrating functions (6.5 pts)

In this exercise, we seek to calculate and study the integral of a function using the method of the rectangles.

The function is numerically defined by two lists `x` and `fx`, where `x` contains the abcissa coordinates and `fx` the corresponding values of the function. The sampling of the points is not necessarily regular : the step between each value of x is not constant. For example the function $f(x) = x^2$ could be represented by the following two lists :

```
x = [-1, 0, 2, 3, 4.5]
fx = [1, 0, 4, 9, 20.25]
```

and is shown graphically in Figure 2 (left).

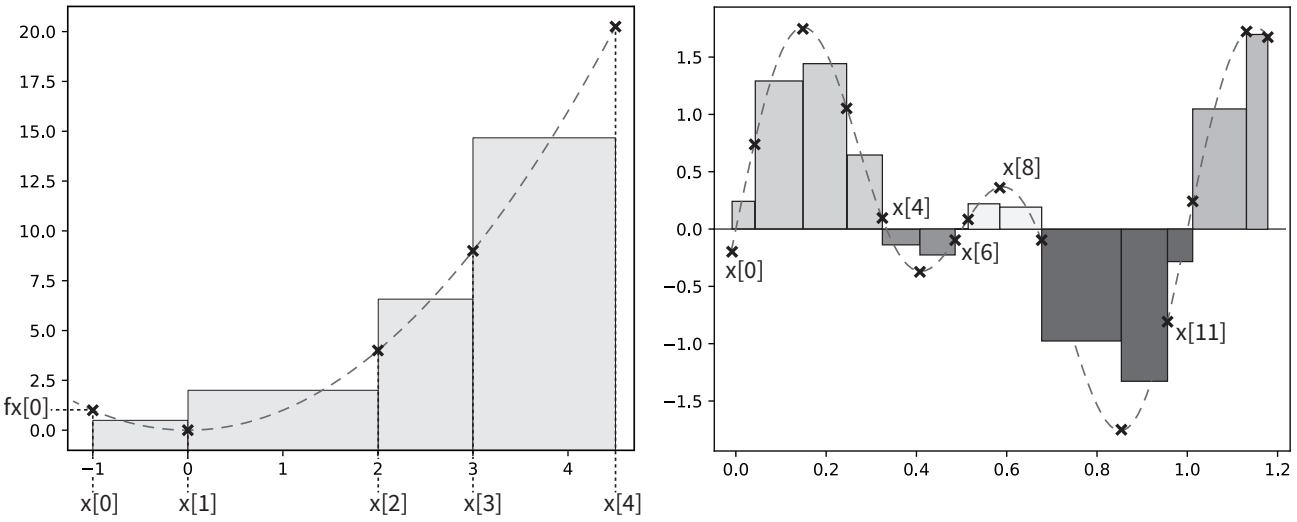

FIGURE 2 – Left : the function $f(x) = x^2$ given in example The height of each rectangle that starts at $\mathtt{x}[i]$ is $\frac{\mathtt{fx}[i]+\mathtt{fx}[i+1]}{2}$. Right : a sinusoidal function that crosses 0 several times (used from Q4.2)

The integral of the curve can be approximated with the rectangle method : the integral is then defined as the sum of the area of each rectangle. With the example function defined above, the integral will be computed as the sum of the 4 rectangles shown in Figure 2 (left). The width of each triangle is the distance between the two sample points while the height is defined as the middle point of the function :

$$\int_{\mathtt{x}[0]}^{\mathtt{x}[n]} f = \sum_i (\mathtt{x}[i] - \mathtt{x}[i-1]) \frac{\mathtt{fx}[i] + \mathtt{fx}[i-1]}{2}$$

With the previous function, the areas of the different rectangles would be [0.5, 4, 6.5, 21.9375] for a total integral of 32.9375.

(Q4.1) Write the function `integral(x,fx)` that takes as parameter the two lists defining the abcissa (x) and values (fx) of the function and that computes the approximate integral of the function using the rectangle method. The function should return the computed integral (32.9375 in previous example) and the list of the areas of all the rectangles ([0.5, 4, 6.5, 21.9375] in previous example).

```
def integral(x,fx):
    liste_rectangles = []
```

```
    integral = 0
    for i in range(1,len(x)):
        width = x[i] - x[i-1]
        height = (fx[i] + fx[i-1])/2
        liste_rectangles.append(width*height)
        integral += width*height
    return integral, liste_rectangles
```

We now want to know where the function changes of sign. The program should give the list of indices i where `fx`$[i]$ has a different sign from `fx`$[i+1]$. For the function shown in Figure 2 (right), the result would be the list `[0,4,6,8,11]`.

(Q4.2) Write a function `change_sign` that takes as parameter the values of the function and that returns all the indices where the function has changed sign (i where `fx`$[i]$ has a different sign from `fx`$[i+1]$).

```
def change_sign(fx):
    indices = []
    for i in range(len(fx)-1):
        if (fx[i] < 0) != (fx[i+1] < 0):
            indices.append(i)
    return indices
```

Having detected where the function changes sign, we now want to compute the integral of the different parts of the function, cutting it where the function change sign. If we ask for the integral of the function shown in Figure 2 (right) cut at indices `[0,4,6,8,11]`, we thus want the integrals of the 5 subparts of the function depicted in different levels of gray.

(Q4.3) Write the function `integral_per_part` that computes the integral of each subpart of the function. The function takes as parameters the area of each rectangle (computed with the `integral` function) and the indices where each subpart of the integral should be computed. It returns a list containing the integral of each subpart.

```
def integral_per_part(aires,chgt):
    seg = []
    for i in range(len(chgt)-1):
        aire = 0
        for j in range(chgt[i], chgt[i+1]):
            aire += aires[j]
        seg.append(aire)
    return seg
```

(Q4.4) Use the functions that your defined to compute and display the integral of the function between each change of sign. You can use the variables `x` and `fx`. You display should be similar to the one below :

```
Between -0.01 and 0.31, the integral is 0.33
Between 0.31 and 0.48, the integral is -0.025
Between 0.48 and 0.58, the integral is 0.014
Between 0.58 and 0.95, the integral is -0.28
```

```
inte, areas = integral(x,fx)
indices = change_sign(fx)
integral_parts = integral_per_part(areas, indices)
```

```
for i in range(len(indices)-1):
    print(f"Between {x[indices[i]]} and {x[indices[i+1]]}, the integral
       is {integral_parts[i]}" )
```

From a starting point `x`[*beg*], we want to compute the integral until the function change sign. In the example above, if we start at *beg* = *2*, we will compute the integral until `x`[4] (and the integral will be the sum of the area of two rectangles).

(Q4.5) Write a function that computes the integral of the function between a given starting point and the next change of sign. Use the areas of the rectangles that can be computed with the `integral` function. If there is no sign change after the starting point, your function will return the integral until the end of the function.

```
def area_until_zero(fx,aires,deb):
    aire = 0
    i = deb
    while i < len(aires)-1 and (fx[i] < 0) == (fx[i+1] < 0):
        aire += aires[i]
        i += 1
    return aire
```

Exercice 5 Code writing : processing sentences (3.5pts)

In this exercise, we will try to modify phrases, for example, change randomly the order of words in a sentence or exchange words.

We provide you with some functions for which you have the docstring on page 9

(Q5.1) Using the functions available page 9, write a code that randomly shuffle the words of a phrase. Your code will first display all the words from the phrase (each word on one line) before printing the phrase with the shuffled words. Be cautious, all the functions might not be useful and you have to initialize all the variable you might use. An example of display for the phrase "Hello SCAN students" is shown below.

```
Hello
SCAN
students
SCAN Hello students
```

```
phrase = "Hello SCAN students"

ind = find_words(phrase)
display_words(phrase, ind)
new_ind = shuffle(ind)
new_phrase = move_words(phrase,new_ind)
print(new_phrase)
```

(Q5.2) Write the function `extract_word` whose docstring is given. You do not have to write the docstring on you copy.

```
def extract_word(phrase, index):
    mot_extrait = ""
    i = index
    while i<len(phrase) and phrase[i] != " ":
```

```
        mot_extrait+=(phrase[i])
        i+=1
    return mot_extrait
```

(Q5.3) Write the function `display_words` whose docstring is given. You do not have to write the docstring on you copy. You will think of using the provided functions if needed.

```
def display_words(phrase, words):
    for i in range(len(words)):
        mot = extract_word(phrase,words[i])
        print(mot)
```

Available functions for exercise 5 :

```
def find_words(phrase):
    """ Find the positions of words in a phrase
    Input:
        phrase (string): phrase to analyse
    Output:
        indices_words (list of integers): the position of each word in
           the phrase (index of the first letter of the world)
    """

def shuffle(numbers):
    """ Randomly shuffle a list
    Input:
        numbers (list): list of elements to shuffle
    Output:
        new_list (list): a list in which all the elements of numbers are
           present in a random order
    """

def extract_word(phrase, index):
    """  Extract the word that starts at a given index in the phrase
    Ex: extract_word("Hello beautiful world",6) => "beautiful"
    Input:
        phrase (string): phrase to treat
        index (int): the index of the first letter of the word to extract
    Output:
        word (string): the extracted word (does not contain any spaces)
    """

def move_words(phrase, words):
    """ Build a new phrase composed of the words of the phrase that are
       placed at indices words (in order)
    Ex: move_words("Hello beautiful world",[6,0,16]) => "beautiful Hello
       world "
    Input:
        phrase (string): phrase to treat
        words (list of int): the indices of the first letter of each word
    Output:
        new_phrase (string): a phase composed of the words at indices
           words from the given phrase
    """

def display_words(phrase, words):
    """ Display each word of the phrase (defined by their indices) on a
       different line
    Input:
        phrase (string): phrase to treat
        words (list of int): the indices of the first letter of each word
    Output:
        None
    """
```