EFS Informatique et Société Numérique 1
SCAN - January 2024

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. » *Martin Golding*

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

For all the exercises : 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.

Unless specified, you do not have to write functions. For each question, you can reuse variable or functions defined in previous questions (even if not answered).

Exercise 1 Code reading (2 pts)

(Q1.1) What does the following code display?

```
1  def f(x,y):
2      z = 2*x + y
3      print(f"z has value {z}")
4
5  def g(a):
6      a = a+1
7      return a%2
8
9  def fonction(a,b):
10     if g(a) == 0:
11         b = b*2
12     return b**2 - g(a)
13
14 a = 1
15 f(3,a)
16 print(fonction(a,5))
17 print(g(a))
18 print(a)
```

(Q1.2) What does the following code display?

```
1  liste = ["baby","arm","blank","armadillo"]
2  j=0
3  for i in range(len(liste)):
4      while j < len(liste[i]) and liste[i][j] != "a":
5          j += 1
6      if (j == len(liste[i])):
7          j -= 1
8      print(j)
```

Exercise 2 Code correction (2 pts)

(Q2.1) The following program contains **4 errors** that will either produce incorrect results, make the program crash in some cases or not execute at all. *These are not syntax error* (such as wrong variable names or forgetting a parenthesis). Identify each error by indicating its line number and how to correct it.

```
1  def find_letter(string,letter):
2      found = False
3      res = -1
4      i = 0
5      while not found and i<len(string):
6          if string[i] == letter:
7              found = True
8      return found,i
9
10 def display(found,i):
11     if not found:
12         print("The letter was not found")
13     else:
14         print(f"The letter was found at position {i}")
15
16 find_letter("coucou","u")
17 print(display(found,i))
```

Exercise 3 The ideal gas law (6.5 pts)

In physics, the ideal gas law is formulated as :

$$PV = nRT$$

In this equation, P refers to the pressure of the ideal gas, V is the volume of the ideal gas, n is the total amount of gas, R is the universal gas constant, and T is the temperature (you can ignore units).
An experiment is conducted to compute the value of the universal gas constant R. In this experiment, the volume and quantity of gas is kept fixed while the gas is heated up. The pressure and temperature are measured through time, the measurement are stored in lists. The following variables are thus available :

```
1 # Fixed values
2 volume = 12.0 #V
3 quantity = 0.5 #n
4 # Values that evolve through time, stored as lists
5 pressions = [101.3, 103.5, 104.7, 105.2, 107.3] #P
6 temperatures = [293.1, 299.0, 302.8, 304.1, 310.5] #T
```

The goal of the exercise is to write a program that find the value of R and verify that it is a constant. We can find the value of R by reformulating the equation as $R = \frac{PV}{nT}$.

(Q3.1) Compute the value of R for each measurement and store the result in a list (it should be the same length as `pressions` and `temperatures`).

(Q3.2) Compute the average of R over time (over all measurements).

We now want to verify if R is indeed a constant as the temperature and pression is changing. A way to verify this property is to compare all values of R to their average : if all values are within 10% of the average (more than 90% of the average and less than 110% of the average), we can consider that R is a constant.

(Q3.3) Write the code to verify if all values of R (computed in Q3.1) are within 10% of the average computed in Q3.2. If it is the case, your program should display `"All values of R are around XXX"` where XXX is the average value. If it is not the case, your program should display `"There is a problem with measurement number XXX"` where XXX is the index of the first value of R that is too far from the average.

We can also compute the moving average on all measured R : for each value, we compute its average with its adjacent values, the number of values depending on the width of the moving average.A visual example with a width of 3 is shown in Figure 1.

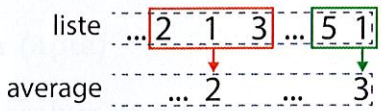

FIGURE 1 – Computing the moving average of `liste` in a list `average` with a width of 3.

We output a list (`average`) of the same size as the initial list (`list`). With a width of 3, `average[i]` contains the average of the 3 values around index `i` (from `list[i-1]` to `list[i+1]` included), except at the extremities where the average will include less values to avoid getting out of range : `average[0]` will contain the average of only 2 values (`liste[0]` and `liste[1]`). With a width of 5, `average[i]` will contain the average of 5 values (from `liste[i-2]` to `liste[i+2]` included), except close from the extremities where it will use only 3 or 4 values.

(Q3.4) Write a function that allows to compute the moving average of a list. The function takes as input a list and the width of the moving average and returns the list of averages. You can consider that the width will always be an odd number. Use this function to display the moving average of all measured R with a width of 3.

Exercise 4 Algorithm writing (9.5 pts)

A restaurant wants to ease the preparation of orders and know the dishes it can prepare with the ingredient in stock. To fullfil this task, you have access to the recipe of each dish (what are the required ingredients and the required quantity), as well as the level of stock for each ingredient, that you can also update. A dish or an ingredient is represented as a string, you can ignore the units of the ingredients (depending on the ingredient, it can be in units, in kilograms or in liter but it does not impact the code). Below are the provided functions :

```
1  # Provided functions
2
3  def ingredients(dish):
4      """
5      Gives the list of ingredients to prepare a dish. Example: ingredients("Lemon
           cake") -> ["flour","eggs","lemon"]
6      Parameter: a dish (string)
7      Returns: a list of ingredients (string)
8      """
9  def nb_ingr(dish, ingr):
10     """
11     Gives the number of ingredient used in a given dish for a given ingredient (
           ex: the "Lemon cake" needs 0.3 kg of ingredient "flour").
12     Parameter: a dish (string), an ingredient (string)
13     Returns: the number of ingredients needed for the recipe (float or int)
14     """
15 def available(ingr):
16     """
17     Gives the number of ingrdients available in the stock (ex: 6 "eggs"
           available, 0.5 kg of "flour").
18     Parameter: an ingredient (string)
19     Returns: the number of ingredients available in stock (float or int)
20     """
21 def remove_ingr(ingr, quantity):
22     """
23     Update the quantity of available ingredients in the stock by removing the
           given quantity  of an ingredient (ex: remove 3 eggs or 0.2 kg of flour)
24     Parameter: an ingredient (string), a quantity (int or float)
25     Returns: Nothing
26     """
```

First, we want to know the quantity of a given dish that can be prepared, depending on the quantity of ingredients that are in stock, as well as the limiting ingredient. For example, if the "Lemon cake" dish requires 0.3 kg of flour and 6 eggs, and if we have 1 kg of flour and 15 eggs in stock, we can prepare the cake only 2 times because we do not have enough eggs to prepare a third one.

(Q4.1) Write a function `nb_prepared` that takes as input a dish and returns the number of dishes that can be prepared (an integer) and the limiting ingredient. The function called with the dish "Lemon cake" would return 2 and "eggs". If several ingredients are limiting (for example, we are missing eggs and lemon for the third cake), you will only give one of them.

We now have a list of dishes that were ordered by clients and we want to know which ones we can prepare, given the stock of ingredients. We consider that we stop preparing as soon as there is one dish that can not be prepared. In the example given above, consider the list of orders `orders = ["Lemon cake", "Chocolate cake", "Lemon cake", "Ice cream"]`. We can prepare only the two first dishes (we can not prepare a third cake because we

will be missing eggs), we thus stop preparing the orders there (even if it could be possible to prepare the ice cream).

(Q4.2) Write a program that lists the dishes that can be prepared from a list of order and indicates the limiting ingredient. In the example above, you should produce the following output :

```
   Preparing Lemon cake
Preparing Chocolate cake
Can not prepare Lemon cake because we are missing eggs.
```

Functional decomposition (4pts)

The restaurant would like to analyze its profit over a year. For this, it has the list of all the orders of the year (`order_year`). From this list, the restaurant would like to see the profit for each month, and the proportion that it represents over the profit of the year in this way :

```
1 January
2 | **                    |  7945 / 84516
3 February
4 | *****                 |  19674 / 84516
5 March
6 | *                     |  4149 / 84516
7 # and so on until December.
```

In this example, the total profit of the year is 84516 euros, the profit for the month of January is 7945 euros. The number of stars represents the proportion of the profits of the month relative to the profits of the year, the numbers on the right directly shows the profit numbers. The goal of this question is to propose a functional decomposition allowing to reach this display.

Your functional decomposition should respect the following rules :

- Use between 3 and 4 functions.
- 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
- You can consider that all previous functions are available to you, as well as functions to access any information about an order `order_year[i]` (date, month, profit...)
- You do **not** have to write any code within the functions.

(Q4.3) Suggest between 3 and 4 functions that you will use in the main program. For each function, write the signature and a (short) docstring .

(Q4.4) Write the main program by using your suggested functions. All your functions should be called in the main program. You will consider that your main program can use the variable `order_year` which contains all the orders of the year.