Computer science (code 30424) a.y. 2020-2021

Exercises lesson 13

EXERCISES ON LECTURE TOPICS

All the exercises are provided with the script of a possible solution, considered the most efficient to illustrate the tools introduced in the theoretical lesson.
To verify if the code is working properly, test different logical flows and use different data types. Use the course textbook[1] as a support.

EXERCISE 13.1

Create in Python the program **Exercise13.1.py** containing the following text string:

```
animals = 'dogcatgiraffejaguarlion'
```

The program must:

a. use slicing to save the names of the five animals in five different variables
b. insert the five names in a list and sort it alphabetically
c. display the list on the screen, printing a single animal per line

EXERCISE 13.2

Create in Python the program **Exercise13.2.py** which contains the following text string:

```
animals = 'mouse, cat, monkey, snake, wolf'
```

The program must:

a. use the proper string method to transform it into a list containing the names of the animals (without punctuation marks or blank spaces)
b. print the list on the screen
c. sort the list in a descending alphabetical order
d. create a string concatenating the list's elements separated by a dash (-)
e. print the text string on the screen

EXERCISE 13.3

Create in Python the program **Exercise13.3.py** to build a dictionary in which:

a. the keys are integer numbers between 1 and a value given by the user
b. the values correspond to the cube of a half of the corresponding key

At last the program must display on the screen the items of the dictionary, printing each key-value pair on a different line, aligning keys and values.

[1] Learning Python, A. Clerici, M. De Pra, M.C. Debernardi, D. Tosi, Egea, 2020

EXERCISE 13.4

Create in Python the program **Exercise13.4.py** which, given a list of integers (starting list), asks the user for a number and displays a new list (result list) containing the numbers of the starting list which are smaller than the given number.

E.g.: considering the list *a = [125, 89, 2, 6, 58, 65, 137, 11, 64, 98, 17, 35]* and 70 as number given by the user, the program must display the following result list: *[2, 6, 58, 65, 11, 64, 17, 35]*.

EXERCISE 13.5

Create in Python the program **Exercise13.5.py** which asks the user for an integer number and calculates its divisors (hint: the remainder is zero if you divide the given number by a divisor), excluding 1 and the number itself. The divisors must be stored in a list.

At last, the program must display on the screen:

- the list of divisors, preceded by an explanatory message, if the given number has one or more divisors
- a message which specifies that the given number is prime, if there are no divisors.

EXERCISE 13.6

Create in Python the program **Exercise13.6.py** which receives a positive integer number from the user and returns the corresponding multiples smaller than 1000, storing them in a list.

The program must then display the list of multiples, preceded by an explanatory message, otherwise it must print on the screen that there are no multiples below 1000.

EXERCISE 13.7

Create in Python the program **Exercise13.7.py** to receive a positive integer number from the user and return a dictionary containing the multiples which are smaller than 1000 (use the multipliers as keys and the multiples as values of the dictionary).

If multiples below 1000 are available, the program must print the multiplication table, aligning the multiplies on the right digit in a 3-characters field and the multiples in a 4-characters field, as in the following output example (multiplication table of number *50*):

```
Table of 50
50 x   1 =   50
50 x   2 =  100
…
50 x   9 =  450
50 x  10 =  500
50 x  11 =  550
…
50 x  19 =  950
50 x  20 = 1000
```

EXERCISE 13.8

Create in Python the program **Exercise13.8.py** containing the tuple:

```
tuplex = (4, 9, ['a','b'], 123.45, 0)
```

At the end of the program the tuple must be changed in the following way, applying the changes in the exact order given below (remember: tuples cannot be modified, they can only be reassigned, thus a list must be used to perform the transition from the initial tuple to the final one):

a. add value 7 at the end of the sequence
b. add tuple (10, 100, 1000) in the fourth position
c. add the string "bob" stored in the position with index 2
d. add number 3.5 in first position
e. add False in position -1
f. delete value 9
g. delete the element stored in the position with index -4

Display each change on the screen (printing each time the intermediate list) and, at last, print the final tuple *tuplex*.

EXERCISE 13.9

Create in Python the program **Exercise13.9.py** to sum the elements of a list of 5 numbers provided by the user. More specifically, the program must:

a. use a conditional loop (*while*) to ask the user for each single element of the list
b. print the list on the screen
c. convert the elements of the list into decimal numbers
d. calculate and display the sum of the five elements, if they are all numbers, otherwise print an error message

EXERCISE 13.10

Create in Python the program **Exercise13.10.py** containing the *palindrome* function which checks whether a word, provided as input, is a palindrome (a word that remains the same if read in reverse order).

The function must receive a word as mandatory parameter, check if it is a palindrome and print the outcome on the screen.

The main program must call the function two times: the first time using "*kajak*" as argument, the second one using "*canoe*".

EXERCISE 13.11

Create in Python the program **Exercise13.11.py** which asks the user for a text string and prints its length. The program must keep on asking new inputs until the user enters *"end"* or *"exit"*.

EXERCISE 13.12

Create in Python the program **Exercise13.12.py** which receives a text string as input and extracts the first 2 and the last 2 characters, thus creating a new text string. The program must return an empty string if the length of the input string is less than 4 characters.

EXERCISE 13.13

Create in Python the program **Exercise13.13.py** which must:

a. ask the user for five numbers and store them in the *start_list* list (initialized with 0 values)
b. create a second list (*square_list*) containing the square of the values stored in *start_list*
c. sort the second list in ascending order and print it on the screen

EXERCISE 13.14

Create in Python the program **Exercise13.14.py** to manage a list containing the guests of party. The program must:

a. consider that two people have already been invited to the party (*Jack* and *Daniel*)
b. ask the user if he/she wants to invite another person
c. allow different positive answers (for example "Yes", "yes", "Y", "y", etc.),
d. in case of a positive answer, ask the user for the new guest's name and add it to the list, asking then if the user wants to invite another person (an infinite number of guests should be allowed)
e. in case of a negative answer, print the guests list on the screen

EXERCISE 13.15

Create in Python the program **Exercise13.15.py** containing a function, called *total_list*, which receives a list as mandatory parameter and sums up its values, returning the result as output. Solve the problem using a loop which allow the function to deal with lists of different lengths.

In the main program the function must be called twice, using lists of different lengths.

EXERCISE 13.16

Create in Python the program **Exercise13.16.py** containing a function, called *char_frequency,* which receives a string as mandatory parameter and returns a dictionary in which the keys are the string's characters (without repetitions) and the values are the corresponding occurrences.

The main program must ask the user for a string, call the function and display the output.

EXERCISE 13.17

Create in Python the program **Exercise13.17.py** to draw simplified horizontal histograms in the shell. More specifically, the program must:

a. contain a function, called *create_list*, which asks the user for the length of the list (i.e. number of series of the histogram) and asks to insert the value of each element (only integer numbers should be allowed), returning the list as output
b. contain the function *draw_hist* which uses the symbol given as optional parameter (default value: "*") to draw horizontal histograms of the list created by *create_list*

E.g.: using the list *[2,7,1,4]* and '#' as optional argument, the program must produce the following output:

```
##
#######
#
####
```

EXERCISE 13.18

Create in Python the program **Exercise13.18.py** containing a function, called *weighted_average*, which calculates the weighted average. More specifically, the function must contain the following mandatory parameters:

- a list containing the values to calculate the average
- a second list containing the corresponding weights (which must be positive numbers between 0 and 1)

The function must perform the following checks: the sum of the weights must be equal to 1 (i.e. 100%); the values list and the weights list must have the same length. If a check fails, a message explaining the error must be shown on the screen and the average must not be calculated; if both checks are positive, the function must calculate the weighted average (i.e. the sum of the products of each value multiplied by the corresponding weight).

Complete the program testing all the outcomes of the function (calculation of the average, weights with sum different from 1, weights list and values list with different lengths).

EXERCISE 13.19

Create in Python the program **Exercise13.19.py** containing a function, called *digits_sum*, which receives a positive integer as mandatory parameter and returns the sum of its digits. E.g.: if the number received by the function is 17, it must return 8 (1 + 7); if the number received by the function is 153, it must return 9 (1 + 5 + 3).

In the main program the function must be called twice, in order to test it with different inputs.

EXERCISE 13.20

Create in Python the program **Exercise13.20.py** containing the function *digits_sum_bis* which asks the user for a positive integer number and returns the sum of its digits.

The function must provide explanatory messages, without calculating the sum, for the following wrong inputs:

- an empty string (press Enter without writing anything)
- a decimal number
- a negative integer
- a sequence made of zeros
- a text string

The main program must call the function to test it.

Remember: the conversion of a *float* number to an integer removes the decimal digits, so *int(5.34)* returns 5, which is obviously different from 5.34!

EXERCISE 13.21

Create in Python the program **Exercise13.21.py** containing a function to count how many numbers and how many text strings are stored in a given list or tuple. The function must print a brief result message and return the counters' values in a list (see the example provided below).

The main program must initialize a list (or a tuple), call the function using the list (or the tuple) as argument and print the list returned by the function.

E.g.: using the list *[2, 'a', 'b', 7, 1, 5, 4, 'z']* the following output must be produced:

```
There are 8 elements:
 5 are numbers
 3 are strings
The values obtained are: [5, 3]
```

EXERCISE 13.22

Create in Python the program **Exercise13.22.py** to create the dictionary *dict_ita_eng* which contains at least 5 Italian words (keys) and the corresponding English translations (values).

The program must also contain a function to consult the dictionary. The function, called *translate*, must ask the user if he/she wants to translate an Italian word and:

a. stop the function's execution, if the user gives a negative answer
b. ask the user for the Italian word and print its translation, if the user gives a positive answer
c. keep on asking if the user wants to translate another word, until he/she decides to stop
d. prevent any error and print a message to tell the user that the word is missing, if the word given by the user is not stored in the dictionary