Luigi De Russis, Alberto Monge Roffarello

LAB 2 – PYTHON BASICS: EXERCISES

GETTING STARTED…

The goal of this set of exercises is to experiment with the basic concepts of Python with the PyCharm IDE.

If two of you share a single computer, adopt the pair programming[1] technique and change role at the beginning of each exercise.

Recap:

1. *Fork your own copy of the Git repository associated with this lab (https://github.com/AmI-2018/python-lab2) to your personal GitHub space*
2. *Open PyCharm Professional and select Checkout from Version Control > Git in the "Welcome to PyCharm" window, to clone your (forked) repository*
3. *Fill the requested fields (repository URL, location on disk, …) and press the "Clone" button*
4. *Once the project is open, you can create a new Python file by right clicking on the project name (Project tab, on the left) and selecting New > Python File*
5. *To execute your program, right click on the Python file (in the Project tab) to be executed and select Run. In this way, PyCharm will create a new "Run/Debug Configuration" that you can find in the "Configurations" list (accessible through the menu Run -> Edit Configurations)*
6. *Commit and push the changes you made back to GitHub, from the VCS menu in PyCharm*

EXERCISE 1 – TODO LIST

Given a list of tasks (i.e., actions that the user wants to do in the future) implement a `todo_manager` program to perform 4 actions:

1. insert a new task (a string of text);
2. remove a task (by typing its content, exactly);
3. show all existing tasks, sorted in alphabetic order;
4. close the program.

At startup, the program shows a menu with the 4 options and, for each choice, performs the requested action. After the action (except action 4), the program returns to the prompt for actions.

***Hint 1**: to show a sorted list of tasks you can use the `sorted()` function.*

[1] In pair programming, two programmers work as a pair, together on one computer. One, the driver, writes code while the other, the navigator, reviews each line of code as it is typed in and helps plan and catch errors.

***Example**:*

Run the program: > `todo_manager`

First screen shown by the program:

```
Insert the number corresponding to the action you want to perform:

   1. insert a new task;
   2. remove a task;
   3. show all the existing tasks in alphabetic order;
   4. close the program.

Your choice:
```

EXERCISE 2 – TODO LIST SAVED TO FILE

Extend the program developed in the previous exercise to save and retrieve the list of tasks to/from a text file. The file name is read as the first parameter from the command line.

Consequently, at startup, the program takes the list of tasks from the file and saves the changes to the file as soon as the user decides to close the program.

For this exercise, you can get the *task_list.txt* file from the GitHub repository of this lab. It contains 6 tasks saved as a separate line on the file.

***Hint 2**: to specify one or more string to be passed as input parameters to the application, you can modify the Configuration created when you run the program for the first time. The Configuration is accessible through the menu Run -> Edit Configurations. Specify your parameters in the "Script parameters" field.*

Example:

Run the program: > `todo_manager task_list.txt`

First screen shown by the program:

```
Insert the number corresponding to the action you want to perform:

   1. insert a new task;
   2. remove a task;
   3. show all the existing tasks in alphabetic order;
   4. close the program.
```

Luigi De Russis, Alberto Monge Roffarello

EXERCISE 3 – TASKS DELETION EXTENSION

Modify the program developed in the previous exercise to remove all the tasks that contains a specified substring. For example, when user types "shopping", the program will use the provided string to delete all tasks that contain the substring "shopping".

Example:

Run the program: > `todo_manager task_list.txt`

First screen shown by the program:

```
Insert the number corresponding to the action you want to perform:

    1. insert a new task;
    2. remove a task;
    3. remove all the existing tasks that contain a provided string;
    4. show all the existing tasks in alphabetic order;
    5. close the program.
```

EXERCISE 4 - FIND URGENT TASKS - DICTIONARIES

Given a "2D dictionary" of tasks, like this one (i.e., tasks):

task1 = {'todo': 'call John for AmI project organization', 'urgent': True}
task2 = {'todo': 'buy a new mouse', 'urgent': True}
task3 = {'todo': 'find a present for Angelina's birthday', 'urgent': False}
task4 = {'todo': 'organize mega party (last week of April)', 'urgent': False}
task5 = {'todo': 'book summer holidays', 'urgent': False}
task6 = {'todo': 'whatsapp Mary for a coffee', 'urgent': False}

return a new dictionary, using the same "2D" format, that contains only the urgent tasks, i.e., take the dictionaries that have a True value in the urgent field and combine them in a single new dictionary as shown in the example.

Example:

Run the program: > `find_urgent_tasks`

Result returned by the program:

```
{'task1': {'todo': 'call John for AmI project organization', 'urgent': True }, 'task2':
{'todo': 'buy a new mouse', 'urgent': True}}
```