Introductory Prac 0: Python Basics

Week 0 - Applied - Practical

Objectives of this Prac

The objective is to become familiar with Python and, in particular, with its:

- List data structure.
- Iteration and selection.
- Input/output.

Useful Material

The following links provide useful info on:

- Iteration and Selection: https://docs.python.org/3/tutorial/controlflow.html.
- Lists and Tuples: https://docs.python.org/3/tutorial/datastructures.html.
- Style guide: https://www.python.org/dev/peps/pep-0008/.
- Docstrings: https://www.python.org/dev/peps/pep-0257/.
- Type hints: The beginning of https://www.python.org/dev/peps/pep-0484/ has the basics.

IMPORTANT
Before attempting these exercises, read example documentation and further explanation of good documentation under Important Documents on Moodle. This will ensure you know how you are expected to comment Python code in this unit.

Documentation Requirements

In Python, a **module** refers to any Python file (suffix .py) with Python code defining functions, classes and/or variables. As shown in example_documentation.py (available on Moodle), the documentation for a Python module is written within the code itself by

1. Using meaningful names for variables, class, functions and methods.
2. Using type hints to indicate the intended input and output types
3. Using docstrings to comment the modules, classes, functions and methods.

The **pydoc** module automatically generates documentation if the programmer adheres to the **pydoc** conventions. For example, the documentation about the Python list type is automatically generated by **pydoc** and can be displayed on-screen in several different ways, including:

- In the Python interactive prompt by typing help(list).
- At your operating system's command prompt (without going through the interactive prompt), by typing **pydoc** list.
- In a browser, either by generating a static HTML file for a particular module, or by starting a live server with all available documentation.

The documentation in **pydoc** works by turning specially-formatted Python strings (called **docstrings**) into professional-looking HTML documentation. The core developers for the Python standard library use **pydoc** to create the documentation for it. As you use the official Python documentation for the language at http://docs.python.org/3/reference/ and for the library at http://docs.python.org/3/library/ you'll become accustomed to the default **pydoc** style.

You will be required to document every module you write in this unit, and to do so using meaningful names, type hints and docstrings. The example provided in the introduction gives you an idea of how to do this. When you start creating longer and more complex code, you will see how useful it can become (particularly, when the code you are using was written by somebody else in a workplace is not yours, or you wrote it years ago...).

Exercise 1

In this exercise you will practice reading a line from input that contains several values, separating it into strings, iterating over a list, and returning values.

1. Using PyCharm create a file **prac0.py** and add a Python function **read_integers** that asks the user to enter some integers separated by spaces in a single line, and returns a list of these integers in the order they were provided.

2. Run your code line by line and see how the value of your integers change. If you are confused about how to do so, please ask your demonstrator.

3. Copy your code for the function and place it in the Ed console and see if you pass the test cases.

The following shows a few examples of what should happen when you run the file in the interactive Ed console:

```
--------------------- 1st run -------------------------
Enter some integers:
[]
--------------------- 2nd run -------------------------
Enter some integers:-6
[-6]
--------------------- 3rd run -------------------------
Enter some integers: 1 -2 6 0
[1, -2, 6, 0]
```

You can use **strings = input("Some string:").split()** to display **"Some string"** and then read the input line, leaving in strings the list of strings in that line separated by spaces.

Please note - Don't worry about error checking your code right, pretend that the user is amazing and always passes the input in the right format

Exercise 2

In this exercise you will practice taking arguments, a bit of maths, and more iteration and returning elements.

Add a Python function **sum_squared_integers** to the **prac0.py** file that, given a list of integers, returns the sum of squaring each of the integers in the list. The following shows a few examples:

```
>>> from prac0 import sum_squared_integers
>>> sum_squared_integers ([])
0
>>> sum_squared_integers([ -6])
36
>>> sum_squared_integers([1, -2, 6, 0])
41
>>>
```

Exercise 3

In this exercise you will practice reading from a file, rather than from input.

Add a Python function **read_from_file_sum_squares** to the **prac0.py** file that, instead of asking users to input the integers, asks them to give the name of a file, where each line in the file has a sequence of integers separated by spaces. For each line in the file, the function should read the integers, compute the sum of their squares and print out the result. If the line is empty, it should print a 0. We have provided 2 test files (**file1** and **file2**) for testing.

The following shows the result of calling function **read_from_file_sum_squares()** for each of these two files:

```
>>> from prac0 import read_from_file_sum_squares
>>> read_from_file_sum_squares()
Enter the filename:file1
9
>>> read_from_file_sum_squares()
Enter the filename:file2
14
20
0
1
>>>
```

You can use **input_file = open(filename, "r")** to open the file named by string **filename** to read, and the line **list_lines = input_file.readlines()** to store in **list_lines** the list of strings, where each string is a line of the file, in the order they appear in the file. Remember to close the file once you do not need it anymore, using **input_file.close()**.

Exercise 4

In this exercise you will practice nested iterations and building lists of lists.

Add a Python function **read_from_file_table()** to the **prac0.py** file that reads a file as before, and stores every non-negative number in each line in a list, in the order provided, obtaining a table (i.e., a list of lists). If a line is empty, it should add the empty list as an element of the table. For the two file above, the result is as follows:

```
>>> from prac0 import read_from_file_table
>>> read_from_file_table()
Enter the filename:file1
[[3]]
>>> read_from_file_sumtable()
Enter the filename:file2
[[1, 2], [4, 2], [], [1]]
>>>
```

Note that you can import several functions in one line (separated by commas), for example:

```
>>> from prac0 import read_integers , sum_squared_integers
```

Appendix

Some things to note about the code snippets:

- Function **input** is what's called a "built-in" function. It's part of Python proper and available for all and every program. There are other ways for you to acquire functions (you may import them from the Python standard library, you may install additional packages, or you may write them yourself).
- Calls to **input()** are usually followed by a call to string method **strip()**.
- Note also how we use not (rather than !) for negating the boolean value of an expression in, for instance, a conditional.
- The call to int(x) function converts the number or string x to an integer. It is also a builtin, and one of the many available for type conversion.