Introduction to Python

STUDENT BOOKLET

Shalev, Omer

About this Booklet

This booklet consists of several basic exercises in Python. It is complementary to the frontal lecture and therefore is worthless without attending class. You are strongly encouraged to invest maximal effort in solving the exercises below because in Python programming, like in any other programming, you learn through your hands.

All your work will be done on the virtual machine that you should have installed as a preliminary work for this course. Working on different environments (your Windows laptop, etc.) is optional but is at own risk.

You are encouraged to use the Internet for any question. Python is so popular that every question that you might think of was probably already asked. Also, don't hesitate to ask for help from the course staff.

Good luck and enjoy the ride!

Exercise #1

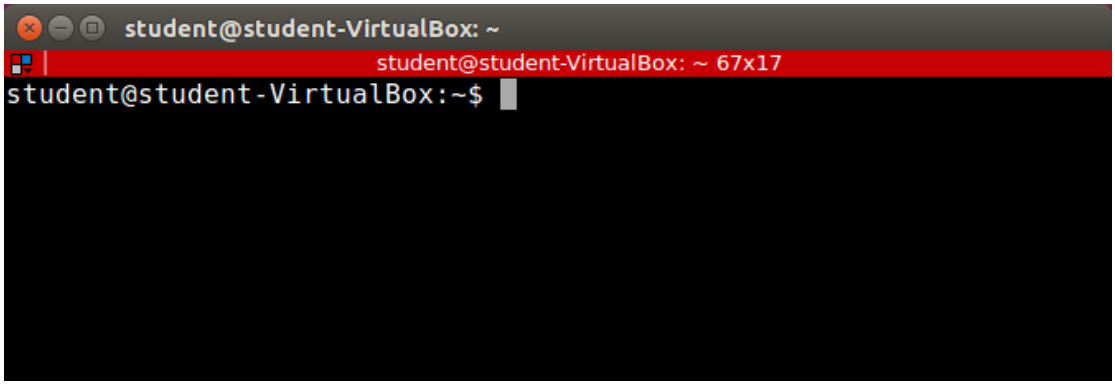
Before we dive deep, let's get acquainted with the Python programming environment on our Ubuntu machine by executing our first "Hello World". There are several ways for doing that and we will elaborate about all of them.

From the console

Open the virtual machine and wait for Ubuntu to load.

Now, open a Linux terminal by clicking on the Terminator icon on the left banner:

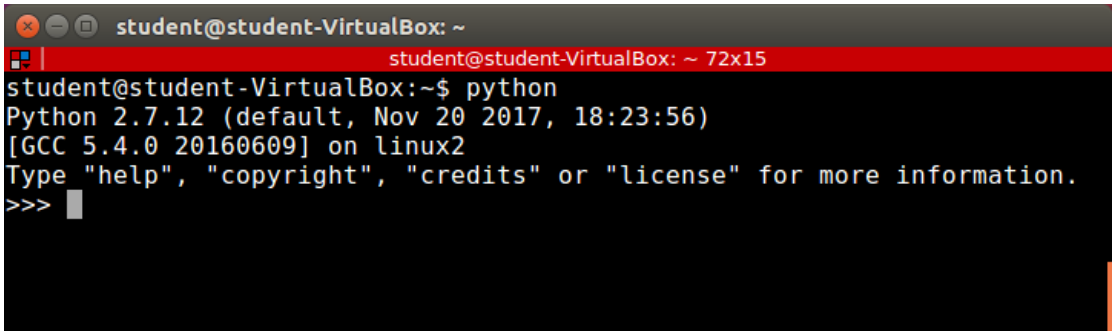
A terminal window should open:



This is the Linux shell and we will open the Python console by typing

```
python
```

and hitting Enter.



The >>> symbol indicate that we're in a Python console. Hooray! The message printed upon launching Python also indicates on the Python version that we're using (2.7.12 in our case).

In order to print "Hello World", all we need to do is (…) type

```
print 'Hello World'
```

(how convenient, ha?)

You may now exist the Python console by typing

```
exit()
```

Now, back in the Linux shell, type

```
ptpython
```

Again, a Python console is opened (it is still the same Python 2.7.12) but with some cool features such as text highlighting, tab completion, etc. and it is recommended to use this terminal from now on.

Repeat the “Hello World” print in this console.

From a script

Apart from the console, Python is scriptable, which means that it can be written as a script and be executed serially.

Open the Geany editor by clicking on the icon on the left banner:

In a new file, type the same statement that you previously typed in the console (“Hello World”…). Save the file as a python script under */home/student/intro_to_python/exercises/ex1* and name it *hello_world.py*.

Now from the Linux shell, execute the commands

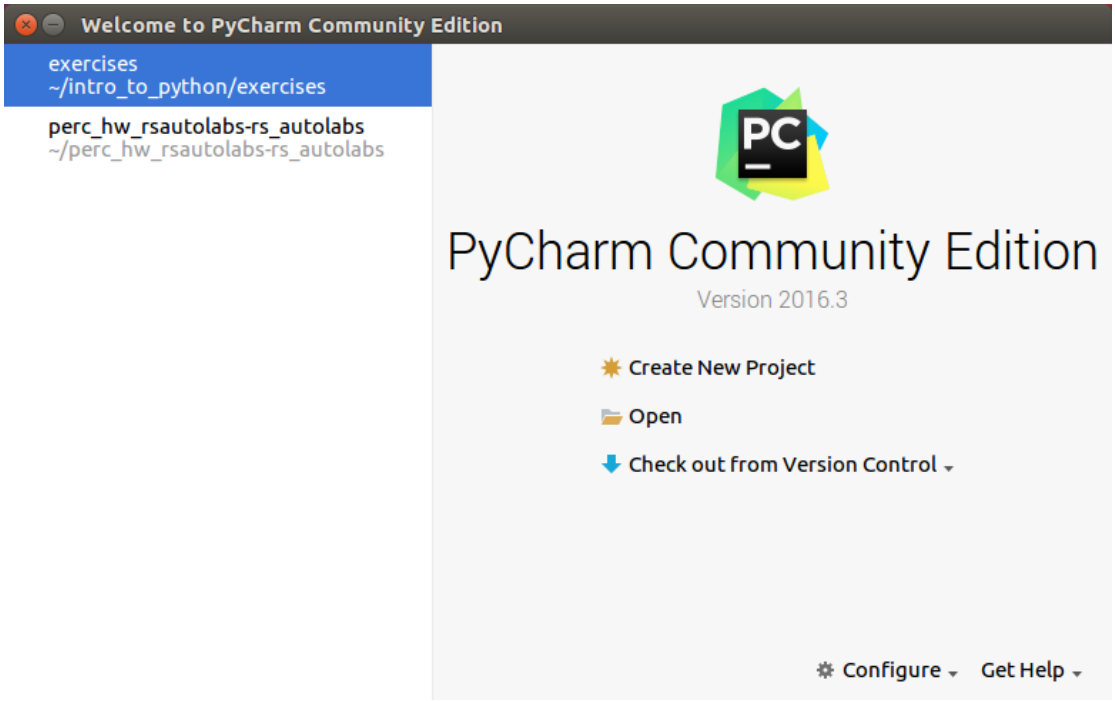
```
cd /home/student/intro_to_python/exercises/ex1

python hello_world.py
```

and the Hello message should be printed. This is basically the way to run a Python script (in a non-interactive mode): type *python* and the name of the script that we want to execute.

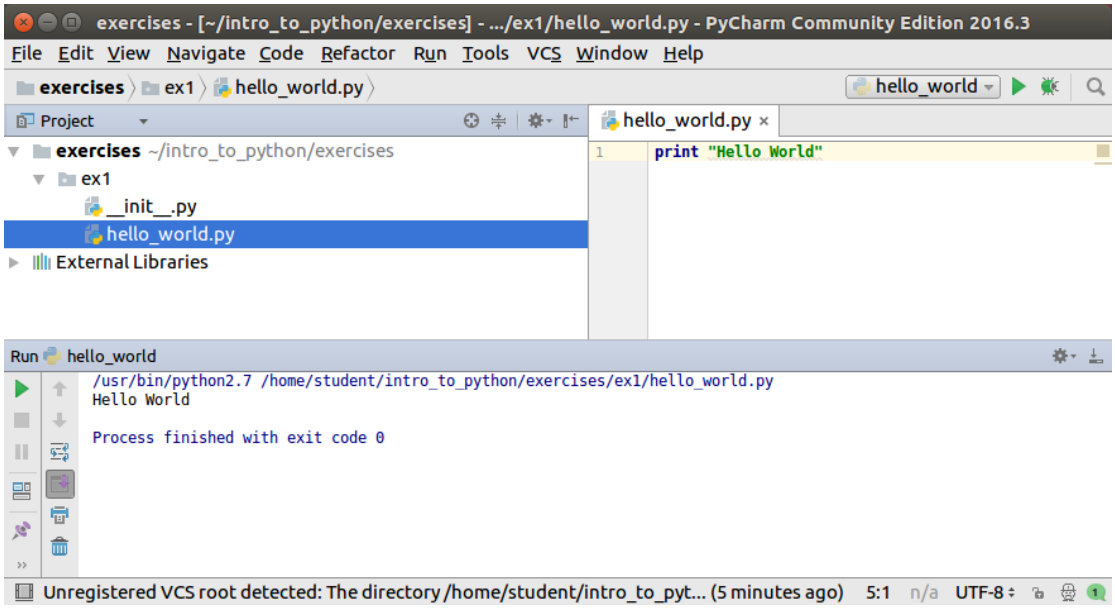
From PyCharm

Last but certainly not least: PyCharm, a wonderful Python IDE (Integrated Development Environment) is recommended for working with large projects. Close Geany and open PyCharm by clicking on the icon:

In the Welcome screen choose the project *exercises*:



Next, under the *project* area, double click on *hello_world.py*. Your script should appear in the main area (to the right). In order to execute this script, right-click on *hello_world.py* and choose *Run 'hello_world'*. The script output should appear in the console area at the bottom:



Exercise #2

Now that we feel more comfortable in the Python environment, we can start do some real (yet simple) programming tasks.

What's the time now?

In your PyCharm project under *exercises* → *ex2* you'll find a file named *hour_getter.py*. Open it.

As you probably figure from the code, *#* denotes a one-line comment. If you wish to comment out an entire block, you can also enclose it with triple quotes (''') or triple double quotes (""").

The block that begins in *if __name__ == '__main__':* can be considered Python's equivalent for a main function. This block will be executed only when running this script. It means that when importing this module in a different one (like we'll do in the next task), the code in this block is not going to be executed.

In parallel with PyCharm, it's always nice to have an open Python console so if you already closed ptpython, launch it again from the Linux terminal.

In this task we will write a function that returns the current hour and prints it. For this purpose we will use the library *datetime*.

In the console, type:

```
import datetime

datetime.datetime.now()
```

This gives complete information about the current time, including date and hour.

Assign the output of the function above to a variable (still in the console):

```
now = datetime.datetime.now()
```

We are specifically interested in the current hour.

It is a good practice to use the *help* function:

```
help(now)
```

This built-in function provides information about (almost) every object in Python. In our case, we easily can see that hour is specified (not so surprisingly) in a field named *hour*. This means that we can get it by:

```
hour = now.hour
```

Back to the script. Complete the function *get_hour* to do the following:

1. Print the message
   The time now is <hour> O'clock.
   Pay attention to casting.
2. Return the current hour as an integer in the range of 0 to 23.

Run this script and see that you're getting the expected result.

Greet me

Open the file named *greet.py* under *ex2*. Complete the function *greet_me* that gets a name of a person (string) and greets him/her. The function should first print the current hour and then greet the person according to the current hour of the, providing that:

- "Good morning" is used from 7:00 till 11:59.
- "Good afternoon" is used from 12:00 till 16:59.
- "Good evening" is used from 17:00 till 20:59.
- "Good night" is used from 21:00 till 1:59.
- Between 2:00 and 6:59, the message should be "Go to sleep".

Examples:

Function call @ time	Output
greet_me('John') @ 10:30	The time now is 10 O'clock Good morning, John
greet_me('Bob') @ 23:59	The time now is 23 O'clodk Good night, Bob
greet_me('Ben') @ 4:30	The time now is 4 O'clock Go to sleep, Ben!

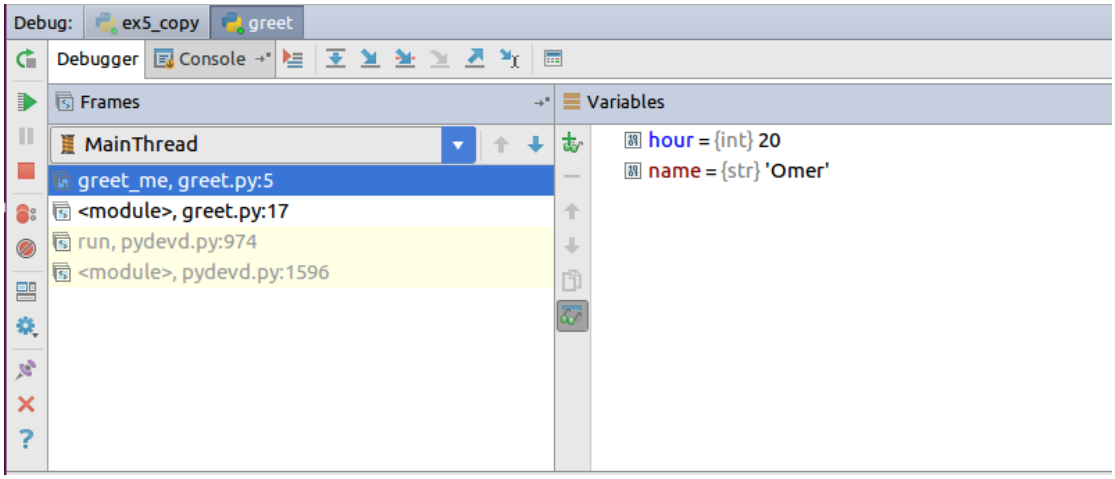
Debugging in PyCharm

Since the *now()* function return the current time, we can't validate all the options unless we "inject" a different time. Let's take this opportunity to get familiar with PyCharm's debugger.

Before debugging a script, we first need to define where we want to stop and place a breakpoint there. In PyCharm, all you need to do is place the cursor on the relevant line and press Ctrl+F8. A red point will appear at the sidebar. Removal of a breakpoint is also done by pressing Ctrl+F8.

In order to run in debug mode, you need to right-click on the module you want to debug under the project area (*greet.py* in our case) and select *Debug 'greet'*. The script will automatically stop at the first breakpoint. From this point, you can execute a single step (F8), step-into (F7) or resume program execution until the next breakpoint (F9).

At the bottom you'll find a watch window with the variables defined in this scope:



For this debug session, assign other values to your hour variable and see if your code behaves as expected.

Exercise #3

This exercise is especially important because it deals with the four common containers in Python which every Python developer can't get along without.

Euclidean distance

Just to remind you, the Euclidean distance between two points in a plane

$$p_1 = (x_1, y_1)$$

$$p_2 = (x_2, y_2)$$

is defined

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Under *ex3*, open the file name *distance.py* and implement the function *euclidean_distance*. The input of this function is two tuples, p1 and p2 and the output should be the Euclidean distance (float).

Run the script and debug your code if needed.

List of distances

In this task we will do a few list manipulations.

Open the file *distances_list.py*. You are given the list below:

```
pairs_of_points = [((0,0), (0,1)),
                   ((0,0), (1.5,0)),
                   ((0,3), (4,0)),
                   ((0,0), (1,1)),
                   ((-1,0), (1,1)),
                   ((0,0), (1,3))
                  ]
```

Every element in the list represents a pair of two point in the plane. Make sure you understand the structure of this list (why are there so double parentheses?).

A list of distances is initialized for you. As a first step, fill this list with the Euclidean distances between every pair of two points given in *pairs_of_points*. Use the function from the previous task (already imported for you). Print this list.

Now, let's try some things on this list. As already advised, you should use the *help*, function from the console, to understand what methods and what attributes an object (list, in our case) has.

- Print this list in ascending order.
- Print the minimal distance value in the list
- Print the two largest distance values in the list

Run the script and debug your code if needed.

BONUS

Round the distances values to closest integers and print only the ones that are even. Execute this in one line.

Hint: use the *round* and *filter* built-in function. Read about list comprehension and lambda functions.

References:

- https://www.programiz.com/python-programming/methods/built-in/filter
- https://www.programiz.com/python-programming/list-comprehension
- https://www.programiz.com/python-programming/anonymous-function

Vectors from points

Let's practice dictionaries. Open the file named *vectors.py*. You are given a dictionary of points in a 3D space (each represented by a 3-element tuple). Our purpose in this task is to create all possible vectors from the given 8 points.

Reminder: the vector between two points

$$A = (x_1, y_1, z_1)$$
$$B = (x_2, y_2, z_2)$$

is

$$\overrightarrow{AB} = (x_2 - x_1, y_2 - y_1, z_2 - z_1)$$

(pay attention to the order).

Fill the dictionary (which is already initialized for you) with all possible vectors, represented as tuples. As keys, use the vector name (e.g. for $\overrightarrow{AB}$ use the string 'AB').

Run the script and debug your code if needed.

String challenge

Open *string_challenge.py*. Write a function that gets a sentence (string) as an input and return this sentence with one change: the word before the last bang (!) in the sentence is in uppercase.

For example: if the input sentence is

Hey! Hooray! This Python course is awesome! We enjoy it very much.

the function will output

Hey! Hooray! This Python course is AWESOME! We enjoy it very much.

Run the script and debug your code if needed.

Exercise #4

Object oriented programming is an important concept in every project that needs to scale up. Let's practice it.

Polygon

Under *ex4* open the file *polygon.py*. We would like to represent a polygon object and for this purpose we will create class. Polygon's initializer gets one argument, *vertices*, which is a list of tuples, each representing a point in the plane. We assume that the vertices are ordered which means that element i of the list is the adjacent vertex of elements $i + 1$ and $i - 1$.

Create a class method named *get_perimeter* which returns the polygon's perimeter (float). You are encouraged to use the function *euclidean_distance* which you already implemented in the previous exercise (already imported for you).

Run this script (an object is instantiated for you and the *get_perimeter* method is called).

Rectangle

Rectangle is in fact a private case of a polygon, and that is an excellent motivation for using inheritance. Open the file *rectangle.py* where you'll find a class named *Rectangle* which inherits from *Polygon*.

As you can see in *polygon.py*, the function *get_area* is not implemented, and this is for a reason – there isn't an easy general for calculating a polygon's area. That is why it makes sense to declare this function in the base class (*Polygon*) but implement it in the inherited classes (e.g. *Rectangle*). Add the implementation of *get_area* in *Rectangle* and run the script.

Triangle

Triangle is another private case of a polygon and thus it will be implemented as a class which inherits from *Polygon*. Our purposes is as before: implement the method *get_area* which is, as said, not implemented in the base class. However, this time we'll take a somewhat different approach. Triangle's area can be calculate in many ways, but the simplest area formula is

$$S = \frac{1}{2}bh$$

where b is a length of one of the bases and h is the height of the opposite vertex relative to this base. For this purpose, let's add b and h as class members and let's ask the user of this class to provide them to the initializer.

Your tasks this time are:

- Complete the *__init__* method of *Triangle* to store *base_length* and *height* as class members. Don't forget about the other arguments and don't forget that the *__init__* of the superclass is not invoked automatically.
- Implement *get_area* of *Triangle*.

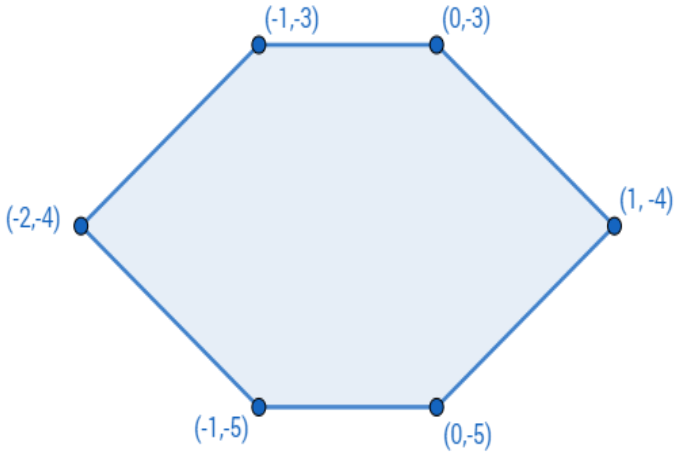
Run this *triangle.py* script to validate your code.

Polygons list

Let's conclude this exercise with a task that involves the three classes that we implemented.

Open the file *polygons_list.py*. Create a list of three elements:

- The triangle instantiated in *triangle.py*
- The rectangle instantiated in *rectangle.py*
- A hexagon with the following vertices:



Note that for a hexagon we don't have a specific implementation, **and we don't want to add one.** Please use the general polygon.

For every item in this list, print the following message:

> *For polygon with <number of vertices> vertices:*
>
> *Perimeter = <perimeter>*
>
> *Area = <area>*

What will happen when applying the *get_area* method on the hexagon? (hint: see the implementation of *get_area* in the base class). Let's print a special message for this case:

Cannot calculate area for a polygon of this degree

Please don't change anything in the implementation of *Triangle*, *Rectangle* and *Polygon* for this task.

Exercise #5

In this concluding exercise we will develop a simple computer vision algorithm using OpenCV and NumPy.

Object segmentation is a basic and important task in many computer vision applications. In this exercise we will develop a module for basketball segmentation from an RGB image. Our task is to draw a circle around the basketball in the image, for examples:

Under *ex5* open the file *basketball_segmentation.py*. Under the same folder you also have a sub folder with images. Running the script will show the images in a row (you need to hit Esc to continue to the next image).

For your convenience, you are given the implementation of four functions (see documentation the code).

- *show_image(image)*
- *draw_circle_on_image(image, x, y, r)*
- create_circle_interior_mask(x, y, r, size)
- create_hsv_mask(image, lower_hue, lower_sat, lowe_val, upper_hue, upper_sat, upper_val)

You are encouraged to use *show_image* for debugging purposes during the development to show images, masks, etc.

Suggested algorithmic approach:

- Circle Hough transform is an excellent technique for detection of circles in an image. Please see the articles below:
    - https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles
    - https://docs.opencv.org/3.1.0/da/d53/tutorial_py_houghcircles.html

- It is recommended to apply the Hough transform on a grayscale image (like we did in class).
- Among all the circles detected by Hough Transform, only one should be chosen. For this decision, the color image can help. HSV is a convenient color space to work with. You are advised to convert the image to HSV representation. Then, create a mask of orange areas. You may use http://colorizer.org/ to choose the appropriate color thresholds.
- For a given circle found by Hough Transform, you may create a mask of a full circle interior. By multiplying this mask with the HSV mask, you get a new combined mask of orange areas inside this specific circle.
- By comparing all the combined masks mentioned above (e.g. comparing their fill factor), you can decide which circle most likely represents the basketball.