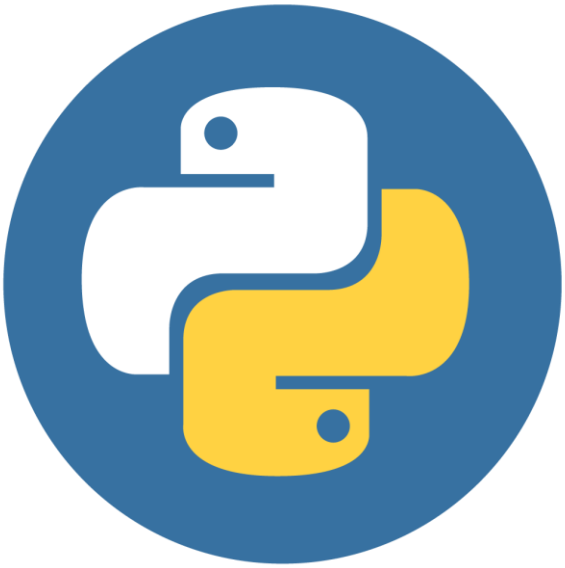
EPITECH.
THE FUTURE OF SOFTWARE
THE BEST OF INNOVATION



Python Short Project

Maze - merizing

-

Adam EMMANUEL

Academic Manager

EPITECH

Preface

This short project aims at teaching the students with the basics of several key aspects of programming.

EPITECH's methodology will be used to achieve the short project.

Depending on the students' level, the project length is divided in several bullet points.

Thus, the students can always get new knowledge although the short project is not finished.

Max. Number of students: 15-20 students

Equipment needed: student's own laptop

Software to be installed the day of the workshop: Python 3 compatible with Mac, Windows or Linux.

Content

The following topics will be covered. The purpose of the workshop is to introduce complex topics using a project-oriented approach so students may enjoy and discover the EPITECH learning process.

Technology:

- Python 3.7
- UNIX/Windows Terminal

Topics:

- Basics of Programming
- Basics of Python Language
- Basics of Algorithm
- Basics of Data structure
- Basics of Object-Oriented Programming[1]
- Basics of Artificial Intelligence
- Basics of Design Pattern[2]

[1] Basics of Object-Oriented programming is a notion covered in the bonus parts of the workshop
[2] Basics of Design Pattern is a notion covered in the bonus parts of the workshop

Subject

Part I: Starter - Introduction to Python

Setting up Python 3

You may find the python installer on the given git repository for windows and UNIX
Just follow the installation steps for Windows or the step by step guide for UNIX and macOS
If the successfully installed python, open a terminal (On windows, Powershell will do).

$> py --version
Python 3.7.0

$> python --version
Python 3.7.0

Use the Python interpreter

What you just used is called the Python interpreter. The purpose of this program is to interpret the code you write and translate it into a language your computer can understand, *on the fly*.

We will use it to execute our code today and the result will be displayed on the terminal

To run the interpreter, just type the following command and press *Enter*:

$> py
Python 3.7.0 (v3.7.0.....) on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>

$> python
Python 3.7.0
Type "help", "copyright", "credits" or "license" for more information.
>>>

To leave the interpreter, just type exit()

You can take your mark doing some simple algebra.

Try something like *3 + 5*, *8 – 2*, ect…

Use Python file extension and run them 

As simple text files have a .txt extension and Excel files a .xlsx extension, python files have their own extension, .py

To be able to do so, let's install your new best friend, *Visual Studio Code*. VSC is what we call an IDE for Integrated Development Environment. Just picture it as a text editor specially made for software development. It comes with a built -in terminal.

All the code you will write will be in a *filename.py* file. To create one, create a *new file*, *save under…* and choose *python*.

To get used to it, let's try our simple algebra inside a python file.
If you try it the same way, you will see that nothing happens. Since we are not using the python interpreter directly, we need to specify in our code that we want *to display* the result. Luckily, the *function* (we will see that later) print(…) do exactly that.

main.py

```
8 + 3
5 + 2
2 – 1
```

$> py filename.py

main.py

```
print(8 + 3)
…
```

Introduction to Variables 

Now that we know how to make simple algebra directly in the Python interpreter or in a file, let's learn how to code.

To start programming, there is some *basics* that need to be mastered. The first one is variables.

A variable is like a small box. You can put one specific data inside. To store more data, you can make more variable.

Once a data store in a variable, you can reuse it as much as you want, but also modify or update it.

There are many types of variable, each one representing a type of data:

Scalar:

- Integer
- String (text)
- Float (floating number)
- Bool (Boolean logic, True or False)

There are more advanced and complex types of variable that I will introduce later.

As a warm-up, let's use our new variables to redo the *simple algebra example:*

variables.py

```
number_one = 5
number_two = 3
result = number_one + number_two
print(number_one)
...
```

$ > py variables.py

```
5
...
```

Try multiple combination, a number + variable, and vice versa until you completely wrap your head around it. We can store data as well as operation result in variable for later use.

Concerning the other variable types:

variables.py

```
text = " this is some text stored in a string"
floating = 5,4
truth = True
....
```

Text also called string variable must be surrounded by single or double quotes.

Try to print them and test it until you get comfortable.

Conditions

You can now *print text* and make *basic algebra* and then *store* the result into a variable.

We will now learn one of the 3 key points of programming, condition.

Condition are a way to control the way your code is executed. You can tell the computer *"in that situation, do this!"* or *"in that case do that or in any other case, do nothing!"*

To say it to your computer, you have way, also called *statement*:

- *If (if this, do that)*
- *Elif (if not possible, then do that)*
- *Else (if nothing stated before is possible, then do this)*

To use them, you also need operator:

- > (greater than)
- < (lesser than)
- <= (lesser or equal to)
- >= (greater or equal to)
- == (not equal)
- != (is different of)

You can also chain several conditions in one statement:

- and (if this *and* that, then…)
- or (if this *or* that, then…)
- *Try it out!* You can use it the same way to test number, string or Boolean

conditions.py

```
nb = 10
condition = 'In that case, it is greater than'
condition_two = 'In  the other case'
final = 'if nothing else is possible'

if nb > 20 :
        print(condition)
elif nb < 20 :
        print(condition_two)
else :
        print(final)
```

Warning

*The **indentation** in Python (tabulation) is important. If you do not respect it,*

it won't work!

*Same goes for the **:** at the end of the condition statement*

Let's train for conditions with a simple age check

- We have 3 students, *Carl* (26 yo), *George* (14 yo) and *Paul* (18 yo)
- Check if they are **under, equal** or **over** 18
- At the end, print *"You are too young to be here"* for *underage*, *"Just enough to go in"* if they are *just 18* and *"It's all good"* if *over 18*

Clue

To make it easy, use the student name *as a variable name*

and assign it his age, for example, carl = 26

List and Dictionary

Before going deeper in the matter, I want you to learn something more about variables.

So far, you learn that *variables* have many *types*, but Python also have *higher types of variables*, that we call **data structure**. Basically, a **data structure** is a *super variable* made of several *basic variables* (the scalar ones)

There are many **data structures** out there but for this workshop we will only need two, **Dictionaries** and **Lists**

Let's start with **list**.

In the previous part, conditions, we tested the age of several different students. There were only 3 of them but still, it was already a bit of a hassle. What if there was 10 of them?

To solve this kind of issue, we create a data structure called list, that act a bit like an indexed container (or a drawer). It would be way more convenient if we could do this:

Students = "Carl", "Paul", "George", etc ***(this is not an accurate syntax, just an example)***

But how do we tell python we want to use a list and how does it look like in code?

list.py

```
# basic way of assign
student_one = “carl”
student_two = “paul”
# with list
students = [] # empty list
students.append(“carl”) #add an element to an empty list
# OR
students = [ “carl”, “paul”] # or assign the list content directly
# then we print
print(paul) # print the basic variable
print(students[0]) # we print the value inside the first entry of our dictionary
print(students)
```

Warning

In computer science and for list in particular, we start counting from 0, not 1

The first element index is 0, the second is 1

Clue

What if you try to put a list in a list?...

Now, let's talk about dictionary. Dictionary is a kind of list, but better! Instead of using number as index, you can use a string!

It is way is way easier to associate data together, like a name with an age

Let's give it a try!

dictionary.py

```
# basic way
carl = 26
paul = 14
# dictionary way
students = {} # empty dictionary
students["paul"] = 14 #assign a value to the entry paul of our dictionary
# OR
students = { "carl": 26, "paul": 14 } # or assign the dictionary  content directly
print(paul)
print(students["paul"]) # we print the value inside the paul's entry of our dictionary
print(students)
```

Clue

What if you try to put a dictionary in a dictionary?

What if you try to put a list in a dictionary?

What if you try to put a dictionary in a list?

Loop

Loops! The second most important principle in basic programming. The purpose of computing is the automation of repetitive tasks. But so far, nothing really « handy » is achievable with what we learn so far.

A loop is a form of condition with a special extra, repetition. Succinctly, as long as your *condition statement* is true, the same action or block of code will be repeated.

Notice the word block, in programming, you can isolate some part of your code inside a block.

For if statement for example and now, for loops. Block of code are *delimited* with tabulation.

You can tell python you want to make a loop with 2 key words:

- while *(basic way to loop)*
- for…in… *(to loop through list or dictionary)*

loops.py

```
counter = 0
while counter < 10 :
    print("plus one !") #here same indentation means
    counter = counter + 1 # same block of code
#here students[0],... is replaced by student, and the loop go through the wall list/dictionary
for student in students:
    print(student)
```

Clue

If you put a list in a list, maybe should you try a for in a for…

Function

Last but not least, functions! Function are the most used and maybe the most important and hardest thing to wrap your head around in software engineering basics.

You can see that if we put everything, we did so far in one same file, it will be quite messy to understand.

Another problem will be to reuse the code. Variables helped us reusing result of arithmetic calculus and functions are just the same, but for *block of code*. If you want to reuse code that do only one thing, like computing the square of a number.

To tell python we are making a function, you need to use the word def.

functions.py

```
def square (nb): # we define our function here
    square_nb = nb  * nb # everything in the function block belong to the function
    return square_nb
nb = 1
result = square(2) # calling the function by its name and giving it 2 as a parameter
print(result)
print(square(nb)) # here we give it the nb variable as a parameter
```

Here there is many things to explain.

To use a function, your first need to define it. Then you can call it every time you need it by its *name*. just like a variable. Here the function name is *square*.

Between *the parenthesis* is what we call a parameter. It's a way to give an input to your function or to pass it data. The data can be a *variable* or an integer or a string right away. You can have as many parameters as you want *but all of them are expected* to be given when the function is called.

The second thing is the return keyword. For the function to *output* the result of what it does, you need to return it. When we type *result = square(…)*, we assign to *result* what square returns.

Try to put code you already done in previous exercise into function and call them!

Clue

*Be careful with block of code (**scope** in Software Engineering). in the same code you can give*

Different variables the same name if they are in different block/scope.

They won't erase or replace each other and doesn't even know they exist,

Just like nb in the previous example. Be careful with that!