



Programming

An Introduction to Python

Vassilis Markos, Mediterranean College

Week 01

Contents



- 1 A Brief Intro
- 2 What is Python?
- 3 Getting Started With Python
- 4 Fun Time!

A Brief Intro

Desperate Times, Desperate Measures

Since we are transitioning on a new platform, things are still a bit quirky. So, to make sure we keep track of who is here and who is not, please **scan the QR code shown next** or **click the link below it** and fill in this form with your information (confidential).



<https://forms.gle/4yiYhonrjuVs4sCv6>

Desperate Times, Desperate Measures

For similar reasons, we will also be using a public shared drive folder to keep our materials as long as our platform is a bit unstable. To visit the platform and download this lecture's materials please use the QR shown right or the link below.




https://drive.google.com/drive/folders/1cUY_XNJLyGNHRRgfJbxXemf8AFgiMnaF?usp=sharing

Some More Technicalities



- Inside computer labs **we are not allowed to eat or drink** (coffee included), **with the exception of water.**
- **Do not try to disassemble any part of the lab equipment!** It might well do so by itself, if needed!
- Attendance will be checked **every hour**:
 - Classes start at xx:00 and end at xx:50.
 - You can enter class with a 10 minutes delay.
 - You can also enter class after that allowed 10 minutes delay, but, in this case, you will have been marked as absent.
 - In case of scheduled absence, please inform me prior to our class via email or in person.

sudo echo whoami



- Vassilis (name) Markos (surname): it also works the other way around.
- My interests include: AI / XAI, Quantum Computing, Data Science, Operator Theory, Statistics, Software Development... (this might be of your interest too in the future, in case you are looking for a **thesis supervisor**).
- I am mostly a Linux / UNIX user, so, please, be kind towards my ignorance regarding MS Windows. :)

What Is This Course?



What is this course about? (multiple choice, one correct answer):

- ☐ a Interesting and cool programming stuff.
- ☐ b A good chance to deeply understand Python.
- ☐ c A process where you have to write computer programs.
- ☒ d **All of the above.**

In case you already know Python, don't worry, we will also cover more complex stuff in this course, so, stay tuned!

Module(s) Assessment



All coursework will be submitted **directly on UDO** (you will get your credentials soon if you have not yet). This means that:

- Delayed submissions are not possible (including submissions by email etc).
- The only way to get an extension is by formally applying for one **at the University of Derby**.
- All UDO submissions are by default checked by Turnitin.

Module(s) Assessment



So, in order for things to run smoothly:

- You should **work** on coursework sufficiently **prior to the deadline!**
- In case of extension, **make sure you have all necessary documents** available, e.g., doctor's written diagnosis, in case of a medical condition.
- I will be accepting **drafts** which can be discussed during **office hours** to make sure things are okay for submission.
- Do not make (excessive) use of Generative AI!

Module's Coursework



There is a single-part coursework for this course:

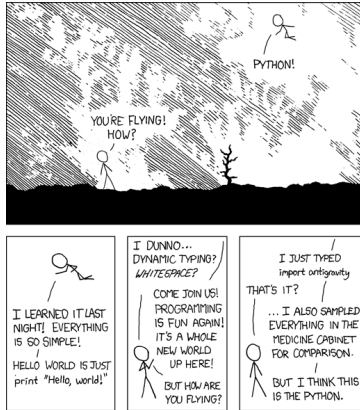
- **Coursework 1 (100%):** A programming portfolio including Python programs developed by you covering various topics of this course (flow control, loops, functions, classes, etc).

We will work on your portfolio in class as well as on similar exercises, so, do not worry about difficulty. Just pay attention to the lectures and play around with Python on your own!



What is Python?

Some Programmer Humour First...



The moment you understand this is the moment you'll have learnt Python. Source: <https://xkcd.com/353/>.

Python in a Nutshell

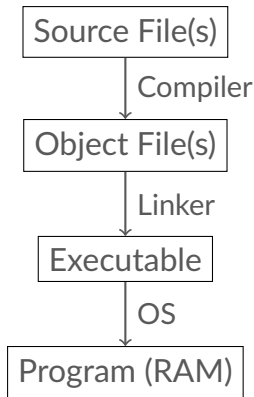


Python is:

- An **interpreted** language (so, no compiler here). This means that commands are executed sequentially, one after the other.
- Quite **high-level**, which means that it makes several decisions for us, making coding much simpler.
- **Tricky**, because of its high level, since you might think that your code works in some way, while it actually works quite differently.
- **Slow**. Really slow, mostly because it has to make all those decisions for us.

The Compilation Process (in general)

- **Source File:** The code file we write on a PC.
- **Object File:** The compiled code file, usually machine / CPU instructions (e.g., assembly).
- **Executable:** A file that can be executed by a certain Operating System (OS).
- **Compiler:** A computer program that transforms a source file to an object file.
- **Linker:** A computer program that interconnects a source file with other required existing programs (libraries).



The Interpretation Process (in general)

The interpretation process is more straightforward:

- The interpreter consumes `x = 5` and executes it.
- The interpreter consumes `y = 7` and executes it.
- The interpreter consumes `z = x + y` and executes it.
- The interpreter consumes `print(z)` and executes it.

```
1 # source/interpreter_sample.py
2
3 x = 5
4 y = 7
5 z = x + y
6 print(z)
```


The Bare Truth

Python is not actually interpreted; it is **compiled and then interpreted**:

- Python code is first **compiled** to Python bytecode, i.e., a lower level code generated by Python's compiler.
- Then, this bytecode is:
 - either **interpreted** by a Python interpreter, or;
 - **compiled**, by a python bytecode compiler.
- In general, whether a certain language is interpreted or compiled is not a feature of the language per se but of the **implementation** of the language. For an interesting discussion of that:

<https://stackoverflow.com/a/6889798/12461122>

Installing Python



In order to play around with Python you need to install it:

- Avoid installing Python 2.x, as it is no longer supported. **Install Python 3.8 or later**, instead.
- For MS Windows, you can find the corresponding releases here:
<https://www.python.org/downloads/windows/>
- For Linux / Unix Python is probably already installed, however you can follow the instructions found here:
<https://docs.python-guide.org/starting/install3/linux/>
- You can also use online interpreters, which I would discourage you to, since we will sooner or later need to work on our own machine.

Installing a Text Editor



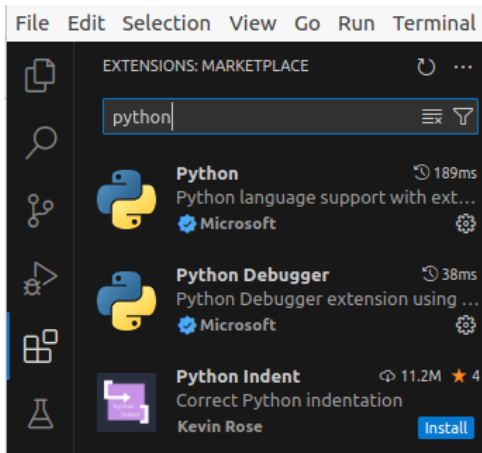
In order to write some Python code you need a text editor:

- **You cannot use MS Word** or some other similar editor.
- One popular choice is MS **Visual Studio Code** with any relevant plugins. We will be using this in our classes as it is quite convenient.
- You can download it from: <https://code.visualstudio.com/>.
- You can also choose any other editor / programming environment, like **PyCharm**.

Configuring VSCode Plugins

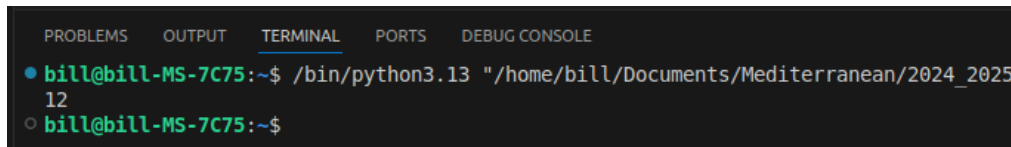
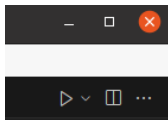
To install python plugins:

- Click on the “four squares” icon on the left sidebar.
- Write “python” in the search bar.
- Install the two first plugins, “Python” and “Python Debugger”, as shown right.



Running Python Code in VSCode

To execute a Python program simply press the “Play” button on the top right of the editor (see right). Any results should be displayed at the bottom of your window, as shown below.

A screenshot of the VS Code interface with the terminal panel open at the bottom. The terminal shows a command being executed: `/bin/python3.13 "/home/bill/Documents/Mediterranean/2024_2025_12"`. The prompt is `bill@bill-MS-7C75:~$`. The terminal output shows the command being executed and the prompt returning.

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE
● bill@bill-MS-7C75:~$ /bin/python3.13 "/home/bill/Documents/Mediterranean/2024_2025_12"
○ bill@bill-MS-7C75:~$
```

PATH Variables



Regardless of your Operating System (OS), you have to be really careful with PATH variables:

- Such variables inform your OS where to look for Python (and other stuff).
- For Windows, when installing Python check the box that asks you whether to append Python to the PATH.
- Instructions for Windows, Linux and Mac may be found here:
<https://realpython.com/add-python-to-path/>

Getting Started With Python

hello_world.py



It is usual to start a course in programming by presenting a simple program which prints “Hello, world!” on our console. So, let’s write this in Python!

```
1 # source/hello_world.py
2
3 print("Hello, world!")
```

Execute the above program! Does it work as expected?

Some Operations



Right you can see a simple Python program.

- Without executing it, can you guess what will be printed on your screen?
- Execute it and verify your conjecture.
- Explain what those peculiar + and * do.

```
1 # source/magic_001.py
2
3 x = 5
4 y = 4 * x + 6
5 print(y)
```

More Operations



Right you can see a simple Python program.

- Without executing it, can you guess what will be printed on your screen?
- Execute it and verify your conjecture.
- Explain how `**` works.

```
1 # source/magic_002.py
2
3 x = 2
4 y = 4 ** x + 6
5 print(y)
```

More Operations



Right you can see a simple Python program.

- Without executing it, can you guess what will be printed on your screen?
- Execute it and verify your conjecture.
- Explain what's the difference between `/`, `//`, and `%`.

```
1 # source/magic_003.py
2
3 x = 19
4 y = x / 5
5 z = x // 5
6 w = x % 5
7 print(y, z, w)
```

More Operations



Right you can see a simple Python program.

- Without executing it, can you guess what will be printed on your screen?
- Execute it and verify your conjecture.
- Explain why the program works the way it does.

```
1 # source/magic_004.py
2
3 x = 6
4 y = x * (x - 4)
5 z = x * x - 4
6 print(y, z)
```

What About = and ==?

Right you can see a simple Python program.

- Without executing it, can you guess what will be printed on your screen?
- Execute it and verify your conjecture.
- Explain why the program works the way it does.

```
1 # source/magic_005.py
2
3 x = 1
4 y = x == 2
5 z = x + 3 == 4
6 print(y, z)
```

Summarising Our Observations



Operator	Description
+, -, *, /, **	Addition, subtraction, multiplication, division and power operators.
//, %	Integer division and modulus (remainder) operators.
==	Comparison operator (equality)
<, >, <=, >=	Comparison operators (inequalities, as in maths)

What's Your Type?



Run the following program on your PC. What does it print? Can you explain its output (or, even better, guess it)?

```
1 # source/data_types.py
2
3 x = 1.2
4 y = 5
5 z = x == y
6 w = "This is a message"
7
8 print(type(x), type(y), type(z), type(w))
```

Python Data Types



Type	Description	Example
int	Represents integer numbers	-5, 0, 82
float	Represents floating point numbers	-6.54, 4.0, 89.2
string	Represents sequences of printable characters	"foo", "Python is FUN"
bool	Represents one of the two available Boolean values	True, False

Reading User Input



So far, we have printed stuff on screen, but it would be nice to ask the user for some input, in order to make our programs more interactive. This can be achieved with Python's `input`, as in the following example (run it on your machine):

```
1 # source/read_input.py
2
3 name = input("What is your name? ")
4 print("Your name is:", name)
```

Can you explain what happened?

What About Non-String Input?



What if we want to read something that is not a string? For instance, execute the following program:

```
1 # source/kaboom.py
2
3 x = input("Please, enter your age: ")
4 y = x + 10
5 print("Your age plus 10 equals: ", y)
```

What did happen? What does the output read? How can we fix that?

Becoming a Developer

You must see something like the following:

```
Traceback (most recent call last):
  File "/home/bill/Documents/Mediterranean/2024_2025/Fa
    y = x + 10
      ~^~
TypeError: can only concatenate str (not "int") to str
```

- Let us look online for what other developers have to say...
- Write on some search engine the main message of your error:
TypeError: can only concatenate str (not "int") to str
- How can we fix this?


Python Errors



In Python, a `TypeError` usually refers to an error regarding some misuse of data types.

- In our case, as you must have found online, the error is about the data type returned by `input()`.
- `input()` by default returns a **string**.
- So, in order to handle its input as a number, say an integer, we should **cast it to an integer**.
- This can be done with the help of Python's function `int()`, which takes a Python object, e.g., a string, and returns an integer, if possible, or raises an error, otherwise.

Handling Non-String Input



The correct way to parse user input as an integer is as follows:

```
1 # source/noboom.py
2
3 x = int(input("Please, enter your age: "))
4 y = x + 10
5 print("Your age plus 10 equals: ", y)
```

What we did above was to wrap `input()` with `int()`, so `int()` parses its output and converts it to an integer.

Python's Casting Functions



Function	Description
<code>int()</code>	Casts its argument to an integer , if possible, or raises a <code>ValueError</code> .
<code>float()</code>	Casts its argument to a float , if possible, or raises a <code>ValueError</code> .
<code>str()</code>	Casts its argument to a string , if possible, or raises a <code>ValueError</code> .
<code>bool()</code>	Casts its argument to a boolean value, if possible, or raises a <code>ValueError</code> .

This, and this, or that, but not the other...



- How would you express in Python the following?
"A number, x , is greater than 5."
- One possible python expression could be: $x > 5$.
- How about that?
"A number, x , is less than 12."
- Similarly, it could be: $x < 12$.
- How about that?
"A number, x , is greater than 5 and less than 12."

Composite Logical Expressions

- In Python, such composite logical expressions are written with the help of **logical operators**.
- Logical operators are the same as mathematical operators (+, −, *, /, etc) just for boolean values instead of numbers.
- The following program checks whether a number is between 5 and 12 (run this a few times with various inputs):

```
1 # source/composite_conditions_001.py
2
3 x = float(input("Please enter a number: "))
4 y = x > 5 and x < 12
5 print("Is it between 5 and 12?", y)
```


Python Logical Operators



Operator	Description
and	Corresponds to “both” in everyday language.
or	Corresponds to an “inclusive” or, i.e., either one of the two options or both may hold.
not	Corresponds to everyday language’s “not” (negation).

How Do Logical Operators Work?

Fill in the following table with the values you think each operator yields in each case. You can draw inspiration from your everyday experience:

x	y	not x	x and y	x or y
True	True			
True	False			
False	True			
False	False			

Testing Your Conjecture

Fill in the following table using Python to evaluate each expression (write a small program that does this for each case):

x	y	not x	x and y	x or y
True	True			
True	False			
False	True			
False	False			

Compare Python's view of logical operators with yours. What do you observe?

Python + Logical Operators



The values Python assigns to those expressions are:

x	y	not x	x and y	x or y
True	True	False	True	True
True	False	False	False	True
False	True	True	False	True
False	False	True	False	False

Useful Resources



There are several high quality Python resources around the web:

- Usually quite in depth and informative tutorials can be found at:
<https://realpython.com/>. I highly recommend this as a source for your personal learning.
- The official Python Tutorial, while quite lengthy, goes in great detail to cover all interesting and useful features of the language:
<https://docs.python.org/3/tutorial/index.html>
- There is also a great (and long) video tutorial available on YouTube:
<https://www.youtube.com/watch?v=rfscVS0vtbw>



Fun Time!

In-class Exercise #001



A foot (imperial distance unit, not part of one's body) is equal to 0.3048 metres.

- ① Write a python program that asks the user a distance in feet and prints on screen the same distance in metres.
- ② Write another python program that asks the user a distance in metres and prints on screen the same distance in feet.

Test your programs by running them for multiple inputs!

In-class Exercise #002



Write a Python program that:

- Asks the user to provide two integer numbers, let m and n .
- Checks if m divides n .
- Prints “Divisor!” if m divides n and “Not Divisor!” otherwise.

Test your program by running it for multiple inputs!

This requires Python machinery we have not discussed yet, so, The Internet is your friend!

In-class Exercise #003



Write a Python program that:

- Asks the user to provide two integer numbers, let m and n .
- Checks if m divides n and m^2 does not divide n .
- Prints “Simple Divisor!” if the above holds and “Non-simple Divisor!” otherwise.

Test your program by running it for multiple inputs!

This requires Python machinery we have not discussed yet, so, The Internet is your friend!

In-class Exercise #004



Run the following Python program:

```
1 # source/tabbing.py
2
3 x = 4
4   y = 3
5 z = y * x
6     print(z)
```

What is the console output? Try to make it work as intended!

In-class Exercise #005

For which values of `x` does the following program print `True` and for which `False` (do not run it!):

```
1 # source/bar.py
2
3 x = int(input("Pleasse, enter an integer: "))
4 a = x == 1 or x == 4
5 b = x < 3 and x > -5
6 c = a or b
7 print(c)
```

Run the program. Did it work as you expected? Explain why it works that way.

In-class Exercise #006



What input should you give in the following program in order to print False?

```
1 # source/foo.py
2
3 x = int(input("Please, enter an integer: "))
4 a = x == 1 or x == 4
5 b = x != 1 and x != 4
6 c = a or b
7 print(c)
```

Explain your answer in detail!

In-class Exercise #007



Write a Python program that:

- Asks the user to provide two floats, a , and b .
- Stores the value of a to b and that of b to a .
- Prints the new values of a and b .

Explain your chain of thought in detail.

Homework 1



As your homework, you will have to explore some features of Python we have not yet explored in class. To do so, you will have to follow this **excellent tutorial**:

<https://realpython.com/courses/basics-conditional-logic-control-flow/>

Take time to **watch all short videos** / read the transcripts (whichever you prefer) and **solve all provided exercises**. Share any thoughts, questions, your solutions to the tutorials problems and anything else (relevant) via email at: `v.markos@mc-class.gr`.

Homework 2



This is a more advanced homework, building on top of the previous one. So, start working on this one once (and if) you are done with the previous one. Otherwise, don't worry, we will cover much of that material in the following lectures. Given that, start working on the following tutorial:

<https://realpython.com/courses/conditional-logic-control-flow-exercises/>

Share anything relevant at: `v.markos@mc-class.gr`.

Any Questions?

Do not forget to fill in
the questionnaire shown
right!



<https://forms.gle/dKSrmE1VRVWqxBGZA>