Python level 1

Introductory course
Class 1

Fábio Neves & Luis Dias

September 9

AGENDA

Part 1 – Introduction to python

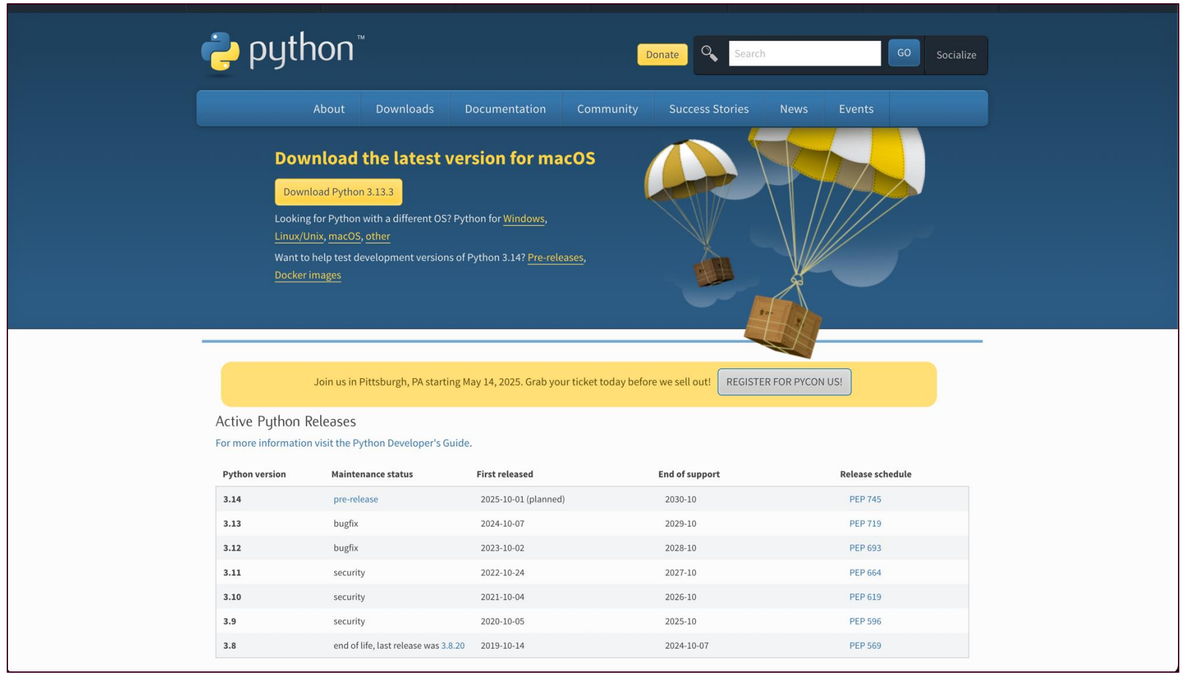
1. **The basis**
   a. Installation
   b. IDE
   c. Main libraries
   d. Syntax
   e. Variables
   f. Data types
   g. Basic mathematical function
   h. Operators
   i. If…Else
   j. Loops

The setup

The basis – installing Python

Download a Python:

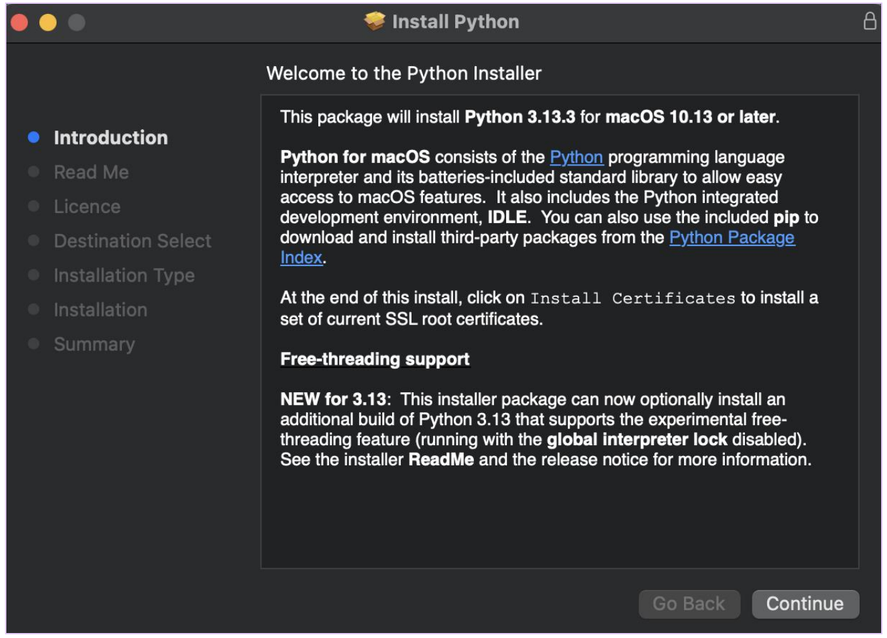
1. Visit the official Python website: Python Downloads (https://www.python.org/downloads/);
2. Click on the "Downloads" tab and select the version suitable for your operating system (Windows, macOS, or Linux);
3. Install the latest Python version.



The basis - installing Python

Run the Installer:

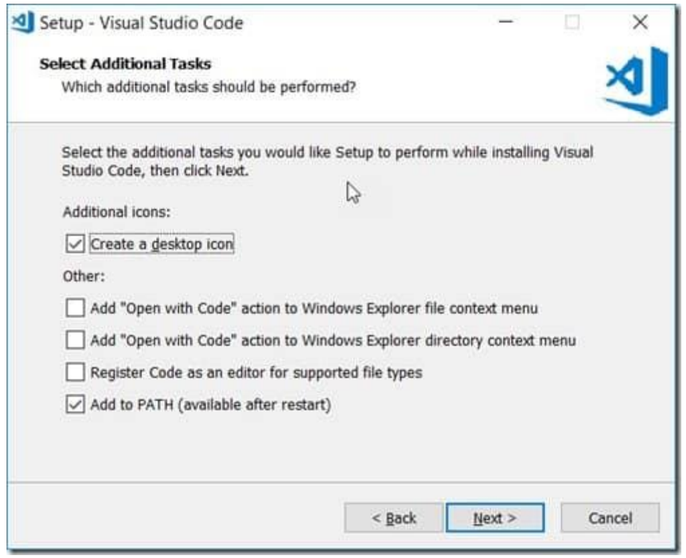
1. For **Windows**: Double-click the downloaded installer (.exe) and follow the installation wizard;
2. For **macOS**: Double-click the downloaded installer (.pkg) and follow the installation instructions;
3. For **Linux**: Open a terminal and navigate to the directory where you downloaded the installer. Run the command: sudo dpkg -i <installer_filename>.



The basis – installing VS Code on Windows

Install VS Code

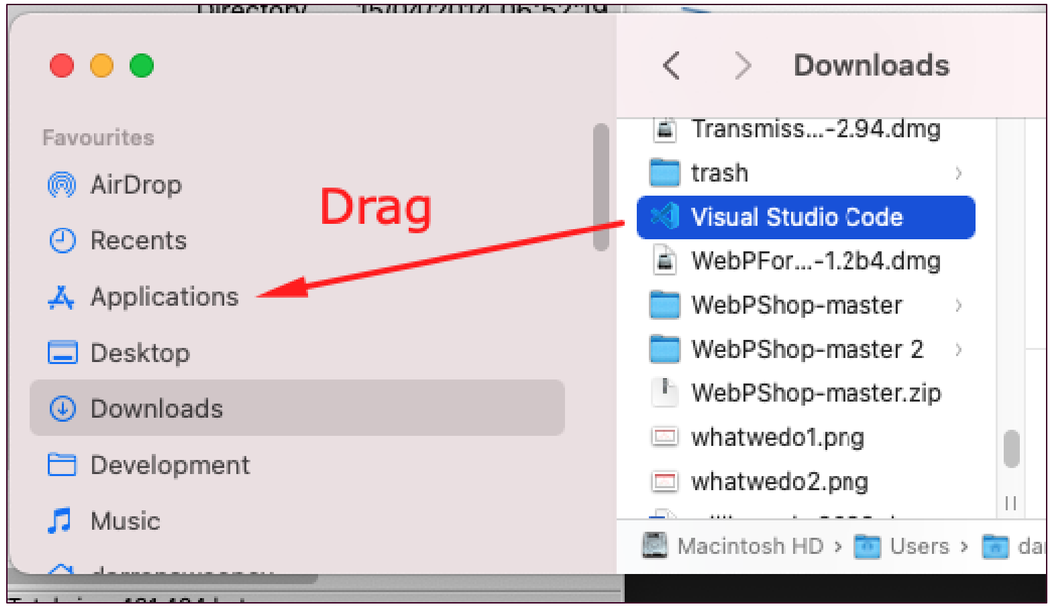
- **For windows**
  - Run the downloaded .exe installer.
  - Accept the license agreement.
  - Choose the installation location (default is fine for most users).
  - **Recommended:** Check the following options during setup:
    - Add "Open with Code" to context menu
    - Register Code as editor for supported file types.
  - Click Install.



The basis – installing VS Code on MAC

Install VS Code

- **For MAC**
  - Open the downloaded .zip file.
  - Drag the Visual Studio Code app into your Applications folder.
  - You can optionally add VS Code to your Dock for easy access.



The basis - VS Code essentials

Set up a virtual environment

- **Create** the virtual environment: In the terminal and insider your project folder type.

```
terminal
cd path-to-your-project-folder python -m venv venv
```

- **Activate** the virtual environment.

Windows `.\venv\Scripts\Activate.ps1` **MAC/Linux** `source venv/bin/activate`

- After activation, your **terminal** will show the environment name at the beginning, like.

```
terminal
(venv) your-computer-name:your-project-folder username$
```

The basis – Virtual Environments

A **virtual environment** is like a **special, isolated space** on your computer where you can install Python packages **just for one project**, without messing with other projects or your system's Python.

Think of it like:

📦 A "project box" that keeps everything you need inside.

🛡️ It protects your project from problems like version conflicts (different projects needing different versions of the same package).

Example:

Project A needs **Pandas 3.2**.

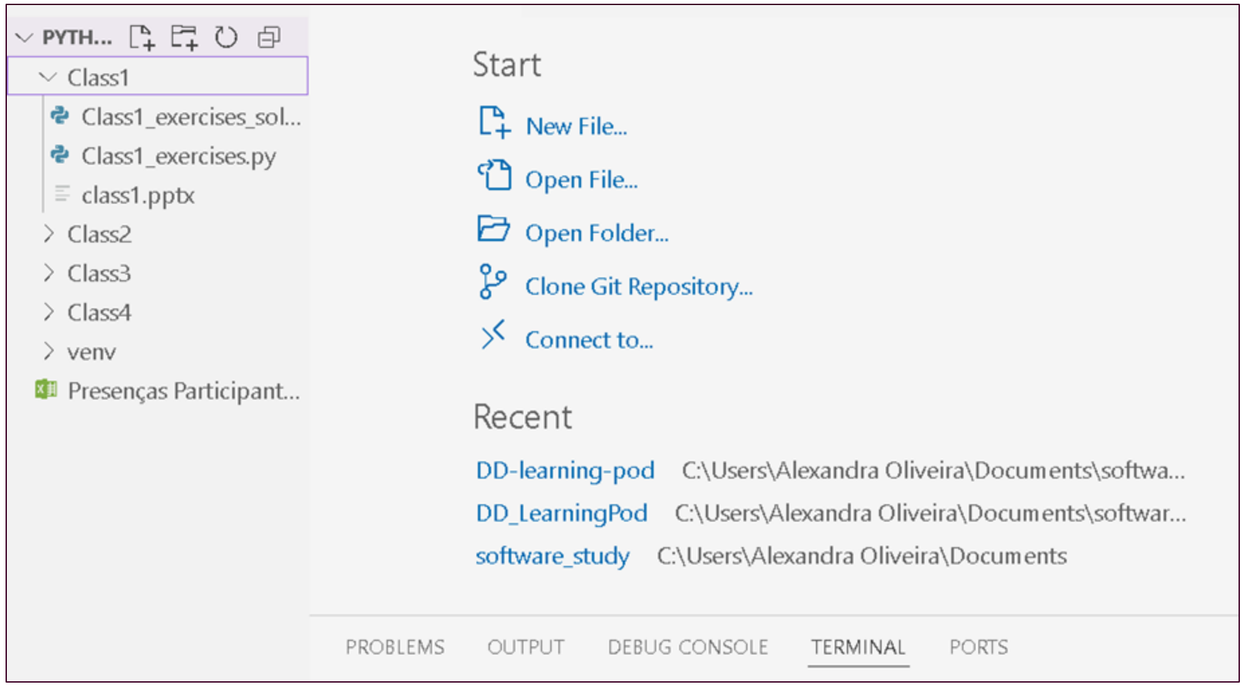
Project B needs **Pandas 4.**

If you use a virtual environment, **each project can have its own Pandas version**, no problems!

Without a virtual environment, everything would install globally, and projects could easily break each other.

The basis - Virtual Environments

An integrated development environment (IDE) is a software application that helps programmers develop software code efficiently. It increases developer productivity by combining capabilities such as software editing, building, testing, and packaging in an easy-to-use application. Just as writers use text editors and accountants use spreadsheets, software developers use IDEs to make their job easier.



VS Code is one of the best IDEs for programming in Python

Python introduction

Important concepts

The Python language relies on 4 types of entities:

- **Variables**: store data of different types by giving them a name;
- **Functions**: used to perform data processing, as well as input/output operations;
- **Libraries:** a collection of precompiled codes that can be used later in a program for specific, well-defined operations. Libraries help reduce coding errors, make programmers more efficient, and make software smaller in size (and lines of code). Instead of writing long lines of code to accomplish a common task, coders can simply call upon a library – often with a single line of code – to perform that task instead;
- **Programs (scripts):** usually a set of instructions, including function calls with a defined flow (order) for the resolution of one (or several) tasks.

Popular libraries

- **Numpy**: stands for “Numerical Python”. It is the most commonly used library that supports large matrices and multi-dimensional data. It consists of in-built mathematical functions for easy computations. Array Interface is one of the key features of this library;
- **Pandas**: pandas are an important library for data scientists. It is an open-source machine learning library that provides flexible high-level data structures and a variety of analysis tools. It eases data analysis, data manipulation, and cleaning of data. Pandas support operations like Sorting, Re-indexing, Iteration, Concatenation, Conversion of data, Visualizations, Aggregations, etc.;
- **Scipy**: the name “SciPy” stands for “Scientific Python”. It is an open-source library used for high-level scientific computations;
- **Matplotlib**: responsible for plotting numerical data. And that’s why it is used in data analysis. It is also an open-source library that plots high-defined figures like pie charts, histograms, scatterplots, and graphs.

Syntax – The print( ) functions

print() is the keyword for printing anything we want: a sentence, a value of variables or any other object.

example.py

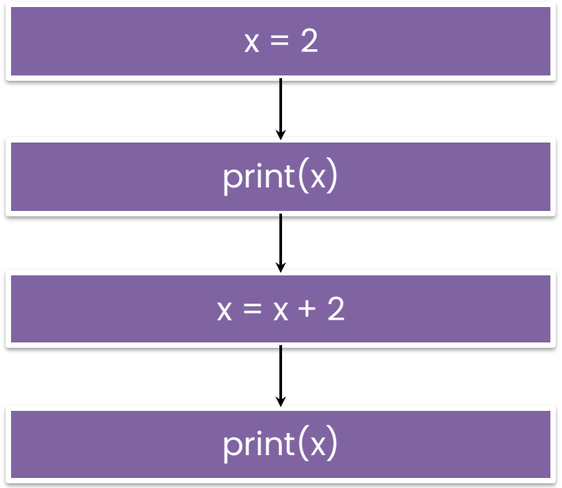
```
print("Hello, World!")
>> Hello, World!

print("This is class 1 of python")
>> This is class 1 of python
```

Syntax – Program flow

Like a recipe or installation instructions, a program is a sequence of steps to be done in order.

1. Sequential Steps



example.py

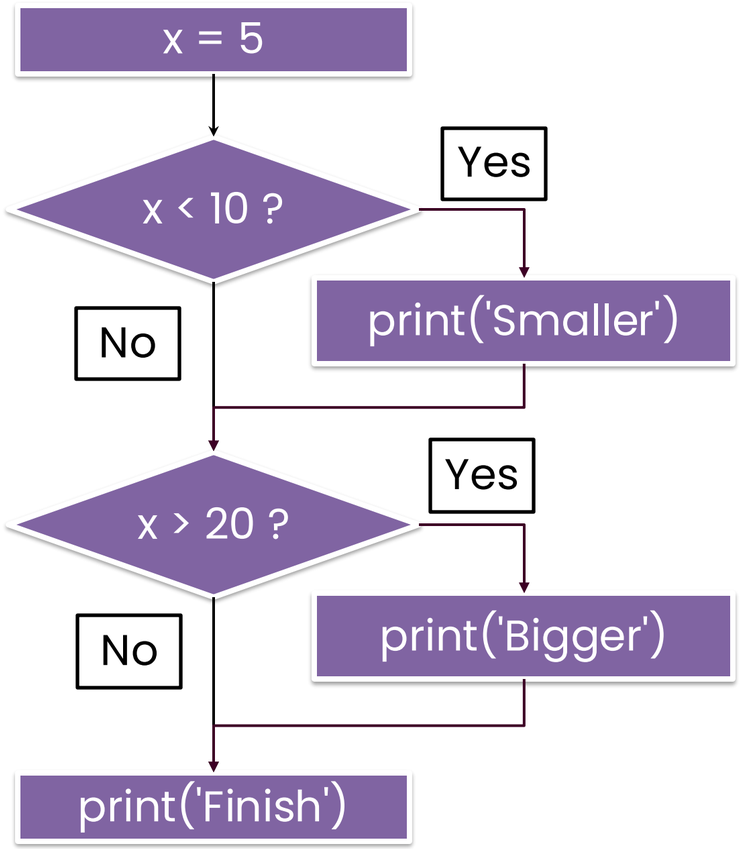
```
x=2
print(x)
>> 2

x = x + 2
print(x)
>> 4
```

Syntax – Program flow

Like a recipe or installation instructions, a program is a sequence of steps to be done in order.

2. Conditional steps



example.py

```
x=5
if x<10:
  print("Smaller")
if x>20:
  print("Bigger")
print("Finish")
```

Correct program

Indentation

```
x=5
if x<10:
print ("Smaller"
if x>20:
print("Bigger")
print("Finish")
```

Missing bracket

Wrong program

Syntax - Variables

Creating variables

```
x = 5
y = "John"
```

Casting

```
x = str(3) # x will be '3'
y = int(3) # y will be 3
z = float(3) # z will be 3.0
```

Get the type of the variables

```
x = 5
y = "John"
print(type(x))
print(type(y))

>> <class 'int'>

>> <class 'str'>
```

Give names to variables

```
myvar = "John"
my_var = "John"
_my_var = "John"
myVar = "John"
MYVAR = "John"
myvar2 = "John"
```

```
2myvar = "John"
my-var = "John"
my var = "John"
```

Many Values to Multiple Variables

```
x, y, z = "Orange", "Banana", "Cherry"
print(x)
print(y)
print(z)

>> Orange

>> Banana

>> Cherry
```

Syntax – Data types

Data types	Notation in python	Example
Text	str	x = "Hello World"
Numeric	int float complex	x = 20 x = 20.5 x = 1j
Sequences	list	x = ["apple", "banana", "cherry"]
Mapping	dict	x = {"name" : "John", "age" : 36}
Set	set	x = {"apple", "banana", "cherry"}
Boolean	bool	x = True
None	None	x = None

Mathematical functions

Some python functions

example.py

```
x = min(5, 10, 25)
y = max(5, 10, 25)
x = abs(-7.25)
x = pow(4, 3) # x has the value of 4 to the power of 3
```

The Math Module

Python also has a built-in module called **math**, which extends the list of mathematical functions. To use it, you must import the **math** module (if it is already installed).

example.py

```
import math
x = math.sqrt(64) # x has the value of the square root of 64
x = math.ceil(1.4) # x is equal to 2
x = math.floor(1.4) # x is equal to 1
x = math.pi # x has the value of pi number
```

It does have many other functions. For more information, check the documentation of the math module (https://docs.python.org/3/library/math.html).

Syntax - Operators

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y
//	Floor division	x // y

Syntax – Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
//=	x //= 3	x = x // 3
**=	x **= 3	x = x ** 3
^=	x ^= 3	x = x ^ 3

Syntax – Comparison and Logical operators

Type of operator	Operator	Description	Example
Comparison	==	Equal	x == y
	!=	Not equal	x != y
	>	Greater than	x > y
	<	Less than	x < y
	>=	Greater than or equal to	x >= y
	<=	Less than or equal to	x <= y
Logical	and	Returns True if both statements are true	x < 5 and x < 10
	or	Returns True if one of the statements is true	x < 5 or x < 4
	not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Syntax - If and else conditions

An "**if statement**" is written by using the if keyword.

example.py

```
a = 33
b = 200
if b > a:
  print(”b is greater than a")

>> b is greater than a
```

The ***elif*** keyword is Python's way of saying, "if the previous conditions were not true, then try this condition”.

example.py

```
a = 33
b = 33
if b > a:
  print(”a is greater than a")
elif a == b:
  print(”a and b are equal")

>> a and b are equal
```

Syntax - If and else conditions

The ***else*** keyword catches anything which isn't caught by the preceding condition.

Motivation.txt

```
a = 200
b = 33

if b > a:
  print("b is greater than a")
elif a == b:
  print("a and b are equal")
else:
  print("a is greater than b")


>> a is greater than b
```

Syntax – Loops

While loop

With the **while** loop, we can execute a set of statements as long as a condition is true.

example.py

```
i = 1

while i < 6: # print as long as i is less than 6
  print(i)
  i += 1

>> 1 2 3 4 5
```

The **while** loop requires **relevant variables** to be ready; in this example, we need to define an **indexing variable**, **i**, which we **set to 1**.

With the **break** statement, we can stop the loop even if the while condition is true.

example.py

```
i = 1
while i < 6: # print as long as i is less than 6
  print(i)
  if i == 3:
    break
  i += 1

>> 1 2 3
```

Syntax - Loops

For loop

A **for** loop is used for iterating over a sequence (that is either a list, a tuple, a dictionary, a set, or a string).

example.py

```
fruits = ["apple", "banana", "cherry"] # This is a list of strings
for x in fruits:
  print(x)


>> apple banana cherry
```

The **for loop does not require an indexing** variable to set beforehand

With the **break** statement, we can stop the loop even if the for condition is true.

example.py

```
fruits = ["apple", "banana", "cherry"] # This is a list of strings
for x in fruits:
  print(x)
  if x == "banana":
    break

>> apple banana
```

Syntax – Loops

To loop through a set of code a specified number of times, we can use the **range()** function. It returns a sequence of numbers, starting from 0 by default, increments by 1 (by default), and ends at a specified number.

The **range()** function defaults to 0 as a starting value. However, it is possible to specify the starting value by adding a parameter: **range(2, 6),** which means values from 2 to 6 (but not including 6).

The **range()** function defaults to increment the sequence by 1. However, it is possible to specify the increment value by adding a third parameter: **range(2, 30, 3)** .

Motivation.txt

```
for x in range(0,3):
  print(x)

>> 0 1 2
```

Note that range(6) is not the values of 0 to 6, but the values 0 to 5, because by default python starts counting on 0.

```
for x in range(2,6):
  print(x)

>> 2 3 4 5
```

```
for x in range(2, 10, 3):
  print(x)

>> 2 5 8
```

Class Wrap-up

What have we learned today?

- Setting up and using an **IDE** to program in **Python**
- Learned about the power of **Python libraries**
- Learned the **Python syntax** language
- Variables **data types**
- To use Python **integrated** mathematical **functions**
- Learned what is a **Python operator** and what is used for
- Main ingredient of programming: **If….else and Loops**

PRACTICE
PRACTICE
PRACTICE

You won't master a skill if you don't practice!

Exercises – Learn by doing!

In order to facilitate the learning process of Python **we have prepared for each session a python file** where you can find **exercises** that will help you to grasp the introduced Python concepts.

Visual Studio Code

We will use **VS CODE** as our Python program IDE

Exercises for today

Why should you deactivate Copilot? (for now)

As **beginners in Python programming**, it's crucial to focus on truly understanding how code works, rather than just seeing it appear. Tools like GitHub Copilot can be tempting, but they **often offer solutions without explanation**, making it easy to skip the learning process. While these tools are designed to assist, **not replace your thinking**, they can encourage you to rely on solutions you don't fully grasp—and they're not always correct. To truly learn, you need to write, debug, and explore code on your own. **By turning off Copilot** during the early stages of your learning, you give yourself the opportunity to develop real problem-solving skills, build confidence, and create a strong foundation. Later, when you have a solid grasp of the basics, Copilot can serve as a useful support tool, but always approach its suggestions with a critical mindset, not blind trust.

Steps to turn-off GitHub Copilot:

1. Go to Settings (File > Preferences > Settings or press Ctrl+,)
2. In the search bar, type: Copilot.
3. Find the setting GitHub Copilot: Enable.
4. Uncheck it to disable Copilot globally.

THANK YOU

Questions?

Fábio Neves fnssm26@gmail.com

Luis Dias luis.f.s.m.dias@gmail.com