Introduction to Bioinformatics using Python

Lecture 1 Introduction to Programming

Dr. Alexey Larionov

28 October 2024

www.cranfield.ac.uk

Lecture plan

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

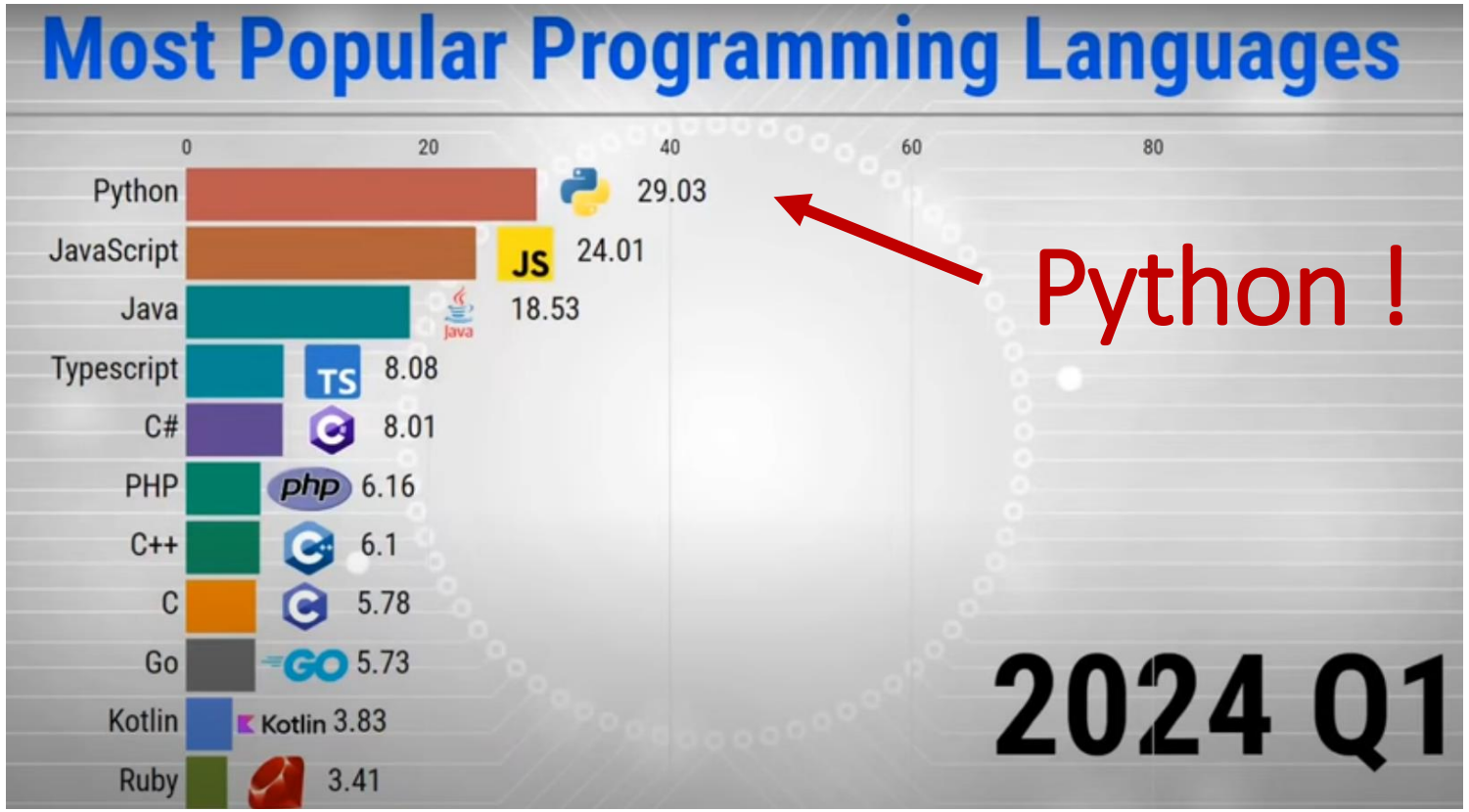
For whom is this lecture ?

For Biologists !

What Programming Languages Exist ?

Object-oriented / Functional

Dynamic / Typed



Interpreted / Compiled

General purpose / Specialised

High-level / Low-level

See ranks from 1965 till 2024 : https://www.youtube.com/watch?v=xOW3Cehg_qg

What is a Programming Language ?

- Language as a ***syntax*** (vocabulary, grammar etc)

"Python 3.10 includes new features …"

Added match – case statement …

Removed some deprecated functions …

- Language as a ***software*** to execute what is written with that "syntax"

(or make an executable file of it …)

"I installed Python 3.12 on my laptop …"

CPython

Cython

PyPy

Jython

Lecture plan

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

Program execution

Compiled *vs* interpreted languages

(Pre)compiled binaries

Human-readable code → Compiled languages (e.g. Fortran, C, C++) → **File with machine code executable by OS**

Compiled (build) ***once*** to executable file

Scripts

Human-readable code → Interpreted/Scripting languages (e.g. Bash, Perl, Python) → **OS runs the code compiled by interpreter on-the-fly**

OS-dependent interpreter ***compiles*** the code behind the scene ***each time*** when we run it

Program execution

Compiled *vs* interpreted languages

Human-readable code → Compiled languages (e.g. Fortran, C, C++) → **File with code executable by OS**

Compile ***once*** to executable file

Human-readable code → **"Bytecode"** → **OS-dependent VM (Runtime Environment)**

Java

Human-readable code → Interpreted/Scripting languages (e.g. Bash, Perl, Python*) → **OS runs the code compiled on-the-fly**

OS-dependent interpreter compiles the code behind the scene ***each time*** when we run it

- Strictly speaking: Python is a *dynamic* interpreted language: it compiles to *bytecode* behind the scene and runs it in the virtual machine (runtime environment) each time when we run the code
  There is also "just-in-time compilation" etc

Program execution

Role of OS and dependencies

Code executable by OS
(compiled in advance or on-the-fly)

Operating System
(Linux, Mac OS, Windows, Android)

Already available code libraries for common tasks

Language-specific libraries	**System libraries**
(modules, packages, …)	(modules, packages, DLL-s, …)
e.g. Python: Argparse, Matplotlib, … R: ggplot, shiny, DESeq2, …	e.g. libraries for remote connection, encryption, files compression …

Dependencies

Packages/modules/libraries for standard tasks

Lecture plan

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

Procedural, Functional & Object-oriented programming

It's about a way of thinking
(and the tools facilitating this way)

Simple **step-by step** execution of code, use loops for repetitive actions

Consider everything as a function, use **recursion** for repetitive actions

Group everything to **objects**: keep Data and Methods together

Procedural programming

The most intuitive approach: at the core of any programming language

Just a sequence of steps (with loops and conditional statements)

```
For Each Row in Table
    If Mark > 80%
        Print (Name, Course, Mark)
```

Student ID	Name	Course	Mark
…	…	…	…

Code example

Print excellent students from first 100 rows

Data example

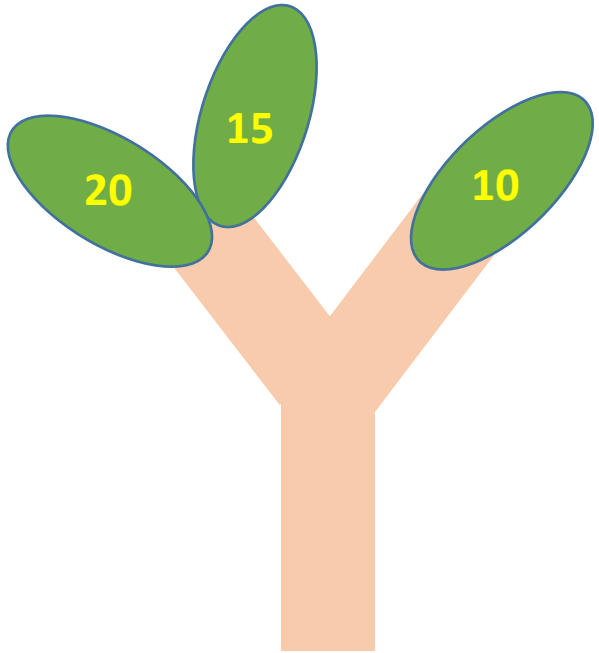
Table with students' academic achievements

Functional programming

Good for hierarchical data (tree instead of a table)

- Theory: Everything is a function, Function output is independent of environment, …
- Practice: ***Recursion*** instead of looping (function calls itself)

```
Define SumLeavesValues (Branch, Sum):
   Go to the next Branching point
   For each Branch:
      If Branch is Leaf: Sum = Sum + LeafValue
      Else: SumLeavesValues (Branch, Sum)
   Return Sum

SumLeavesValues(Root, 0)
```




```
<root>
   <branch>
      <leaf>20</leaf>
      <leaf>15</leaf>
   </branch>
   <branch>
      <leaf>10</leaf>
</branch>
</root>
```

Code example

Sum Leaves Values on a Tree

Root

Data example

XML file

https://realpython.com/python-functional-programming

Object-Oriented Programming (OOP)

Imagine that you run paper-records in a university…

It could be many books …

Demographic data

ID	Name	Age	Gender

Contact details

ID	Name	Address	Phone

Scholarships & awards

ID	Name	Award	Sum

Academic achievements

ID	Name	Assignment	Mark

Course A

ID	Name	Assignment	Mark

Course B

ID	Name	Assignment	Mark

Course C …

Try to calculate average course mark for a student …

Data-type focused records keeping

Keep separate books for separate types of data: Contacts, Demographics, Academic results … (each book containing records for all students)

Problem when you have too many books

Difficult to manage data in a student-focused way, e.g. obtain data for the same student from different books or add a new student to all the books …

Object-Oriented Programming (OOP)

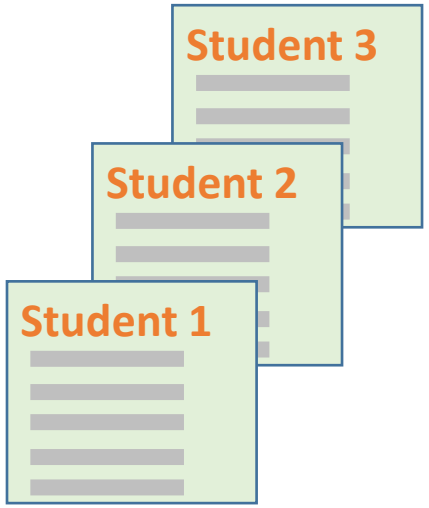
An intuitive way of dealing with complexity

An alternative to keeping and synchronising all these books:

Student Form

Demographic data
Contact details
Scholarships & awards
Academic achievements
…

Template



Individual records
based on the template

Student focused records keeping

- Make a standard template to record all details for a single student in a single form
- Fill the form for each student, when student arrives

Object-Oriented Programming (OOP)

An intuitive way of dealing with complexity

What is an "Object" here ?

For each student: keep all properties (members) and functions (methods) within the same memory object = instance …

Declare Class Student:

ID
Name
Address
Phone
Age
Gender
achievements_table[]
…
average_mark(…)

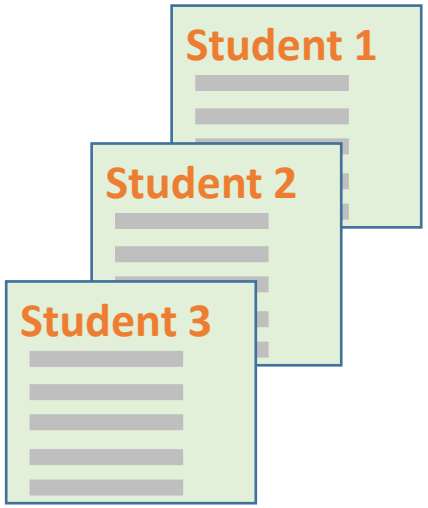
Student Form

Demographic data
Contact details
Scholarships & awards
Academic achievements
…

Student1 = New Student
Student1.ID = s012345
Student1.Name = John

Student2 = New Student
Student2.ID = s543210
Student2.Name = Jane

…



Class
Student

Instances
of the class Student

Lecture plan

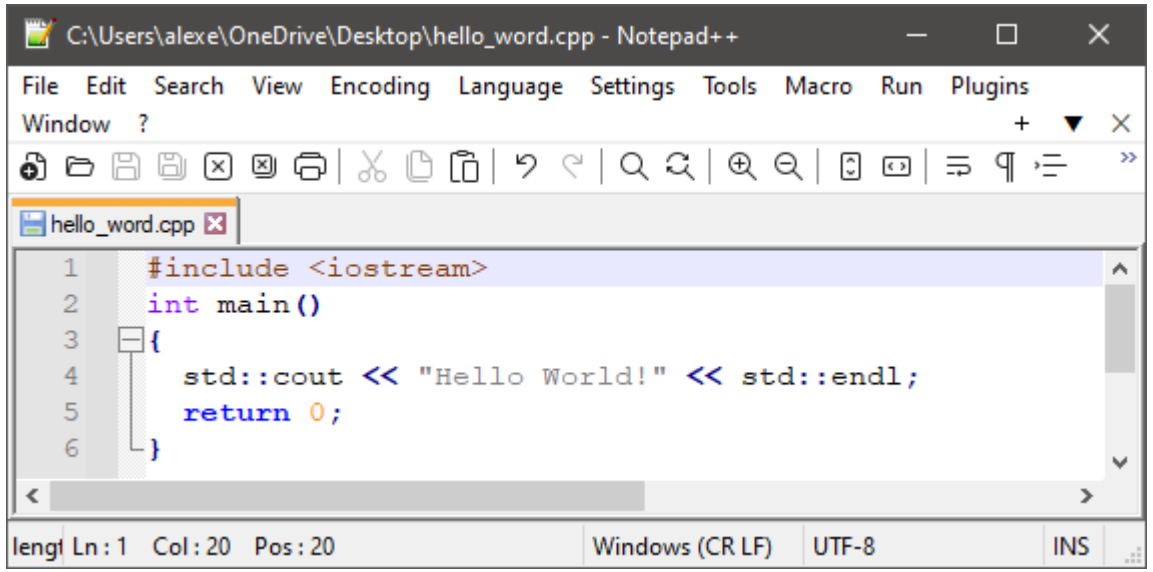
- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

Writing code in a text editor

Compiled language: CPP

Write

Any text editor
(Atom, Notepad++, …)



Compile

Compiler for a specific OS
(the OS terminal)

Compiler	Script		Executable file
> g++	hello_world.cpp	-o	hello_word

Execute

Command line
(the OS terminal)

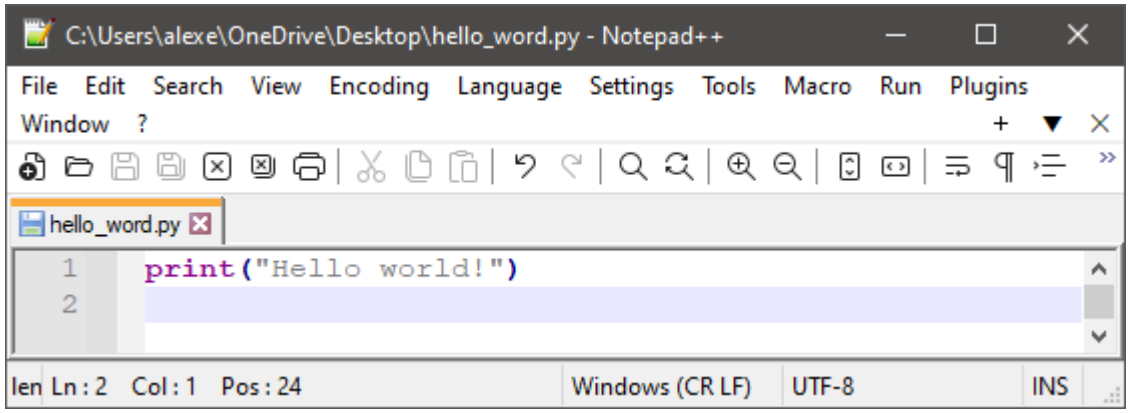
```
> hello_word
Hello World!
```

Writing code in a text editor

Interpreted language: Python

Write

Any text editor
(Atom, Notepad++, …)



Execute

Terminal
(any OS with Python
interpreter installed)

Interpreter Script

```
> python hello_world.py
Hello World!
```

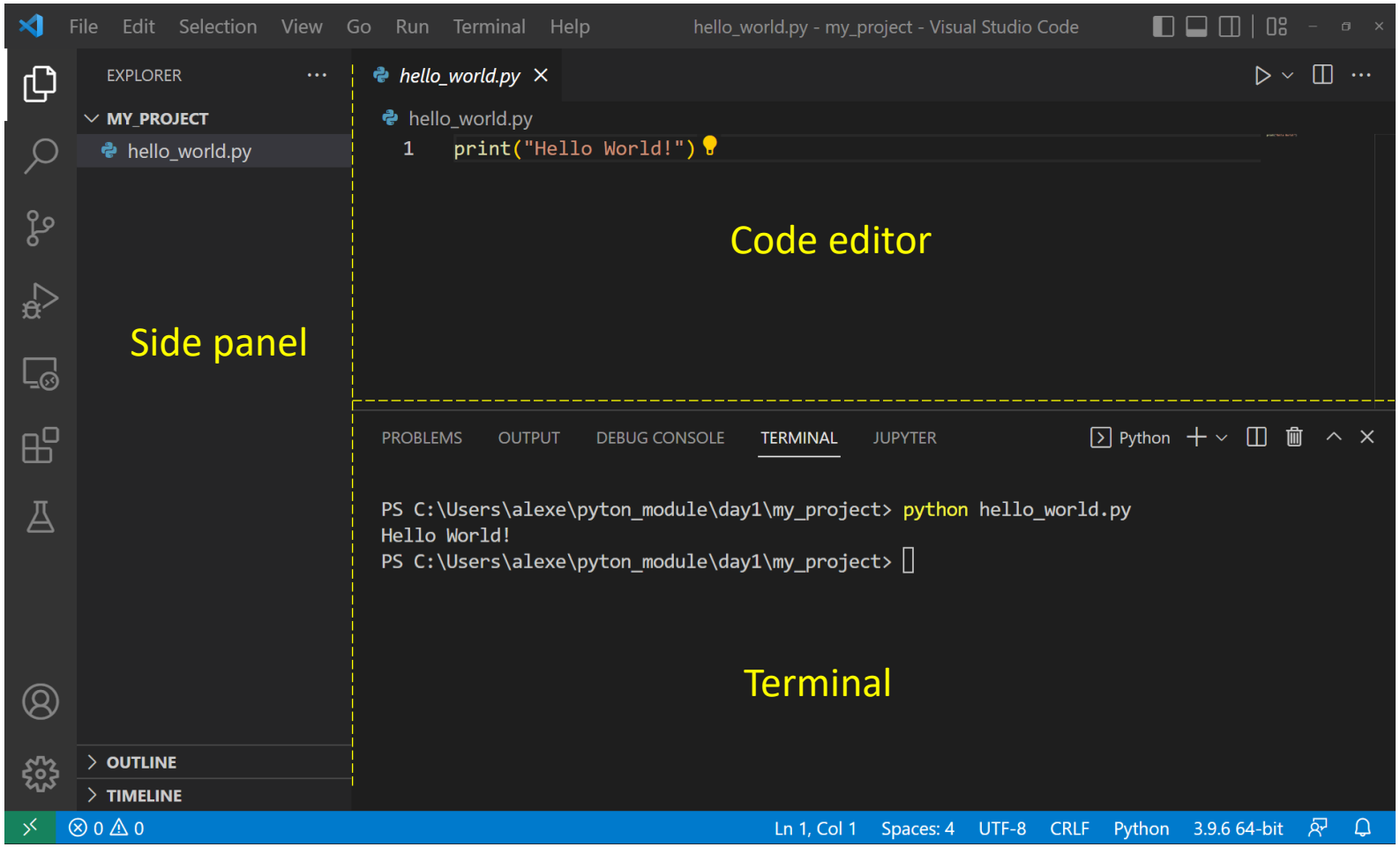
Integrated Development Environment (IDE)

Visual Studio Code, NetBeans (Java), R-Studio (R), PyCharm (Python), IntelliJ …

Menu

Taskbar

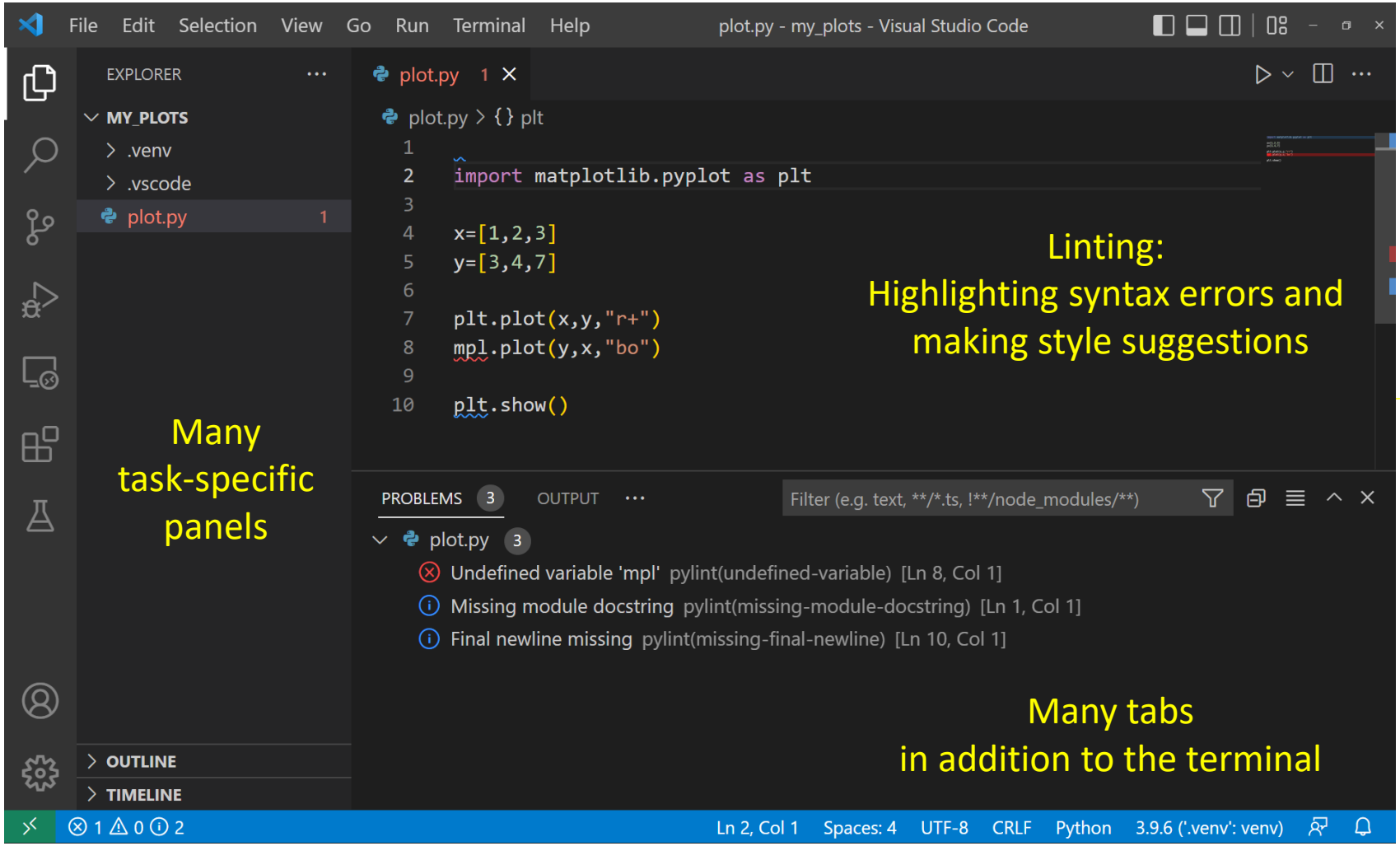
- File explorer
- Search
- Git integration
- Debugging
- Remote access
- Extensions
- Unit tests
- …



Status bar

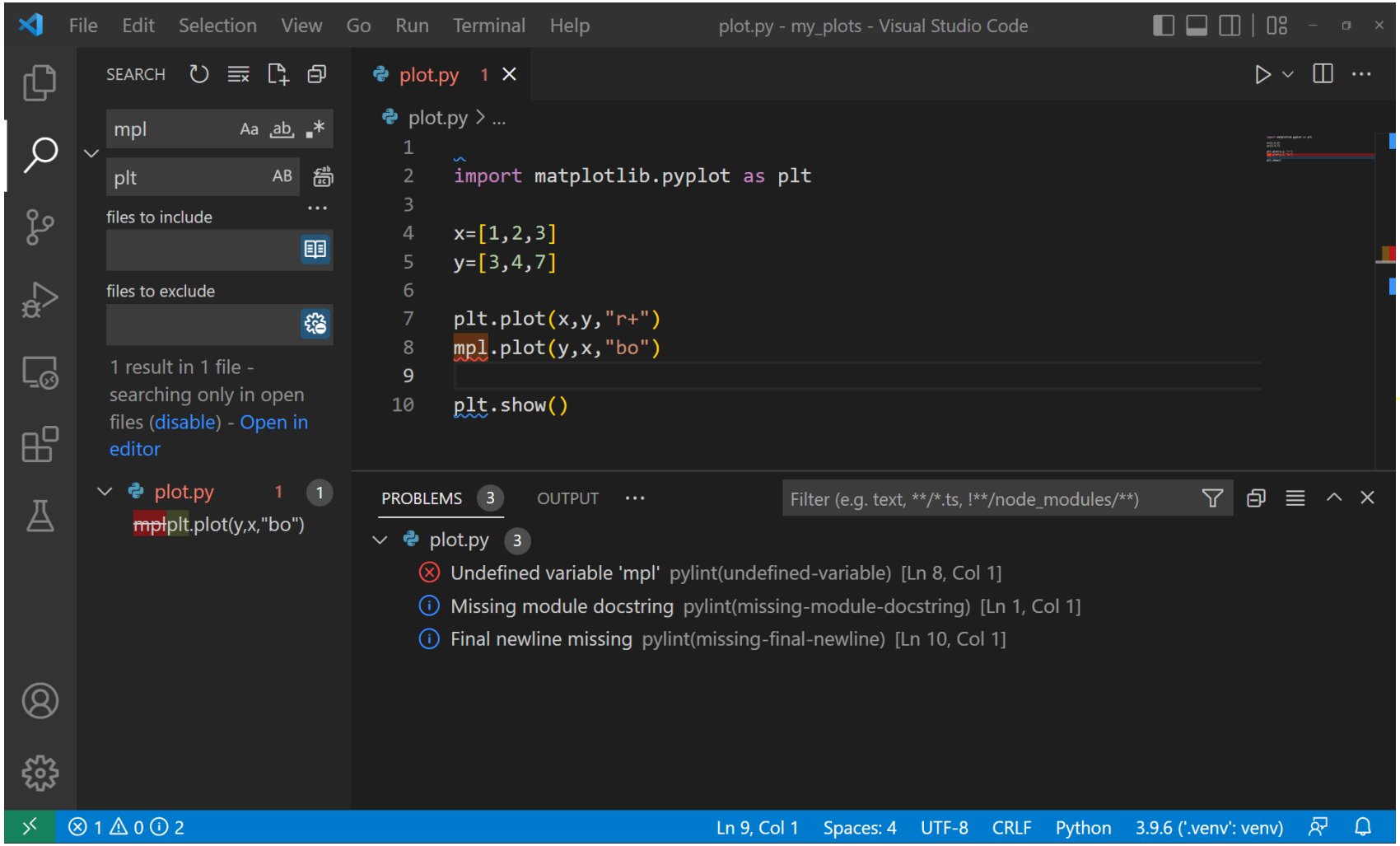
VS Code: File explorer & Syntax highlighting

Files & Folders



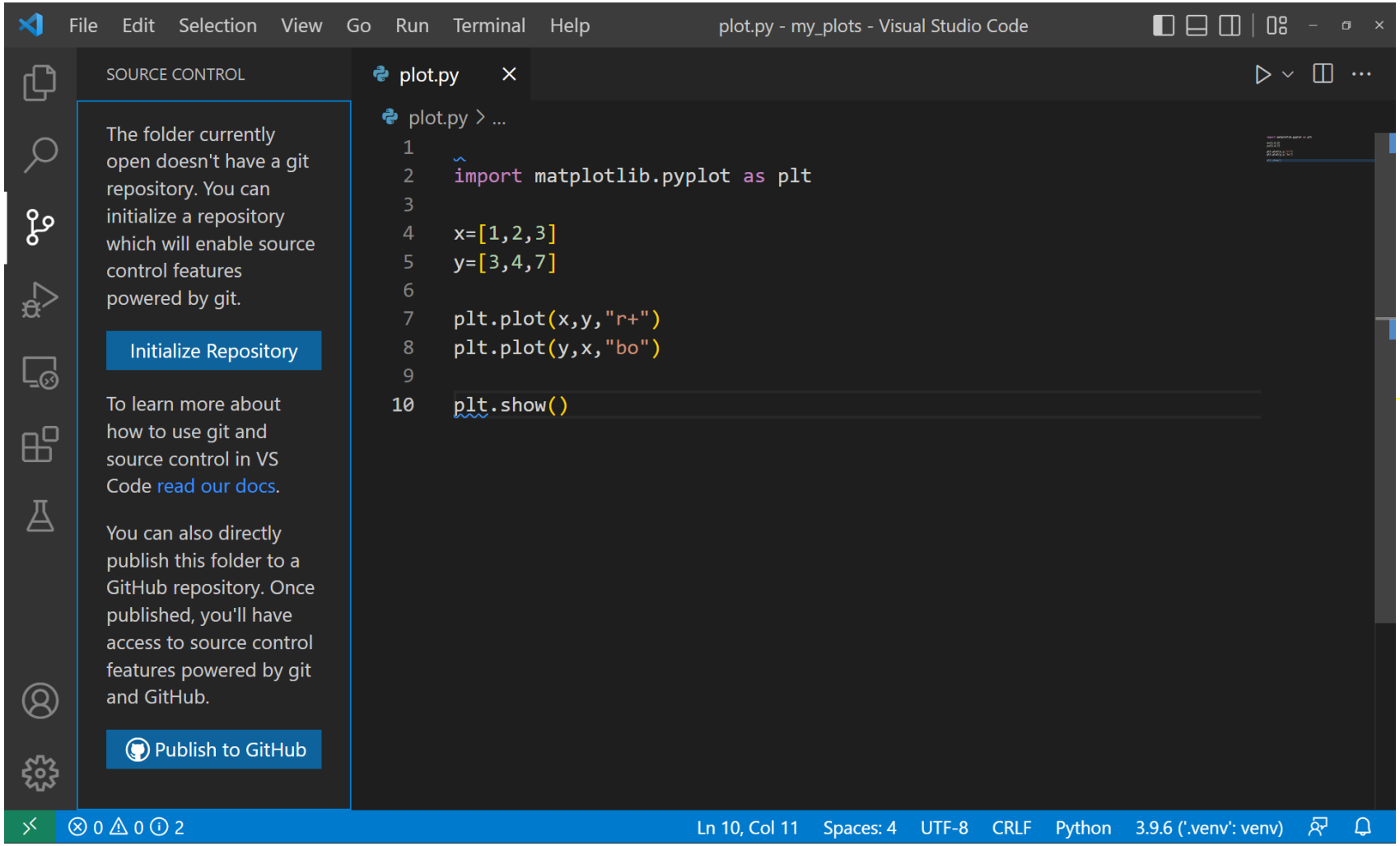
VS Code: Search & Replacement

Search



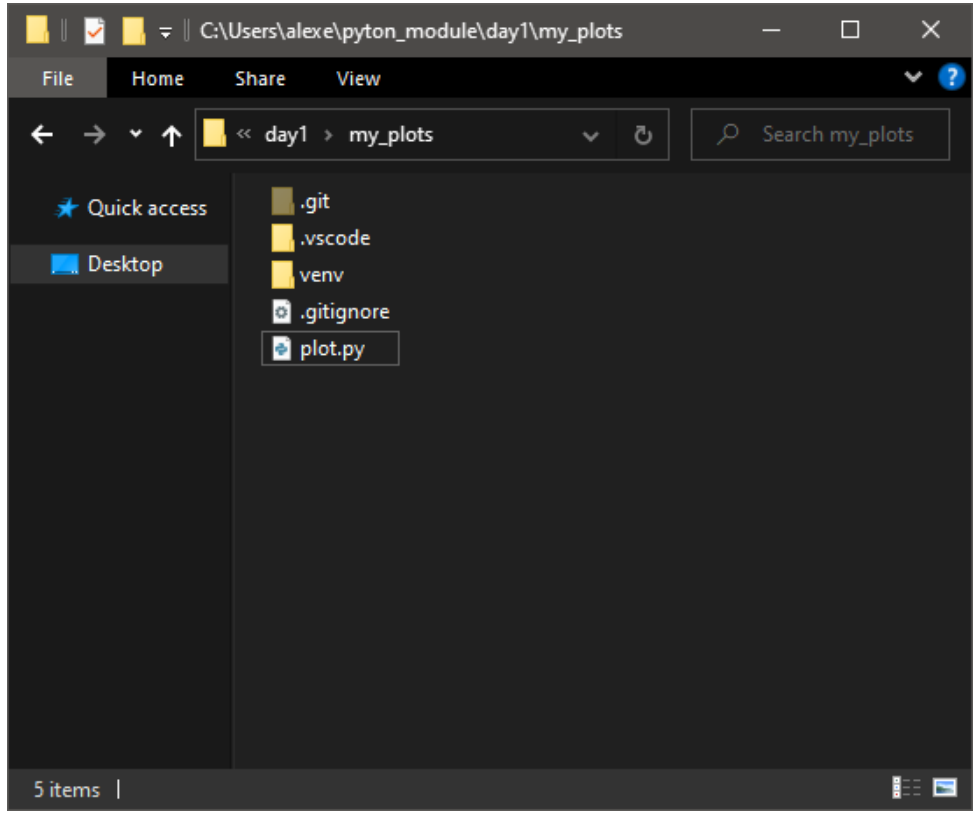
VS Code: Git integration

Git



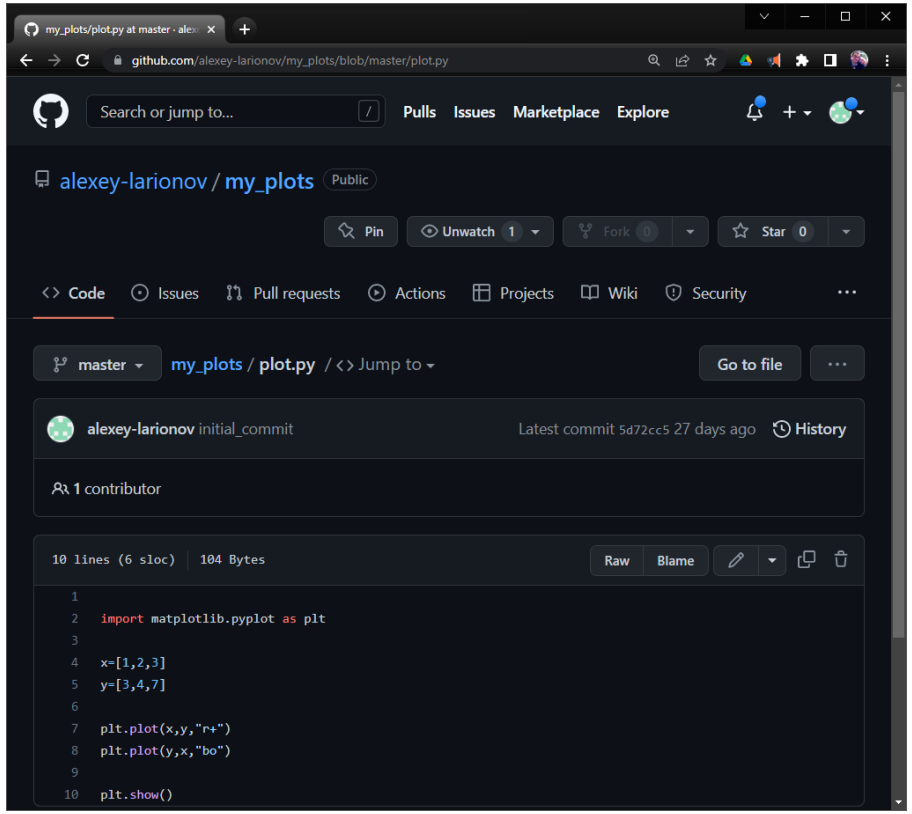
Git: backup, version control, collaboration …

Git is a command-line tool that helps to back-up your files (and much more …)



Local repository

.git



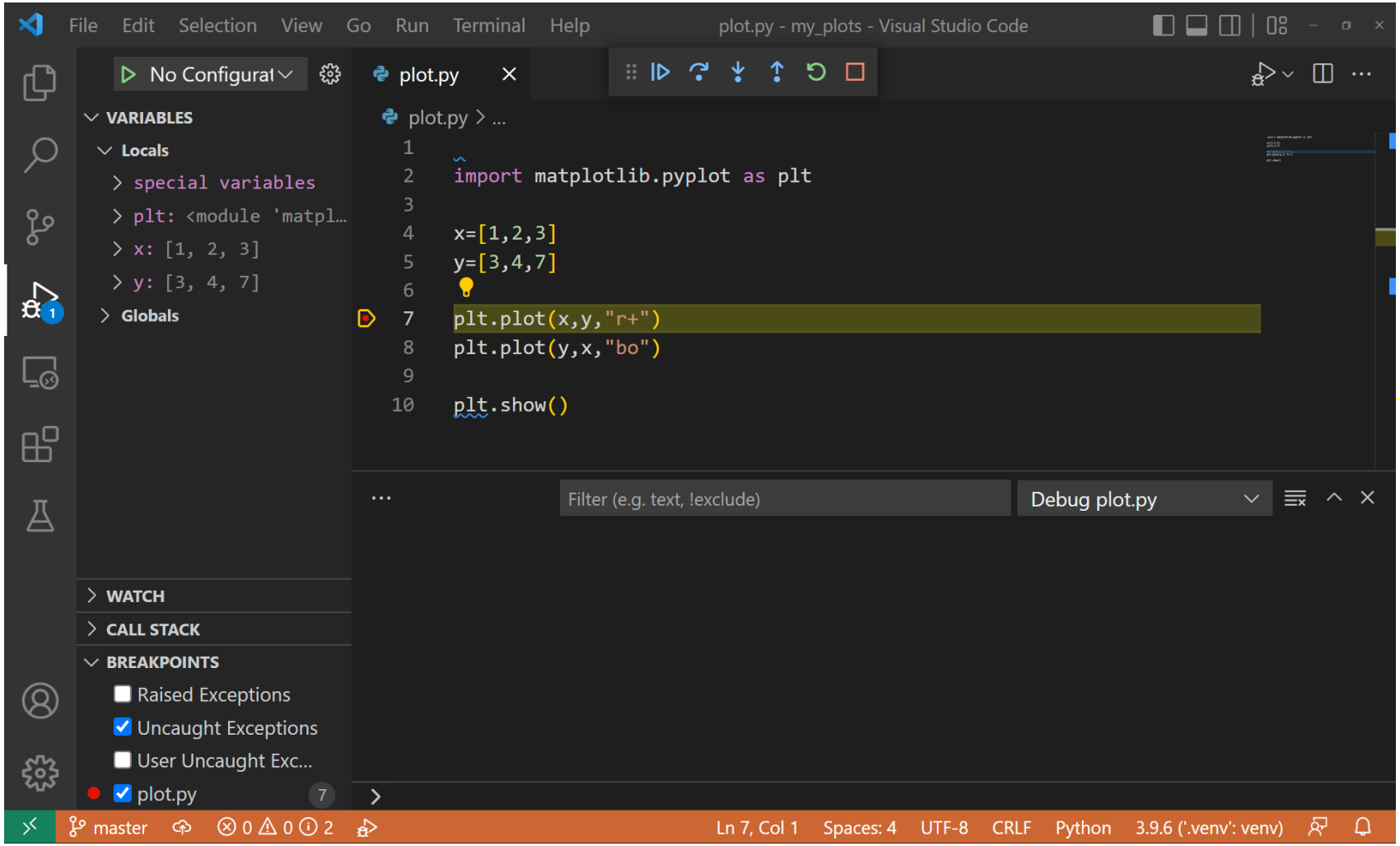
Remote repository

on github.com

Debugging (Runtime and Logical Errors)

Step-by-step execution + Variables explorer

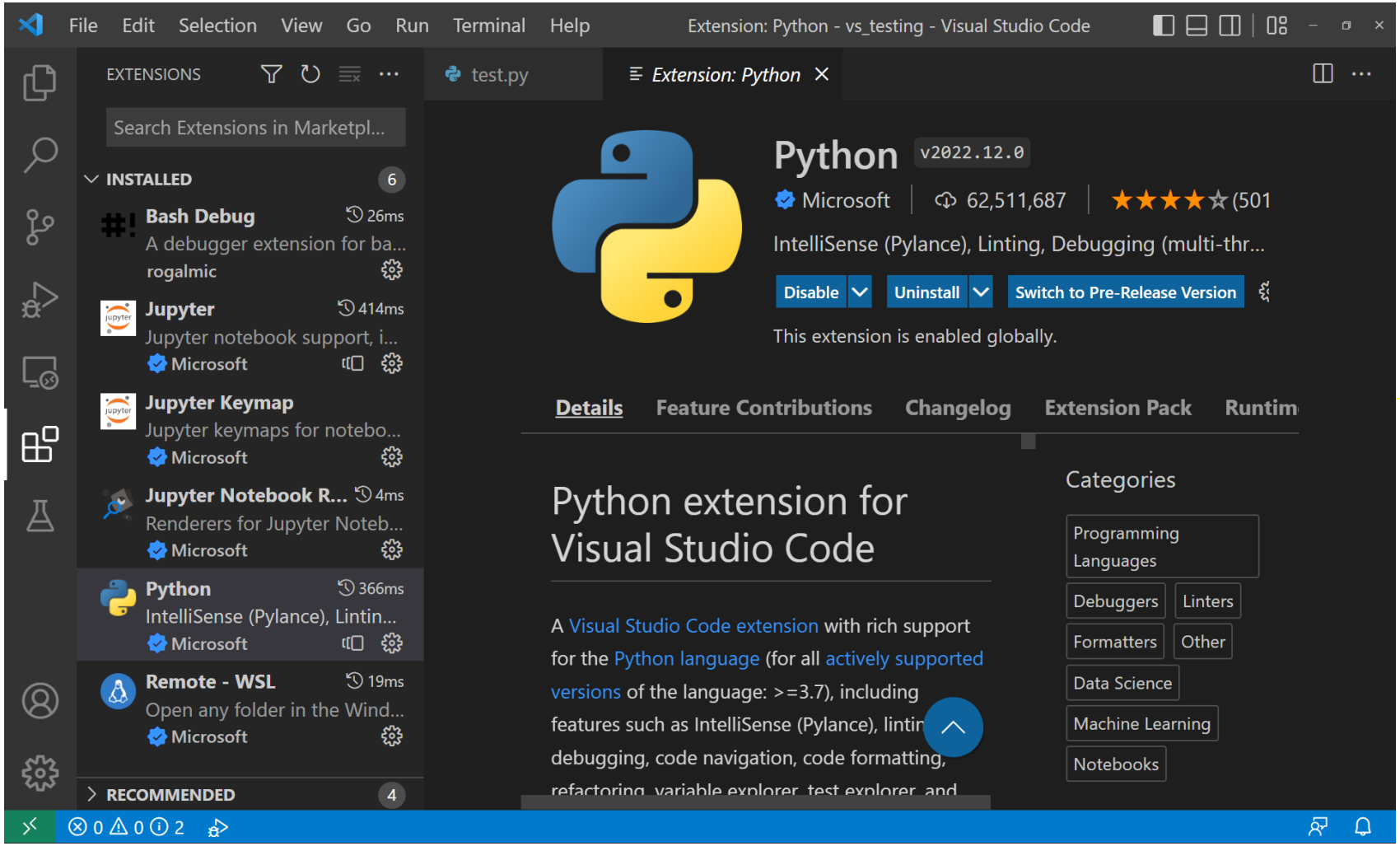
Debugging



VS Code : Extensions

VS Code does not understand Python until you install the Python Extension !

Extensions



Many IDE-s for many languages and tasks

There is much more in programming than coding …

Highlighting syntax

Version control

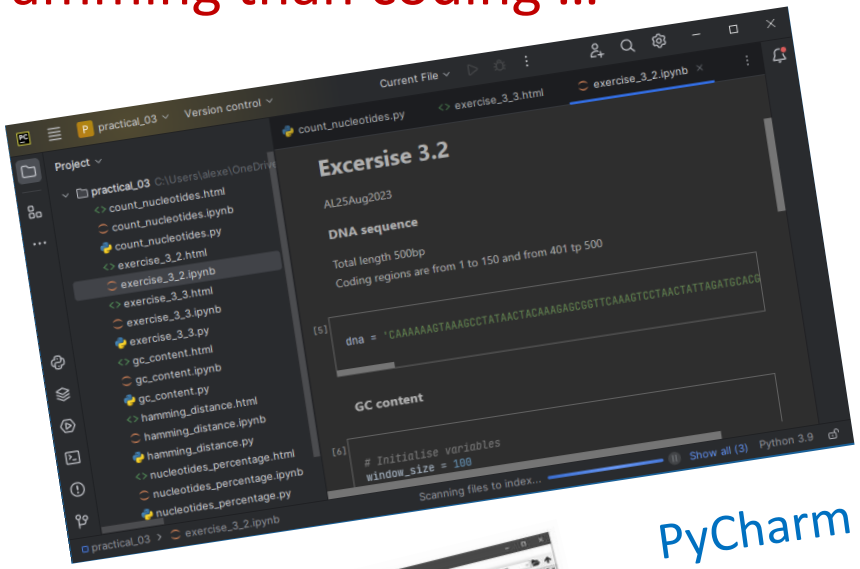
Collaborative development

Debugging

AI-assistants

Unit tests

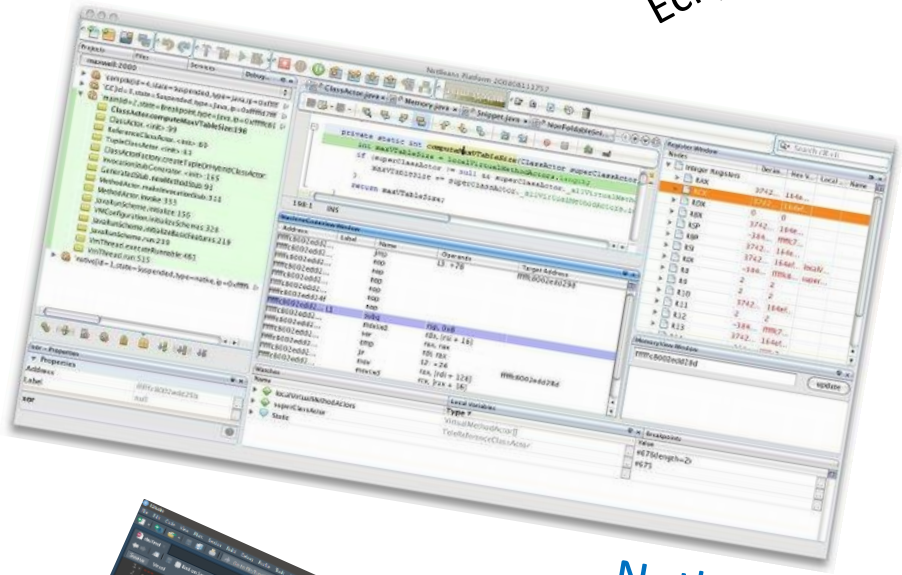
…



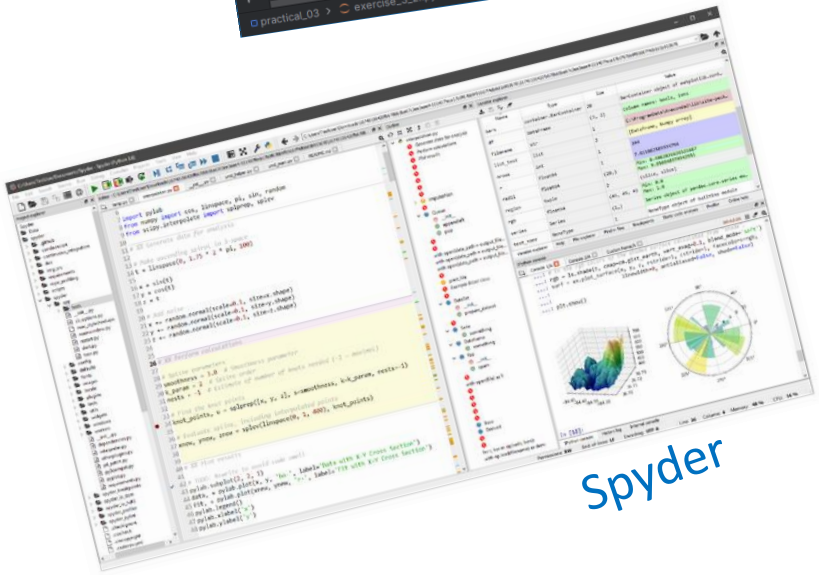
PyCharm

Komodo

Eclipse



Netbeans



Spyder

R Studio

Intelli-J IDEA

AI Assistants

OpenAI

ChatGPT

https://chatgpt.com

Google

Gemini

https://gemini.google.com

Microsoft

COPILOT

https://copilot.microsoft.com

Extension in VS-Code

Let's try these 3 prompts:

- Python: read CSV file …
- Calculate GC content from FASTA file in Python
- Calculate mean enrichment efficiency from Picard HSMetrics outputs

Use of AI by university students outside of coding:

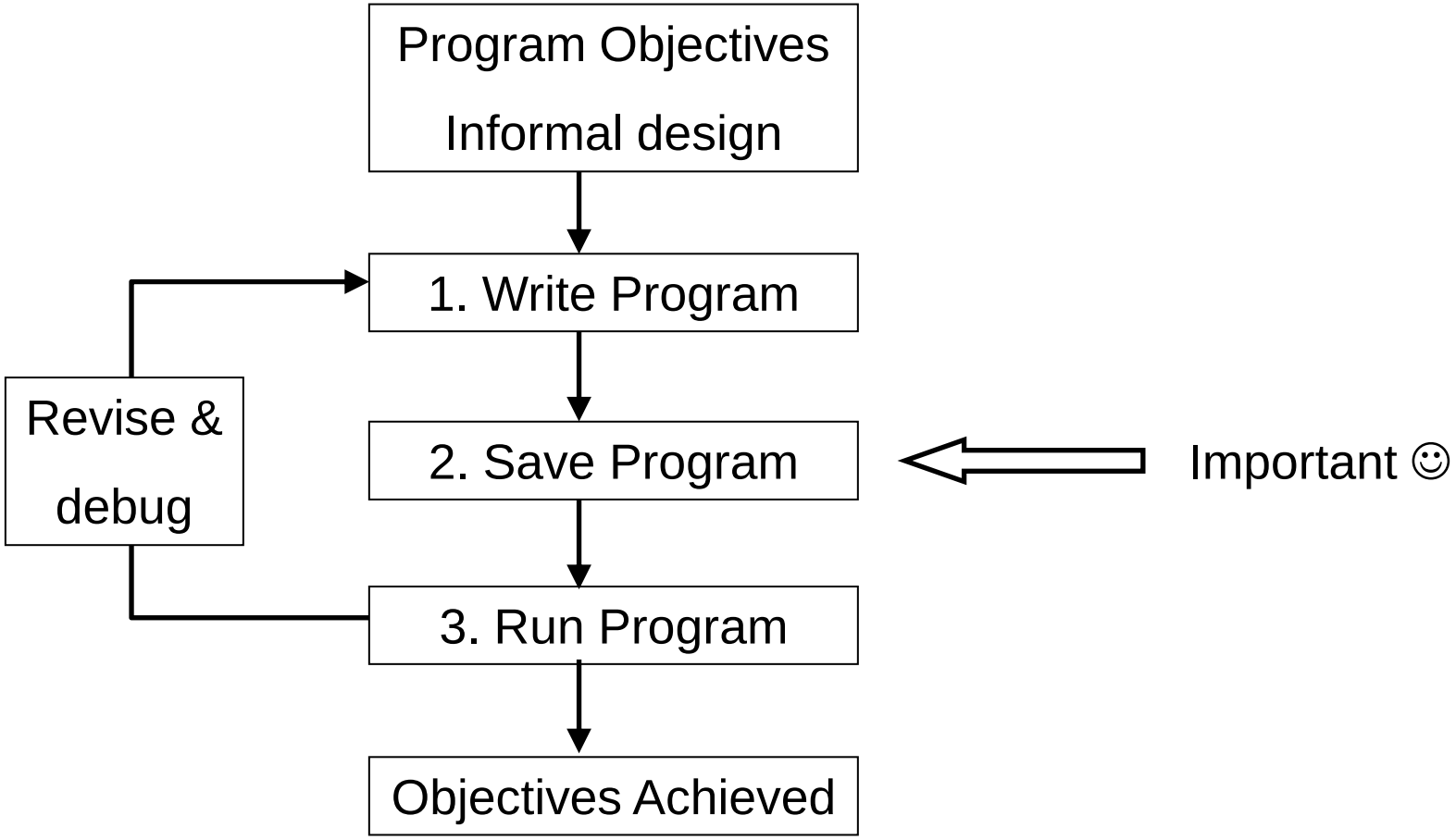
https://intranet.cranfield.ac.uk/Pages/Generative-AI.aspx

https://www.cell.com/patterns/fulltext/S2666-3899(24)00208-3

Lecture plan (Learning outcomes)

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

Programming Process (simplified ☺)



Program Objectives & Informal Design

Program Objectives

A. What the program needs to do ?

B. Identify the required inputs

C. Identify the required outputs

...

Informal design

Sketch / Flowchart / Spider Diagram

Pseudo Code

Calculating mean: Objectives

Program Objectives

A. What the program needs to do ?

Calculate mean of numbers

B. Identify the required input(s)

Two or more numbers

C. Identify the required output(s)

Mean of those numbers

...

Informal design

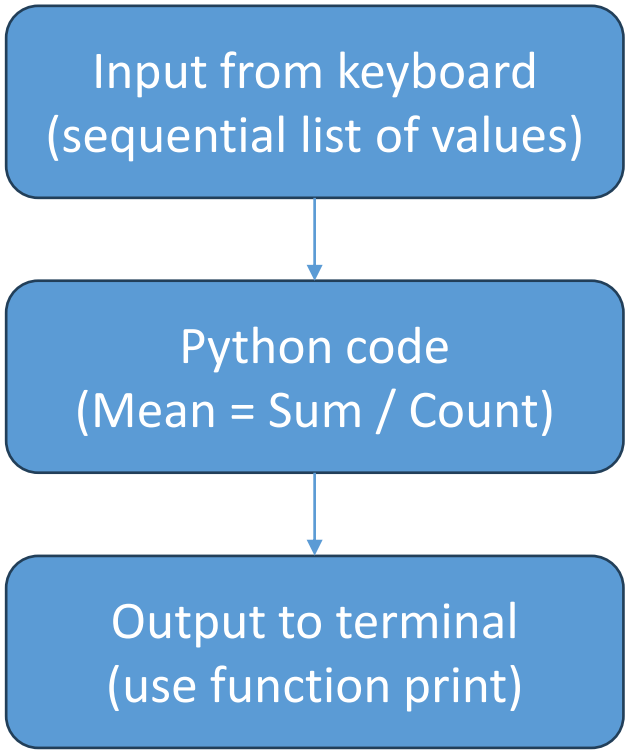
Sketch / Flowchart / Spider Diagram

Pseudo Code

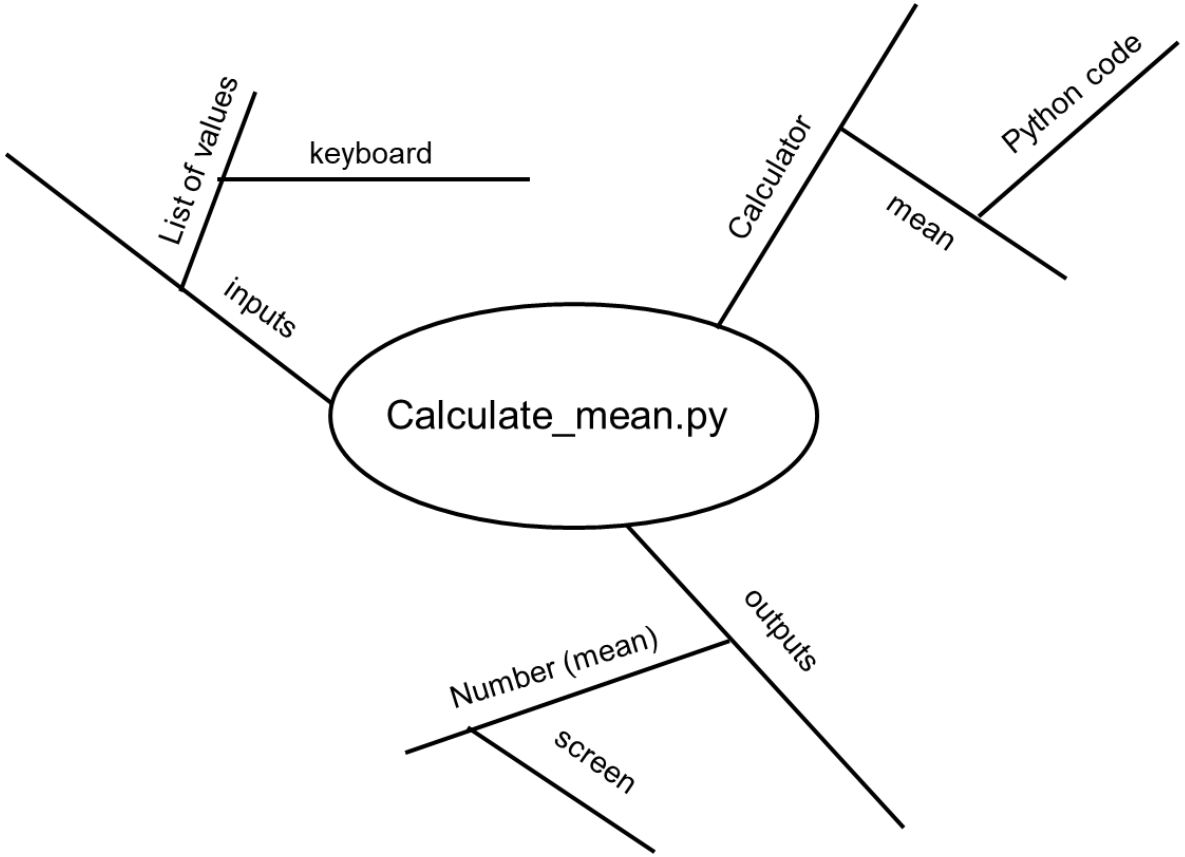
Calculating mean: design sketches

a picture says thousand words…

Flowchart



Spider diagram



Informal design: Pseudo Code

What is it?

- Sort of Code
- Sort of English

Bridge between

- Written objectives
- Written Code

Especially useful for complex tasks

- Multi-step algorithms
- Complicated pattern matches

Calculating mean: Pseudo Code

```
Title: Pseudocode for calculating mean

Get numbers into numbers_list

Required initialisations

For each number in the numbers_list
        add the number to total
        add one to count

Divide total by count

Print mean
```

total and ***count*** to start at zero

Actual Code →

Calculating mean: Code

Title: not executed

Get numbers into numbers_list

Required intialisations

For each number in numbers_list

add number to total

add one to count

Divide total by count

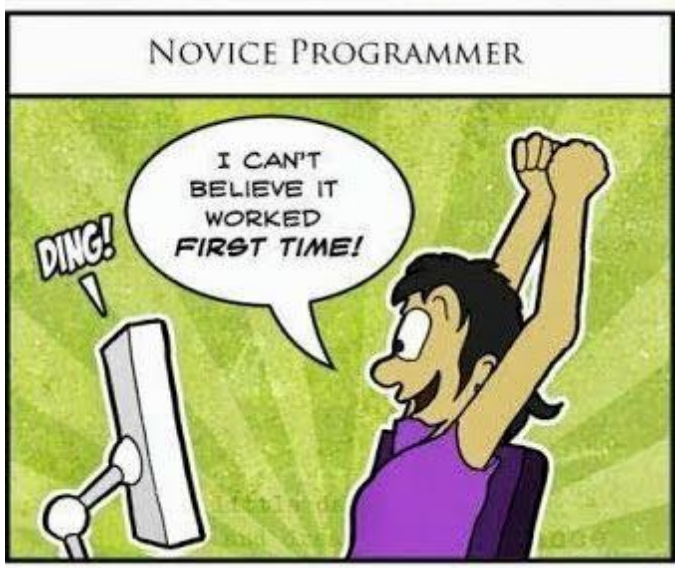
Print mean

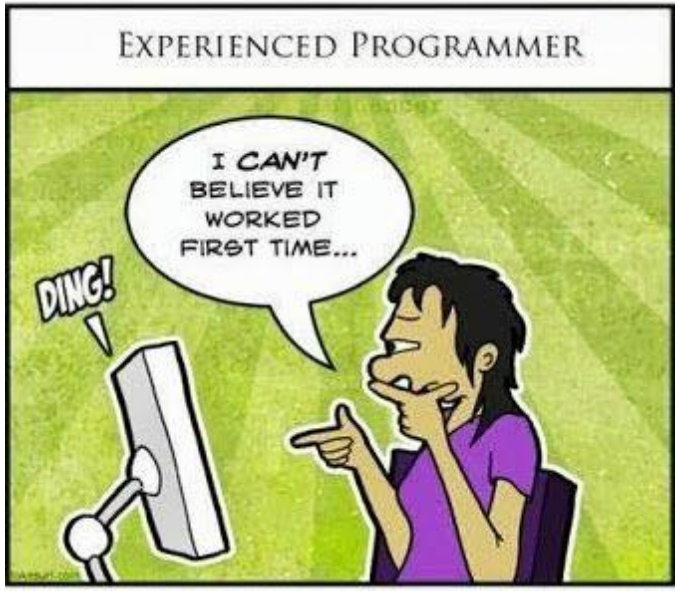
```
# This program calculates mean of numbers in a list
numbers_list = [1, 2, 3, 4, 5]
total = 0
count = 0
for number in numbers_list :
      total = total + number
      count = count + 1

mean = total / count
print(mean)
```

Lecture plan (Learning outcomes)

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

Debugging means removing Errors





Syntax errors

- Miss-spelling names of keywords
- Forgetting to match closing quotes or brackets
- Using variables before they've been named

```
Print("Hello Word)
```

Runtime errors

- Occurs during program is executing
- A common example of a runtime error is a division by zero error

```
x = 0
y = 5
z = y / x
```

Logic errors

- Program does what you asked… but not what you wanted !

```
x = 5
y = 4
area = 5 + 4
```

Lecture plan (Learning outcomes)

- Programming languages: Python, Java, R, C++ …
- Program execution: Compiling *vs* Interpreting, role of Operating system & Dependencies …
- Main programming paradigms: Procedural, Functional, Object-oriented …
- Programming tools: IDEs, Versioning & Collaboration (Git-Hub), AI-assistants …
- Writing a program: Concept, Pseudocode, Code, Debugging …
- Main types of programming errors: Syntax errors, Run-time errors, Logic errors …

For whom was this lecture ?

For Biologists !

Python module KSBs for apprenticeships

Knowledge

- Common bioinformatics programming languages; algorithm design, analysis and testing
- Retrieval and manipulation of biological data, including data mining, from public repositories

Skills

- Make use of suitable programming languages and/or workflow tools to automate data handling and curation tasks
- Apply a range of current techniques, skills and tools (including programming languages) necessary for computational biology practice
- Recognise and critically review the format, scope and limitations of different biological data
- Maintain a working knowledge of a range of public data repositories for biological data
- Carry out the analysis of biological data using appropriate programmatic methods, statistical and other quantitative and data integration approaches – and visualise results

Questions