1. Introduction

Python Programming / Operating Systems

Team

Prof. Federico March

Computer Science

Prof. Marco Cascio

Computer Science

Dr. Flavio Giorgi

Computer Science

Dr. Leonardo Picchiami

Computer Science

Scope and Goal

- The basic concepts of programming
- Programming in Python
- The tools you need to program (as far as necessary)
- Basics of computer networks
- Operating systems and shell programming
- Gaining practical experience with all of the above
- Disclaimer: the actual work starts after this course!

Organization

- We provide you with:
  - Basics of computer networks and Operating systems (the theory)
  - Concepts and tools for Python program development (the theory)
  - Exercises and programming tasks (the practice)

- You need to:
  - Ask questions (at any time) and engage with your classmates! (we are a small group)
  - Work through the tasks (on your device)

- Course Material:
  - Moodle
  - Slides
  - Lecture Notes

Content

Python Programming

- Foundation of Programming
- Primitives (Data Types)
- Operations
- Collections
- Control Statements
- Functions
- Classes
- Threads
- Docker

Basics of Computer Networks

- Network Protocols
- ISO/OSI Model
- TCP/IP Model

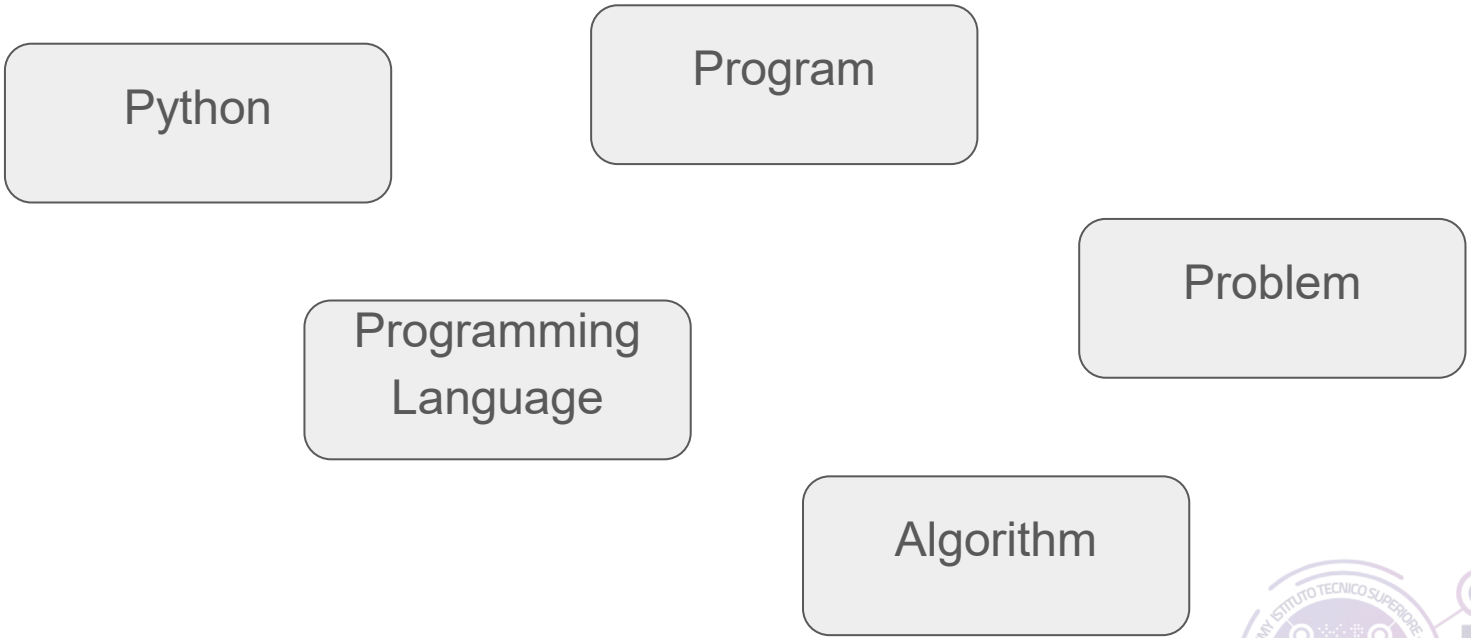
Operating Systems (OS)

- Binary Code
- Computer Architecture
- Fundamentals of OS
- OS services and functions
- Linux
  - Shell Programming

1.1 Getting Started

- Python Program
- Environment

Python Program



Problem 1/2

Description:
A **task** or **question** that requires a solution or answer is a **type of problem** that challenges you to think critically and apply your knowledge to solve it. This can range from a simple question to more complex problems that require **multiple steps** to find a solution.

Problem 2/2

Example:
Calculate the mean of some given numbers. For instance, if you are given the numbers 4, 8, 15, 16, 23, and 42, you would:

- Add them together: 4+8+15+16+23+42=108.
- Count the total numbers: 6.
- Divide 108 by 6: 108/6=18.

Algorithm 1/3

Description:
An **abstract step-by-step procedure** to solve a problem is often called an **algorithm**. Formally, it is a defined **sequence of instructions** or a **systematic approach** used to achieve a particular goal or solve a specific problem.
An algorithm is fundamental in both computer science and everyday problem-solving, as it helps **break down complex challenges into manageable steps** that can be followed to reach a solution.

Algorithm 2/3

A good algorithm is:

- **Clear:** each step is unambiguous, and there is no confusion about how to execute it.
- **Efficient:** it solves the problem in a reasonable amount of time without unnecessary steps.
- **Generalizable:** it can be applied to different sets of data, not just a specific example.

Algorithm 3/3

Example:
Let's take the example of calculating the mean of a set of numbers, and see how this can be expressed in a step-by-step algorithm:

1. Sum up all numbers.
2. Count all numbers.
3. Divide the sum by the count.

Program

Description:

A **set of instructions** telling your computer how to execute an algorithm.

Example:

```
sum = 0
count = 0

for number in numbers:
    sum = sum + number
    count = count + 1

mean = sum / count
```

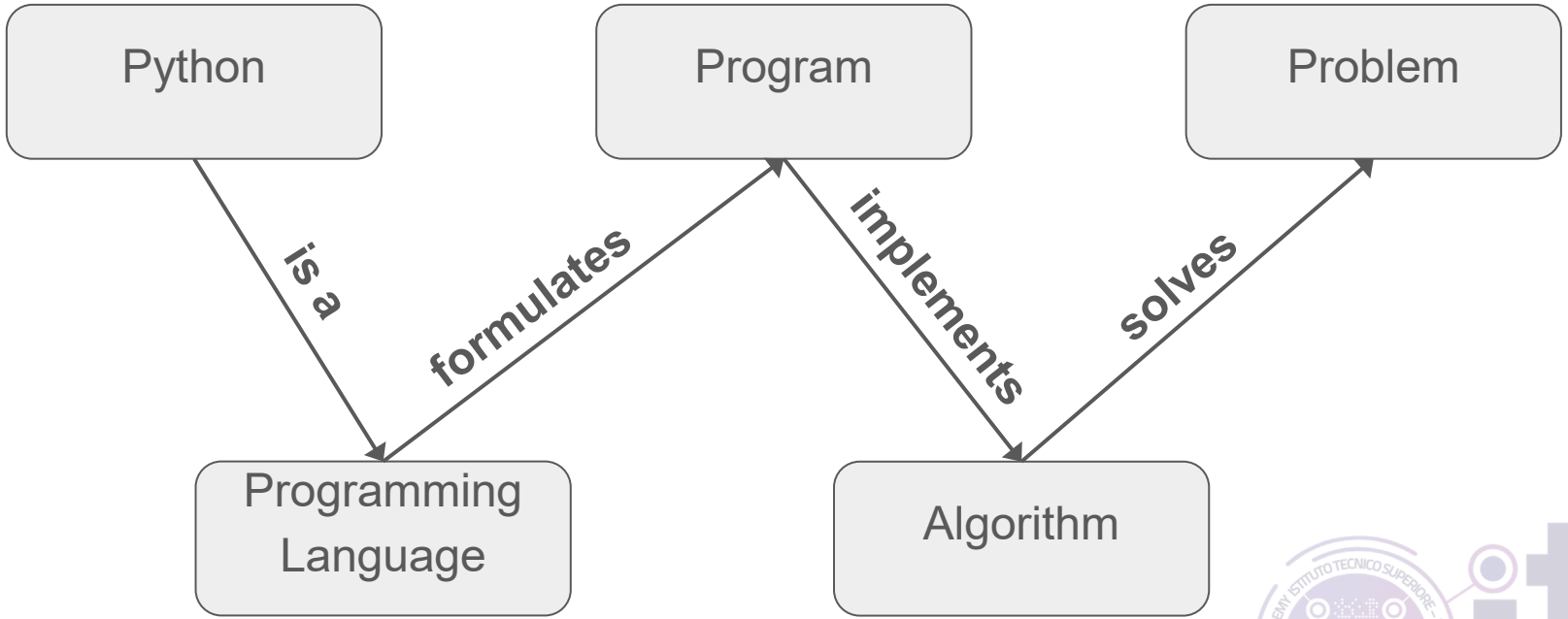
The **programm text** is the **program code**

Programming Language

Description:
A **specific language for formulating programs** is called a **programming language**. Programming languages are used by developers **to communicate instructions to a computer** in a way that it can understand and execute. They are the medium through which we create software, automate tasks, and solve computational problems.

Example:
Python, Java, or C++

Python Program



Environment 1/2

Permanent Storage
- Slow to access
- E.g. HDD, SSD, NAS, cloud storage, etc.

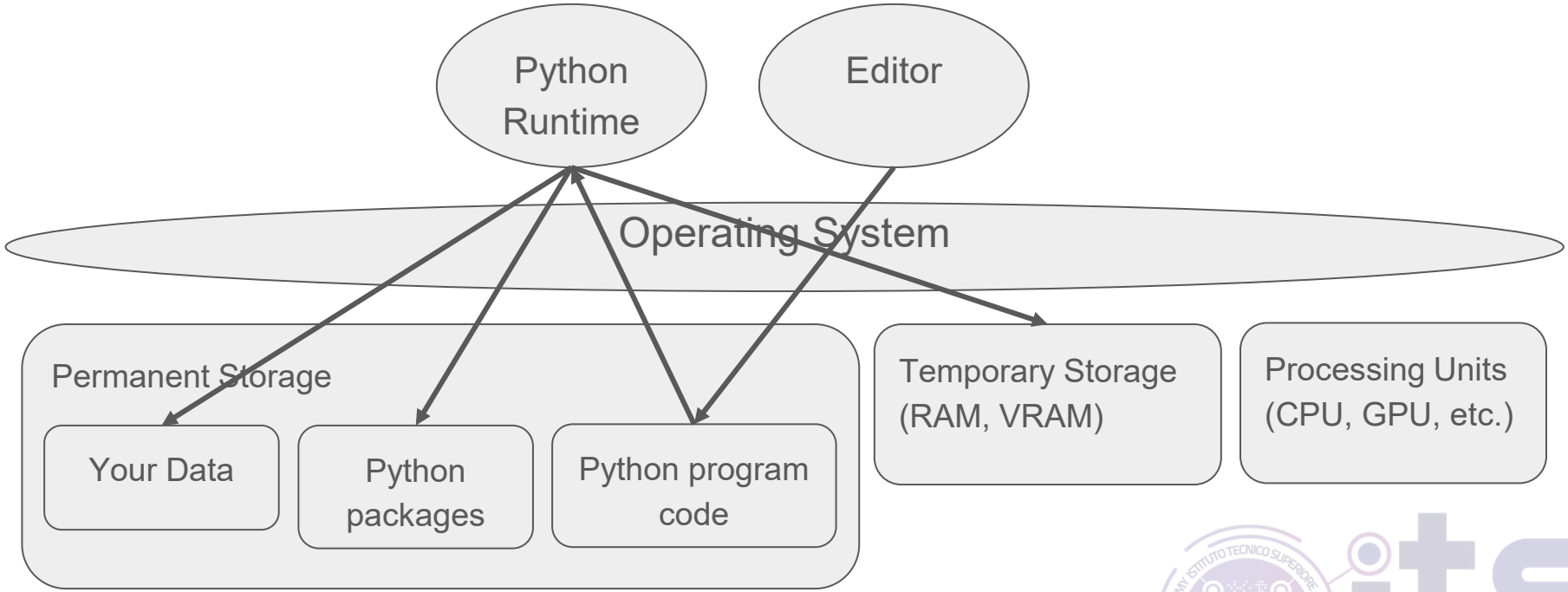
Your Data

Python program code

Processing Units
- Do the actual computation
- Central Processing unit (CPU)
- Graphics Processing Unit (GPU)

Temporary Storage
- Fast to access
- Only temporary
- E.g. RAM, VRAM

Environment 2/2

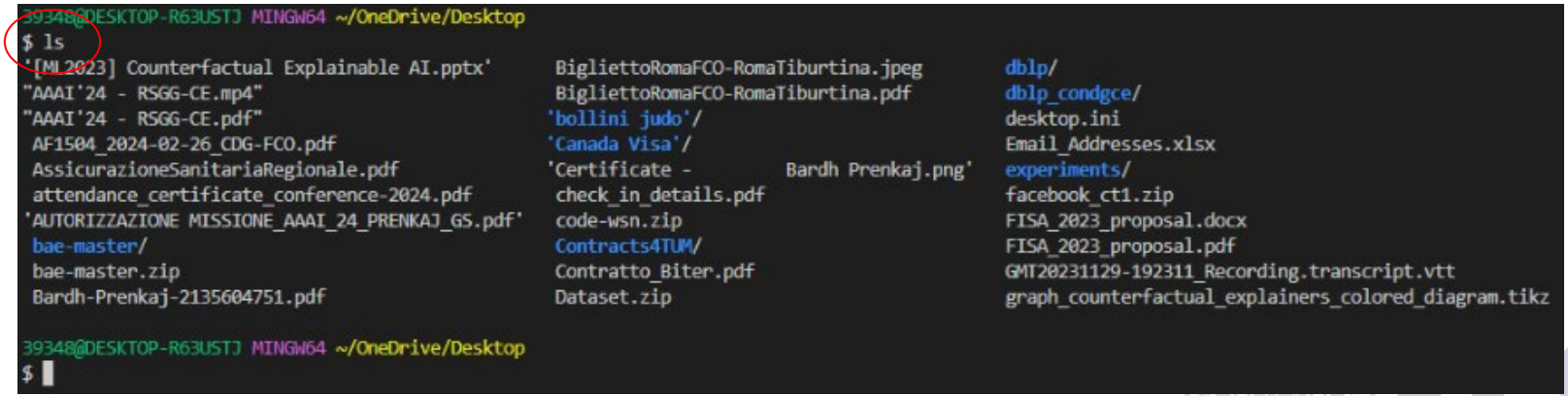


1.2 Setting up our tools: Command-line

- Commands:
  - ls
  - cd
  - mkdir
  - touch
  - rm
  - pwd
  - mv

Command-line

- With the **command-line** (cmd) you can communicate directly with your computer (i.e. operating system)
- Example: For listing the content inside a folder (`ls`)



Command-line - Try it yourself

1. Open command-line: (Use the shortcut **ctrl+alt+t**)
2. Go to your desktop: **`cd`** `Desktop`
3. Print your current working directory: **`pwd`**
4. Create now folder: **`mkdir`** `<name of your folder>`
5. Show your `Desktop` directory content:
    - Linux: `ls`
6. Remove folder: **`rm -r`** `<name of your folder>`

Command-line - Navigating directories

- Change your current directory (**working directory**) with `cd` <target>

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop
$ cd dblp

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$ mkdir cartella-di-prova
```

- For going up the folder structure use “..” as target

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$ cd cartella-di-prova/

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$
```

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ cd ..

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp
$
```

Create your first “empty” Python file

You can do that using the command **`touch`** <*file_name*>

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ touch first_python_program.py

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$ ls
first_python_program.py

39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova
$
```

Command-line - Moving and renaming files

The **mv** command is used to move files or directories to a new location or rename files.

- **mv** `file.txt /path/to/destination/`

  This moves file.txt into the specified directory (/path/to/destination)

- **mv** `old_name.txt new_name.txt`

  This renames old_name.txt to new_name.txt

- **mv** `file.txt /new/path/renamed_file.txt`

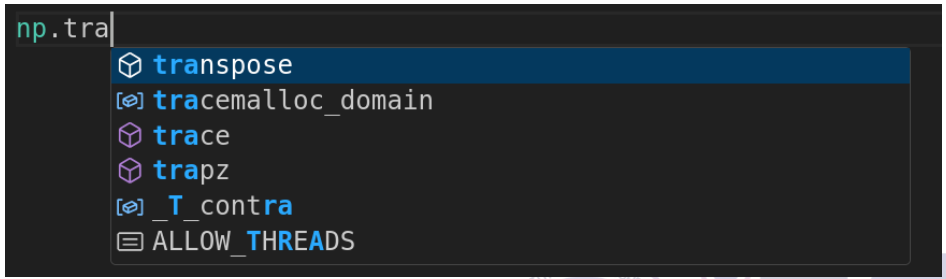
  This moves file.txt to a new directory and renames it

1.3 Setting up our tools: Integrated Development Environment (IDE)

- IDE
- VSCode

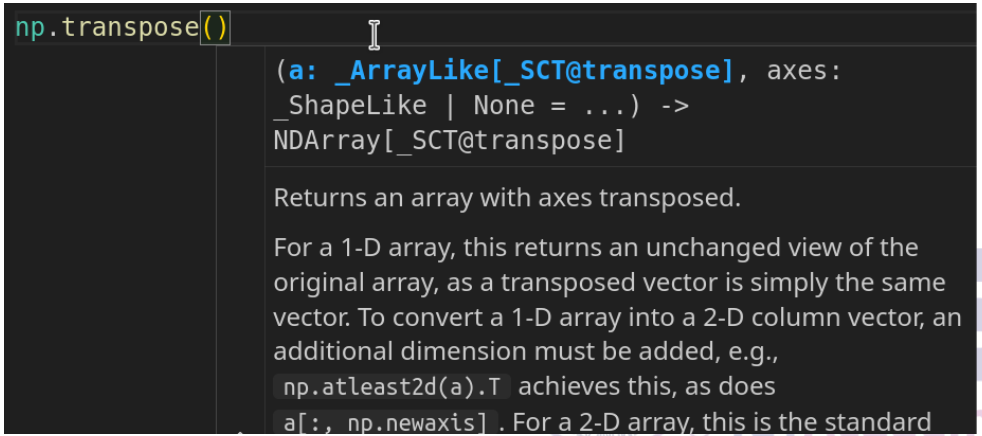
Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations



Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations
- **Documentation** and references

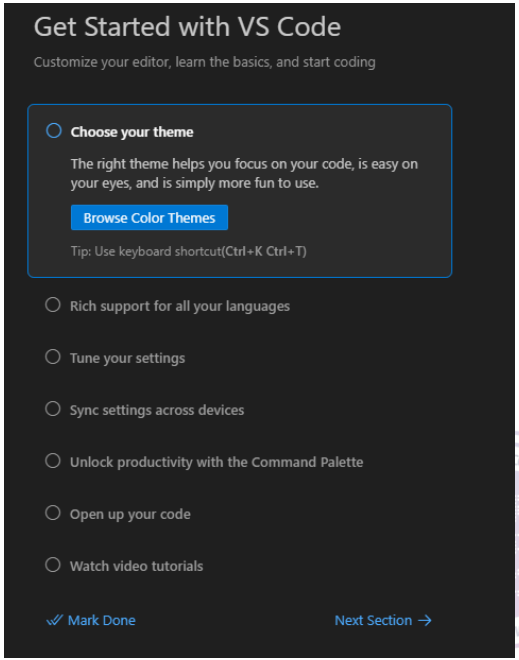


Visual Studio Code - Programming more comfortably!

- **Code editors** are great for developing programs!
  - Visual Studio code is actually more like a *Integrated Development Environment* (IDE)
- Write code and **save** it as a file
- **Highlight errors** in code
- **Autocomplete** and recommendations
- **Documentation** and references
- Code **versioning** control with Git → more on Git later!

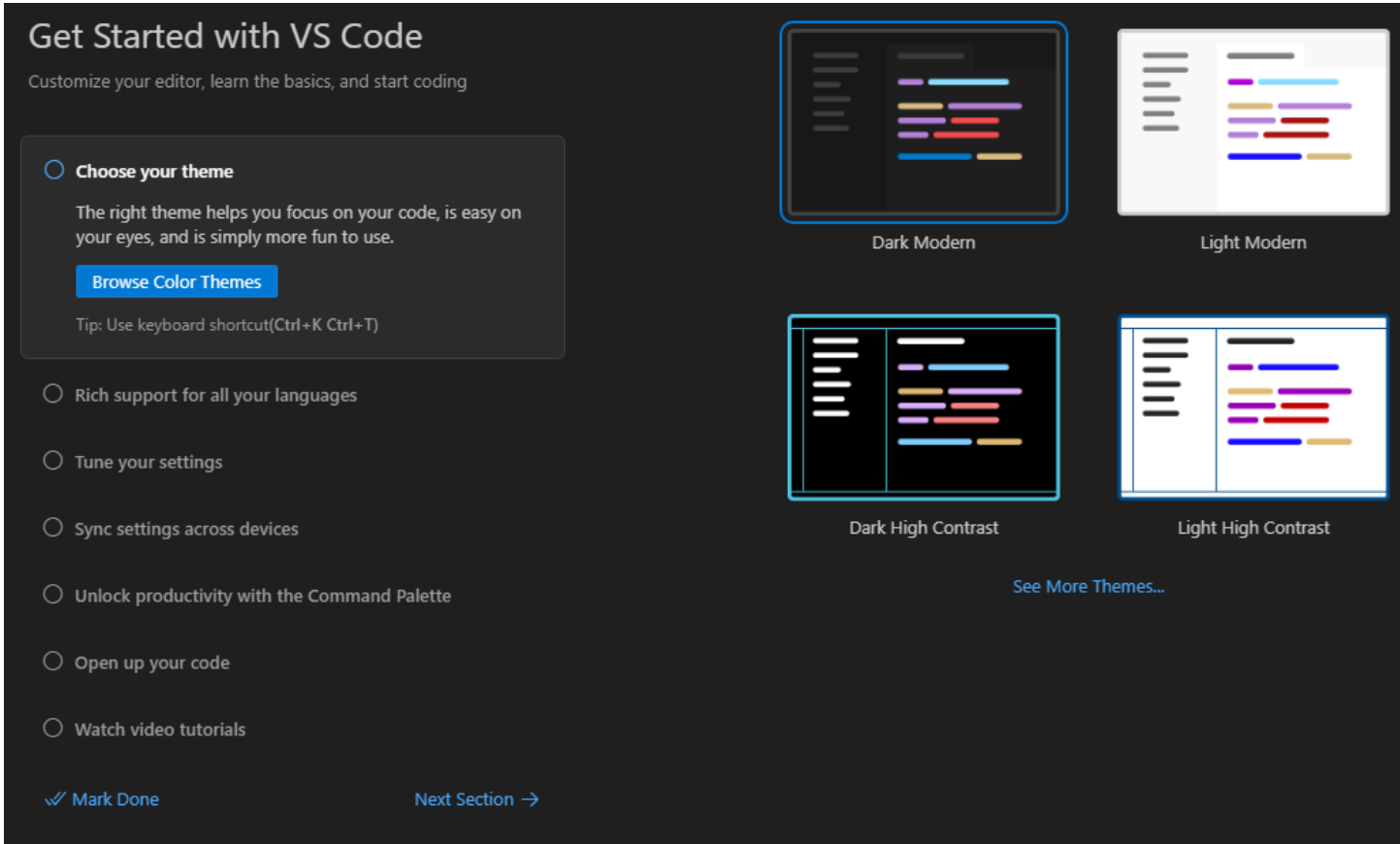
Visual Studio Code - Installation

- Go to code.visualstudio.com
- **Download** installer & **Install** (should be similar for Windows & Mac)
  - Select **all** additional tasks
- **Wait** for the others if you are at "Get Started with VS Code" (We will do the setup together)



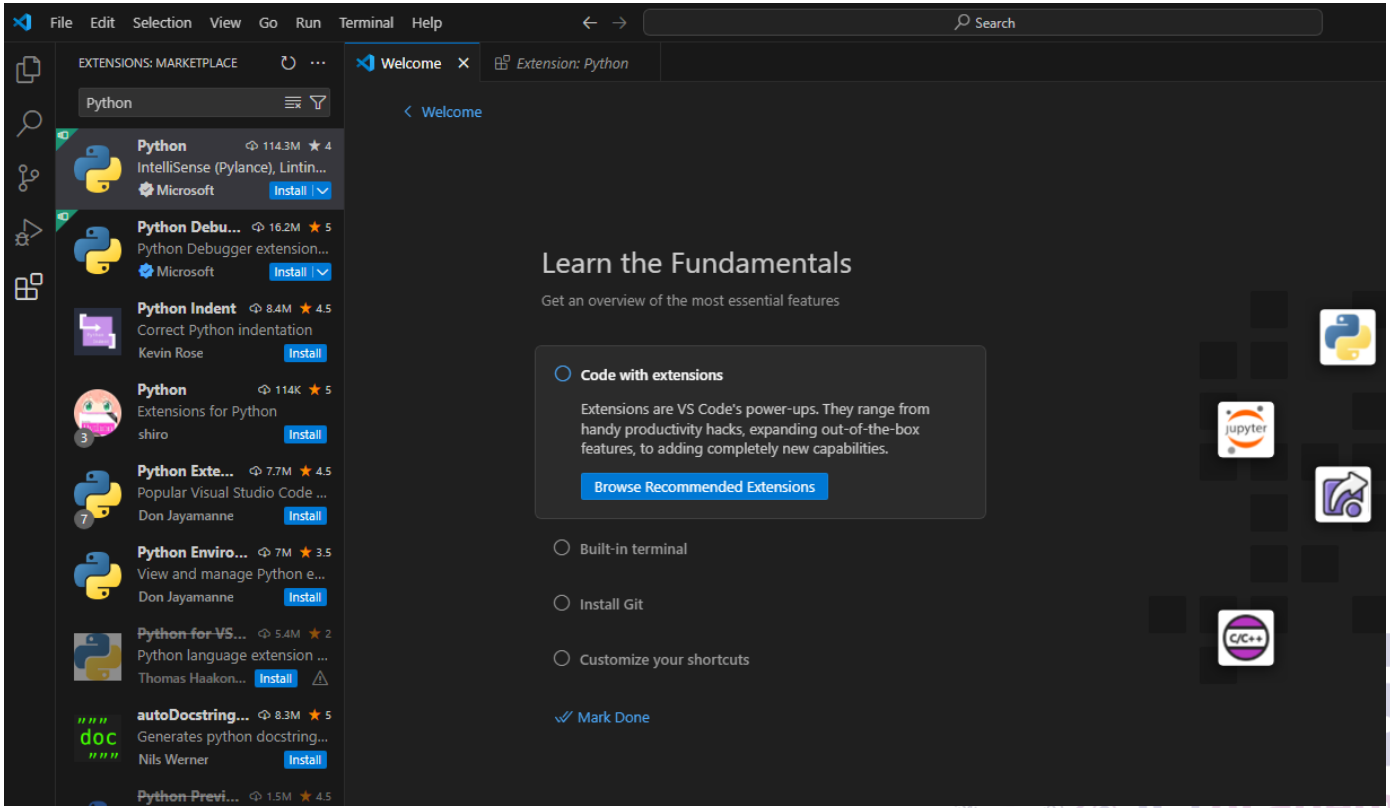
Visual Studio Code - Setup - Theme

1. **Choose** Theme
2. Click “**Next Section**”

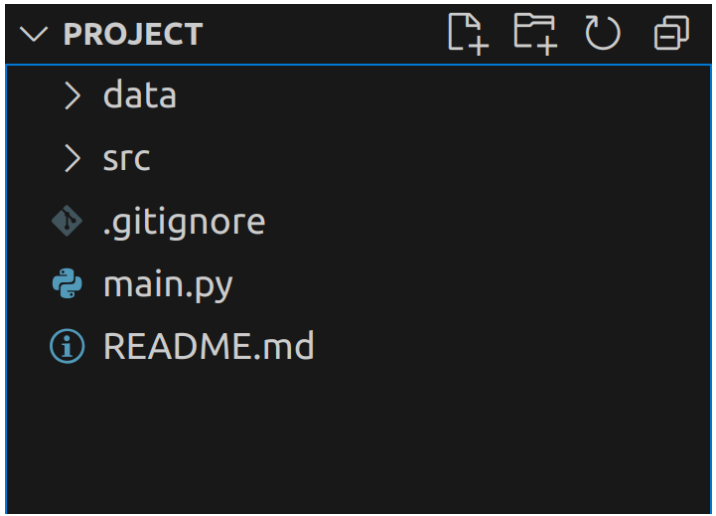


Visual Studio Code - Setup - Python Extension

1. Click “**Browse Recommended Extensions**”
2. Type “**Python**” in the extension **search** bar
3. **Install** the “**Python**” extension
4. If you get the “Get Started with Python Development” Tab opens; **wait** for the others

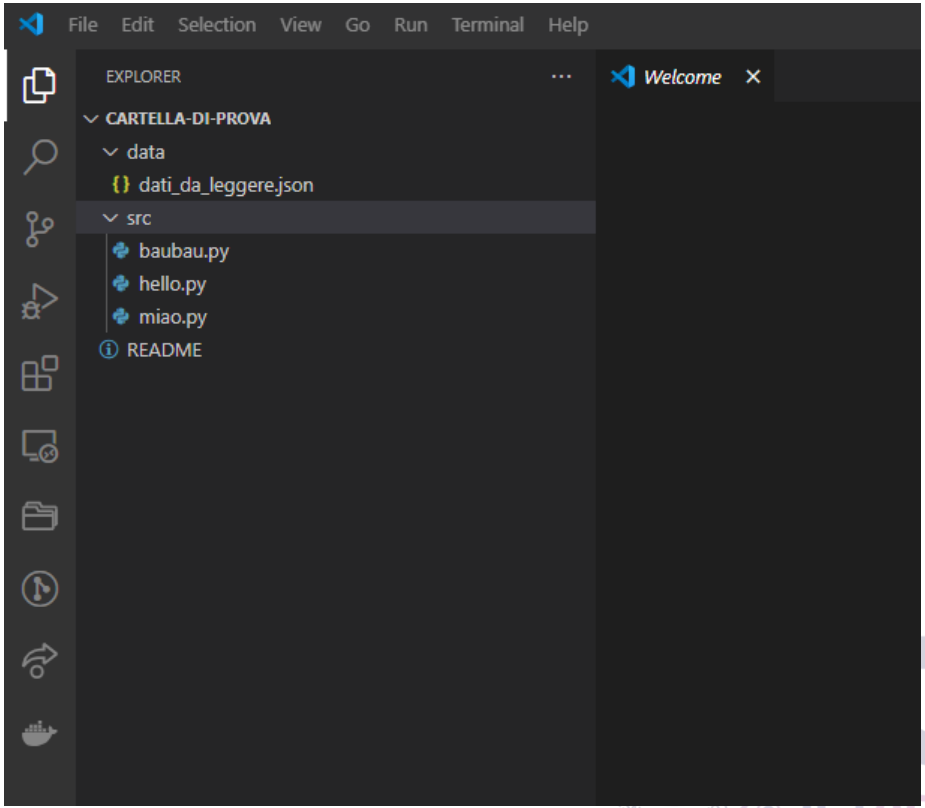


Exercise: Let’s create a “standardized” Python project structure using: mkdir, cd, touch, ls



Visual Studio Code - Setup - Open the project

1. Click “**File > Open Folder**”
2. Choose the parent folder you just created
3. Open the file `hello.py` and write `print(“Hello World”)`



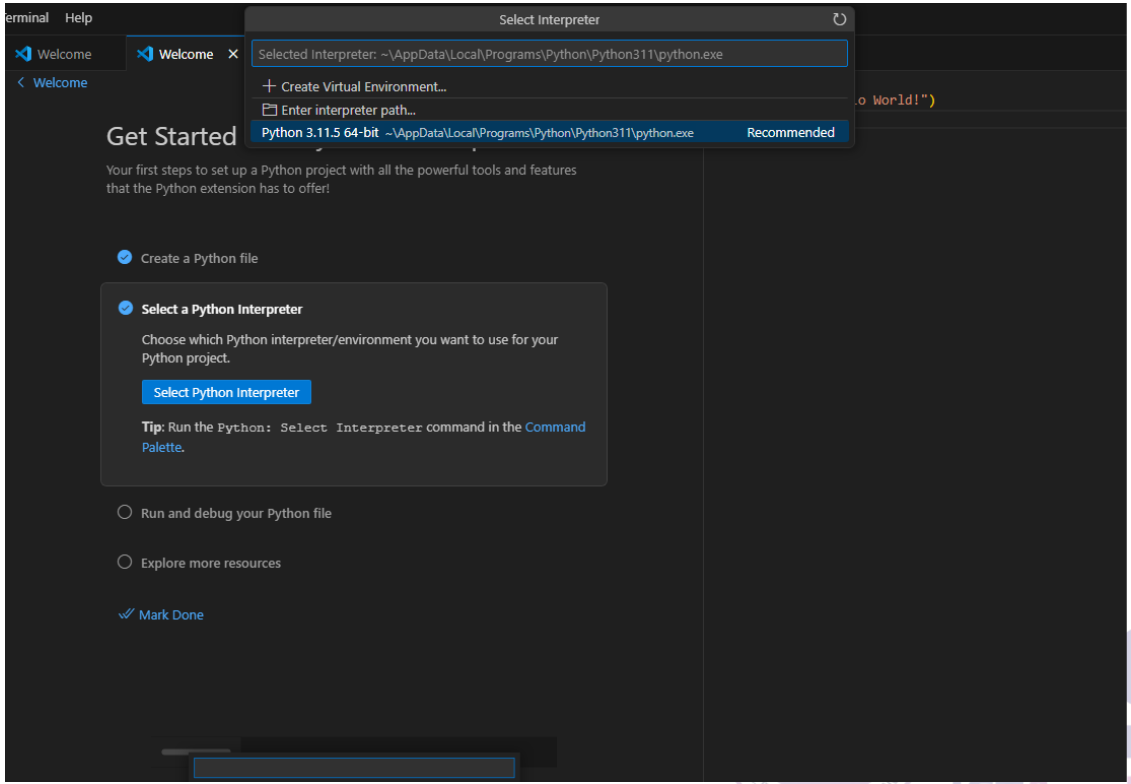
Visual Studio Code - Setup - Running hello.py

1. Click “**Terminal > New terminal**”
2. Navigate to the parent folder of `hello.py`
3. Once there, write `python hello.py`

```
39348@DESKTOP-R63USTJ MINGW64 ~/OneDrive/Desktop/dblp/cartella-di-prova/src
$ python hello.py
Hello World
```

Visual Studio Code - Setup - Python environment

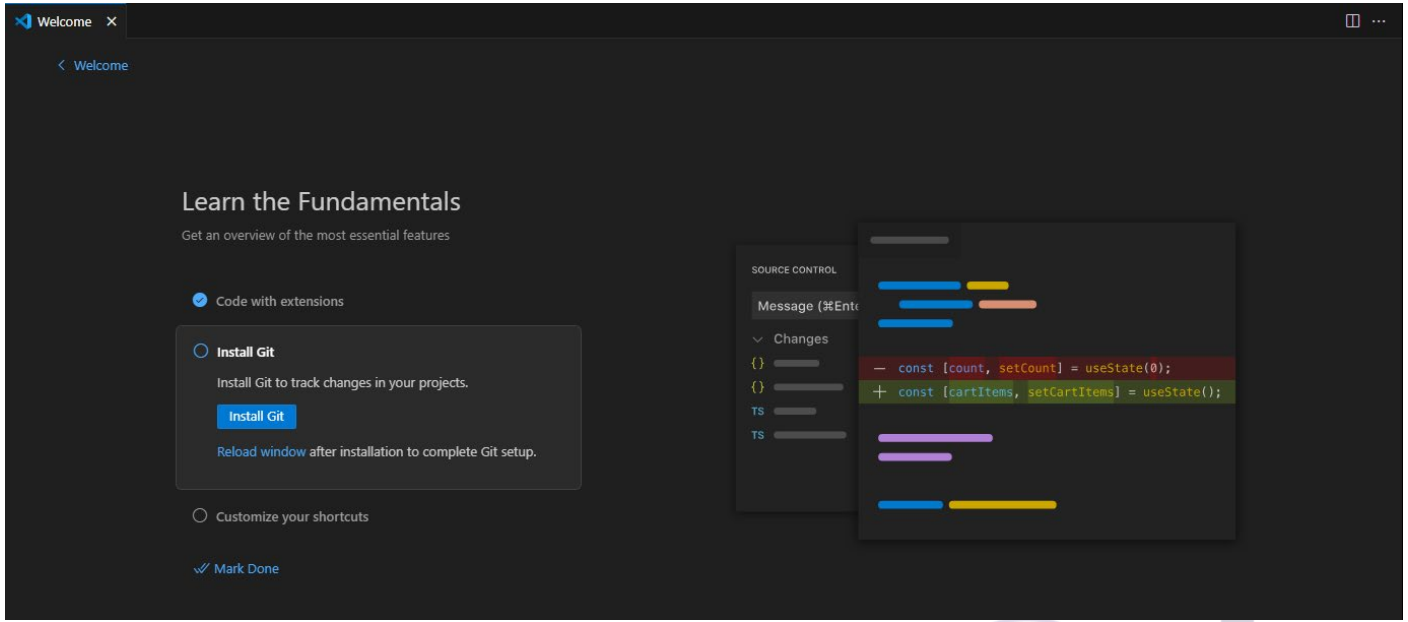
1. Click “**Create Python File**”
2. **Write** in file:
   `print(“Hello World!”)`
3. Go to “**Select** a Python **Interpreter**”
4. **Select** your python **version**.
5. **Go back** into the python **file**



Visual Studio Code - Setup - Git Installation

1. Click “**Install Git**”
2. **Follow** the **instructions** on the pop-up Website.

A. **Download** & **execute** Installer
B. Use **default** settings.
C. **Use VScode** as “**default editor**”
D. Otherwise default options again (there are many)



3. Click “**Reload window**” in VScode

1. Introduction

Understanding Block Diagrams in Algorithms

Algorithm

An **algorithm** is the description of the **problem-solving path** to reach the final results starting from initial data.

We write the algorithm as if addressing an **executor**, capable of performing actions described as **instructions**, written in a specific **language**.

We assume that the executor is available to a user (not necessarily the person who wrote the algorithm) who uses the execution of the algorithm.

Instructions for	Purpose
input	receive data (numbers, expressions, texts...)
output	send messages and communicate results
assignment	store a piece of data by associating it with a variable name
calculation	perform operations on data

Fundamental Structures

Sequence

Start
<instruction>
<instruction>
<instruction>
...
End

Selection

IF <condition>
    THEN
        <instructions1>
    ELSE
        <instructions2>

or

IF <condition>
    THEN
        <instructions1>

Iteration

REPEAT
    <instructions>
UNTIL <condition>

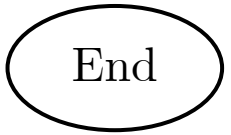
or

WHILE <condition> DO
    <instructions>
END WHILE

or

FOR <index> FROM <start> TO <end>
    DO<instructions>
NEXT <index>

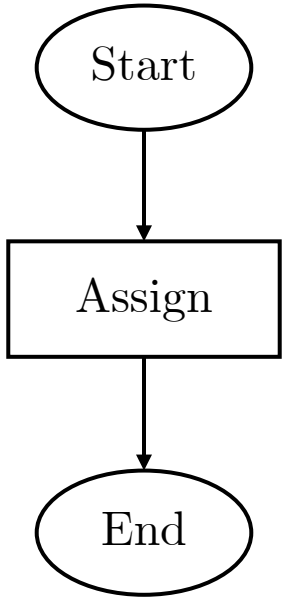
Block Diagram

Opening of the algorithm	Closing of the algorithm	Reading input data (input)	Communicating messages and/or results (output)	Assigning data and/or performing calculations	Controlling the truth value of a condition
					

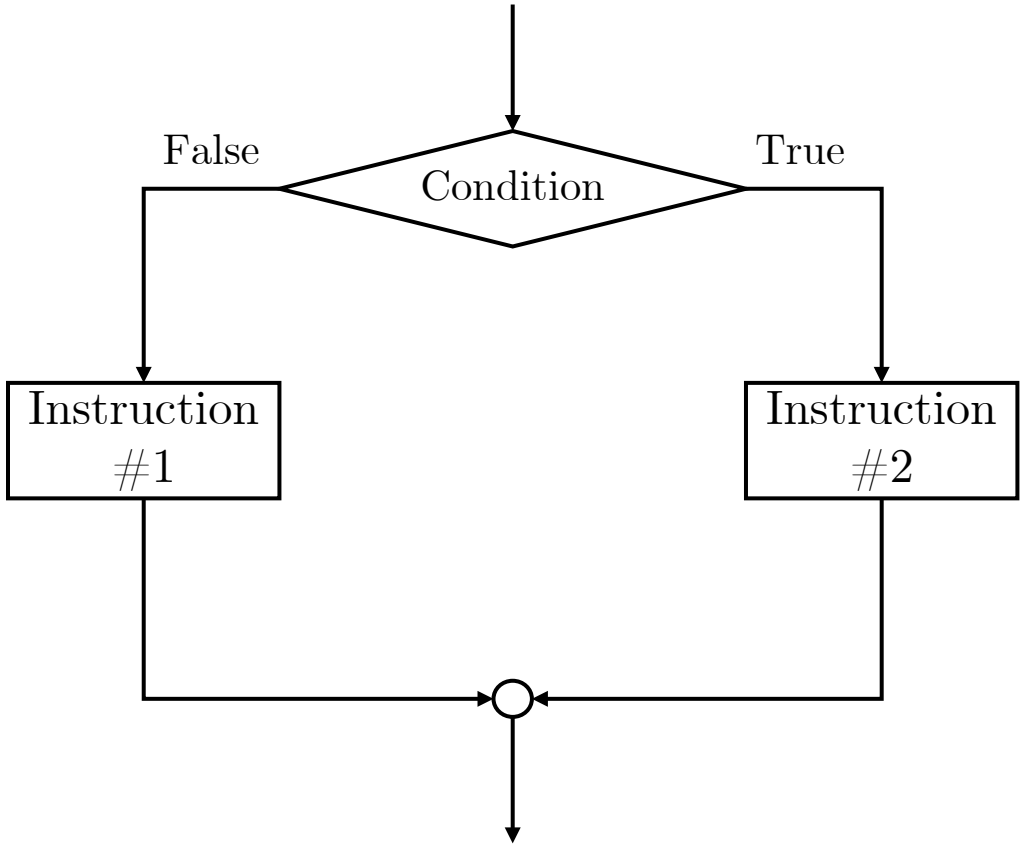
The **three fundamental structures** of algorithms can therefore be described using **flowcharts**. The blocks are represented and connected by arrows, which indicate the flow of execution of the algorithm.

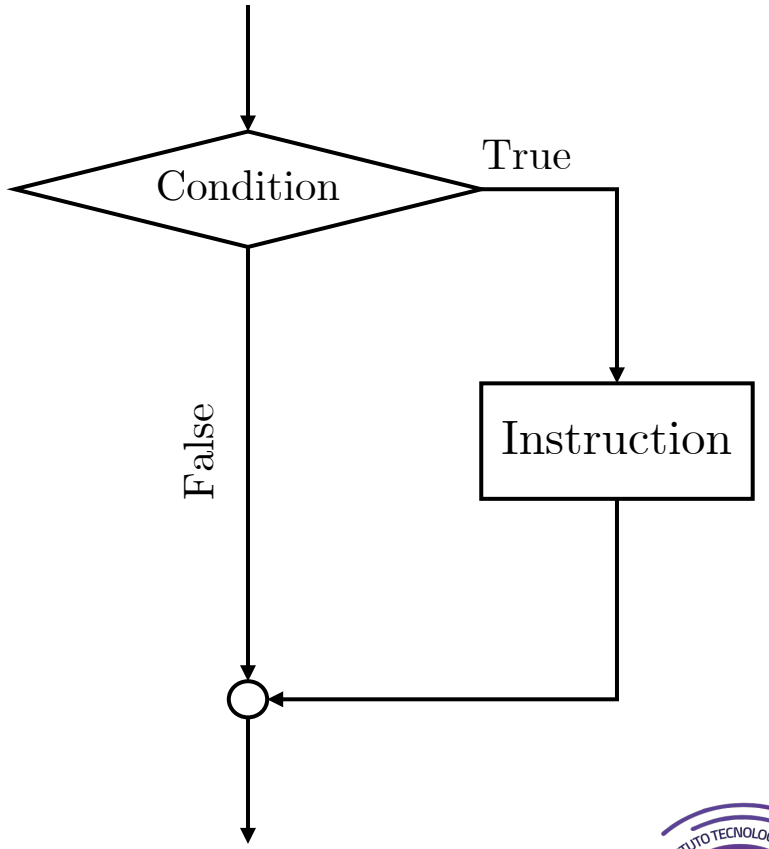
Sequence

The **opening** and **closing blocks** of the algorithm have only **one arrow**, respectively outgoing and incoming, while the **intermediate blocks** generally have **two arrows**: one incoming arrow and one outgoing arrow.

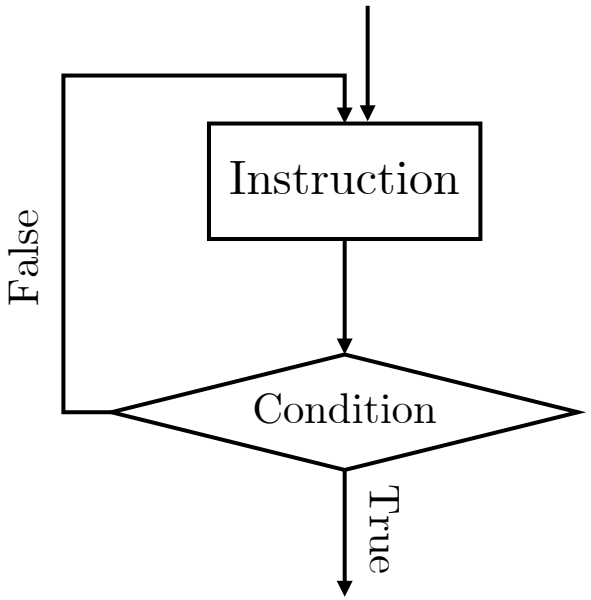


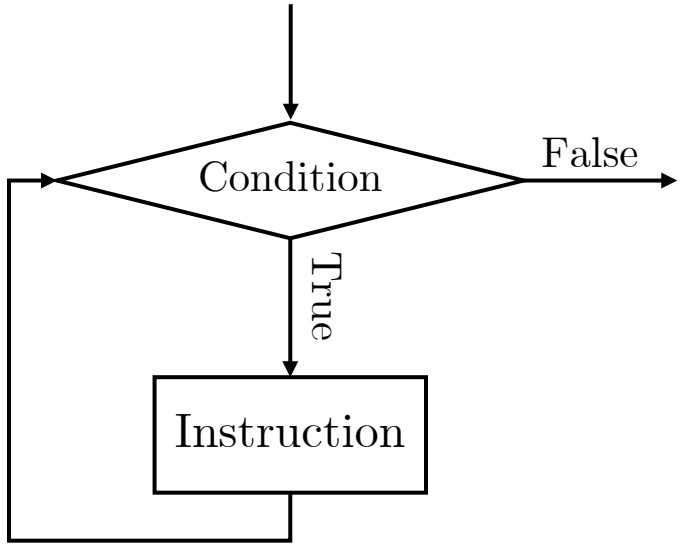
Selection Block

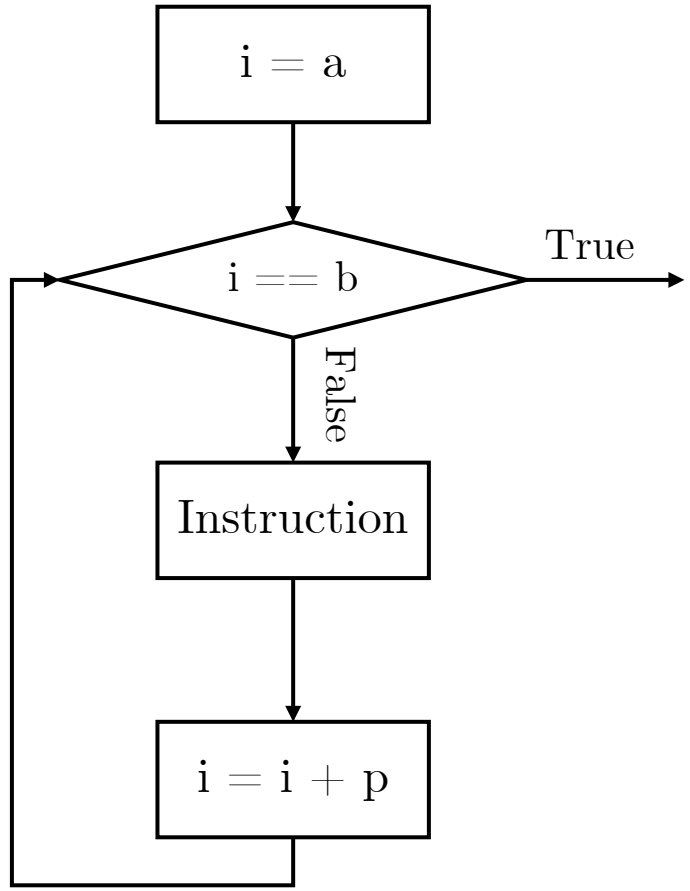




Control Block







Example: perimeter of a rectangle