Programming Techniques

Lecture 01

23 lectures | theory + laboratory

Python

Suggested books:

Think Python
How to Think Like a Computer Scientist
2nd Edition, Version 2.4.0
Allen Downey
(free download)

Python For Everyone
2nd Edition
Cay S. Horstmann and Rance D. Necaise

Lisbon School of Economics & Management
Universidade de Lisboa

Course structure and examination

- Classes:
  - Monday 13.00 - 15.00
  - Wednesday 13.00 - 14.30

- All lessons will be theorethical and include practical exercise.
- Guided/Supported exercises in class + individual/group work

- Evaluation method:
  - 70%: Individual laboratory test (Suff. = 9.5)
  - 30%: Discussion of a computer project (Group of 2/3 or individual)

  - The project will be selected from a list provided (no duplicates), or agreed with the professor.

Lecture 1 - Introduction to Programming and Python

- Introduction to computer systems
- What programming is
- Algorithms, programs, interpreter
- Python philosophy and readability
- Working environments: Anaconda, VS Code, Colab
- First programs and comments

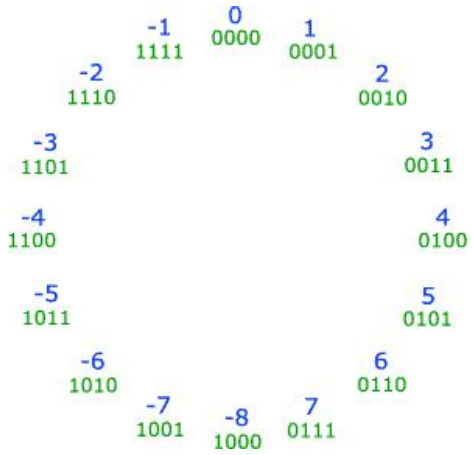
Today's learning goal

Students should be able to explain the core ideas and get familiar with the tools and use them to write short programs before moving to the laboratory exercises.

Focus on reasoning before syntax memorization.

Lecture 1 - Introduction to Computer Systems

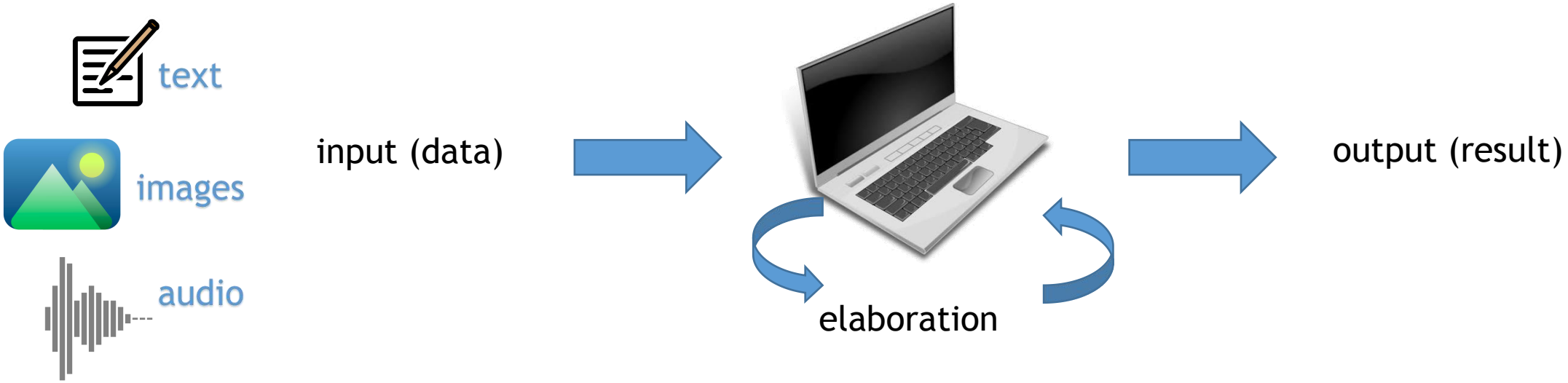
- **Computer Science** (definition): the science that studies the representation and manipulation of information.



TQRCG

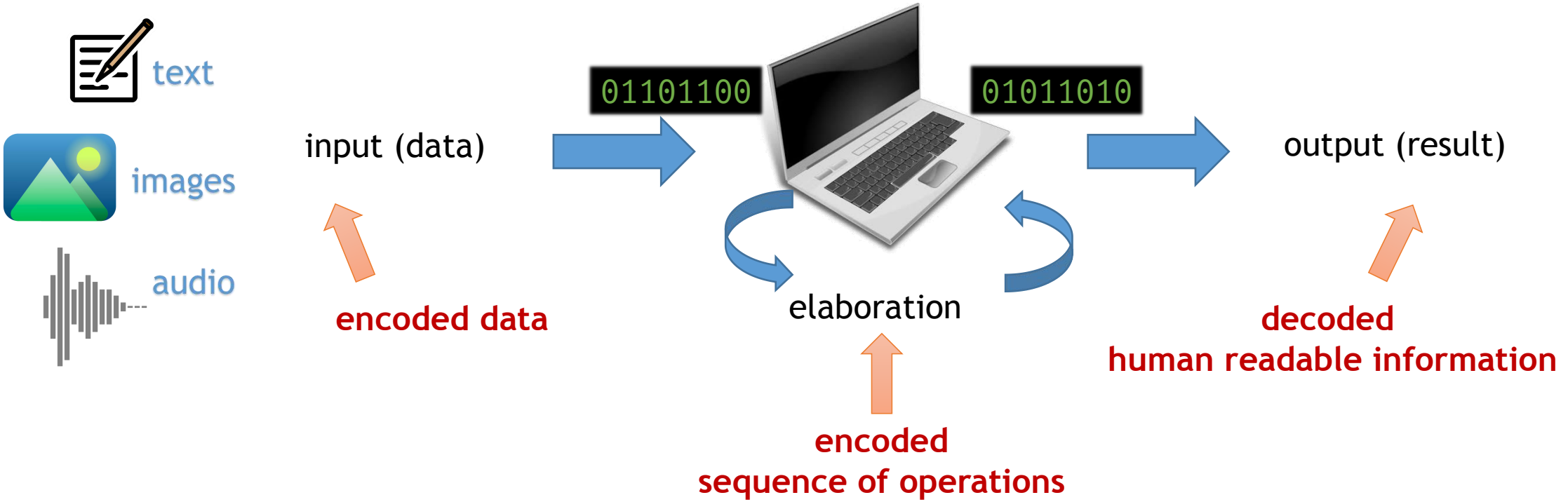
Lecture 1 - Introduction to Computer Systems

- **Computer Science** (definition): the science that studies the representation and manipulation of information.



Lecture 1 - Introduction to Computer Systems

- Computer Science (definition): the science that studies the representation and manipulation of information.



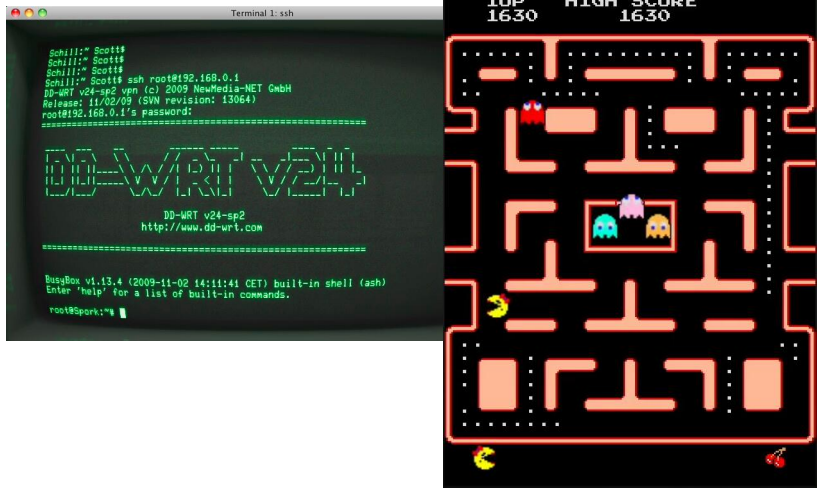
Lecture 1 - What programming is

- **What a program is**
  - a computer program is an encoding that defines **unambiguously** the sequence of steps to complete a task.
  - It is composed by a (huge) number of basic instructions (AND, OR, SUM...)
- Programming means to **project**, **implement** and **test** a program
- The product obtained by running a program is what we call a Software or Application

Lecture 1 - What programming is

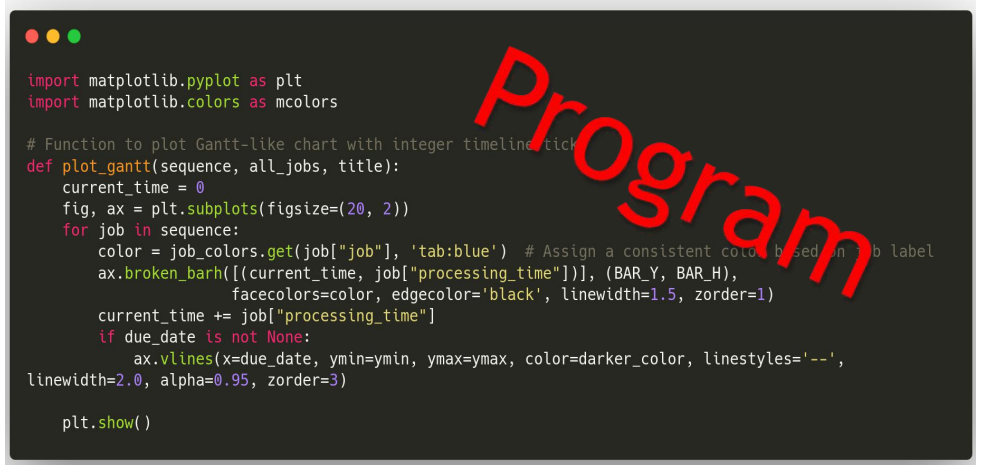
- **What a program is**
  - a computer program is an encoding that defines **unambiguously** the sequence of steps to complete a task.
  - It is composed by a (huge) number of basic instructions (AND, OR, SUM...)
- Programming means to **project**, **implement** and **test** a program
- The product obtained by running a program is what we call a Software or Application

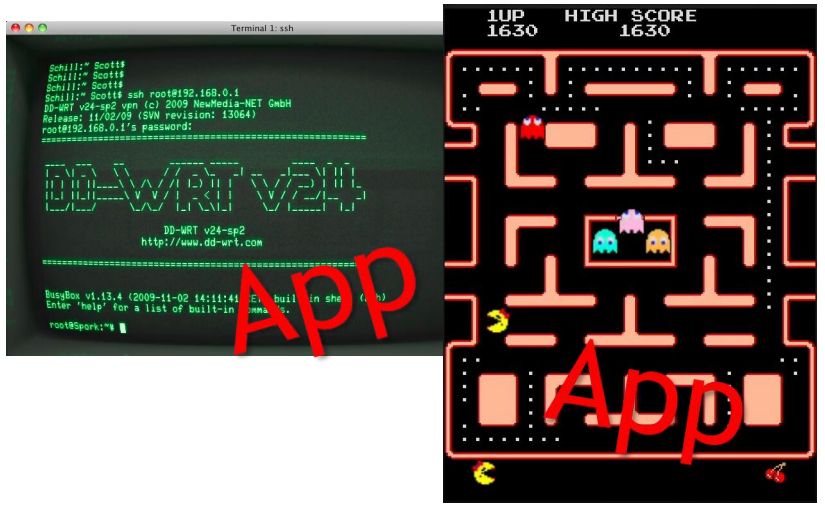
```python
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors

# Function to plot Gantt-like chart with integer timeline ticks
def plot_gantt(sequence, all_jobs, title):
    current_time = 0
    fig, ax = plt.subplots(figsize=(20, 2))
    for job in sequence:
        color = job_colors.get(job["job"], 'tab:blue')  # Assign a consistent color based on job label
        ax.broken_barh([(current_time, job["processing_time"])], (BAR_Y, BAR_H),
                       facecolors=color, edgecolor='black', linewidth=1.5, zorder=1)
        current_time += job["processing_time"]
        if due_date is not None:
            ax.vlines(x=due_date, ymin=ymin, ymax=ymax, color=darker_color, linestyles='--',
linewidth=2.0, alpha=0.95, zorder=3)

    plt.show()
```



Lecture 1 - What programming is

- **What a program is**
  - a computer program is an encoding that defines **unambiguously** the sequence of steps to complete a task.
  - It is composed by a (huge) number of basic instructions (AND, OR, SUM...)
- Programming means to **project**, **implement** and **test** a program
- The product obtained by running a program is what we call a Software or Application





Algorithms, programs and interpreter

Thinking as a programmer

Lecture 1: Algorithms - thinking as a programmer

Wife

"Honey, please, go to the supermarket and get 1 bottle of milk. If they have bananas, bring 6."

Lecture 1: Algorithms - thinking as a programmer

Wife

"Honey, please, go to the supermarket and get 1 bottle of milk. If they have bananas, bring 6."

Husband

*comes back home with 6 bottles of milk

Lecture 1: Algorithms - thinking as a programmer

Wife

"Honey, please, go to the supermarket and get 1 bottle of milk. If they have bananas, bring 6."

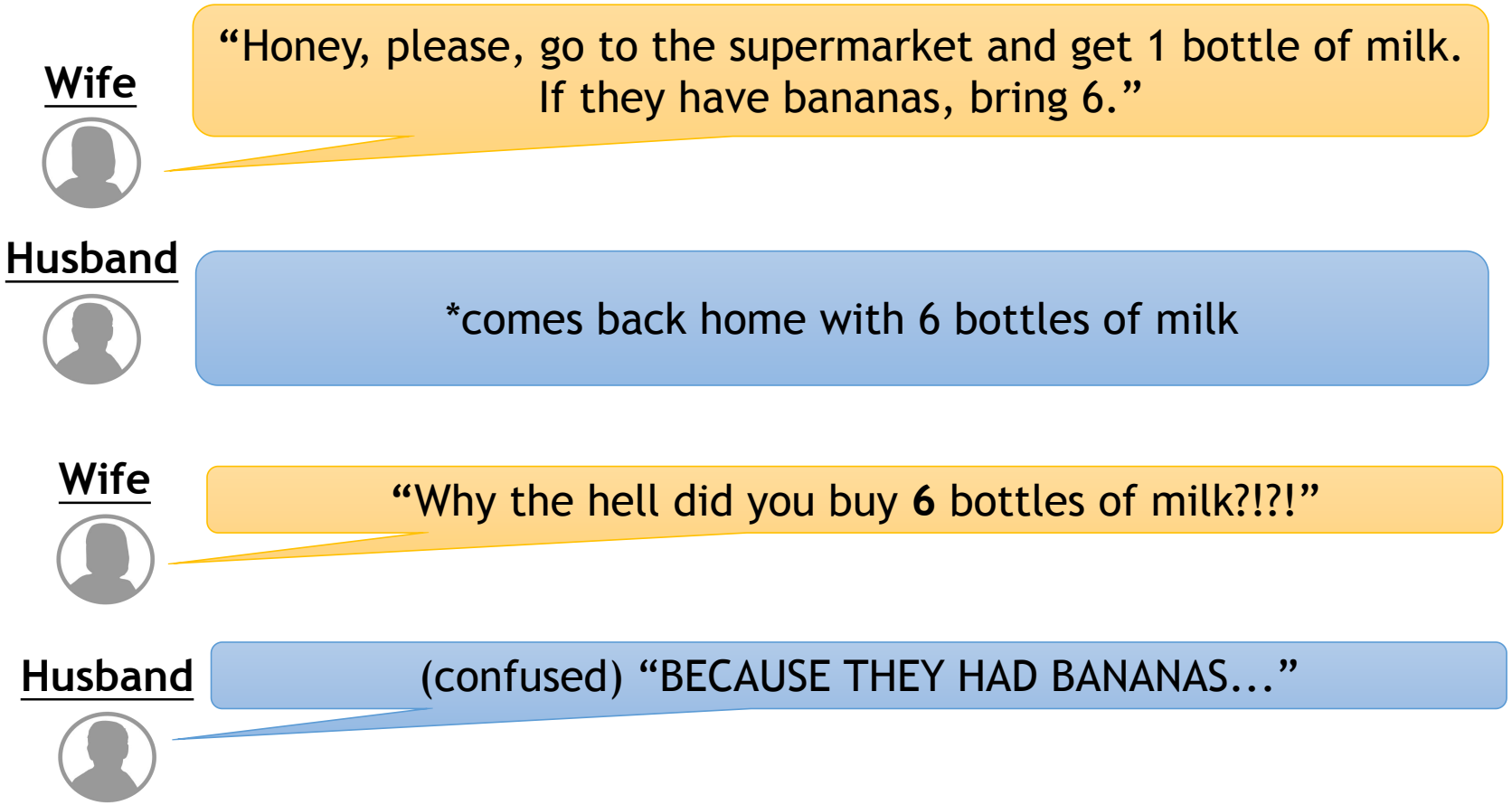
Husband

*comes back home with 6 bottles of milk

Wife

"Why the hell did you buy **6** bottles of milk?!?!"

Lecture 1: Algorithms - thinking as a programmer



He still doesn't understand why his wife yelled at him since he did exactly as she told him.

Lecture 1: Algorithms - thinking as a programmer

- **Algorithm (definition)**: a sequence of steps such that:
  - **Non-ambiguous** (no previous assumption)
  - **Executable** (the operations in sequence make sense)
  - **Terminates** (at a certain point it stops)

- Algorithms are operative plans to solve a problem

- Some of them are well known and easy: compute the area of a circle, the hypotenuse of a triangle...
- Some are more complex: equations, GCD, the trajectory of a rocket...

Lecture 1: Algorithms - formalization

- everyday algorithms: recipes
- example: cooking pasta (as an italian)

PASTA ALGORITHM

Material and ingredients:

- water 1L
- salt 10g
- pasta 100g
- pot
- cooker
- colander

Recipe

1. put water in the pot
2. turn on the cooker fire
3. wait: if it boils then continue
4. put the salt
5. put the pasta
6. wait a short time
7. taste it: if undercooked go to 6. else continue
8. Drain the pasta in a colander

Lecture 1: Algorithms - formalization

- everyday algorithms: recipes
- example: cooking pasta (as an italian)

```
# Ingredients / materials
water = 1        # liter
salt = 10        # grams
pasta = 100      # grams

# Recipe
put_water_in_pot()
turn_on_heat()

while not water_is_boiling():
    wait()

add(salt)
add(pasta)

while pasta_is_undercooked():
    wait_a_short_time()
    taste(pasta)

drain(pasta)
```

PASTA ALGORITHM

Material and ingredients:

- water 1L
- salt 10g
- pasta 100g
- pot
- cooker
- colander

Recipe

1. put water in the pot
2. turn on the cooker fire
3. wait: if it boils then continue
4. put the salt
5. put the pasta
6. wait a short time
7. taste it: if undercooked go to 6. else continue
8. Drain the pasta in a colander

Lecture 1: Algorithms - formalization

- everyday algorithms: recipes
- example: cooking pasta (as an italian)

```
# Ingredients / materials
water = 1        # liter
salt = 10        # grams
pasta = 100      # grams

# Recipe
put_water_in_pot()
turn_on_heat()

while not water_is_boiling():
    wait()

add(salt)
add(pasta)

while pasta_is_undercooked():
    wait_a_short_time()
    taste(pasta)

drain(pasta)
```

Declaration + Initialization

Procedure

PASTA ALGORITHM

Material and ingredients:

- water 1L
- salt 10g
- pasta 100g
- pot
- cooker
- colander

Recipe

1. put water in the pot
2. turn on the cooker fire
3. wait: if it boils then continue
4. put the salt
5. put the pasta
6. wait a short time
7. taste it: if undercooked go to 6. else continue
8. Drain the pasta in a colander

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Analytical solution

$$10000 \cdot (1.05)^n = 20000$$
$$\rightarrow (1.05)^n = 2$$
$$\rightarrow n = \left\lceil \frac{\log 2}{\log 1.05} \right\rceil = \lceil 14.21 \rceil = 15$$

Algorithmical solution

Year	Balance	
0	10000	
1	10000 * 1.05 = 10500	
2	10500 * 1.05 = 11025	
...	...	
14	18856 * 1.05 = 19799	
15	19799 * 1.05 = 20789	> 20000

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Pseudo-code (formal, but natural language or schema)

Algorithmical solution

Year	Balance

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Pseudo-code (formal, but natural language or schema)

1. Set value of `year` to 0
2. Set value of `balance` to 10000

Algorithmical solution

Year	Balance
0	10000

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Pseudo-code (formal, but natural language or schema)

1. Set value of `year` to 0
2. Set value of `balance` to 10000
3. **While** `balance < 20000` do:

Algorithmical solution

Year	Balance
0	10000

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Pseudo-code (formal, but natural language or schema)

1. Set value of `year` to 0
2. Set value of `balance` to 10000
3. **While** `balance < 20000` do:
    1. add 1 to the value of `year`
    2. Multiply `balance` for 1.05

Algorithmical solution

Year	Balance
0	10000
1	10000 * 1.05 = 10500
2	10500 * 1.05 = 11025
...	...
14	18856 * 1.05 = 19799
15	19799 * 1.05 = 20789

Lecture 1: Algorithms - formalization

- **Example: Bank account**
  - You have depositated €10,000 in a new bank account with **annual** interest rate of 5%. How many years does it take to double the initial amount?

Pseudo-code (formal, but natural language or schema)

1. Set value of `year` to 0
2. Set value of `balance` to 10000
3. **While** `balance` < 20000 do:
   1. add 1 to the value of `year`
   2. Multiply `balance` for 1.05
4. Return the value of `year`

Algorithmical solution

Year	Balance	
0	10000	
1	10000 * 1.05 = 10500	
2	10500 * 1.05 = 11025	
...	...	
14	18856 * 1.05 = 19799	
15	19799 * 1.05 = 20789	> 20000

From the algorithm to the code

A language to speak to your computer

Lecture 1: From the algorithm to the code

- Starting from a pseudo-code or a flow chart, coding is (almost) an immediate operation (if you know the language).
- A language includes predefined constructs (“IF”, “WHILE”...) and instructions (+, * ...) with different levels of complexity
- The set of tools depends on the language

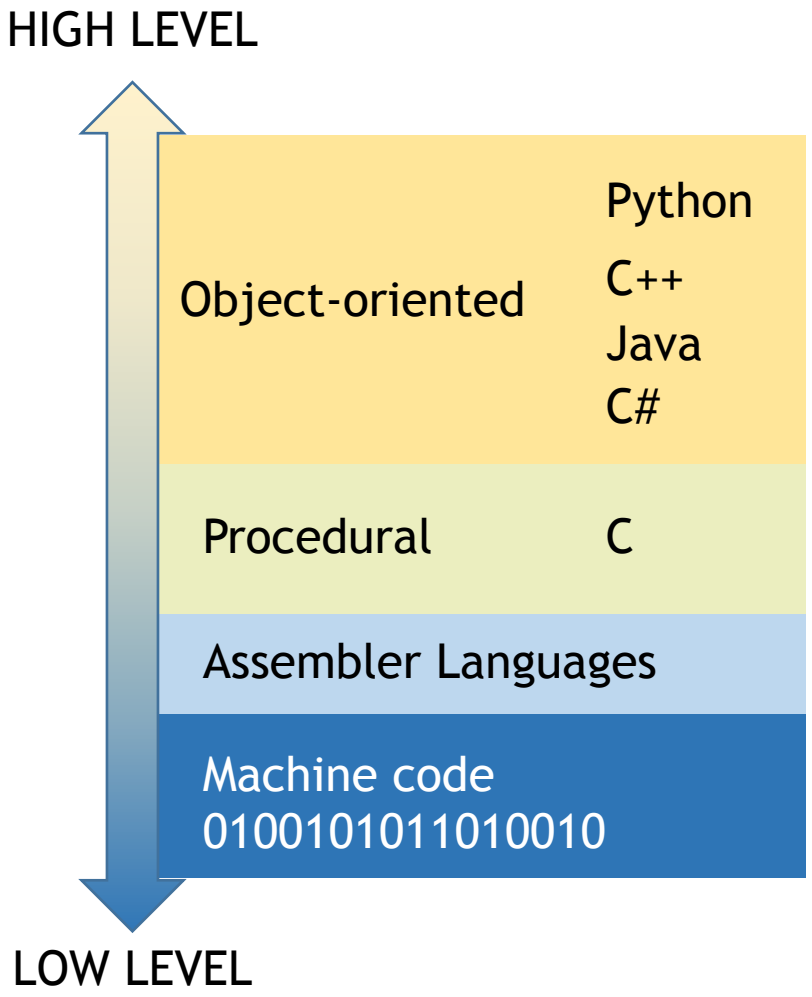
Lecture 1: From the algorithm to the code

- **Level of abstraction**
  - **High level languages**: the language provides elements equivalently complex wrt the human readable flow diagram or pseudocode (conditionals, loop...)
    - **Hardware independent**: the machine is a black box and the same code can be run on different hardware (just becouse someone else translates it into the machine specific language).
    - e.g.

```
if x < 10:
    y = y + 1
```

  - **Low level languages**: instructions are microarchitechtural
    - depends on the hardware
    - e.g. Assembler for Intel Core microprocessor

```
LOAD Reg1, Mem[1000]
ADD Reg1, 10
```



Introduction to Python

IDE, interpreter and language

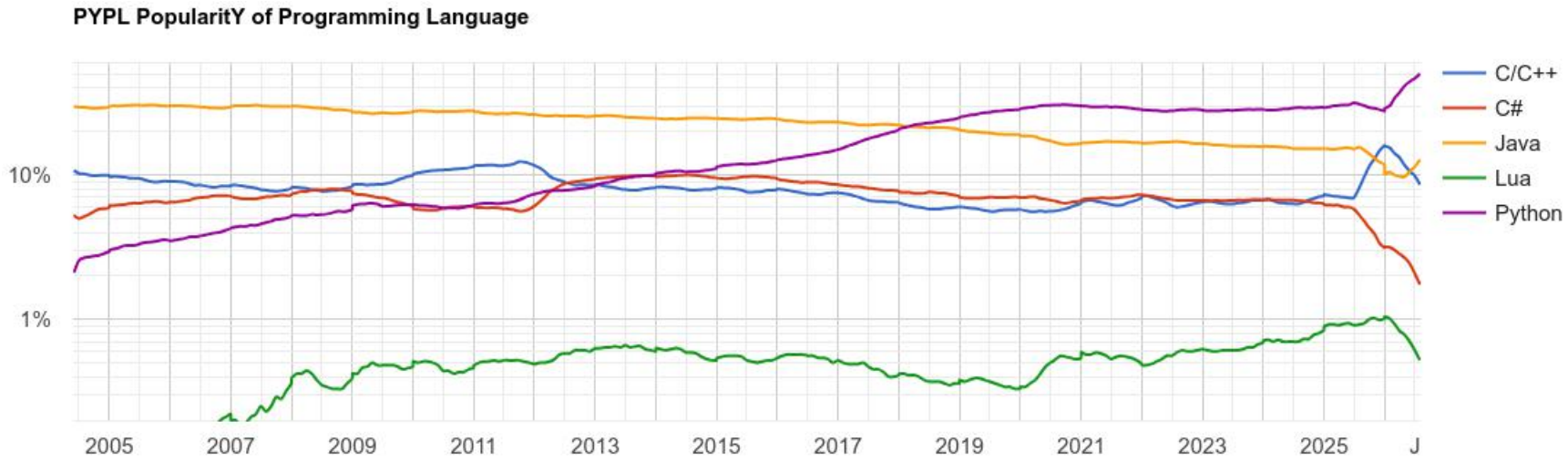
Lecture 1: Introduction to Python

- Early 90s: Guido Van Rossum projects Python
  - Named after his passion for the *Monty Python* show
- Quick and easy sintax for quick prototyping VS languages such as C and C++ optimized to be efficient and write robust large code
- "Battery included" approach: reach of functionality without external libraries and plug-ins
- ***Interpreted* language:**
  - (simplification) The interpreter reads the source code line by line, translates and executes it on the CPU (Python Virtual Machine), at running time
  - VS *compiled* languages (C, C++, Java...) where the code is first translated in machine language, machine specific, and then executed.

Lecture 1: Introduction to Python

- **Why Python**
  - Worldwide, Python is the most popular language, Python grew the most in the last 5 years (21.8%) and C# lost the most (-5.5%) [https://pypl.github.io/PYPL.html]
  - low barrier to entry and beginner-friendly

The Economist [Python is becoming the world's most popular coding language]



Lecture 1: Introduction to Python

- 2 main version: Python 2 (2000 - 2010) and Python 3 (2008 - current)
- Python 3 is not backward compatible with Python 2
- Python 2.7, released in 2010: intended as an intermediate version, to facilitate migration and support partial compatibility. Still used, but not maintained.

- In this course we will use **Python 3**

Lecture 1: Python IDE and tools

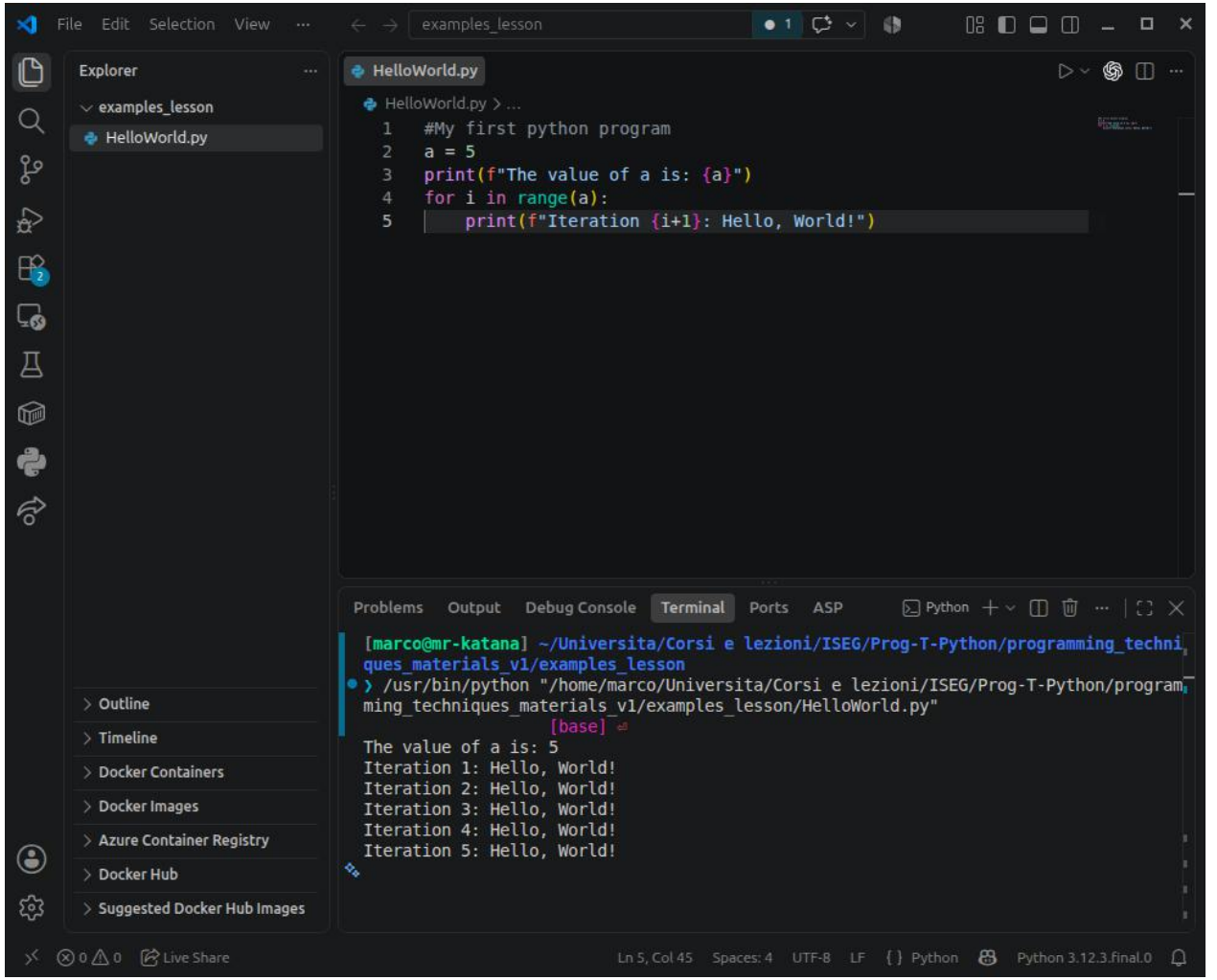
- **Many programming environments:**
  - **IDE** (Integrated Developement Environment): PyCharm, Visual Studio Code...
  - **Simple text editor** (Notes, Notepad, vi, gedit...)
  - **Jupyter Notebook**: interactive environment, deliverable as a file, combining code, text, figures and videos; can be used through an IDE (Jupyter or VS Code extension) or the dedicate Google Colab online tool (no installation required, requires Google account)
  - **Interactive interpreter** directily in the terminal


```
marco@mr-katana ~
❯ python                                                    [base]
Python 3.10.15 (main, Oct  3 2024, 07:27:34) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 2
>>> while a < 5:
...     print("a is less than 5")
...     a = a+1
...
a is less than 5
a is less than 5
a is less than 5
>>> print(f"now a = {a}")
now a = 5
>>>
```

  - Choose your favorite
  - During classes I use VS Code and deliver exercises through Jupiter Notebook or text files.

Lecture 1: Python IDE and tools

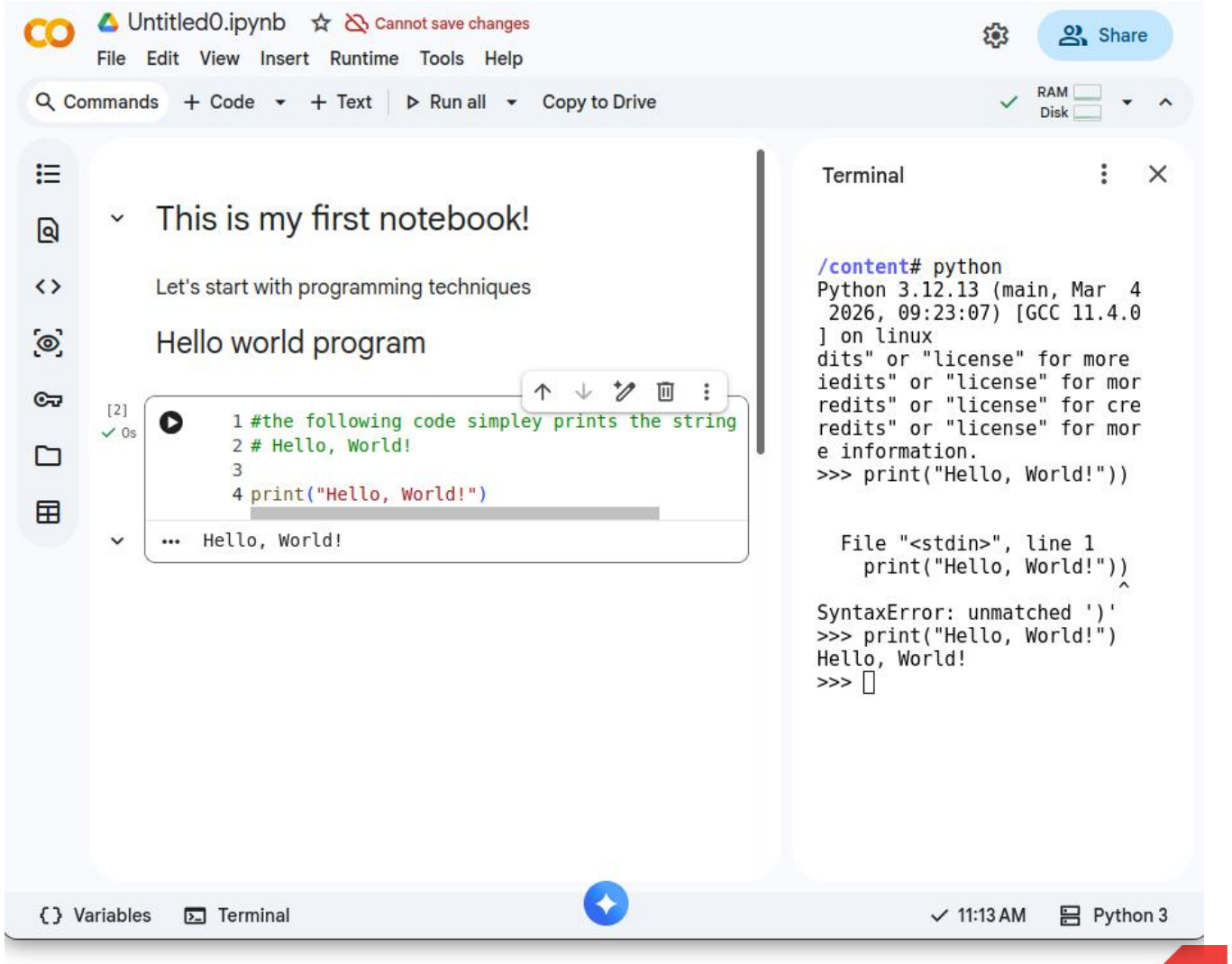
- **IDE:** The programmer's best friend
  - Code visualization and writing
    - Highlight with colors (#comments, numbers, keywords, variables, strings...)
    - Visualize line number
    - Support to indentation (python)
    - Sintax error hilight
    - auto-completion/suggestion
    - Group and recognize *projects* composed of many files
  - Output window
    - to see what the program produces (text)
  - Debugger
    - set of tools to analyze logic errors of the code



Lecture 1: Python IDE and tools

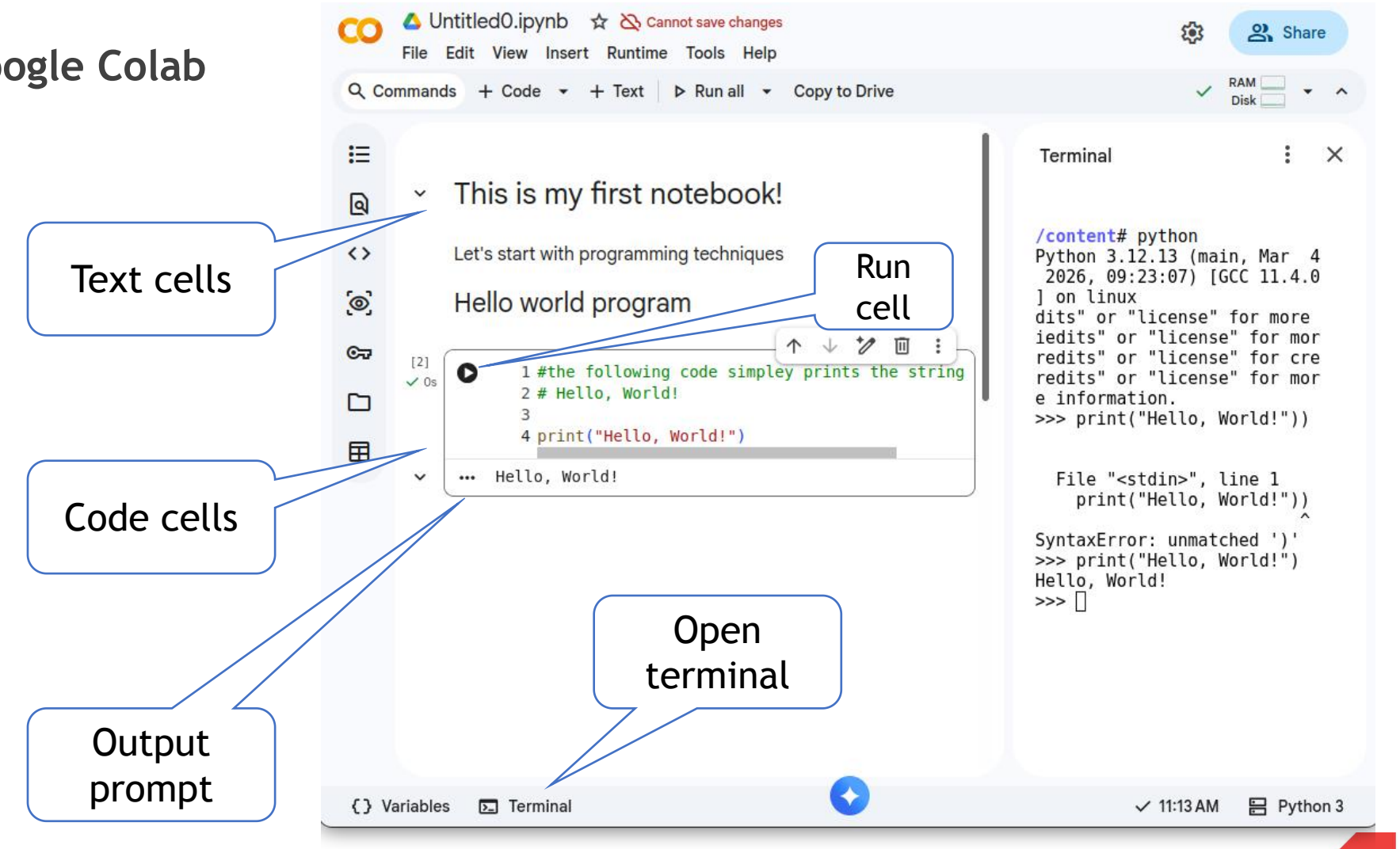
- Jupyter and Google Colab
  - Let's try it!

- Create a new folder in GDrive
- Right click > other > Colaboratory
- Automatically a new notebook is opened, located in your folder



Lecture 1: Python IDE and tools

- Jupyter and Google Colab



Python Syntax

First bites of code writing

Lecture 1: Python syntax

- Typos errors: `primt("Hello World!")`
- Case sensitive:
  - `Variable != variable != VARIABLE`
  - `PyTHon != Python != python`
- Spaces: separate different elements (no names with spaces for variables or functions)
  - `player1_name != player1 name`
  - but `points=points+1` is the same of `points = points + 1`
- New Line: differently from C-style, instructions are separate by the end-of-line (instead of `;`). `;` still can be used to separate same line instructions (not recommended)
- Indentation: no brackets, the indentation defines nested instructions

```
x = False
if x:
    print("x is True")
```

Prints nothing

```
x = False
if x:
print("x is True")
```

Prints "x is True" anyway

Lecture 1: Python syntax

- `print()` **function**
  - A ***function*** is a stand-alone sequence of operations **with a name**, so that we can reuse it. It takes some data (with a particular structure), and executes some task.
  - Basically, a function is code written by someone else for our use.
  - Documentation of `print` function: https://docs.python.org/3/library/functions.html#print

```
print(*objects, sep=' ', end='\n')
```

Print `objects` to the text stream `file`, separated by `sep` and followed by `end`. `sep` and `end`, if present, must be given as keyword arguments ... Both `sep` and `end` must be strings; they can also be `None`, which means to use the default values. If no objects are given, `print()` will just write `end`.

  - NOTE: the symbol * means that `objects` can be any number of elements (0, 1 or more).
  - Basic concept: a `string` is a sequence of charachters, that is plain text. It can be manipulated, and stored into a variable; it is indicated with quotes or duble quotes, e.g. “`Hello, World!`” or ‘`hello world`’

Lecture 1: Python syntax

- Sintax for built-in functions
  - To call (use) a function we need to specify:
    - name of the function, e.g. `print`
    - values needed for the input data, they are called *arguments* or parameters, e.g. “`Hello World!`”
    - Arguments are included between `( )` and separated by commas.
    - the first arguments without default value (see `*objects` VS `sep=' '`) are mandatory, the others can be modified by explicit overwrite
  - Most functions return a value (or more than one)
    - This value can be stored in a variable or used in other operations
      - example: `abs(-15)` returns value `15`
    - `print` write text in terminal, but does not return any value.
  - We can use nested invocations

```
print(abs(-15))
```

Lecture 1: Python syntax

- Sintax for built-in functions
  - To call (use) a function we need to specify:
    - name of the function, e.g. `print`
    - values needed for the input data, they are called *arguments* or parameters, e.g. “`Hello World!`”
    - Arguments are included between `( )` and separated by commas.
    - the first arguments without default value (see `*objects` VS `sep=' '`) are mandatory, the others can be modified by explicit overwrite
  - Most functions return a value (or more than one)
    - This value can be stored in a variable or used in other operations
      - example: `abs(-15)` returns value `15`
    - `print` write text in terminal, but does not return any value.
  - We can use nested invocations

```
print(abs(-15))
```

HANDS-ON! (Guided session 1.1)

Lecture 1: Python syntax

- Let’s see in a Notebook (Colab or IDE):
  - print multiple objects as strings and expressions
  - print using variables
  - print with different separator
  - print using different “end” parameters

HANDS-ON! (Guided session 1.1)

Lecture 1: Errors

- Different types of errors:
  - Syntax Errors (pre-execution)
    - early detection with common IDEs
    - Python recognizes it, and identifies the origin
  - Runtime errors
    - execution starts, but an error occurs while an instruction is being executed
    - In general, refers to a violated rule of the language (often because someone didn't read the docs...) and the application "crashes"
  - Logical errors
    - the program runs without exceptions, but the result is not what was intended
    - It depends on what the programmer wants to do, and it is up to him to find it
    - Helpful testing and debugging tools

Lecture 1: Errors

- Different types of errors:
  - Syntax Errors (pre-execution)
    - early detection with common IDEs
    - Python recognizes it, and identifies the origin
  - Runtime errors
    - execution starts, but an error occurs while an instruction is being executed
    - In general, refers to a violated rule of the language (often because someone didn't read the docs...) and the application "crashes"
  - Logical errors
    - the program runs without exceptions, but the result is not what was intended
    - It depends on what the programmer wants to do, and it is up to him to find it
    - Helpful testing and debugging tools

HANDS-ON! (Laboratory 1.2)