LECTURE 1 - MARCH 6, 2024

SOME INFORMATION

PROF. DAVIDE PETTURITI

EMAIL: davide.petturiti@unipg.it

E-LEARNING: UniStudium → PYTHON LAB (2023/24)

PROGRAM IN BRIEF

1. INTRODUCTION TO PYTHON
2. CONTROL STRUCTURES AND PROGRAMMING
3. FUNCTIONS AND HINTS OF OBJECT-ORIENTED PROGRAMMING
4. STRUCTURED DATA TYPES AND CONTAINERS
5. ACCESS, MANIPULATION AND VISUALIZATION OF FINANCIAL DATA

REFERENCE BOOKS

- P.J. DEITEL, H. DEITEL: INTRO TO PYTHON FOR COMPUTER SCIENCE AND DATA SCIENCE, PEARSON, 2020.
- Y. HILPISCH: FINANCIAL THEORY WITH PYTHON – A GENTLE INTRODUCTION, O'REILLY MEDIA, 2021.

DIDACTIC MATERIAL AND LECTURE NOTES WILL BE PROVIDED THROUGH THE UniStudium PAGE OF THE COURSE.

EXAM MODALITY

THE EXAM CONSISTS IN A PRACTICAL TEST OF PROGRAMMING IN PYTHON LANGUAGE FOLLOWED BY A DISCUSSION OF THE WRITTEN CODE.

INTRODUCTION TO PYTHON

QUESTION: WHAT IS PYTHON?

PYTHON IS A HIGH-LEVEL PROGRAMMING LANGUAGE SUPPORTING VARIOUS PROGRAMMING PARADIGMS:

- PROCEDURAL PROGRAMMING,
- FUNCTIONAL PROGRAMMING,
- OBJECT-ORIENTED PROGRAMMING.

PYTHON NATURALLY EMBRACES THE OPEN SOURCE PHILOSOPHY SO A LARGE AMOUNT OF LIBRARIES ARE FREELY AVAILABLE. SUCH LIBRARIES ALLOW US TO EASILY DEAL WITH, E.G.:

- DATA MANIPULATION AND VISUALIZATION,
- MACHINE LEARNING,
- DEEP LEARNING,
- OPTIMIZATION,
- MANY BRANCHES OF ARTIFICIAL INTELLIGENCE,
- AND SO ON...

SOME HISTORY

1991: GUIDO VAN ROSSUM PUBLISHED PYTHON'S FIRST VERSION

1994: PYTHON 1.0 WAS RELEASED

2000: PYTHON 2.0 WAS RELEASED

2008: PYTHON 3.0 WAS RELEASED

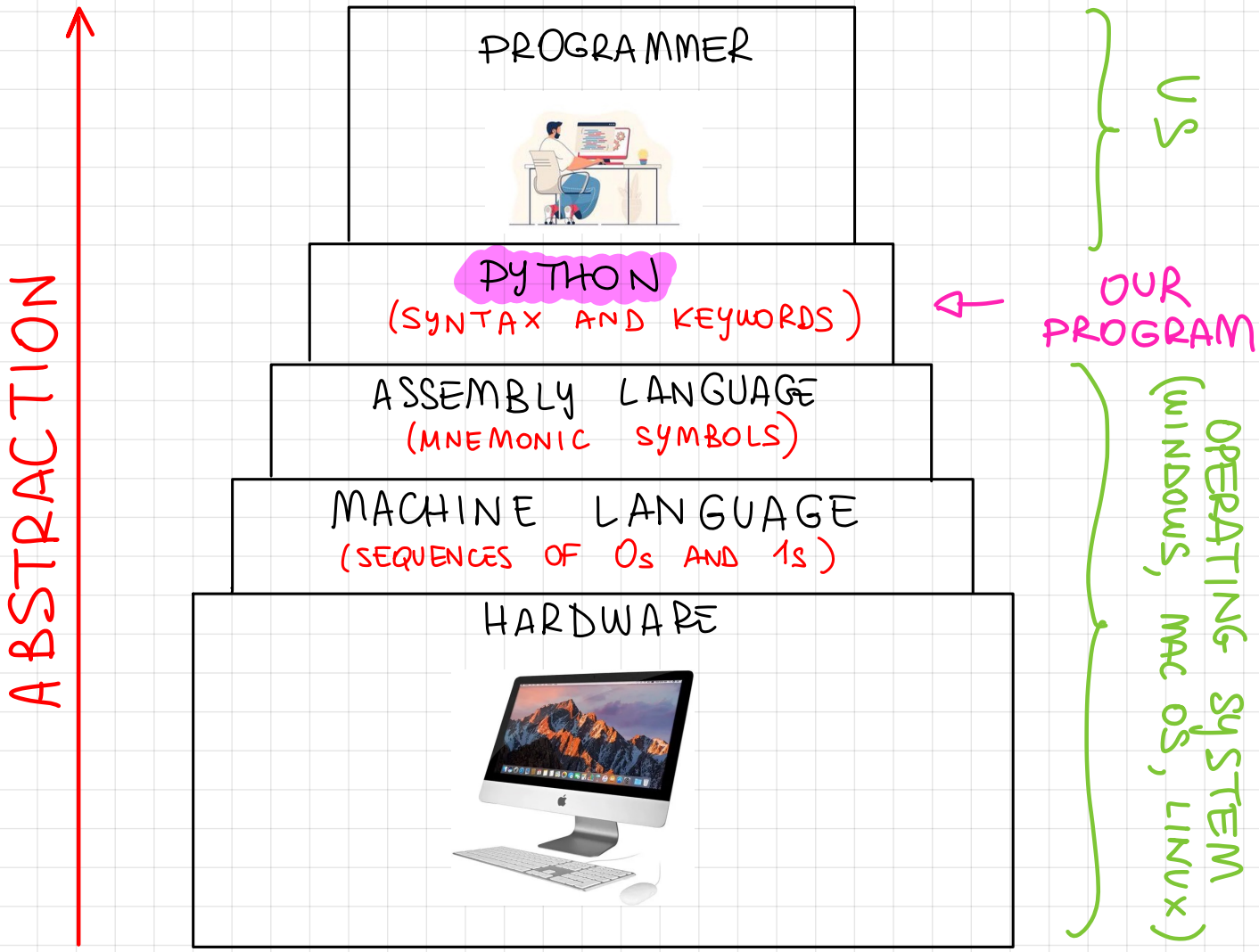
NEW FEATURES ADDED TO THE LANGUAGE

2024: THE CURRENT VERSION IS PYTHON 3.11.X

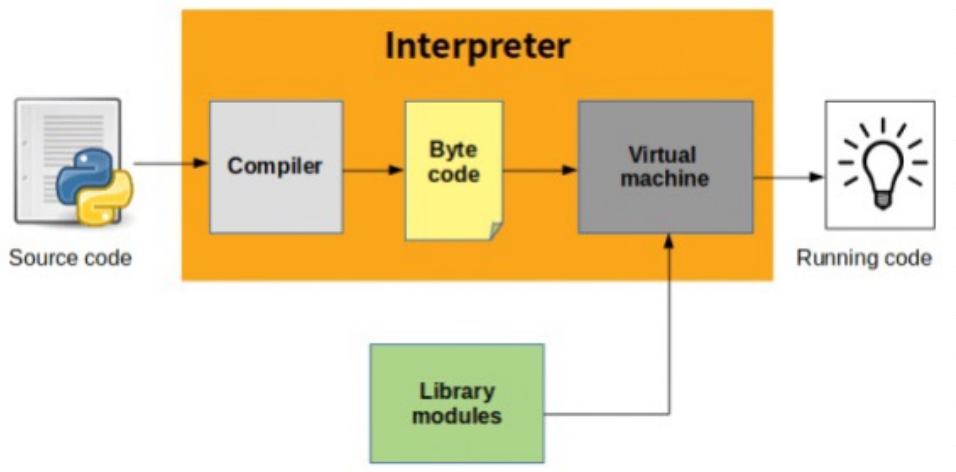
SUBVERSION

CURIOSITY: THE NAME "PYTHON" IS INSPIRED TO THE BRITISH COMEDY GROUP MONTY PYTHON.

WHAT IS A PROGRAMMING LANGUAGE?



PYTHON IS AN INTERPRETED LANGUAGE, I.E., THERE IS A SOFTWARE, CALLED IPython, THAT TRANSLATES OUR PROGRAM (WRITTEN IN Python LANGUAGE) IN RUNNING CODE.



THERE ARE 3 MODES OF INTERACTION WITH IPython:

1. INDIVIDUAL CODE SNIPPETS IN THE IPython INTERACTIVE ENVIRONMENT.
2. COMPLETE PROGRAMS (CALLED SCRIPTS) SAVED AS FILES WITH EXTENSION .py PASSED TO IPython: PYTHON CODE IS OFTEN WRITTEN AND EXECUTED IN AN INTEGRATED DEVELOPMENT ENVIRNOMENT, LIKE THE Spyder IDE.
3. Jupyter Notebooks, WHERE PYTHON CODE IS EXECUTED IN A WEB-BROWSER-BASED ENVIRONMENT. Jupyter Notebooks HAVE EXTENSION .ipynb

ANACONDA DISTRIBUTION

WE USE THE Anaconda Python Distribution SINCE IT IS EASY TO INSTALL AND CONTAINS:

- THE IPython INTERPRETER,
- MOST OF THE LIBRARIES WE WILL USE,
- A LOCAL Jupyter Notebook SERVER,
- THE Spyder IDE.

FOR YOUR PERSONAL COMPUTER:

DOWLOAD AND INSTALL Anaconda AT:

https://www.anaconda.com/download

THEN CLICK ON THE Anaconda ICON 

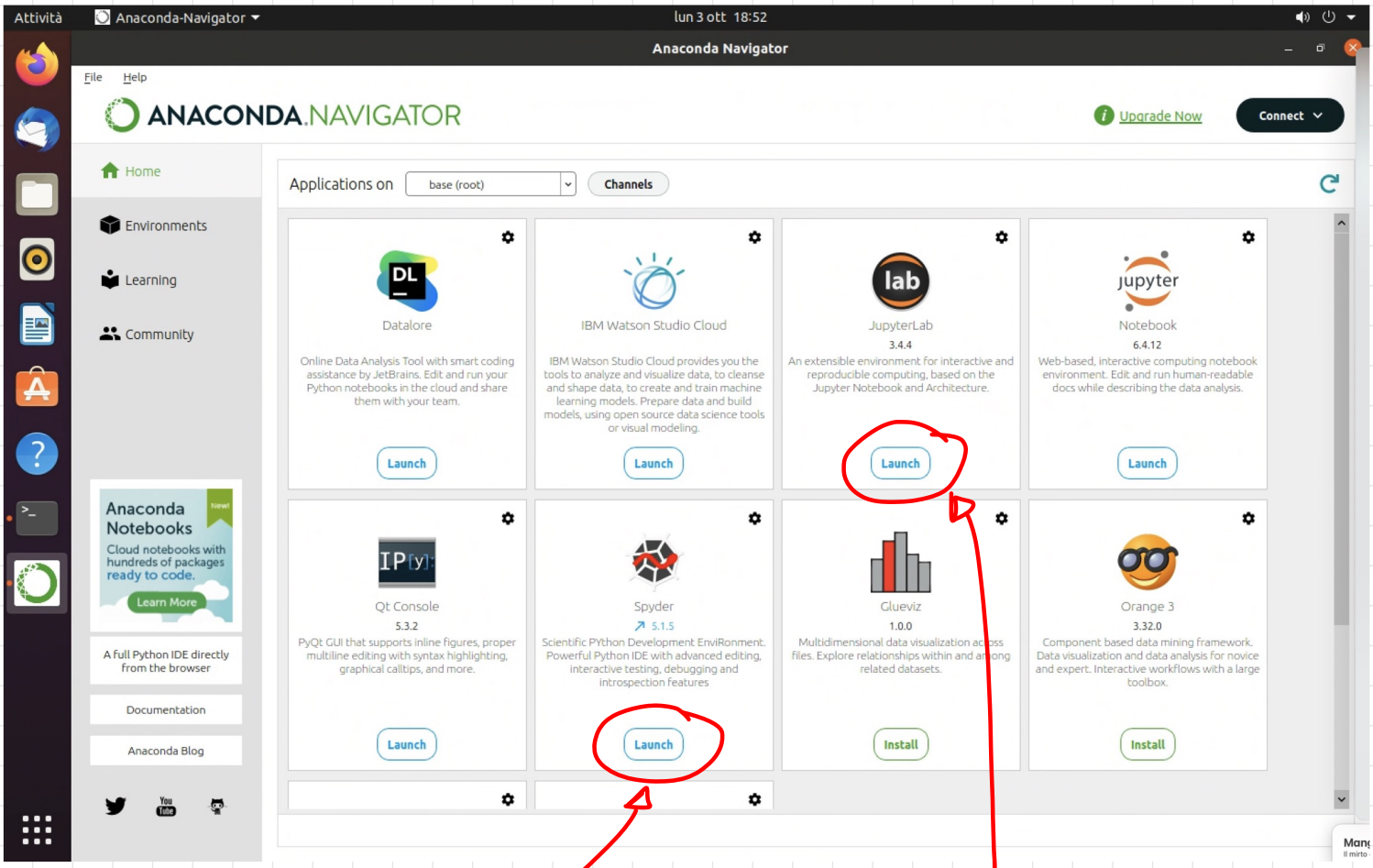
IT IS A LINUX BASED OS

IN THE LAB COMPUTERS:

Anaconda IS ALREADY INSTALLED IN THE Ubuntu OS VIRTUAL MACHINE. TO LAUNCH IT, OPEN THE Terminal  AND WRITE (THEN CONFIRM WITH ↵)

```
:~$ anaconda-navigator
```

PROMPT OF THE Ubuntu OS



② CLICK TO USE THE SPYDER IDE

③ CLICK TO CREATE A JUPYTER NOTEBOOK

① TO ACCESS IPython INTERACTIVELY, WRITE IN THE TERMINAL:

```
:~$ ipython
```

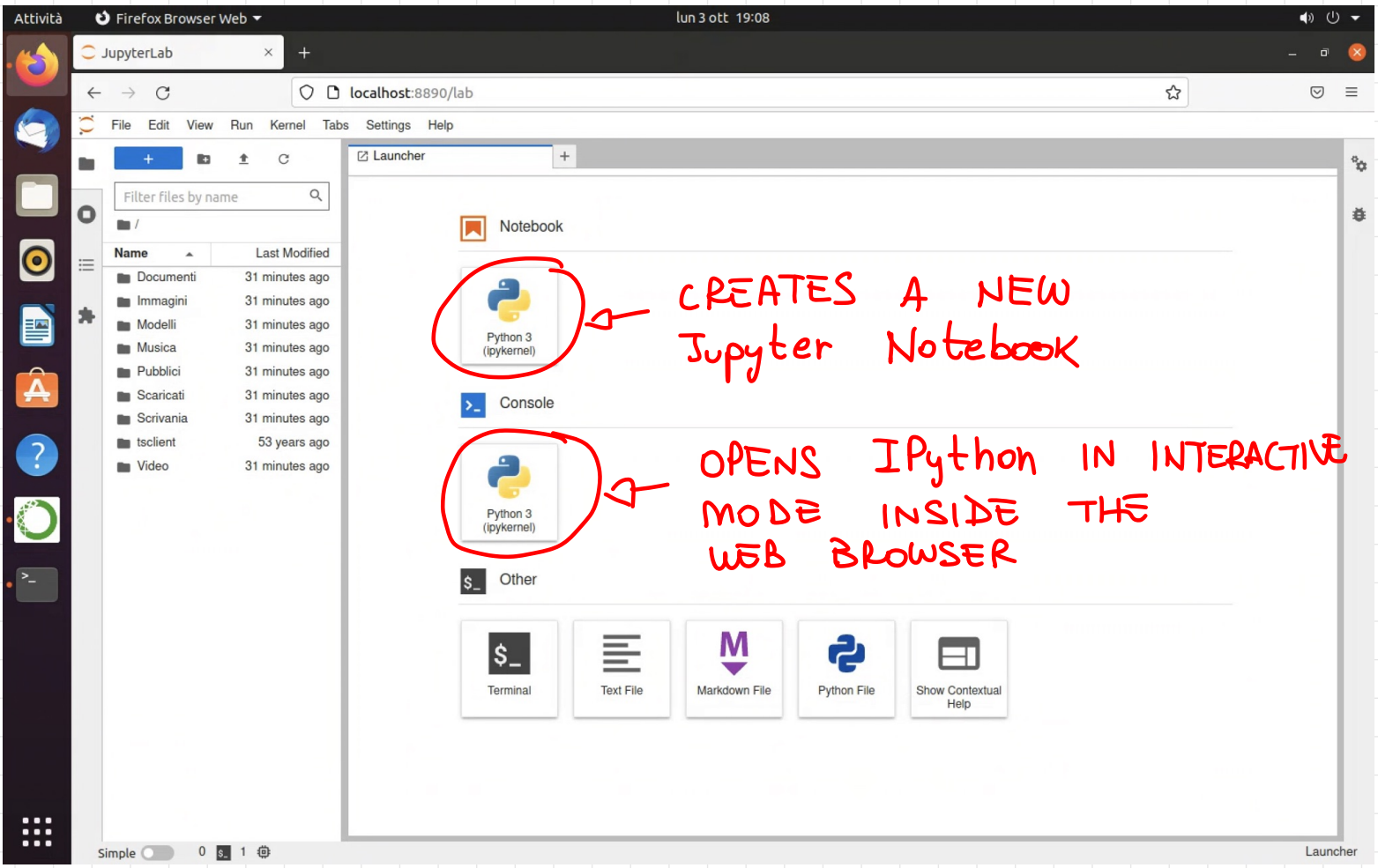
PACKAGE MANAGERS

Anaconda COMES WITH TWO PACKAGE MANAGERS: conda AND pip. TO INSTALL A PACKAGE WE WILL WRITE IN THE TERMINAL:

- conda install <package_name>
- pip install <package_name>

} IN PRACTICE THEY DO THE SAME JOB

USING JupyterLab



VARIABLES AND ASSIGNMENT

WE CAN USE IPython AS A CALCULATOR:

```
[]: 45 + 72 * (5 + 78) ** 2
```

REMARK: THE ARITHMETIC OPERATORS ARE

ADDITION	+
SUBTRACTION	-
MULTIPLICATION	*
EXPONENTIATION	**
TRUE DIVISION	/
FLOOR DIVISION	//
REMAINDER	%

ROUND PARENTHESES CAN BE USED TO HAVE A CLEARER VIEW OF PRIORITY AMONG OPERATIONS.

MOST OFTEN WE USE A VARIABLE TO STORE VALUES THROUGH AN ASSIGNMENT STATEMENT, USING "=":

```
[]: x = 3
    y = 5
    total = x + y
```

REMARK: IN INTERACTIVE MODE OR IN A Jupyter Notebook, TO PRINT THE VALUE OF A VARIABLE WE CAN SIMPLY WRITE ITS IDENTIFIER. IN THE SCRIPT EXECUTION WE NEED THE INSTRUCTION print.

```
[]: total
```

IMPORTANT: NAMES OF VARIABLES, FUNCTIONS AND CLASSES ARE CALLED IDENTIFIERS. THEY ARE CASE SENSITIVE: number ≠ Number ≠ NUMBER! IDENTIFIERS MUST BE DIFFERENT FROM PYTHON'S RESERVED KEYWORDS AND CANNOT BEGIN WITH A NUMBER.

PYTHON IS A STRONGLY-TYPED LANGUAGE: EVERY VARIABLE IS ASSOCIATED TO A PARTICULAR TYPE OF DATA. THE BASIC TYPES ARE

- int FOR INTEGER NUMBERS, LIKE -10
- float FOR REAL NUMBERS, LIKE 1.2
- str FOR STRINGS, LIKE 'hello'
- bool FOR BOOLEAN VALUES: True AND False

THESE TYPES ARE ACTUALLY PARTICULAR "CLASSES". THE OBJECT-ORIENTED PARADIGM USES CLASSES TO DEFINE STRUCTURED TYPES OF DATA WITH PREDETERMINED FEATURES (CALLED ATTRIBUTES) AND FUNCTIONALITIES (CALLED METHODS). INTUITIVELY

- A CLASS IS LIKE THE DESIGN OF A CAR,
- AN OBJECT IS A PARTICULAR INSTANCE OF THE CAR.

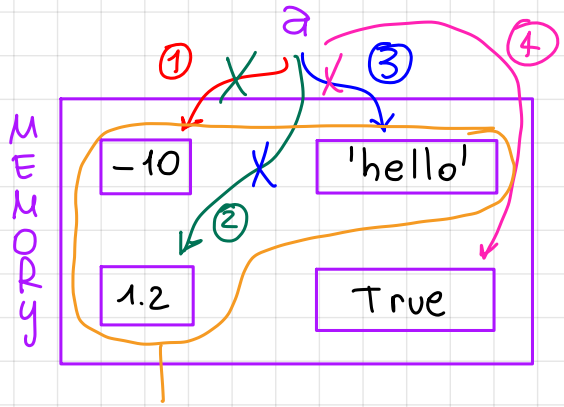
PYTHON USES DYNAMIC TYPING: IT DETERMINES THE TYPE OF THE OBJECT A VARIABLE REFERS TO WHILE EXECUTING THE CODE.
THE BUILT-IN FUNCTION type TELLS US THE TYPE OF A VARIABLE.

```
[ ]: a = -10
     type(a)
```

```
[ ]: a = 1.2
     type(a)
```

```
[ ]: a = 'hello'
     type(a)
```

```
[ ]: a = True
     type(a)
```



PERIODICALLY THE IPython GARBAGE COLLECTOR FREES UNUSED MEMORY

TO READ A VALUE FROM THE KEYBOARD AND STORE IT IN A VARIABLE WE USE THE FUNCTION input THAT RETURNS A STRING: TO CONVERT IT INTO A NUMBER WE USE int OR float.

```
[ ]: b = int(input('Insert a number >'))
     print('The number is', b)
```

PROGRAMMING

A **PROGRAM** IS A SEQUENCE OF INSTRUCTIONS THAT CONTROL HOW A COMPUTER PURSUES A PARTICULAR TASK BY IMPLEMENTING AN **ALGORITHM**: INSTRUCTIONS ARE SPECIFIED THROUGH A HIGH-LEVEL PROGRAMMING LANGUAGE.

IN OUR CASE, A PROGRAM IS A PYTHON LANGUAGE DESCRIPTION OF AN **ALGORITHM**.

AN **ALGORITHM** IS A PROCEDURE FOR SOLVING A PROBLEM IN TERMS OF:

1. THE **ACTIONS** TO EXECUTE,
2. THE **ORDER** IN WHICH THESE ACTIONS EXECUTE.

ALGORITHMS CAN BE DESCRIBED THROUGH:

- **PSEUDOCODE**: AN INFORMAL ENGLISH-LIKE LANGUAGE FOR "THINKING OUT" ALGORITHMS;
- **FLOWCHART**: A GRAPHICAL REPRESENTATION OF AN ALGORITHM;
- **HIGH-LEVEL PROGRAMMING LANGUAGE**: E.G, PYTHON, C, C++, JAVA AND MANY OTHERS.

THE FAMOUS **BÖHM-JACOPINI THEOREM** (1966) STATES (BOTH ITALIANS!!) THAT ALL PROGRAMS CAN BE WRITTEN USING ONLY 3 FORMS OF CONTROL: SEQUENCE EXECUTION, CONDITIONAL STATEMENT AND ITERATION STATEMENT.

IN PYTHON:

- CONDITIONAL STATEMENT:
  - if STATEMENT
  - if ... else STATEMENT
  - if ... elif ... else STATEMENT
- ITERATION STATEMENT:
  - while STATEMENT
  - for STATEMENT

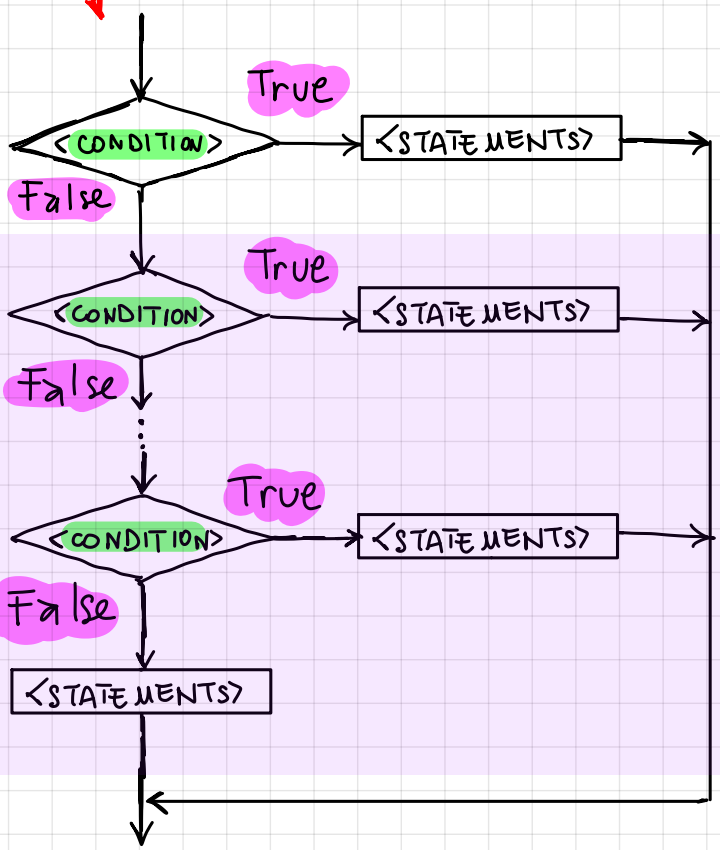
CONDITIONAL STATEMENT

PSEUDOCODE

```
if <CONDITION>:
    <STATEMENTS>
elif <CONDITION>:
    <STATEMENTS>
    .
    .
    .
elif <CONDITION>:
    <STATEMENTS>
else
    <STATEMENTS>
```

OPTIONAL

FLOWCHART



IMPORTANT: INDENTATION MAKES THE DIFFERENCE AS IT IDENTIFIES BLOCKS OF CODE CALLED SUITES.

WHAT WE WRITE IN <condition> IS A BOOLEAN EXPRESSION THAT CAN BE EITHER True OR False.

COMPARISONS: >, <, >=, <=, ==, !=

LOGICAL CONNECTIVES: not, and, or

PAY ATTENTION: = IS AN ASSIGNMENT, WHILE == IS THE EQUALITY COMPARISON!

EXAMPLE CONSIDER THE VARIABLE b

```
[]: if b < 50:
        print('The number is less than 50')
```

```
[]: if b > 50:
        print('The number is greater than 50')
    else:
        print('The number is less than or equal to 50')
```

```
[]: if b > 50:
        print('The number is greater than 50')
    elif b < 50:
        print('The number is less than 50')
    else:
        print('The number is equal to 50')
```

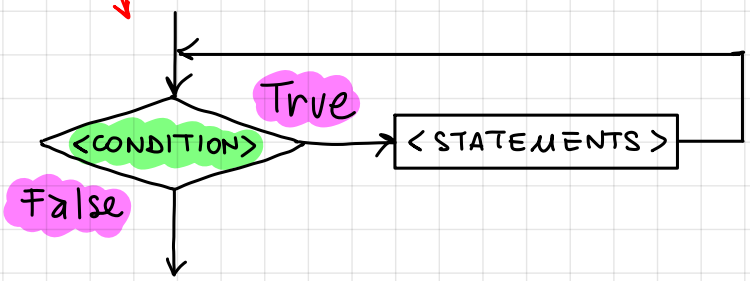
while STATEMENT

IT IS USED TO REPEAT ONE OR MORE ACTIONS WHILE A CONDITION REMAINS True

PSEUDOCODE

```
while <CONDITION> :
    <STATEMENTS>
```

FLOWCHART



EXAMPLE FIND THE LARGEST POWER OF 3 WHICH IS LESS THAN OR EQUAL TO 100

```
[]: product = 3

    while product <= 100:
        product = product * 3

    product
```

TO WRITE LESS:
└ product *= 3

↑ THE RESULT IS 243 BUT IT SHOULD BE 81!
THE CYCLE IS ENTERED WITH 81 AND SHOULD BE STOPPED BEFORE MULTIPLYING AGAIN BY 3!

AUGMENTED ASSIGNMENTS: TO WRITE LESS, WE USE

a += 3	EQUIVALENT	TO	a = a + 3
b -= 3	//	//	b = b - 3
c *= 3	//	//	c = c * 3
d **= 3	//	//	d = d ** 3
e /= 3	//	//	e = e / 3
f //= 3	//	//	f = f // 3
g %= 3	//	//	g = g % 3

WE NEED TO CHANGE THE CONDITION TO BE SURE THAT MULTIPLYING product BY 3 WE STILL ARE ≤ 100.

[ ]:
```
product = 3

while product * 3 <= 100:
    product *= 3

product
```

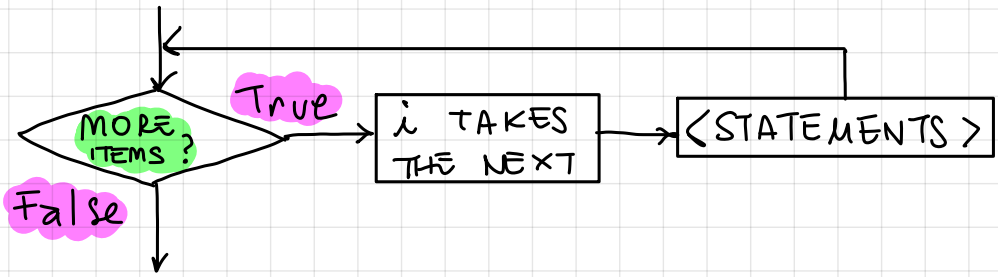
for STATEMENT

IT IS USED TO REPEAT AN ACTION OR SEVERAL ACTIONS FOR EACH ITEM IN AN ITERABLE SEQUENCE, LIKE A STRING, A LIST AND SO ON. WE WILL SEE LISTS IN NEXT LECTURES.

PSEUDOCODE

```
for i in <ITERABLE SEQUENCE>:
    <STATEMENTS>
```

FLOWCHART



ITERATION FROM range

INCLUDED EXCLUDED

```
for k in range(0, 5):
    <STATEMENTS>
```

k = 0, 1, 2, 3, 4

INCLUDED EXCLUDED NEGATIVE STEP MEANS BACKWARD

```
for k in range(4, -1, -1):
    <STATEMENTS>
```

k = 4, 3, 2, 1, 0

ITERATION FROM A LIST

```
items = [10, 20, 30, 40]
        0   1   2   3
for i in items:
    <STATEMENTS>
```

THE ORDER OF ITEMS MATTERS AND ITEMS ARE MODIFIABLE (A LIST IS A MUTABLE OBJECT)

THE ELEMENTS OF A LIST ARE INDEXED WITH INTEGER NUMBERS STARTING WITH 0.

THE FUNCTION len RETURNS THE LENGTH OF A LIST

SO, TO VISIT A LIST WE CAN ALSO WRITE

```
          0   1   2   3
list = [10, 20, 30, 40]
n = len(list)
for k in range(0, n):
    print(list[k])
```

← CREATES A LIST

← GETS THE LENGTH OF THE LIST

COMMENTS

WE CAN ADD SINGLE LINE COMMENTS STARTING A LINE WITH #. COMMENTS ARE IMPORTANT TO EXPLAIN WHAT WE ARE DOING.

```
[]: # This is a comment
```

RECAP ON CONTROL STATEMENTS

- CONDITIONAL STATEMENT:
  - if STATEMENT
  - if ... else STATEMENT
  - if ... elif ... else STATEMENT

  } TO DECIDE IF TO DO AN ACTION DEPENDING ON A CONDITION

- ITERATION STATEMENT:
  - while STATEMENT
  - for STATEMENT

  } TO REPEAT AN ACTION UNTIL A STOP CONDITION IS MET

EXAMPLE TAKE THE SUM OF NUMBERS INSERTED FROM THE KEYBOARD, UNTIL -1 IS INSERTED.
WE NEED A while ← WE DON'T KNOW THE NUMBER OF ITERATIONS IN ADVANCE

```
[]: # Store the partial sum
    tot = 0

    # Read a number from the keyboard
    n = int(input('Insert a number:'))
```

```
while n != -1:
    tot += n
    # Read a number from the keyboard
    n = int(input('Insert a number:'))
print('The total is:', tot)
```

IT MEANS tot = tot + n

EXAMPLE: COMPUTATION OF THE FACTORIAL OF A NUMBER n:

$$n! = \begin{cases} 1 & \text{IF } n = 0, 1 \\ n \cdot (n-1) \cdot \ldots \cdot 1 & \text{OTHERWISE} \end{cases}$$

WE CAN DO IT USING:

- if ... else
- for ← WE ALREADY KNOW THE NUMBER OF REPETITIONS

```
[]: # Read a number from the keyboard
n = int(input('Insert a number:'))
factorial = 1

if n == 0 or n == 1:
    factorial = 1
else:
    for k in range(n, 0, -1):
        factorial *= k

print('Factorial =', factorial)
```

START, EXCLUDED, 1 STEP BACKWARD

← k = n, n-1, ..., 1