

+
•
◦

Python Hackathon

María Navas-Loro



Outline

0. Introduction and setup

1. Python Basics

- Types and variables
- Operations
- Strings

2. Python data structures

- Tuples
- Lists
- Dictionaries

3. Python fundamentals

- Conditionals
- Loops

4. Python functions and libraries

- Functions
- Read/Write
- Libraries

5. Hands-on!

0. Introduction and setup

About me

María Navas Loro

Professor at Technical University of Madrid

Working in Artificial Intelligence:

- Natural Language Processing
- Data Science
- Machine Learning and Deep Learning
- Ethics of AI

You can reach me at any point at
navasloro@gmail.com



You reported different Python levels! So...

If you have no previous Python experience:

- We will start from zero, no worries!

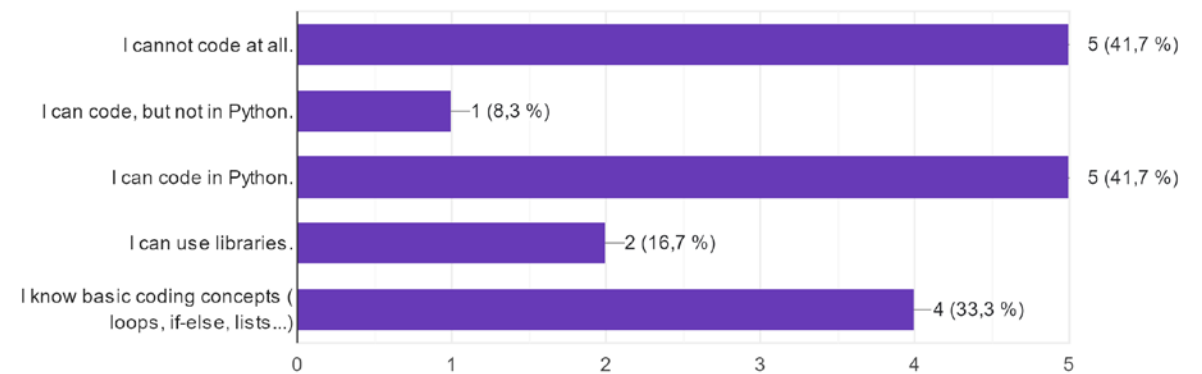
If you have a basic knowledge but you do not feel confident:

- Join us and help your classmates!

In any case:

- It is normal for programming to be difficult at first.
- Please ask if you do not understand any term
- If you are interested in a specific topic, ask!

What do you know about programming (tick all that apply, or the first one if you can't code at all)?
12 respuestas



For Python experts

If you consider you have some Python knowledge, feel free to check the slides and go directly to section “5. Hands-on”. There you will find:

- Different project proposals, so you can try to build some code from scratch.
 - Notebooks from different domains ready to be executed and explored.
 - You can also learn how to use pandas, a library used for data analysis, right after the notebooks. I will not explain that part (no time!), but feel free to ask anything!
- * I will solve any doubts you might have while not explaining the basics to your classmates!

Algorithms

—

What is an algorithm?

What is an algorithm?

“An ordered and finite
set of operations to find
the solution to a
problem.”



REAL ACADEMIA ESPAÑOLA



Can you think of an example of a day-to-day algorithm?

Algorithms everywhere

← desde trabajo (Universidad Politécnica de Madrid ...
a: C. del Príncipe de Vergara, 156, 28002 Madrid

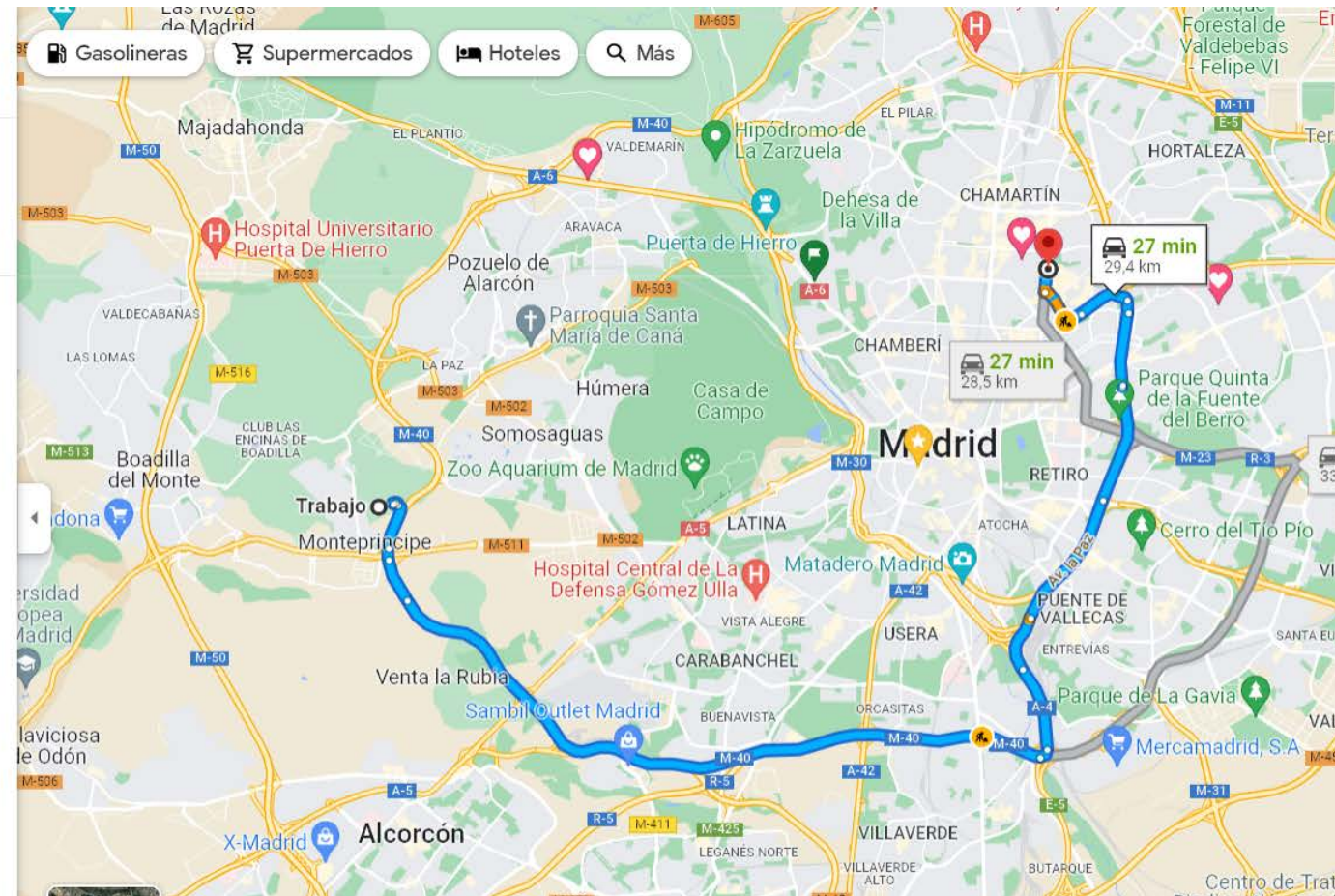
27 min (29,4 km)   

por M-40
La ruta más rápida debido al estado del tráfico

Universidad Politécnica de Madrid
Montegancedo
Universidad Politécnica de Madrid, Av. de Montepríncipe,
28223 Madrid

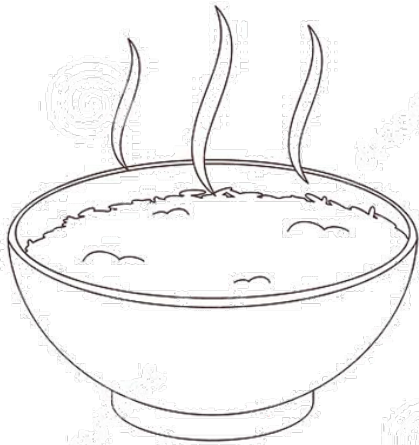
- > Sigue hasta Madrid.
3 min (2,4 km)
- > Sigue por M-40. Toma Av. la Paz y M-30 hacia
Avenida de América.
18 min (25,5 km)
- > Toma C. de Cartagena hacia C. del Príncipe de
Vergara.
6 min (1,5 km)

C. del Principe de Vergara, 156
28002 Madrid

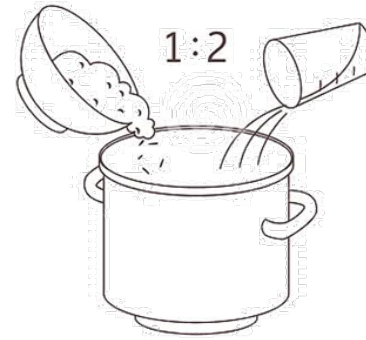


Algorithms everywhere

HOW TO COOK RICE



RINSE THE RICE



ADD WATER AND RICE
IN THE POT



ADD SALT AND OIL
OR BUTTER



COOK ON HIGH HEAT UNTIL
WATER STARTS TO BOIL

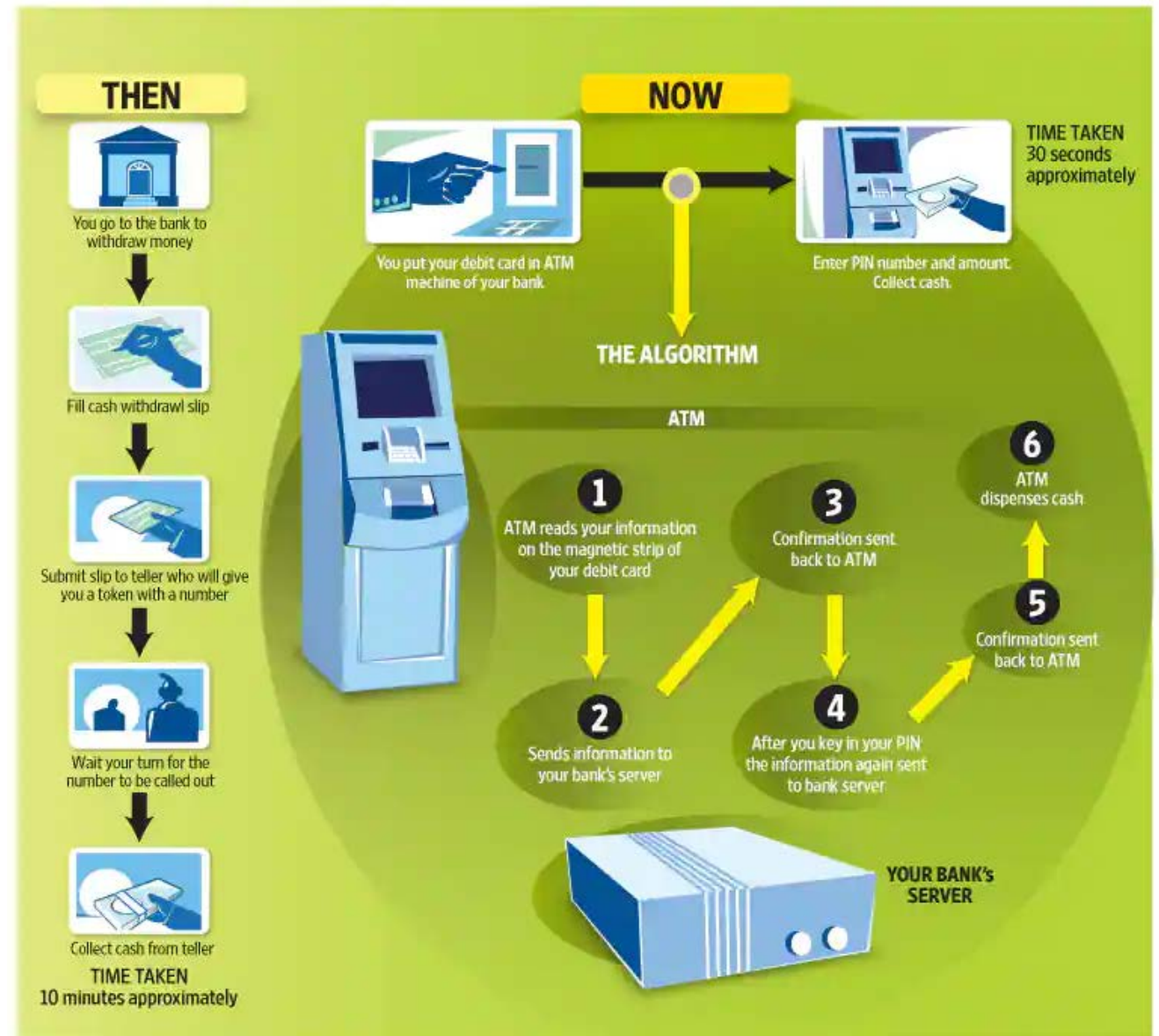


TURN THE HEAT DOWN
TO LOW



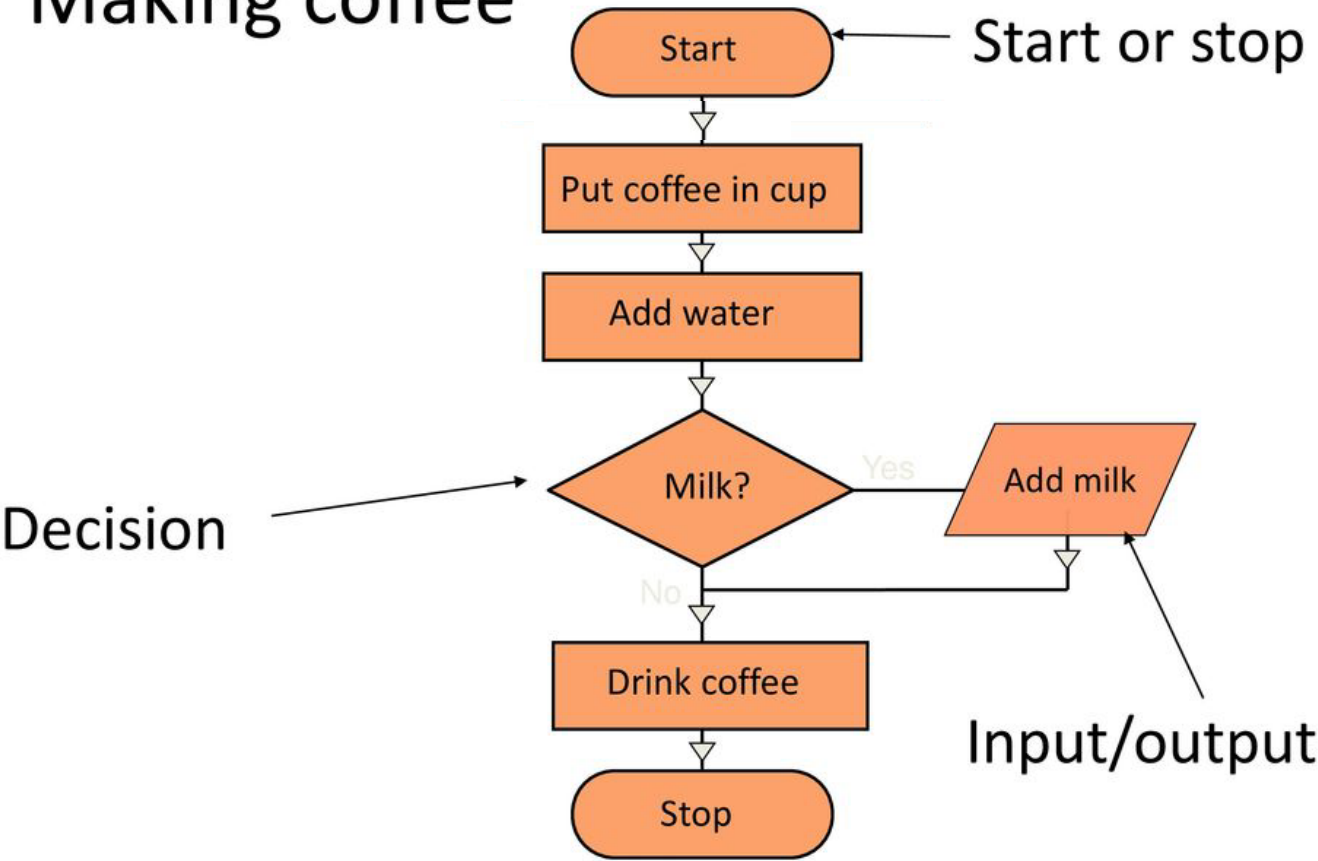
TURN THE HEAT OFF AND
LEFT FOR 10 MIN

Algorithms everywhere



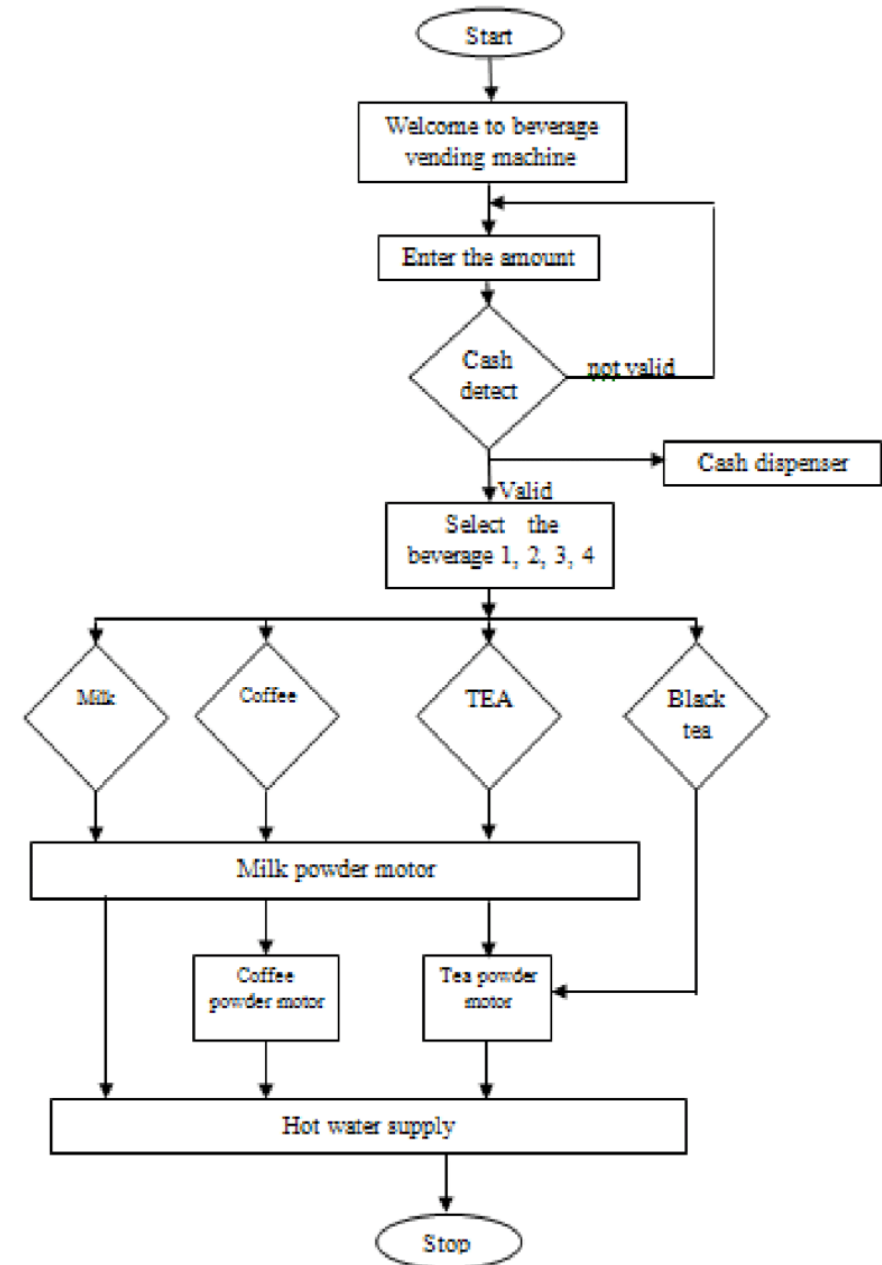
Flowchart

Making coffee

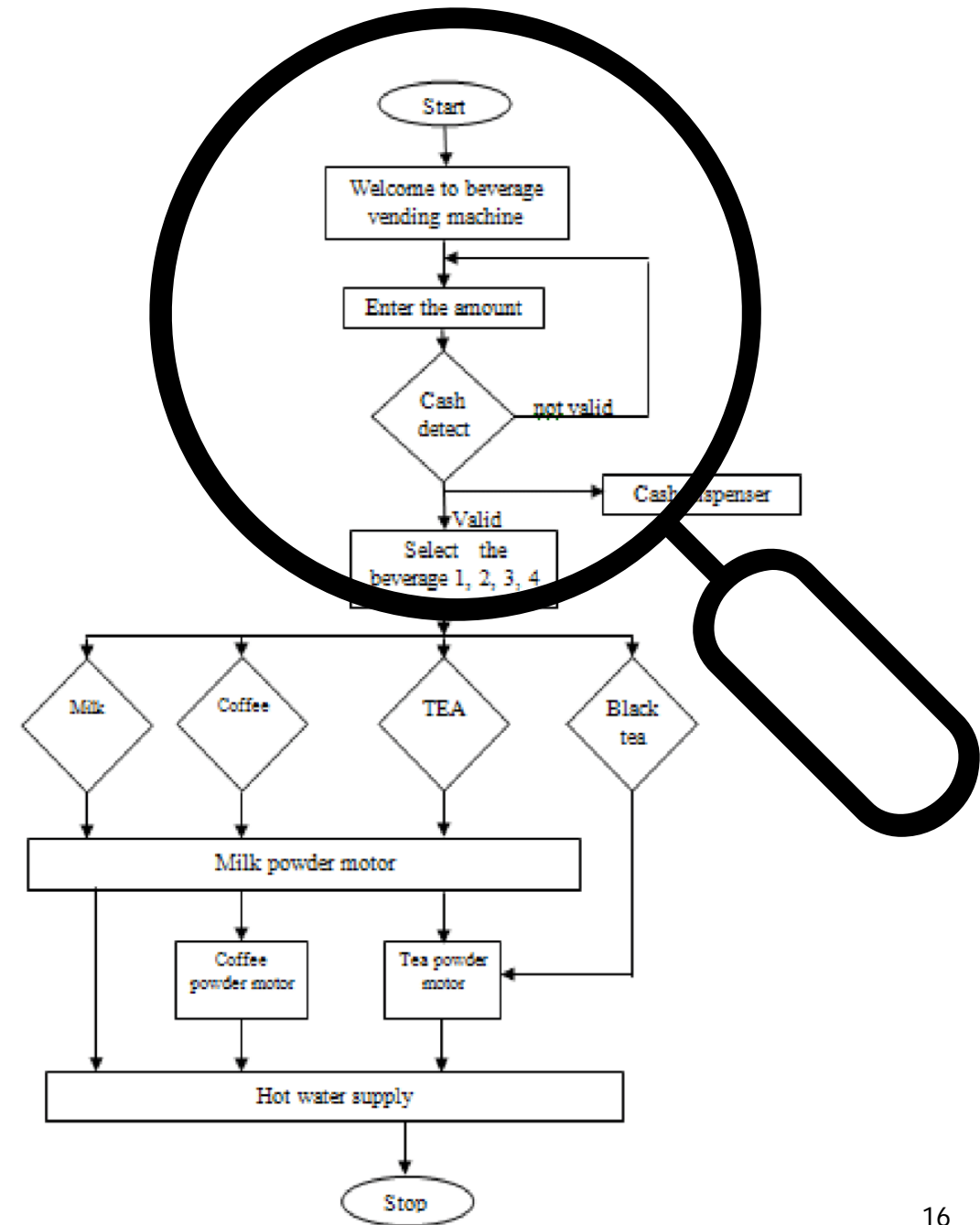
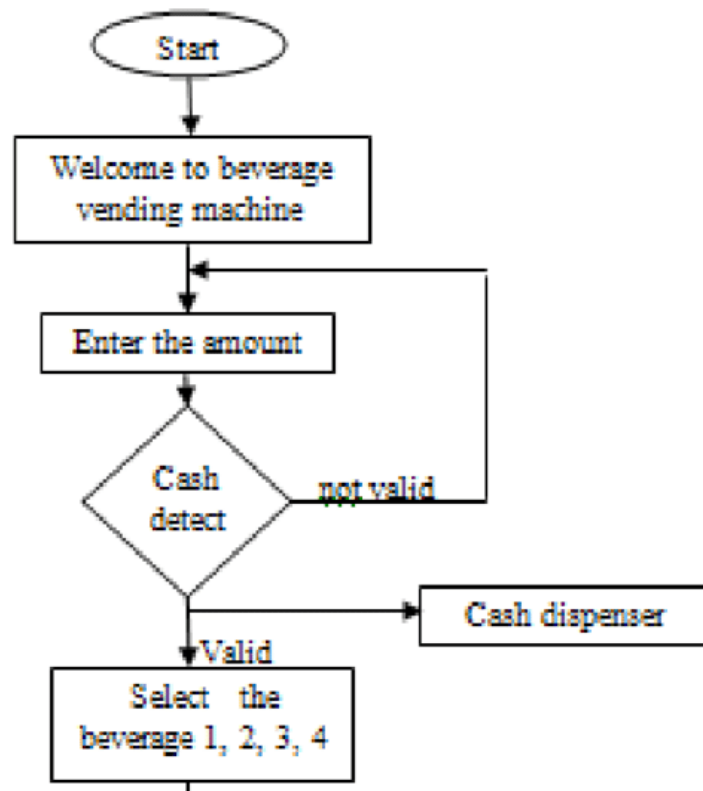


Adapted from
https://slideplayer.com/12977715/79/images/slide_1.jpg

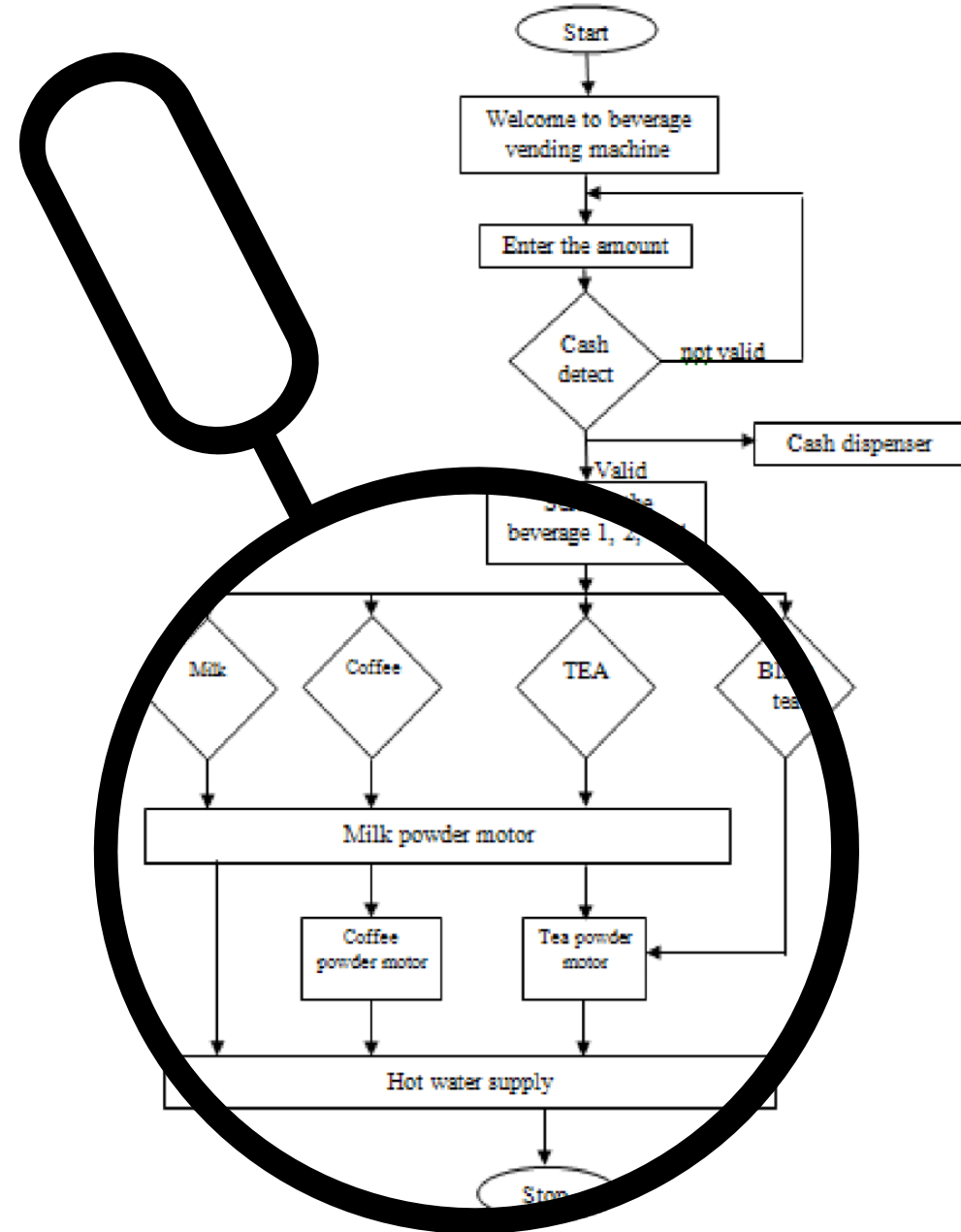
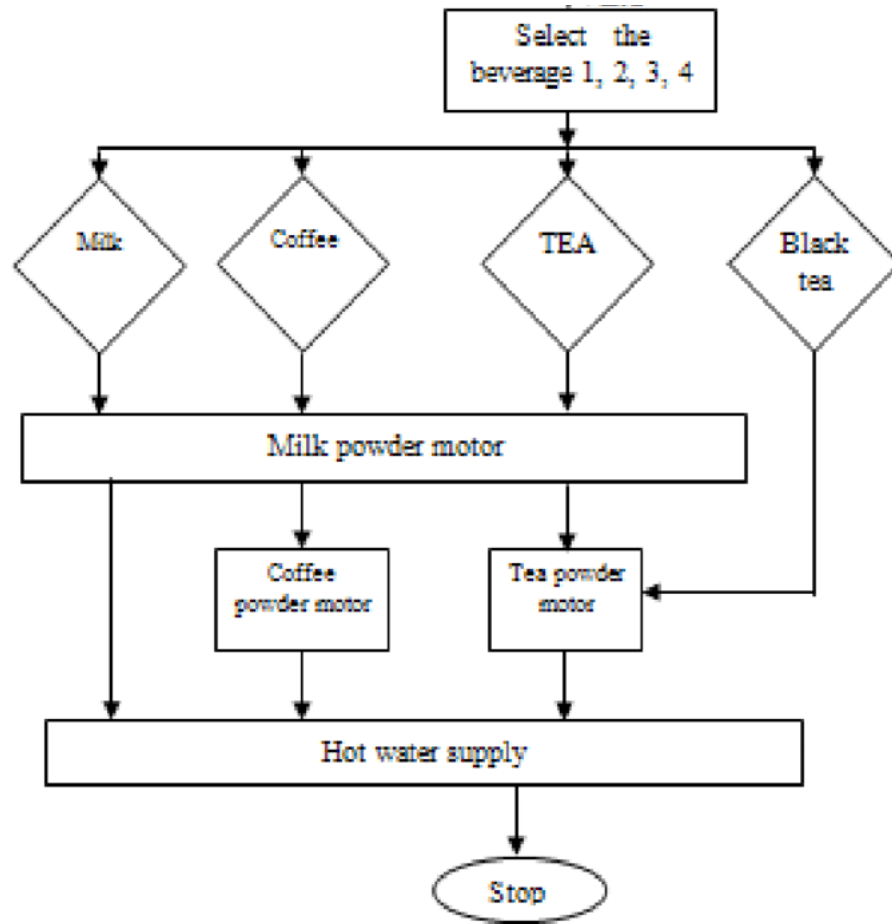
Example: vending machine



Example: vending machine



Example: vending machine



How can we automate all these?

Intro to Python

—

About Python

One of the most used programming languages.

Most popular language in online search.

Used in multiple use cases:

- To create Instagram
- To interpret data from the LHC at CERN
- For drone flight software on Mars
- At J.P. Morgan Chase
- Deep Learning: have you heard about DALL-E, ChatGPT...?
- Video games...

Python Installation

—

Download Python

Windows

<https://www.python.org/downloads/windows/>

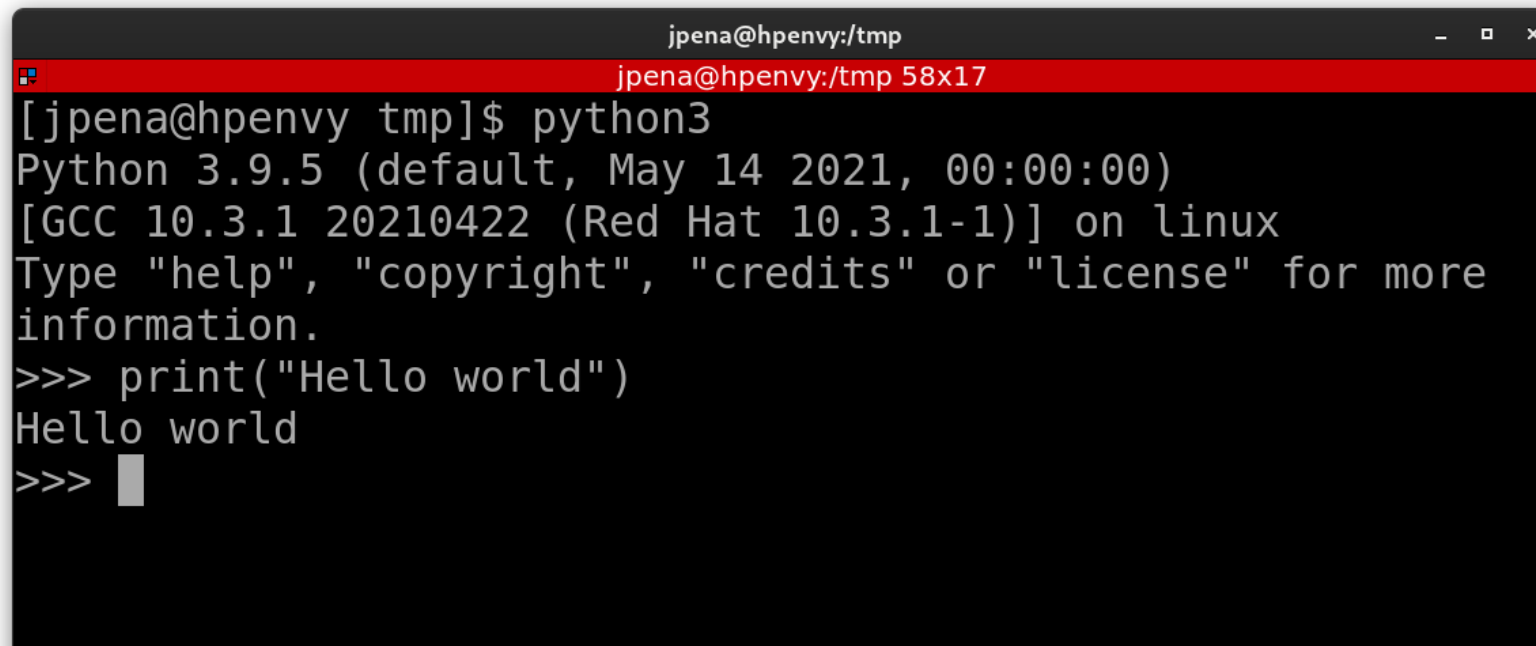
Download the last version (3.10.x), *Windows Installer (32-bit)* or *Windows Installer (64-bit)* depending on your computer

MacOS

<https://www.python.org/downloads/mac-osx/>

Download the last version (3.10.x), *macOS 64-bit Intel installer* or *macOS 64-bit universal2 installer* depending on your computer

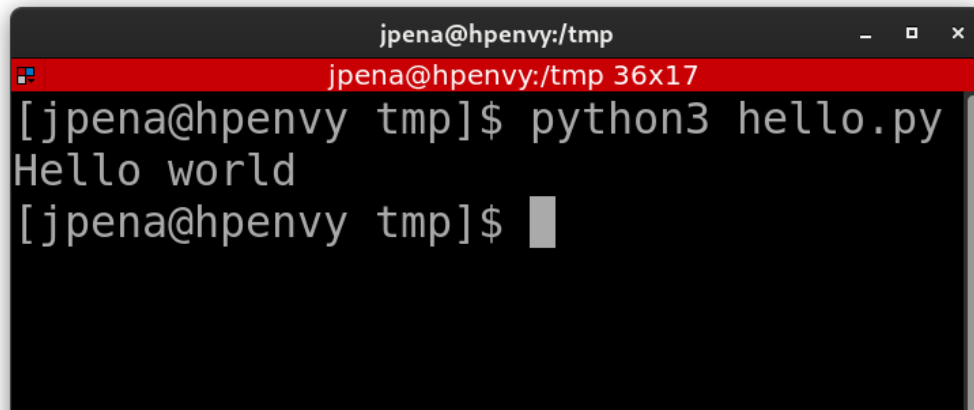
Executing code from the interpreter (command line, CMD...)

A terminal window with a dark background and a red title bar. The title bar contains the text 'jpena@hpenvy:/tmp' and window control icons. Below the title bar, the terminal shows the command 'python3' being executed. The output displays the Python version (3.9.5), the default installation path, the GCC version (10.3.1), and the operating system (linux). It then prompts the user to type 'help', 'copyright', 'credits', or 'license' for more information. Finally, the user enters the command 'print("Hello world")', and the terminal outputs 'Hello world'. The prompt '>>>' is followed by a cursor.

```
jpena@hpenvy:/tmp
jpena@hpenvy:/tmp 58x17
[jpena@hpenvy tmp]$ python3
Python 3.9.5 (default, May 14 2021, 00:00:00)
[GCC 10.3.1 20210422 (Red Hat 10.3.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> print("Hello world")
Hello world
>>> █
```

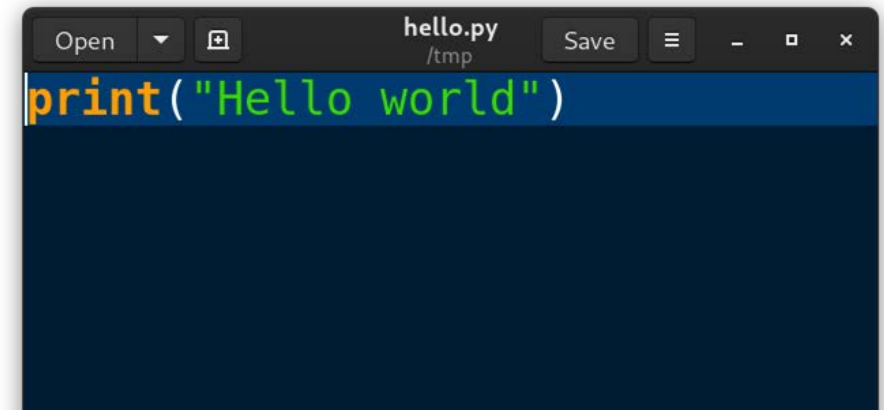
Executing code from a file in the command line

- .py is the extension of Python files
- Similar to .docx for Microsoft Word or .pdf for Adobe Reader
- When you see a .py file, it means it contains Python code!



```
jpena@hpenvy:/tmp
jpena@hpenvy:/tmp 36x17
[jpena@hpenvy tmp]$ python3 hello.py
Hello world
[jpena@hpenvy tmp]$
```

A terminal window with a dark background. The title bar shows 'jpena@hpenvy:/tmp'. The prompt is 'jpena@hpenvy:/tmp 36x17'. The user enters the command 'python3 hello.py' and the output 'Hello world' is displayed. The prompt returns to 'jpena@hpenvy tmp\$'.



```
hello.py
/tmp
print("Hello world")
```

A code editor window with a dark blue background. The title bar shows 'hello.py' and '/tmp'. There are buttons for 'Open', 'Save', and a menu icon. The code 'print("Hello world")' is visible, with 'print' in orange and the string in green.

No need to do this for now!

- Not necessary to install Python!
- Today we will use Google Colab.

<https://colab.research.google.com/?hl=en>

- Just an email account needed if you want to save the code online.
- This way we can do everything online without installation 😊
- Let's see how!

Google Colab

—

Google Colab(oratory)

- <https://colab.research.google.com/?hl=en>
- It uses a file type named .ipynb (we call them “notebooks”)
- Very visual, allows to integrate text and images
- You can share code with your classmates
- A lot of libraries installed by default (code we can call)
- Commonly used, you will find many examples online
- Limitations
 - Google is not private
 - Free usage (CPU, memory, disk space...)

Basic usage (I)

ExamplesRecentGoogle DriveGitHubUpload

Filter notebooks

Title

Last opened ▾First opened ▾

Welcome To Colaboratory6:39 PMJuly 12

Untitled0.ipynb6:38 PM6:38 PM

New notebook

Cancel

Basic usage (II)

Rename
the
notebook

The screenshot shows the Google Colaboratory web interface. At the top, the browser tab is titled 'Untitled - Colaboratory' and the address bar shows the URL 'https://colab.research.google.com/drive/1jNChpmjkUDEfHw001_5VD--CGs2PCbmC?authuser=1'. The interface includes a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu bar, there are buttons for '+ Code' and '+ Text'. The main workspace contains a single code cell with a play button icon on the left and a vertical cursor. On the right side of the workspace, there is a status bar showing 'RAM' and 'Disk' usage with progress bars, and a button labeled 'Editing'. At the bottom of the workspace, a status bar indicates '0s completed at 6:41 PM'. Three red arrows point to specific elements: one to the Colaboratory logo in the top left, one to the play button in the code cell, and one to the RAM and Disk usage indicators in the top right.

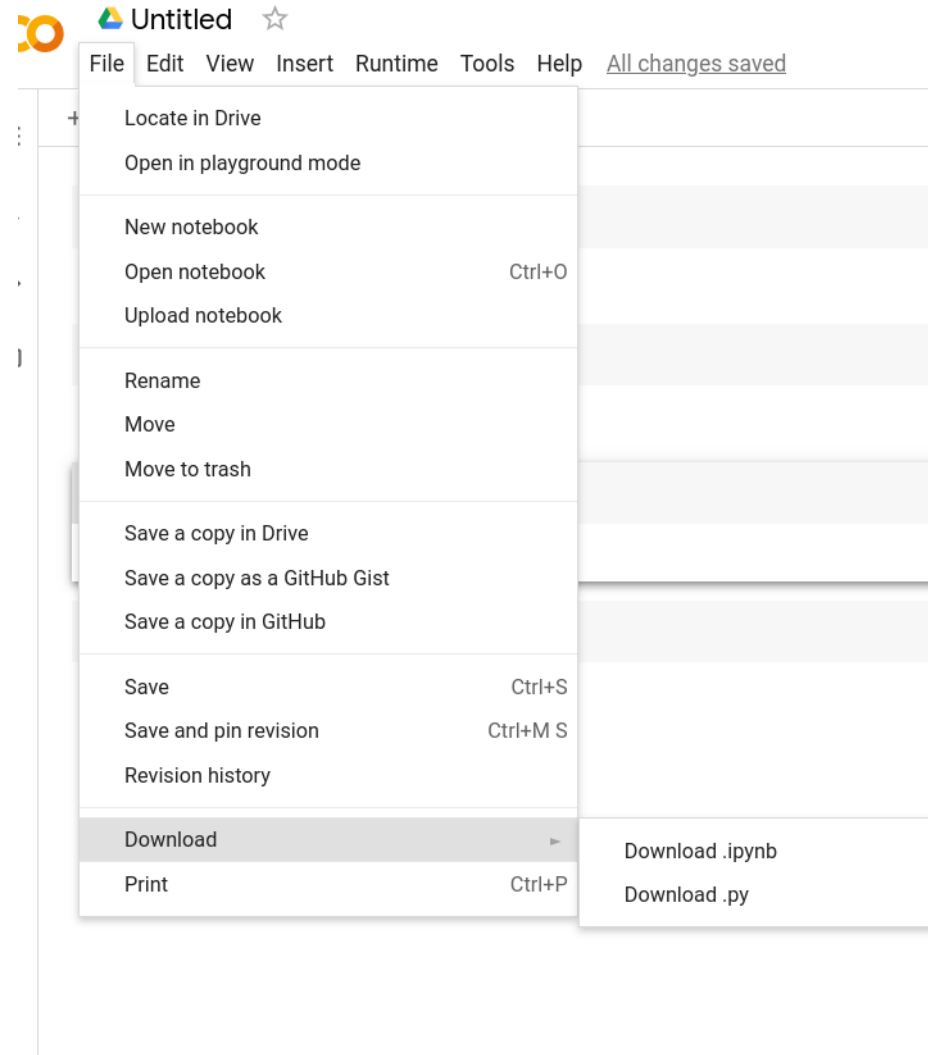
Execute cell

Edit cell

RAM and disk usage

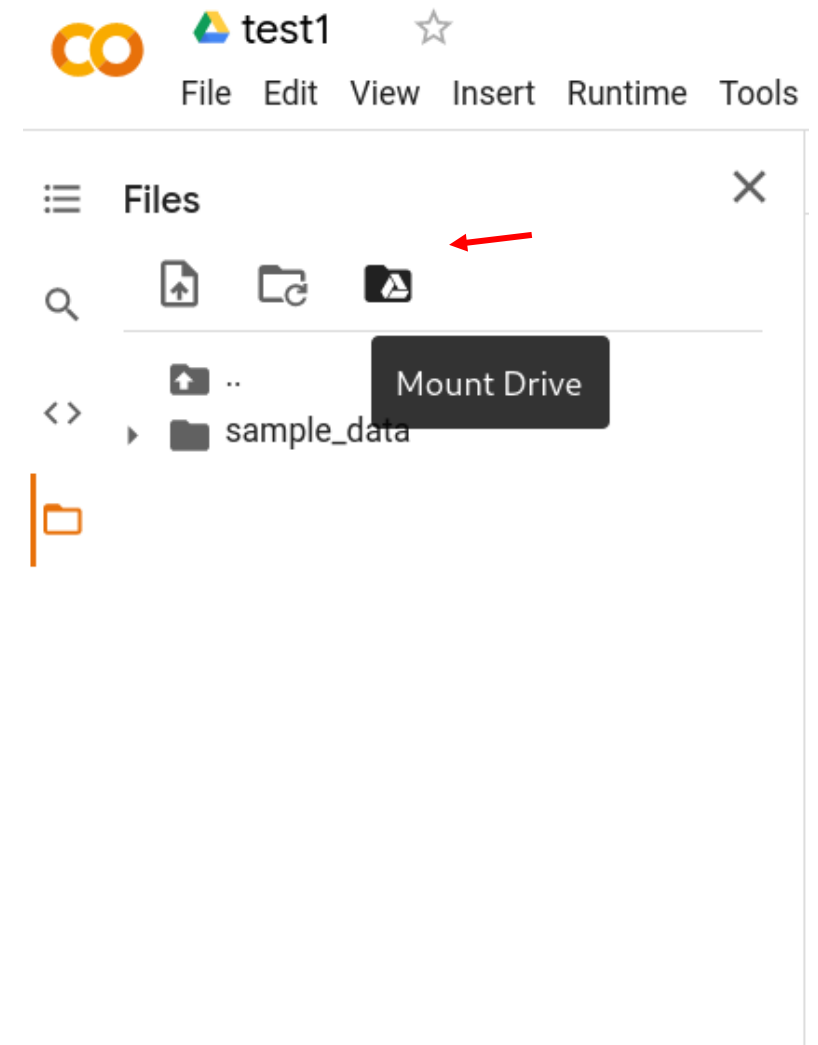
Basic usage (III)

- We can save the file in:
 - Google Drive
 - Locally
 - Other options



Basic usage (IV)

- If logged, we can “connect” (mount) it to our Google Drive, being the path `/content/drive/MyDrive`
- Useful to upload data, for instance.



Let's try!

1. Rename the notebook as “My notebook”.
2. Edit the first cell adding the following:

```
print(“Hello world!”)
```
3. Run the cell (the “play button” on the left).
4. See the result! You can try changing the text inside of print and re-run.
5. Download the file and check it is now in your computer (My notebook.ipynb”).



1. Python Basics

- Types and variables
- Operations
- Strings

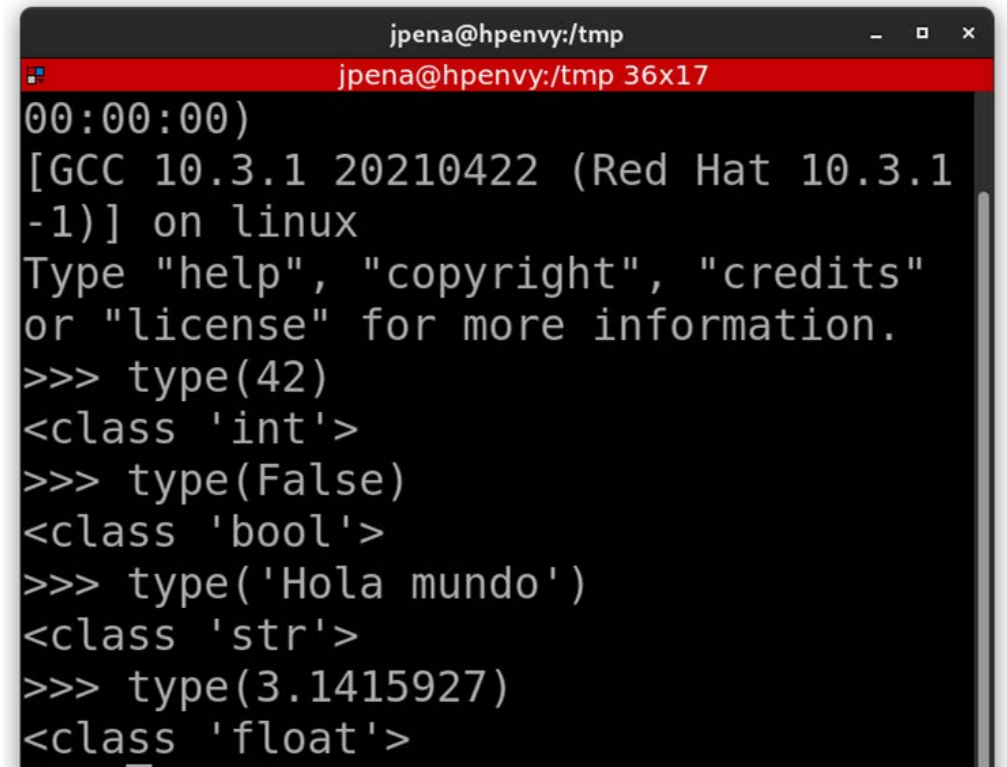
Types and variables

—

Data Types in Python

- Basic datatypes store a single value:
 - Integer
 - `int`
 - Example: 42
 - Floating point (this is, decimal numbers)
 - `float`
 - Example: 42.42
 - Text strings (between " or ")
 - `str`
 - Example: 'Hola mundo'
 - Boolean values
 - `bool`
 - Example : False

To check the datatype: `type(value)`

A terminal window with a dark background and a red title bar. The title bar contains the text 'jpena@hpenvy:/tmp' and 'jpena@hpenvy:/tmp 36x17'. The terminal shows the output of a Python shell session. It starts with a prompt '00:00:00)' followed by system information: '[GCC 10.3.1 20210422 (Red Hat 10.3.1-1)] on linux'. Then it prompts the user to type 'help', 'copyright', 'credits', or 'license'. The user enters '>>> type(42)', and the output is '<class \'int\'>'. The user enters '>>> type(False)', and the output is '<class \'bool\'>'. The user enters '>>> type(\'Hola mundo\')', and the output is '<class \'str\'>'. The user enters '>>> type(3.1415927)', and the output is '<class \'float\'>'.

```
jpena@hpenvy:/tmp
jpena@hpenvy:/tmp 36x17
00:00:00)
[GCC 10.3.1 20210422 (Red Hat 10.3.1-1)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> type(42)
<class 'int'>
>>> type(False)
<class 'bool'>
>>> type('Hola mundo')
<class 'str'>
>>> type(3.1415927)
<class 'float'>
```

* The special word None represents the null value (not important for today)

Variables in Python

- A variable is a data warehouse, a place to store a value (like in maths!)
 - `a = 1`
 - `a = 'Testing one two three'`
 - `a = True`
 - `a = 3.1415927`
- The result of any operation can be stored in a variable
 - `a = 1 + 3`
 - `a = a + 1`
- We can assign to one variable the value of another one
 - `a = 3`
 - `b = a`

Naming variables in Python

- **Basic rules**
 - Start with letter or _
 - Contain letters, numbers, and _
 - Case-sensitive/lowercase
 - They can't use reserved words of the language
- **Advice**
 - Use meaningful names
 - Suitable length
 - “*Snake case*” for names with multiple words
 - Ex: `average_result`, `total_amount`

Operations

—

Math operations

- Usual ones (precedence is the same as in maths, parenthesis also possible)
 - +
 - -
 - *
 - ** (power)
 - /
 - // (Integer Division)
 - % (Module)
- We can always assign the result to a variable
 - $a = 9 / 3$
- Operation & Assignment
 - $a += 1 \rightarrow$ this means $a = a + 1$, or “add one to a”
 - $b *= 5$

Logical Operations

Apply to bool values

and

value1	value2	value1 and value2
True	True	True
True	False	False
False	True	False
False	False	False

or

value1	value2	value1 or value2
True	True	True
True	False	True
False	True	True
False	False	False

not

value1	not value1
True	False
False	True

Strings in Python

—

Define Strings in Python

- We can define strings in Python with
 - 'This is a string' (single quotation marks)
 - "This too" (double quotes)
- A string can contain any type of character
 - Letters
 - Numbers
 - Special Characters
- It's easy to tell the length of a string
 - `my_text = 'Hello World'`
 - `len(my_text)`

Strings as Ordered Sequences

- Strings are an **ordered and immutable sequence** of characters

We can get the character that's in a position

- `my_text = "Hola mundo"`
- `my_text[0]`

Character	H	o	l	a		m	u	n	d	o
Index	0	1	2	3	4	5	6	7	8	9

Creating substrings (slicing)

	my_text[:4]							my_text[-2:]		
Character	H	o	l	a		m	u	n	d	o
Index	0	1	2	3	4	5	6	7	8	9
	my_text[0:2]						my_text[6:]			

String Operations in Python

- **Concatenate strings**
 - `text1 = "Hola"`
 - `text2 = "mundo"`
 - `text1 + " " + text2`
- **Multiply strings**
 - `text = 'nana'`
 - `text * 6`
- **We can combine operations**
 - `text1 * 3 + ' ' + text2`
- **Special sequences start with **
 - `\n` (newline)
 - `\t` (tab)
 - To include the symbol `'\'`, we duplicate it (`\\`)

Methods for Handling Strings

- `upper()`
 - `text = 'example'`
 - `text.upper()`
- `lower()`
 - `text = 'EXAMPLE'`
 - `text.lower()`
- `replace()`
 - `text = "I want potatoes"`
 - `text.replace('potatoes', 'chips')`
- `find()`
 - `text = "Somewhere over the rainbow"`
 - `text.find('over')`

All methods return a **new string**, they do not modify the current one



Formatting Strings with Variables

- We can create strings by combining variable values of any type
- Methods:
 - With %
 - `var1 = 'hola'`
 - `var2 = '%s mundo' % var1`
 - `var3 = "%s %s mundo" % (var1, var1)`
 - Using `.format()`
 - `time = 'afternoon'`
 - `name = 'María'`
 - `greeting = 'Good {}, my name is {}'.format(time, name)`
 - With `+`:
 - `var3 = var1 + " mundo"`

Let's start practicing!

https://colab.research.google.com/drive/1z9XYefkP0P7lhjt-IGdSEdsR1MjFp_sn?usp=sharing



Recap

—

What have we learned for now?

- How to use Colab
- How to Define Variables, and Basic Data Types in Python
- Basic Math Operations in Python
- How to Define and Handle Text Strings in Python

Some extra references/exercises

- *Swaroop C H: A Byte of Python*
 - Available online at: <https://python.swaroopch.com/>
- <http://openbookproject.net/thinkcs/python/english3e/>
- <https://docs.python.org/3/tutorial/interpreter.html>
- <https://docs.python.org/3/tutorial/introduction.html>
<https://holypython.com/beginner-python-exercises/exercise-3-python-data-types/>
 - (Exercise 4 too)

2. Python data structures

- Tuples
- Lists
- Dictionaries

Python data structures

- Lists
- Tuples
- Dictionaries
- Sets (not explained)

Tuples

—

Tuples

- An **ordered sequence** of elements
- We can use elements of the same type, or different
- They are defined in parentheses, and each element is separated by a comma.
- Examples
 - (15, 18, 29, 32)
 - ('one', 'two', 'three')
 - (1, 2, 3, 'catorce')
 - (True, 'False', 1)
- Very similar to strings, also immutable.
 - We can access individual elements:
 - `a = (1, 2, 3)`
 - `a[0]`
 - Also add tuples
 - `a = (1, 2, 3)`
 - `b = (4, 5, 6)`
 - `a + b`
 - Create a *slice*
 - `a = (1, 2, 3, 4)`
 - `a[0:2]`
 - Get its length
 - `a = (1, 2, 3, 4)`
 - `len(a)`

Lists

—

Lists

- An **ordered sequence** of elements
- We can use elements of the same type, or different.
- They are defined in brackets, and each element is separated by a comma.
- Examples
 - [15, 18, 29, 32]
 - ['uno', 'dos', 'tres']
 - [1, 2, 3, 'catorce']
 - [True, 'False', 1]

- Very similar to tuples... But mutable!
 - `a = [1, 2, 3, 4]`
 - `a[3] = 1`

Therefore:

- Lists are good to store data we want to be able to modify.
- Tuples are good if data should not be modified.

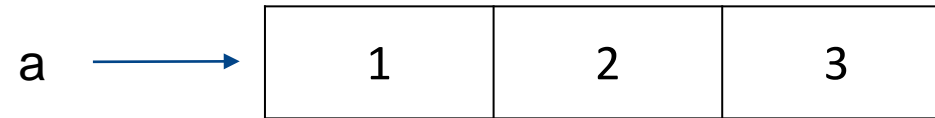


List: how to modify and operations

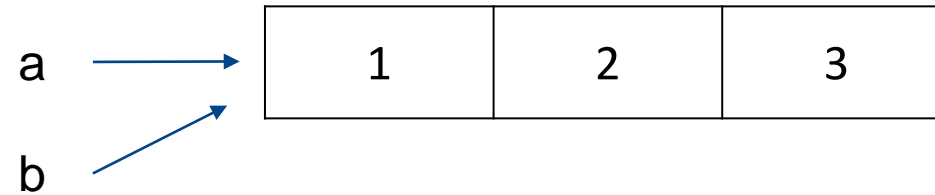
- **Extend a list**
 - `a = [1, 2, 3]`
 - `a.append(4)`
- **Join lists**
 - `a = [1, 2, 3]`
 - `b = [4, 5, 6]`
 - `a.extend(b)`
- **Delete an element**
 - `a = [1, 2, 3]`
 - `del(a[1])`
- **Sort a list:**
 - `a = [1, 5, 7, 1, 4, 9]`
 - `a.sort()`
- **Reverse a list:**
 - `a = [1, 2, 3, 4]`
 - `a.reverse()`
- **Insert an element in a certain position**
 - `a = [1, 2, 4]`
 - `a.insert(position, value)`

Aliasing in lists

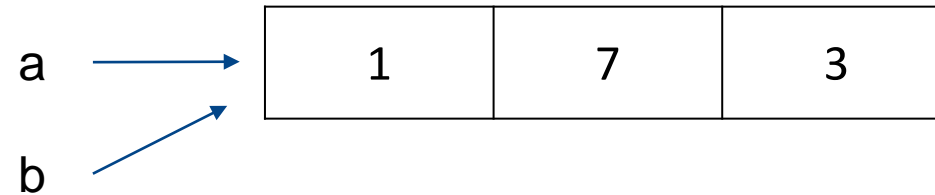
```
a = [1, 2, 3]
```



```
b = a
```



```
a[1] = 7  
print(b[1])
```



Dictionaries

—

Dictionary

- It is a collection of data stored in the form key:value
- Both keys and values can have different types
- We define them:
 - Between curly brackets
 - We declare key and value with :
 - We separate each element with ,

Example:

```
mydict = { 'one': 1, 'two': 2,
           'three': 3, 'four': 4,
           'five': 5, 'six': 6 }
```

Usefulness of dictionaries

- They allow you to associate data with a key
- The value of an element can also be a dictionary
- We can represent very complex structures of information

```
team = {  
    9: {'name': 'Pau Gasol',  
        'age': 30,  
        'height': 193},  
    6: {'name': 'Marc Gasol',  
        'age': 35,  
        'height': 191},  
}
```

team[9]['age']

Working with dictionaries

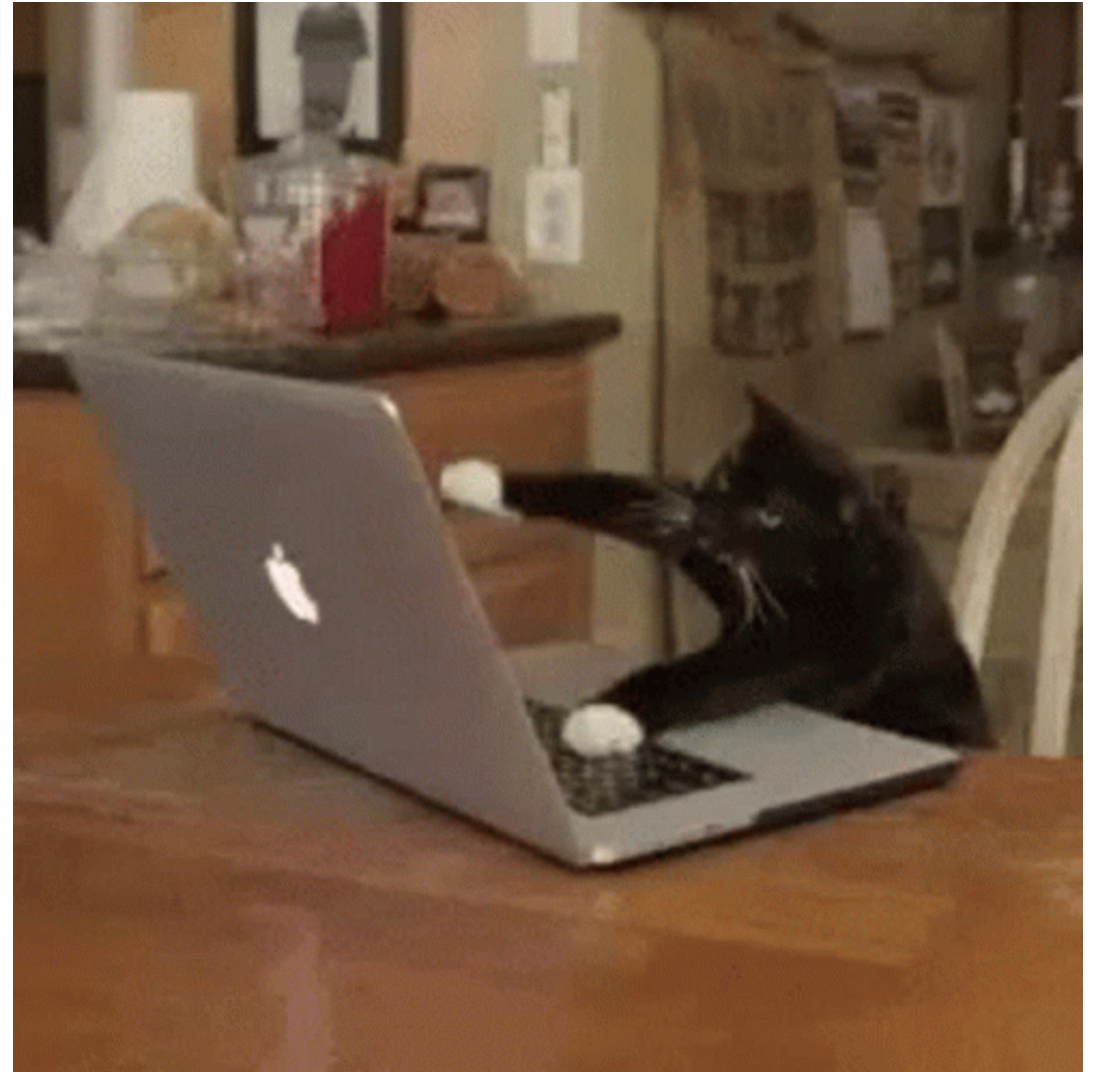
- To access an element, we use `variable['key']`
 - `a = { 'one' : 1, 'two' : 2 }`
 - `a['one']`
- Keys are **immutable**, but values aren't
 - `a['one'] = 11`
- We can add new key:value pairs, as well as delete existing ones
 - `del(a['one'])`
 - `a['three'] = 3`

Working with dictionaries (II)

- We can tell if a key is in the dictionary:
 - `a = { 'one' : 1, 'two' : 2 }`
 - `'four' in a`
 - Returns a value of type bool
- Get all the keys from the dictionary:
 - `a.keys()`
- Get all dictionary values:
 - `a.values()`

Let's practice!

https://colab.research.google.com/drive/1_B5aHQSxoxjnz0YZc1DP3XjjULBkg8ik?usp=sharing



Recap

—

What have we learned?

- **Tuple: Ordered sequence of items**
Immutable – (X , X , X)
- **List: Ordered sequence of items**
Mutable – [X , X , X]
- **Dictionaries: Collection of pairs **key:value****
{ key1 : value1 , key2 : value2 , key3 : value3 }
- They can all contain elements of different types

References and exercises

- <https://docs.python.org/3/tutorial/datastructures.html>
- <http://esparta.github.io/python-data-intro/core/data.html>
- <https://holypython.com/beginner-python-exercises/exercise-5-data-structures/>
 - (also from exercise 6 to 9)

3. Python fundamentals

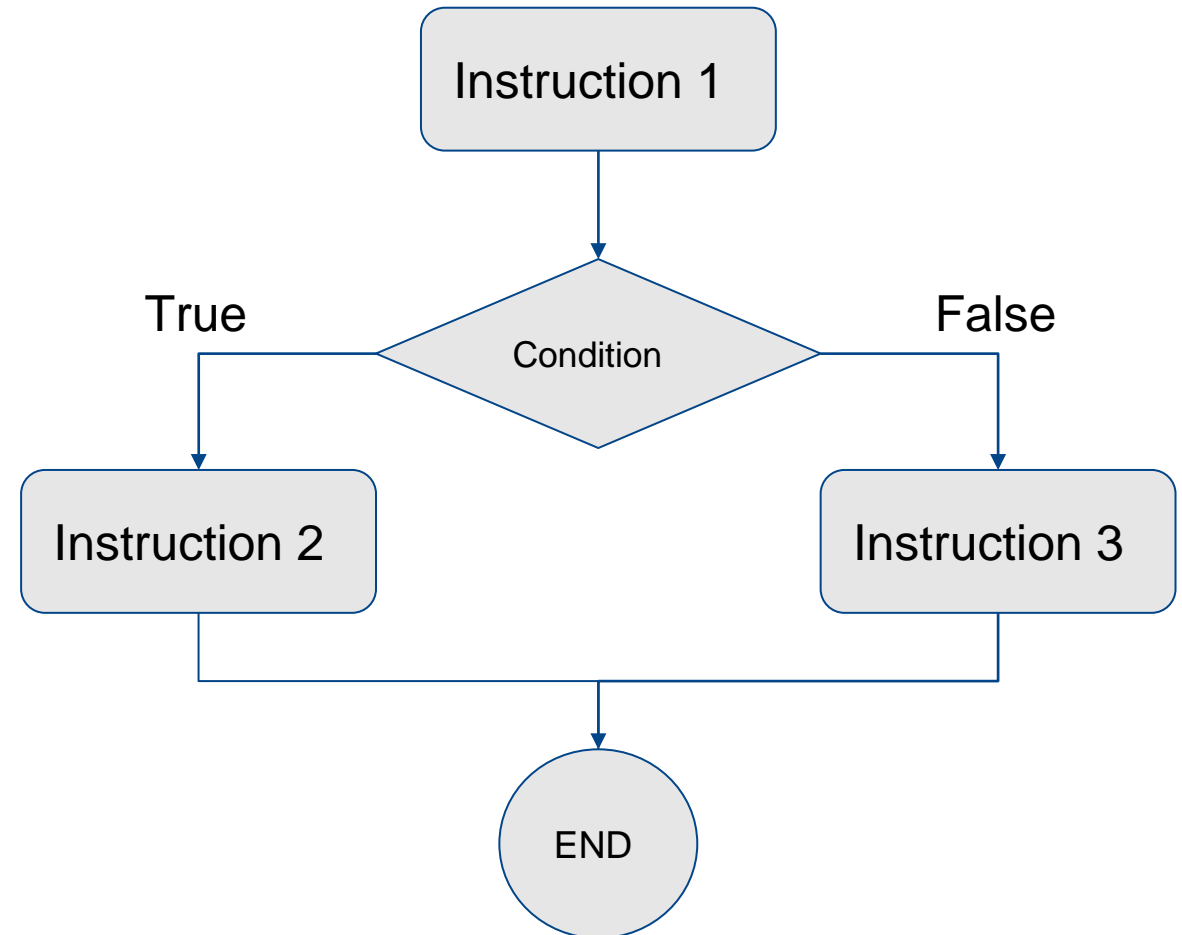
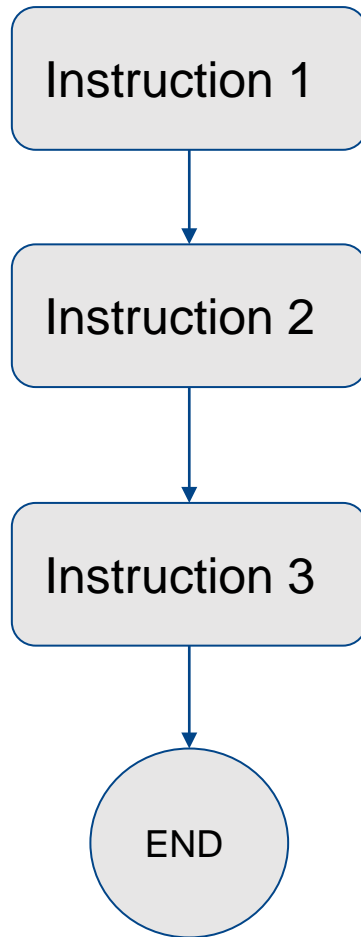
- Conditionals

- Loops

Conditionals

—

Code Blocks in Python



Python Code Blocks (II)

- They are marked with indentation (spaces or one tab)
- Anything that is at the same level of indentation is at the same level of Python code.
- The size of the indentation is arbitrary (4 spaces are recommended)

```
if value > 25:  
    print("That is ok")  
else:  
    print("Not enough!")
```


Condiciones en Python

- They compare two values and return a bool value.
- Types of conditions
 - == (equal)
 - != (not the same)
 - >= (greater than or equal)
 - <= (less than or equal)
 - > (greater than)
 - < (less than)
- Not just for numeric data types

if block

- It allows us to run different code based on the result of a condition
- Syntax:
 if condition:
 do_sthg

```
[4] side1 = 4  
    side2 = 3  
    hypotenuse = 5  
  
    if side1**2 + side2**2 == hypotenuse**2:  
        print("Pythagoras was right")  
    else:  
        print("We have been fooled")
```

Pythagoras was right

What if the condition is not met?

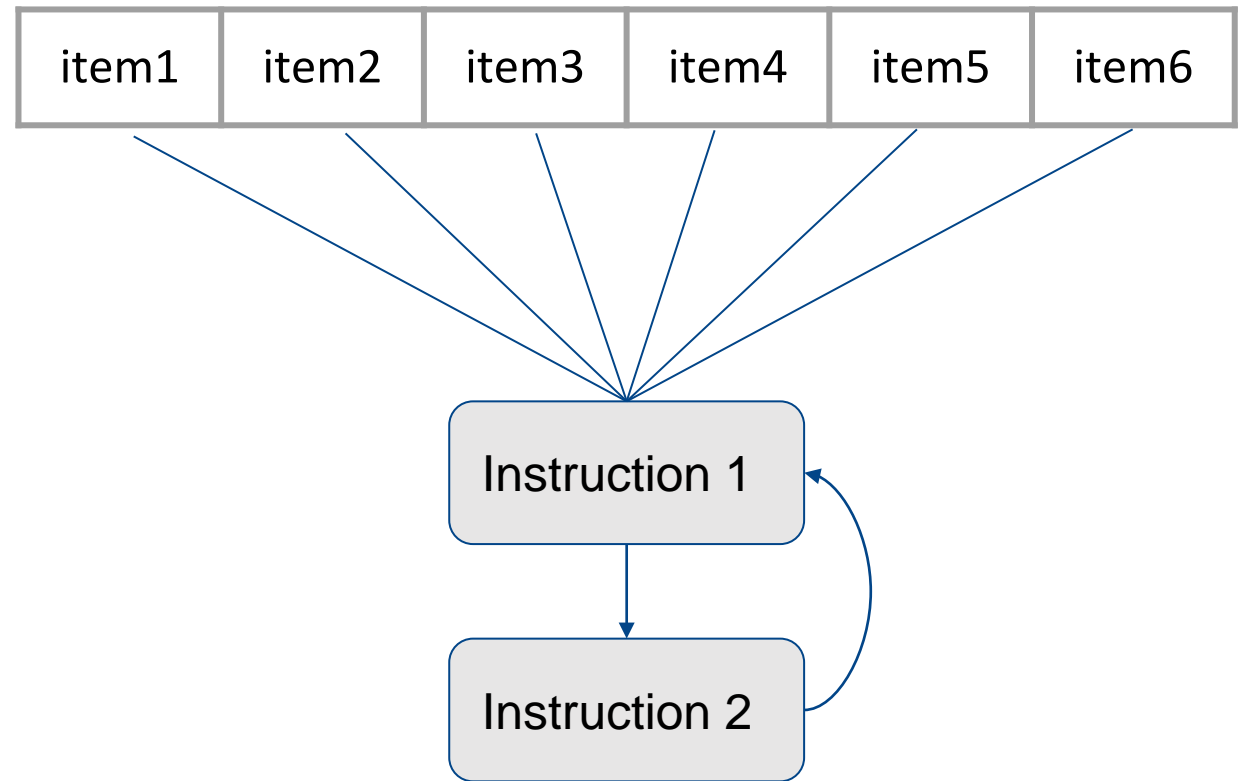
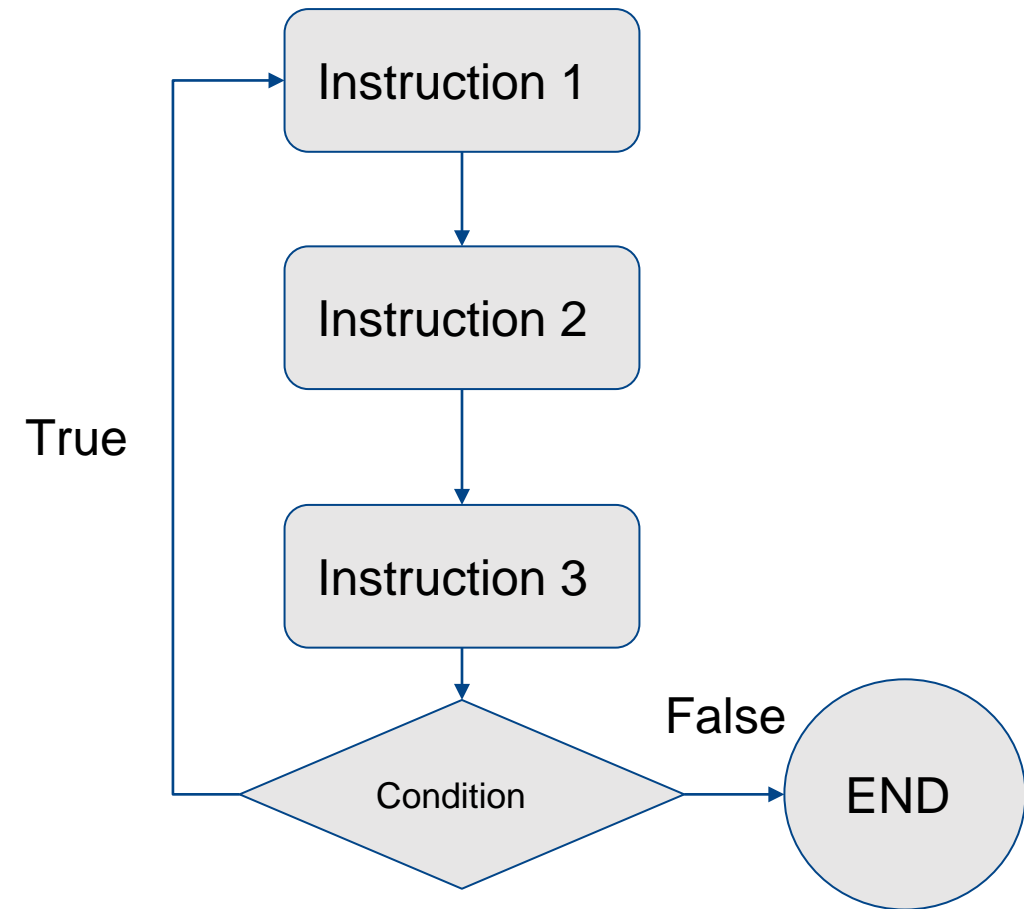
- We can use `else` to check another condition
- If we want to check another condition, we can use *elif* (*else-if*)
- Remember indentation!
- Using **and/or/not** we can add more complex conditions

```
if blood_alcohol > 1.6:  
    print("Criminal!")  
elif blood_alcohol > 0.5:  
    print("You have to pay a fee")  
elif blood_alcohol > 0.0:  
    print("Please do not drink if you drive!")  
else:  
    print("You are a good citizen")
```

Loops

—

Loops in programming



for and range

- The range() function returns a sequence of numbers

range(start, stop, step)

- *step* is optional
- *start* is the starting point
- The range ends in *stop - 1*

- for variable in range(start, stop)
 - We go through the range
 - For each value, we run the code block inside of the loop

```
for x in range(0, 5):  
    print(x)
```

Iterating over a list/tuple

- We can resort to the list using `for ... in range()`, and indexing the list:
- We can also directly iterate on the list, the result would be exactly the same:

```
list_of_numbers = [1,2,3,4]
for item in range(0,4):
    square = list_of_numbers[item]*list_of_numbers[item]
    print("The square of", list_of_numbers[item], "is", square)
print("The loop ended")
```

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The loop ended
```

```
list_of_numbers = [1,2,3,4]
for item in list_of_numbers:
    square = item*item
    print("The square of", item, "is", square)
print("The loop ended")
```

Loop with while

- Repeats the instruction block **as long as a condition is met**

```
list_of_numbers = [1,2,3,4]
item = 3
index = 0
while item > list_of_numbers[index]:
    print(item,"is greater than", list_of_numbers[index])
    index += 1 # if we do not increase this variable, we would always check the same number!
print("We left the loop at element", index, "because", item,"is not greater than", list_of_numbers[index])
```

3 is greater than 1

3 is greater than 2

We left the loop at element 2 because 3 is not greater than 3

Beware of the Exit Condition when using while

- If the output condition is never met, we can have an **infinite loop**
- If we're going through a list, we can go **out of bounds**

```
value = 5
while value > 3:
    value += 1
```

```
list_of_numbers = [1,2,3]
item = 10
index = 0
while item > list_of_numbers[index]:
    index += 1
```

Another way to get out of a loop

- The instruction `break` it will immediately exit the loop we are in:

```
list_of_numbers = [1,7,3,4]
for i in list_of_numbers:
    if i == 3:
        print("Found it!")
        break
    else:
        print("I found", i, "so I keep trying")
print("Out of the loop")
```

```
I found 1 so I keep trying
I found 7 so I keep trying
Found it!
Out of the loop
```

Let's practice

<https://colab.research.google.com/drive/1S995md4zZseeKHYYKW5509XnZs0l6J5l7?usp=sharing>



Recap

—

What have we learned?

- How to Create Conditional Blocks in Python
- How to iterate
 - for loop
 - while loop
 - Iterate through the items in a list

References and exercises

- <https://docs.python.org/3/tutorial/controlflow.html>
- <https://docs.python.org/3/tutorial/errors.html>
- <https://docs.python.org/3/tutorial/classes.html>
- <https://holypython.com/beginner-python-exercises/exercise-14-range-function/> (also exercises 15 and 16)
- <https://pythonprinciples.com/challenges/> (some challenges)

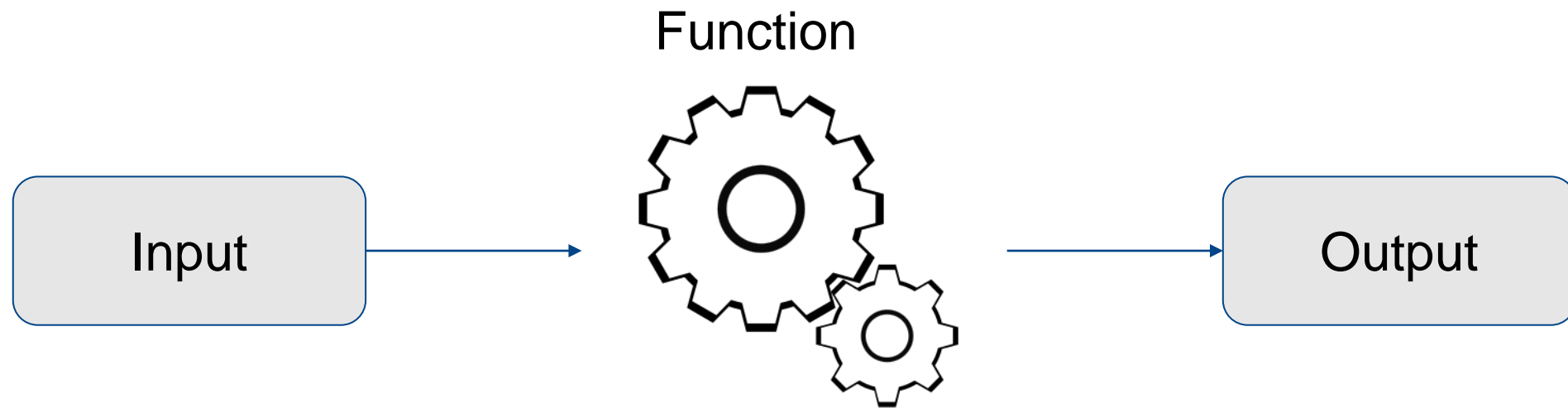
4. Python functions and libraries

- Functions
- Read/Write
- Libraries

Functions

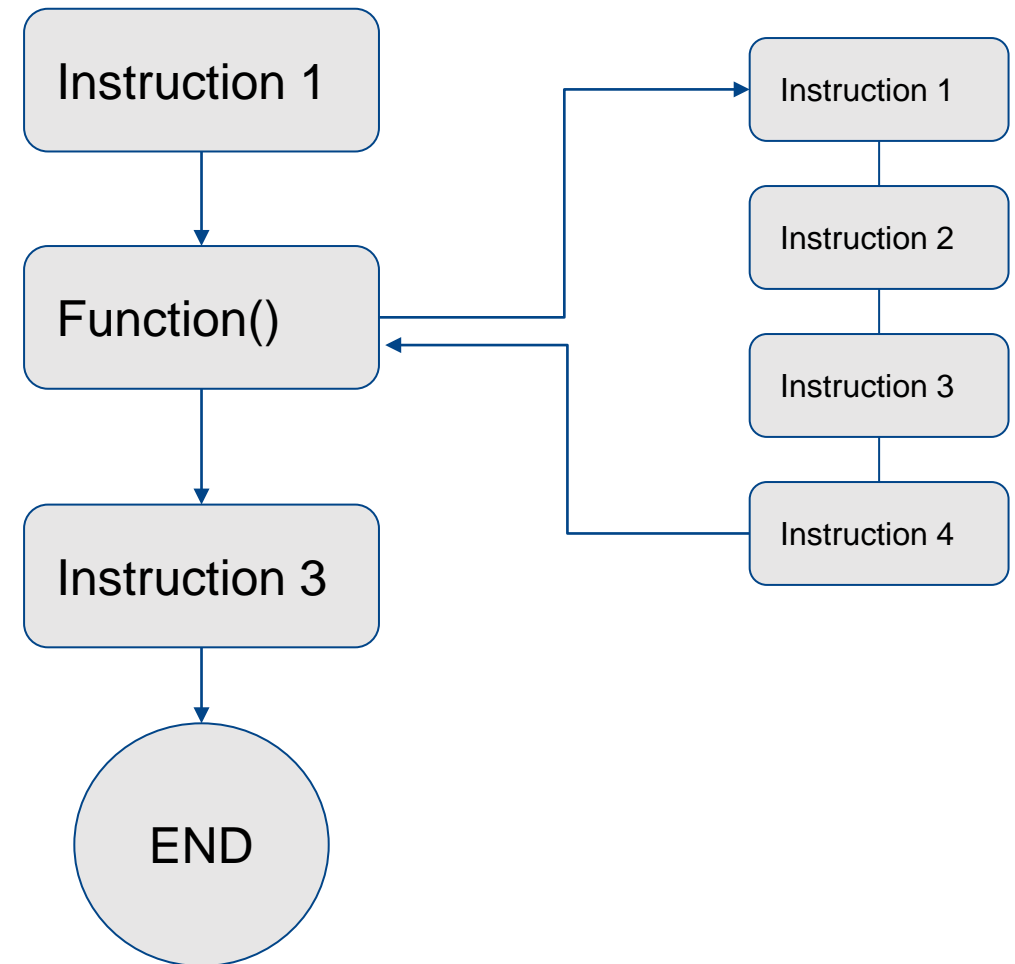
—

Functions in programming



Benefits of Using Functions

- Code reuse, simplicity and legibility
- No need of understand its inner working, just what we must pass them.
- Ease of error correction
- Actully we used a few already!
 - `len()`
 - `type()`
 - `print()`



Defining Functions in Python

```
def function (param1, param2):  
    instruction1  
    instruction2  
    ...  
    return value
```

Defining Functions in Python (II)

- Input parameters can be 0, 1 or multiple
- They can have values by default (must be at the end, can be omitted when the function is called).
- The return statement specifies the return value
 - If it is not included, it is equivalent to “*return None*”

```
[2] def legalAge(age, country="USA"):
    limit_age = 18
    if country == "USA":
        limit_age = 21

    if age >= limit_age:
        return True

    return False

legalAge(18, "Spain")
```

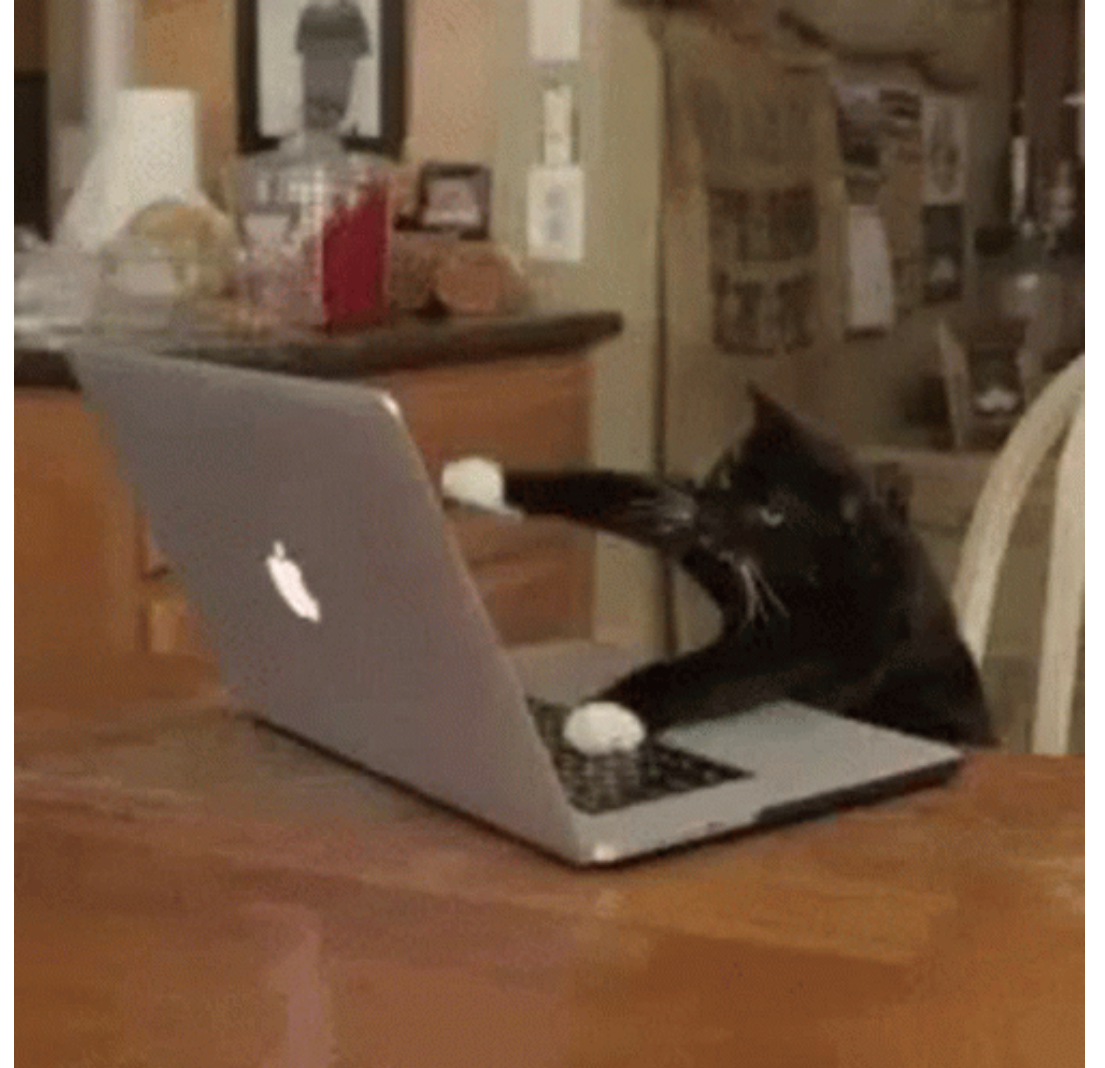
True

```
▶ legalAge(18)
```

False

Let's practice

<https://colab.research.google.com/drive/1lu2u2pmQMnrlqiKL7J-aobDN5otXxIUx?usp=sharing>



Functions for reading/writing files

—

Open and close files

- You can open and read a file as follows:

```
with open('myfile.txt', 'r') as variable:  
    texto = variable.read()  
  
print(texto)
```

- File opening modes
 - r: Reading
 - w: Writing
 - a: Add to end (*append*)

Creating files in Python

- We use `open()`, but using as open mode `'w'`
- Important methods:
 - `variable.write(line)`
 - `variable.writelines(list_of_lines)`

```
texto1 = 'Línea uno\nLínea dos\n'  
texto2 = ['Línea tres\n', 'Línea cuatro\n']  
  
with open('myfile.txt', 'w') as variable:  
    variable.write(texto1)  
    variable.writelines(texto2)
```


Libraries

—

Python libraries

- Reusable parts of code.
- May contain variables, functions...
- Python includes "standard libraries"
- There are also external libraries.
- You can find a list here:

<https://docs.python.org/3/library/index.html>

Some libraries:

- numpy (matrix handling)
- **pandas** (data analysis, a short intro can be found at the end of the slides)
- datetime
- math
- random
- os
- threading
- urllib
- argparse

Including external libraries in our code

```
import matplotlib
import pandas as pd
from math import sqrt

matplotlib.pyplot(...)
var1 = pd.Series(data=[100, 200, 300, 400, 500], index=['tom', 'bob', 'nancy', 'dan', 'eric'])
var2 = sqrt(25.0)
```

Name of the library

Import only a specific function

Name the library
differently

Recap

—

What have we learned?

- How to declare and use functions
- Read and write files
- What is a library and how to use it in our code

5. Hands-on!

Project ideas you can try, from easier to more difficult!

Number guesser

Create a program able to guess a number X you think just by saying a number and providing it as feedback if the number is correct, greater or lesser than X .

You might need to import random to generate a guess between two numbers, this example might be helpful: <https://www.programiz.com/python-programming/examples/random-number>

Trivia

Create a program able to ask questions and possible answers and decided if the selected answer is the correct one. You can create the questions from scratch or read them from some file.

Tic Tac Toe

Create a program able to play tic tac toe with you (it doesn't have to be smart, just to respect the rules!). You can try with a different game, such as **rock-paper-scissors**

Hint: If you prefer to modify existing code, here goes some help:

https://colab.research.google.com/drive/1LiOGRaWAK_B9vv0EbAQxEi_cyKKem89S?usp=sharing

Complex notebooks for Python experts

—

Machine Learning and Natural Language Processing (intermediate)

Using a Natural Language Processing library (SpaCy)

The following notebook shows how to use the NLP SpaCy library step by step:

<https://colab.research.google.com/drive/1OzKmoLNPMh2-t2YcPpTaiK0YlFsai2L3?usp=sharing>

Use Machine Learning to classify tweets (using NLP for preprocessing)

If you prefer to train your own model from scratch, you can use one of Python basic libraries for NLP (NLTK) and train two machine learning algorithms (Naive Bayes and SMV) for tweet classification.

https://colab.research.google.com/drive/1Ul6oTPcVEYH7hjhj7DoEao54YQCkC_qh?usp=sharing

Image classification using neural networks (very difficult!)

The following notebooks includes code for image classification step by step, feel free to ask questions while your classmates are coding!

- Here you will build a model to classify several images with types of clothes:

https://colab.research.google.com/drive/1ieSPD_NIdRpBlMnbiS0Z1gpvDpq1vOCk?usp=sharing

- This notebook shows you how to build a model that classifies if an image is a horse or a human. You will be able to upload an image to see if it is correctly classified!

<https://colab.research.google.com/drive/1cXazy2sPwW1cZp2C9XS4-0XMqiFlO88f?usp=sharing>

Data Analysis (with soccer data)

Someone asked for sports data! You can analyse this data using pandas library (a short introduction can be found in the following slides).

Notebook: <https://github.com/fodra/datascience/blob/master/Week-1-Intro-new/Introduction%20to%20Data%20Science%20in%20Python%20-%20Soccer%20Data%20Analysis.ipynb> (you can upload it to Google colab)

Data is available here: [direct download link](#)]

Information on the dataset can be found here:

<https://www.kaggle.com/datasets/hugomathien/soccer>

Data analysis with other data

More information on pandas and datasets to play:

- <https://pandas.pydata.org/docs/>
- Where to get datasets for analysis:
 - <https://opendata.aemet.es>
 - <https://datos.gob.es>
 - <https://www.kaggle.com>
It includes datasets, sample notebooks, challenges...

After reading the next section slides about pandas, you will be able to perform data analysis! Look for a dataset on the webs above and play 😊

EXTRA: how to use pandas library

—

Pandas

- Library for data analysis and manipulation
- Key Benefits
 - Can read/manipulate heterogeneous data
 - Allows you to combine datasets
 - Makes it easy to create visualizations
 - Supports time series



Data Structure in Pandas: DataFrame

- 2-D Structure
- Heterogeneous data
- Labels for rows and columns

	Population	PIB	Area
Madrid	3223000	240130	604.3
Barcelona	1620000	236814	101.9
Zaragoza	666880	38044	973.8

Defining DataFrames

Label of each column

✓
0s

```
[▶] import pandas as pd

mydict = {'one' : pd.Series([100., 200., 300.], index=['apple', 'ball', 'clock']),
          'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball', 'cerill', 'dancy'])}

mydf = pd.DataFrame(mydict)
print(mydf)
```

```
↗
  one  two
apple 100.0 111.0
ball  200.0 222.0
cerill  NaN 333.0
clock 300.0  NaN
dancy  NaN 4444.0
```

Undefined values receive the NaN value

Load DataFrames

- In general, we're going to load external data
- Pandas supports multiple formats

They can also be URLs



Format	Function for loading the data
CSV	<code>var = read_csv('/path/a/file.csv')</code>
Excel	<code>var = read_excel('/path/a/file.xlsx')</code>
JSON	<code>var = read_json('/path/a/file.json')</code>
SQL	<code>var = read_sql_query(...)</code> <code>var = read_sql_table(...)</code>
HTML	<code>var = read_html('http://path/to/url')</code>

Working with DataFrames (I)

- **Basic Info**
 - `dataframe.index`
 - `dataframe.shape`
 - `dataframe.columns`
 - `dataframe.head()`
 - `dataframe.info()`
- **Creating a dataframe with parts of another**
 - `newvar = pd.DataFrame(oldvar, index=['row1', 'row2', 'row3'])`
 - `newvar = pd.DataFrame(oldvar, columns=['col1', 'col2', 'col3'])`

Working with DataFrames (II)

- Extract a column
 - `df['nombre_columna']`
- Extract a row, without a label
 - `df.iloc[22]`
- Extract a row, labeled
 - `df.loc['etiqueta']`
- We can filter the data based on the value of a column, creating a *slice*
 - `df[valor > 25]`
- And extract to a new dataframe
 - `newvar = df[df[valor> 25]]`

Working with DataFrames (III)

- Add rows
 - `df.loc[indice] = [dato1, dato2, dato3]`
It must have the same number of values as the current dataframe
- Add Columns
 - `df['new'] = df['col1'] * df['col2']`
Any operation will do, even duplicate columns
 - `df.insert(column_index, column_name, data)`
- Delete rows
 - `df = df.drop(index=row)`
- Delete columns
 - `df.pop('column')`
Deletes the column and returns it to us as a result
 - `del df['column']`
It simply deletes it

Working with DataFrames (IV)

- **Grouping by a column**
 - `df.groupby('column')`
- **With the resulting object we can make operations**
 - `df.groupby('column').min()`
 - It groups the data by a column, and generates a dataframe with the minimum values
 - `.sum()`
 - `.max`
 - `.mean()`
 - `...`

String Operations

- The `str` property allows us to work with data as strings
 - `str.split()`
`df['column'].str.split(',')`
 - `str.contains()`
 - `str.replace()`
 - `str.extract(regex)` <- returns the first occurrence of the regular expression
- We can convert strings into other types
 - `str.astype(int)`
 - `str.astype(bool)`
 - `str.astype(float)`

Statistical functions

- We can derive multiple statistics from the data:
 - `df['column'].describe()`
 - `df['column'].mean()`
 - `df['column'].min(), df['column'].max()`
 - `df['column'].mode()`
 - Also applicable to all numeric columns in the dataframe
- `.std()`
 - Standard deviation
- `.corr()`
 - Data Correlation
 - `dataframe.corr()` shows us a table with the correlation between the data
 - `dataframe['column'].corr(dataframe['column2'])` shows us the correlation between 2 columns
- `.any()` / `.all()`
 - Check if any or all of the values in the dataframe/column are True.

Limpieza de datos con Pandas

- Los datos no siempre son perfectos
 - Faltan datos en alguna muestra
 - Datos no válidos
- Estrategias de solución
 - Reemplazar el valor
 - Rellenar huecos
 - Eliminar muestras / columnas
 - Interpolar

Data Cleaning with Pandas

- Data isn't always perfect(missing or not valid)
- Some strategies are replacing, filling empty data, deletion or interpolation.
- Data Cleaning functions:
 - `df.replace(value1, value2)`
Replaces data with one value for another
 - `df.fillna(method='ffill')` / `df.fillna(method='backfill')`
Replaces NaN data with the previous row value (method='ffill') or the following one ('backfill')
 - `df.dropna(axis=0)` / `df.dropna(axis=1)`
Deletes rows NaN values (axis=0), or columns (axis=1)
 - `df.interpolate()`
Interpolates missing data.

Saving data with Pandas

- We can save a dataframe to disk in CSV format
 - `df.to_csv(' /path/al/file')`
- Other available formats
 - `df.to_excel()`
 - `df.to_json()`
 - `df.to_html()`
 - `df.to_sql()`
 - ...