Introduction to Python

(Courtesy of Filippo Aleotti, https://filippoaleotti.github.io/website/)

Digital Systems M, Module 2
Matteo Poggi, Università di Bologna

Introduction to Python

What is Python

Python is a programming language originally developed by Guido van Rossum and now supported by a large community.

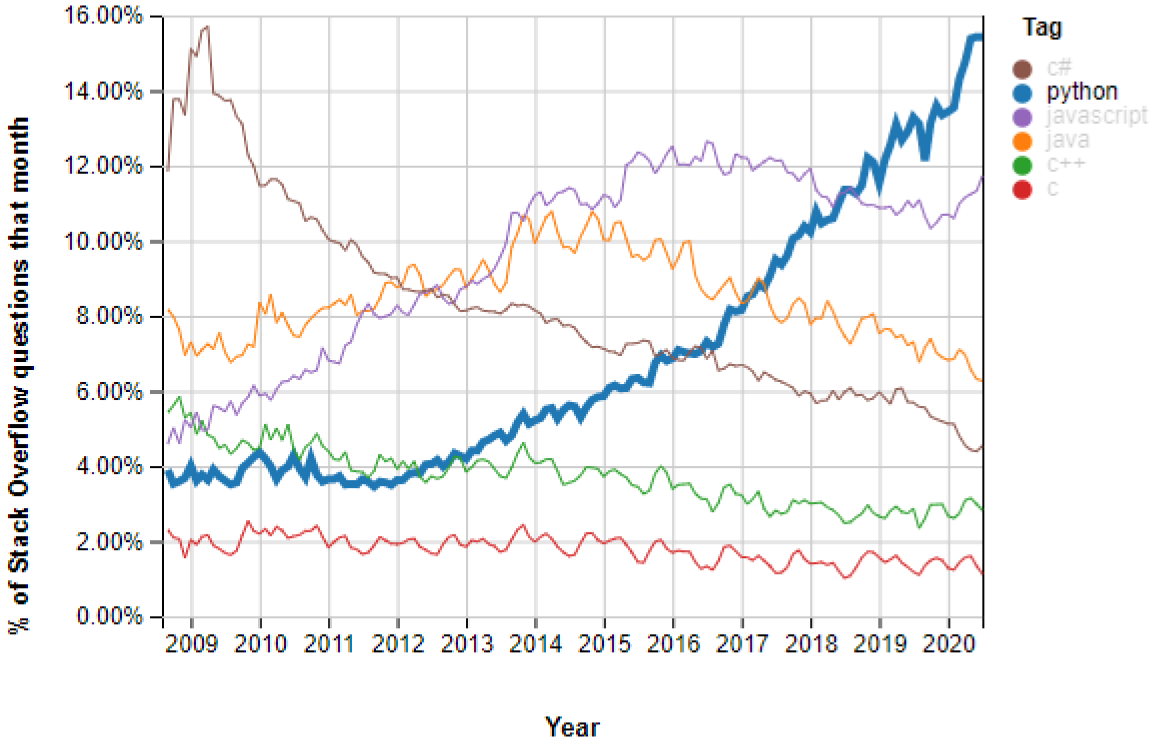
It is object oriented and dynamically typed, particularly suited for scripting but it can be used also for developing applications (web, GUI, etc)

Nowadays, it is largely adopted by researchers since it is easy to learn and use, allowing to fast prototype applications and tests

Python is an interpreted language

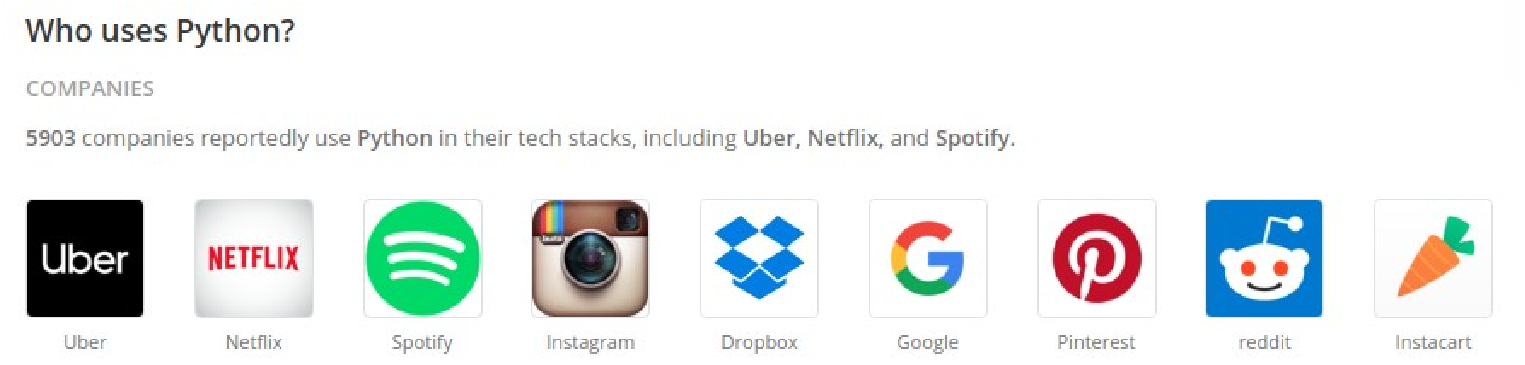
Introduction to Python

Programming languages trend (last updated 18/08/2020) on StackOverflow



Introduction to Python

Some companies that uses Python in their stack (according to stackshare)



Who uses Python?

COMPANIES

5903 companies reportedly use **Python** in their tech stacks, including **Uber**, **Netflix**, and **Spotify**.

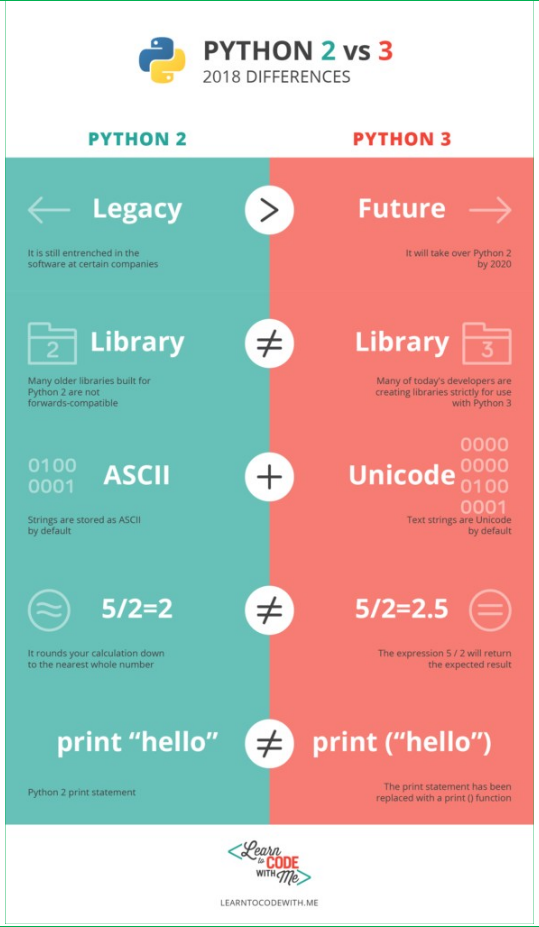
Introduction to Python

There exists two main versions of Python:
Python 2.x and Python 3.x

Although very similar, Python 2.x code may not be compatible with 3.x and vice-versa

Most libraries and frameworks (e.g., NumPy) are available for both

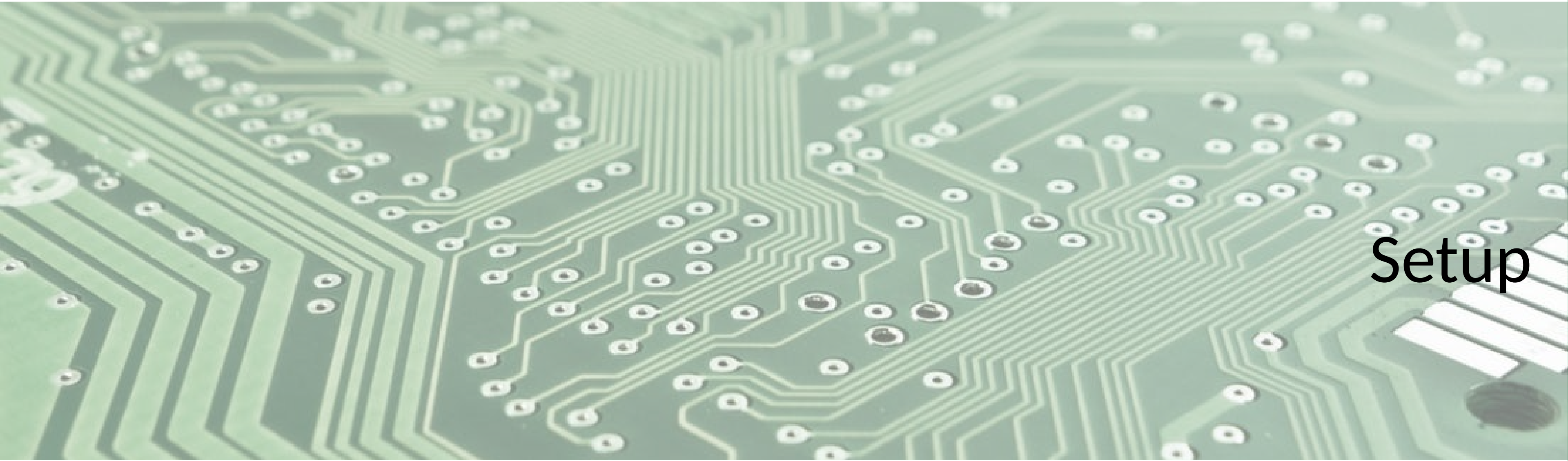
Python 2.x support ceased in January 2020.
Thus, in this course we will use Python 3.x



PYTHON 2 vs 3
2018 DIFFERENCES

PYTHON 2		PYTHON 3
Legacy It is still entrenched in the software at certain companies	>	Future It will take over Python 2 by 2020
Library Many older libraries built for Python 2 are not forwards-compatible	≠	Library Many of today's developers are creating libraries strictly for use with Python 3
ASCII Strings are stored as ASCII by default	+	Unicode Text strings are Unicode by default
5/2=2 It rounds your calculation down to the nearest whole number	≠	5/2=2.5 The expression 5 / 2 will return the expected result
print "hello" Python 2 print statement	≠	print ("hello") The print statement has been replaced with a print () function

Learn to CODE WITH Me
LEARNTOCODEWITH.ME



Setup

Introduction to Python

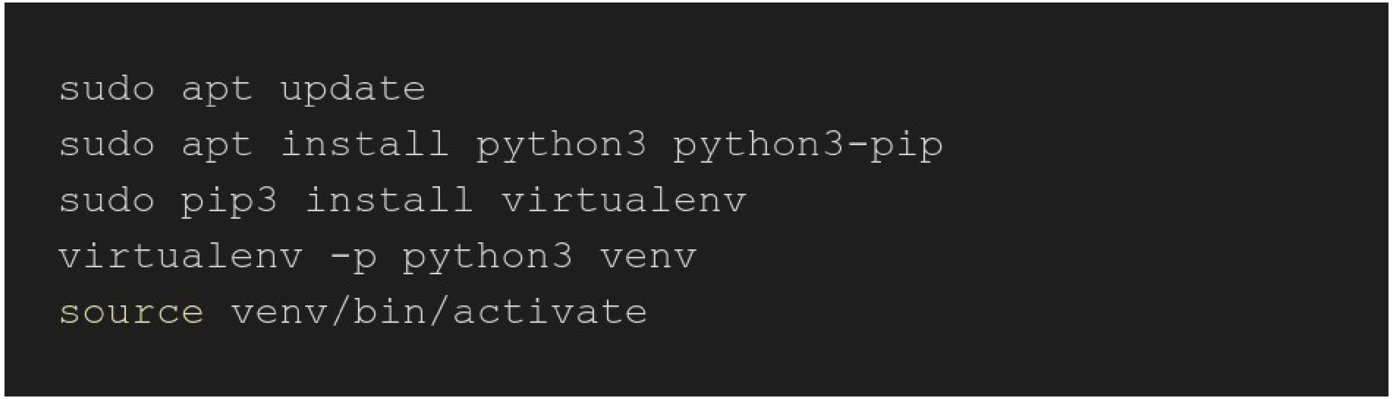
Before we start to code, we need to install Python 3.x if not already installed on our machine

Linux machines already have Python, while Mac and Windows users can download and install it respectively from here and here

It is not mandatory, but for this course we suggest to use Ubuntu:
some libraries (e.g., TensorFlow) will result easier to install

Introduction to Python

It is a good practice to create a virtual environment to install dependencies and packages, avoiding conflicts with other environments (like in a sandbox).



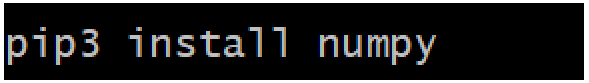
```
sudo apt update
sudo apt install python3 python3-pip
sudo pip3 install virtualenv
virtualenv -p python3 venv
source venv/bin/activate
```

venv is the name of the new environment that will be created, while the last command activates the env

Introduction to Python

Pip (for Python 3.x is Pip3) is a package manager: it allows to install modules and packages available in a store

For instance, if we need NumPy, a popular library handling multi-dimensional arrays, we need to run

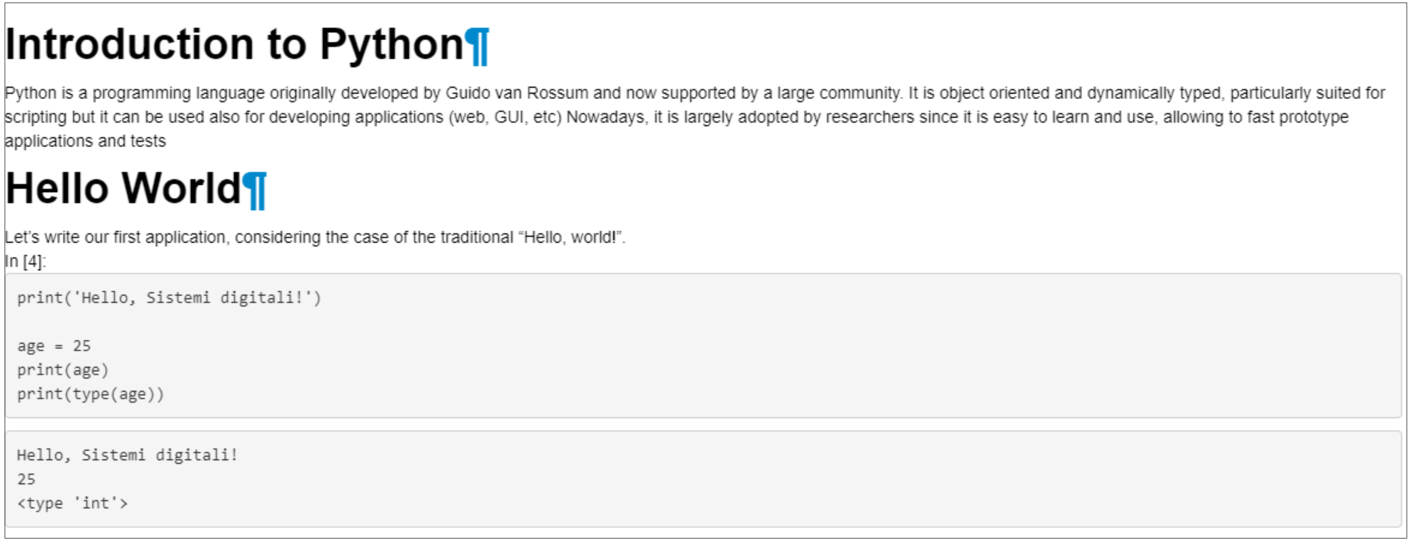


```
pip3 install numpy
```

In the PyPI store we can search if the package we need is available

Introduction to Python

Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations or plain text



Introduction to Python¶

Python is a programming language originally developed by Guido van Rossum and now supported by a large community. It is object oriented and dynamically typed, particularly suited for scripting but it can be used also for developing applications (web, GUI, etc) Nowadays, it is largely adopted by researchers since it is easy to learn and use, allowing to fast prototype applications and tests

Hello World¶

Let's write our first application, considering the case of the traditional "Hello, world!".

In [4]:

```
print('Hello, Sistemi digitali!')

age = 25
print(age)
print(type(age))
```

```
Hello, Sistemi digitali!
25
<type 'int'>
```

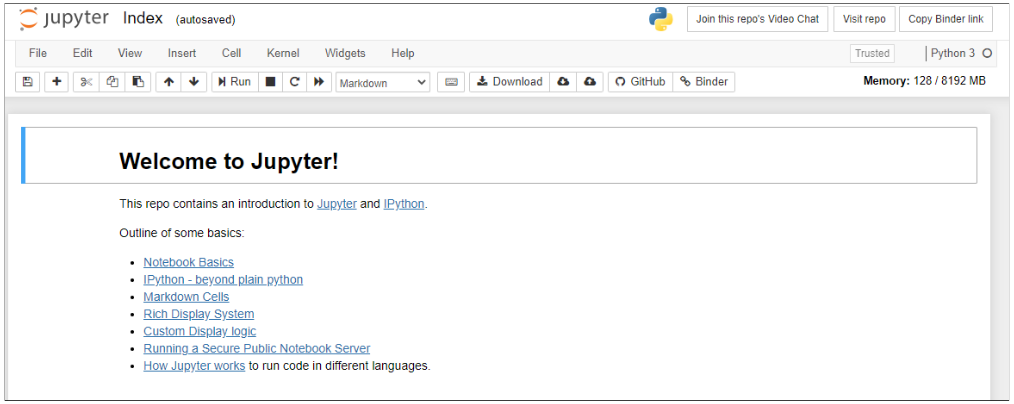
Introduction to Python

It can be installed with pip as well

```
pip install notebook
```

and launched with

```
jupyter notebook
```


jupyter Index (autosaved)
Join this repo's Video Chat
Visit repo
Copy Binder link
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Markdown
Download
GitHub
Binder
Memory: 128 / 8192 MB

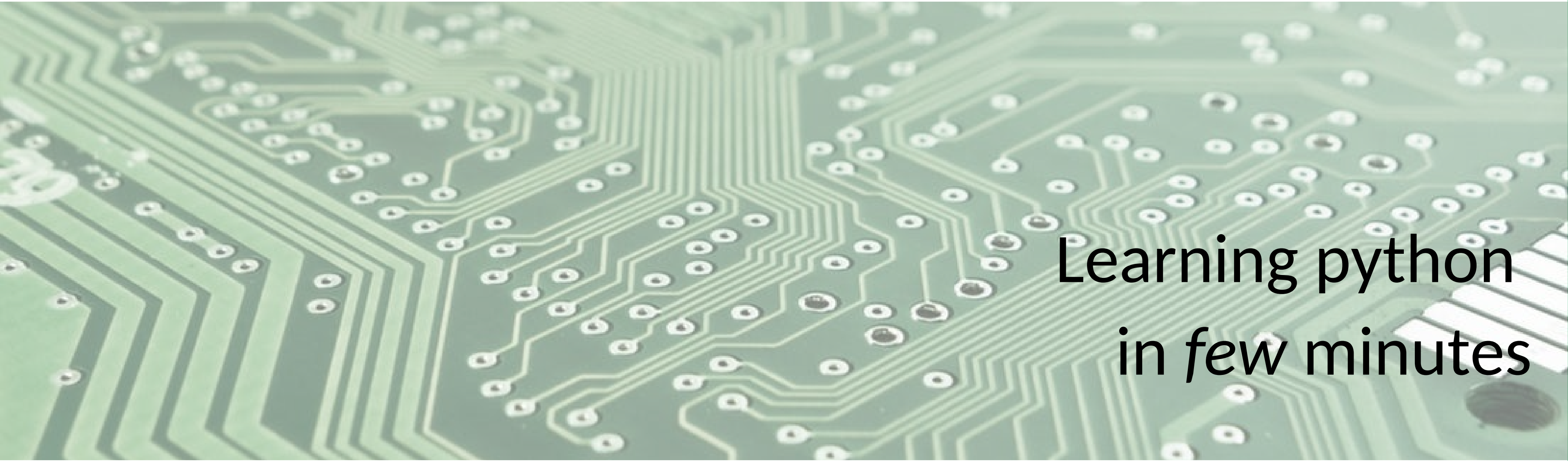
Welcome to Jupyter!

This repo contains an introduction to Jupyter and IPython.

Outline of some basics:

- Notebook Basics
- IPython - beyond plain python
- Markdown Cells
- Rich Display System
- Custom Display logic
- Running a Secure Public Notebook Server
- How Jupyter works to run code in different languages.

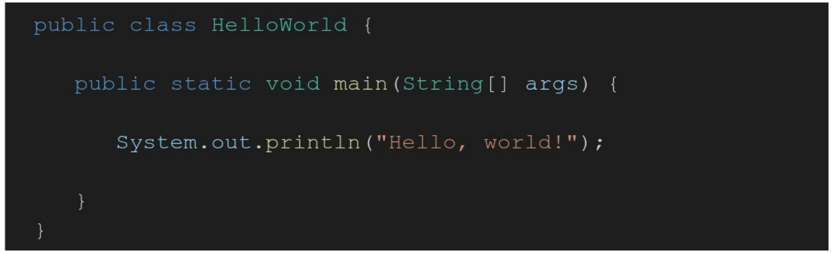
(you can also try it live online)



Learning python in *few* minutes

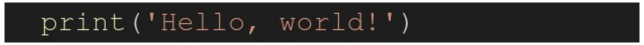
Hello, world!

Let’s write our first “*Hello, world!*” program



```
public class HelloWorld {

    public static void main(String[] args) {

        System.out.println("Hello, world!");

    }
}
```

Java source code



```
print('Hello, world!')
```

Python source code

...and run it

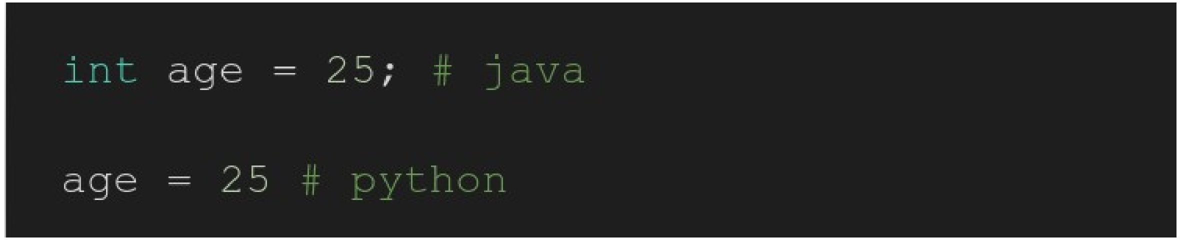
javac HelloWorld.java
java HelloWorld

vs

python hello_world.py

Introduction to Python

Python is a dynamically typed language. This means that we are not forced to explicit the type of each variable, since the compiler is smart enough to understand the type by itself.

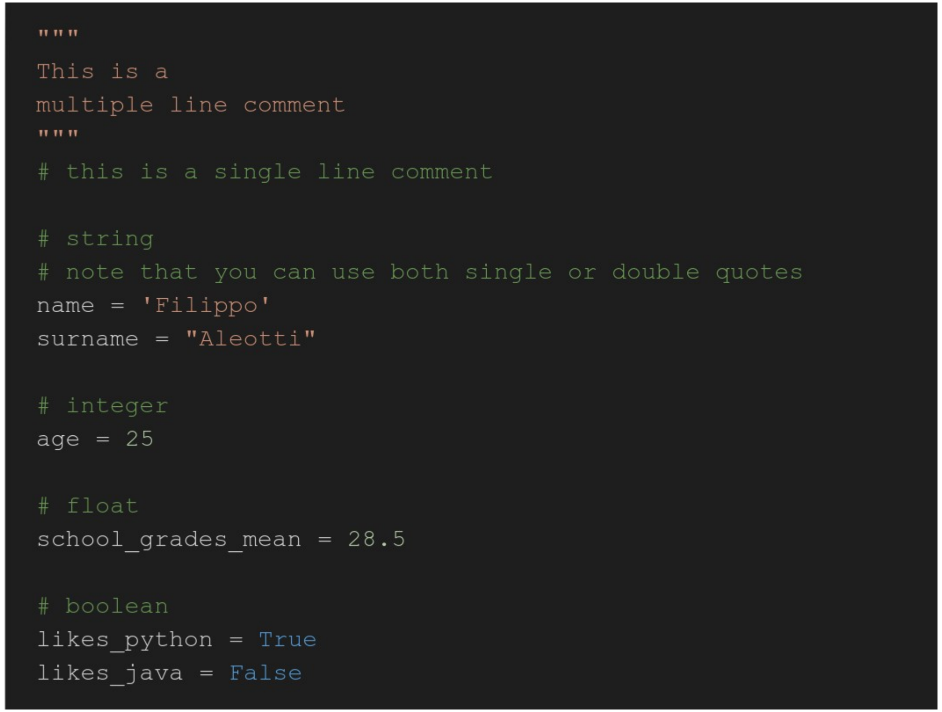


```
int age = 25; # java

age = 25 # python
```

Notice also that the ; character is no more required at the end of the line

Introduction to Python

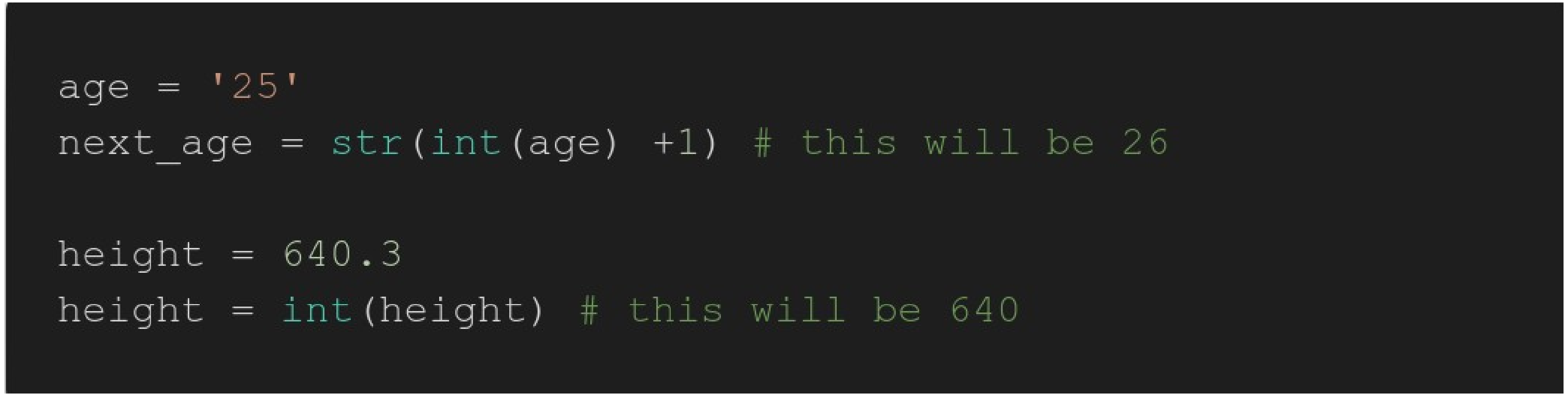
Primitive types of the language are integer, float, strings and boolean



```python
"""
This is a
multiple line comment
"""
# this is a single line comment

# string
# note that you can use both single or double quotes
name = 'Filippo'
surname = "Aleotti"

# integer
age = 25

# float
school_grades_mean = 28.5

# boolean
likes_python = True
likes_java = False
```

Introduction to Python

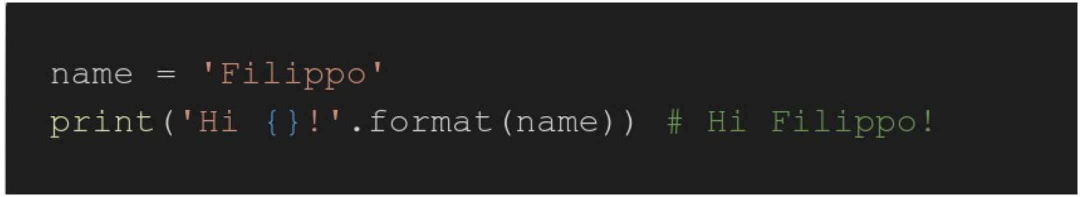
Changing from a type to another is supported as well.
To do so, we can cast the type (paying attention to avoid exceptions)



```
age = '25'
next_age = str(int(age) +1) # this will be 26

height = 640.3
height = int(height) # this will be 640
```

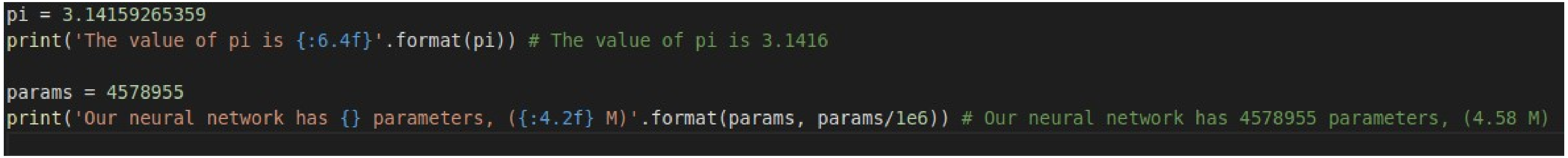
Introduction to Python

It is often desirable to make a string more readable
We can format our string using the format method of strings



```python
name = 'Filippo'
print('Hi {}!'.format(name)) # Hi Filippo!
```

we can format floating point numbers as well

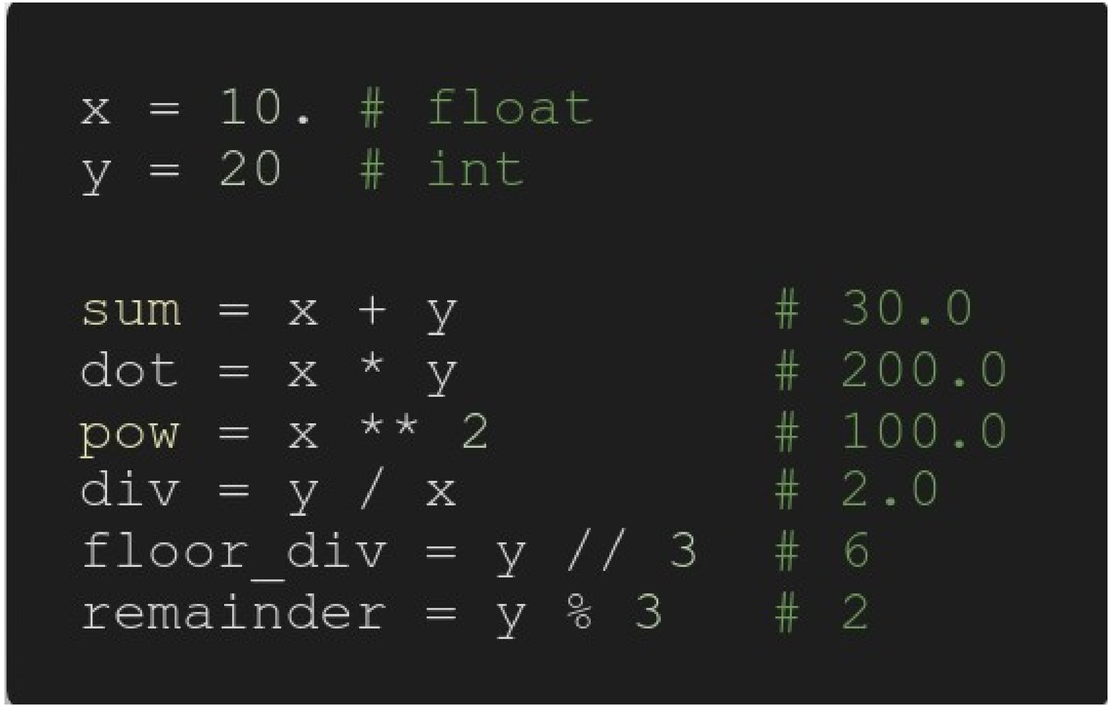


```python
pi = 3.14159265359
print('The value of pi is {:6.4f}'.format(pi)) # The value of pi is 3.1416

params = 4578955
print('Our neural network has {} parameters, ({:4.2f} M)'.format(params, params/1e6)) # Our neural network has 4578955 parameters, (4.58 M)
```

More details are available here

Introduction to Python

Math operations are performed as usual



```python
x = 10. # float
y = 20  # int

sum = x + y             # 30.0
dot = x * y             # 200.0
pow = x ** 2            # 100.0
div = y / x             # 2.0
floor_div = y // 3      # 6
remainder = y % 3       # 2
```

Introduction to Python

Strings are sequences of characters

```python
name = 'Filippo'
first_char = name[0] # F
last_char = name[-1] # o
# as for list, len returns the number of elements
length = len(name)  # 7
# as for list, the + allows to concat strings
full_name = name + ' Aleotti' # Filippo Aleotti
name = name.replace('F','f')  # replacing all F with f
```

They <u>do not</u> support item assignment or mixing characters with int or float

```python
name[0] = 'F'    # TypeError: 'str' object does not support item assignment
name = name + 2 # TypeError: cannot concatenate 'str' and 'int' objects
```

Introduction to Python

Collections

Python offers more complex data structures to handle collections.
Tuples are immutable collections of comma-separated values

```
languages = 'python', 'java', 'c++', 'c#', 'ruby'   # this is a tuple
languages = ('python', 'java', 'c++', 'c#', 'ruby') # but this is the convention for tuple
```

They can be accessed by index

```
first = languages[0]  # get the first element of the tuple
last  = languages[-1] # get the last element of the tuple
third_and_fourth = languages[2:4] # it returns the tuple ('c++','c#')
```

But the values cannot be changed (otherwise TypeError will be raised)

```
languages[1] = 'javascript' # TypeError: 'tuple' object does not support item assignment
```

Introduction to Python

To collect mutable data, a better option consist into using lists

```python
languages = ['python', 'java', 'c++', 'c#', 'ruby']  # this is a list
void = list() # empty list
void = []     # another empty list
mixed_list = ['1', 1, True, None] # we can create list with element of different types
```

As for tuples, we can get elements by index

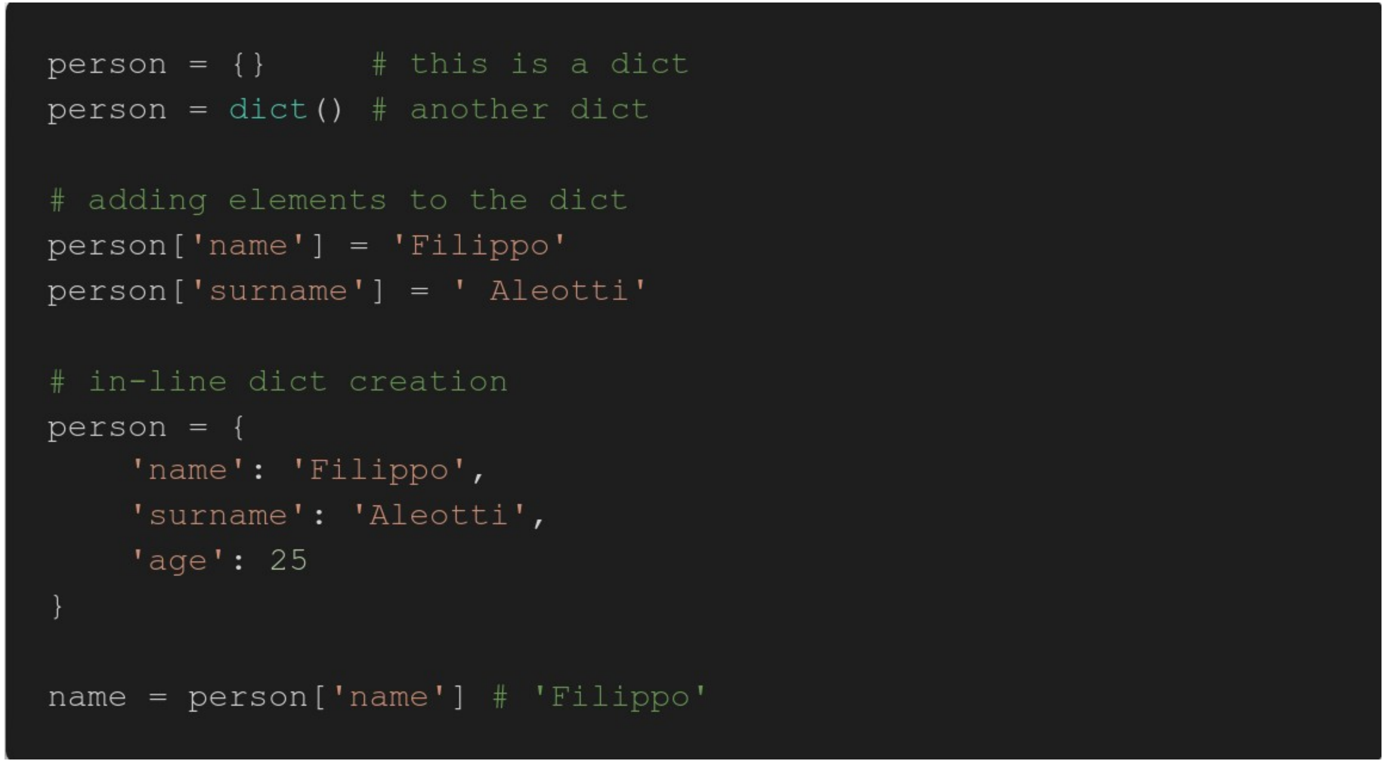
```python
first = languages[0]   # get the first element
last  = languages[-1] # get the last element
third_and_fourth = languages[2:4] # it returns the list ['c++','c#']
```

But, differently from tuples, lists are editable

```python
languages[1] = 'javascript' # now the list will be ['python', 'javascript', 'c++' , 'c#', 'ruby']
languages.append('c') # now the last element of the list is 'c'
languages.insert(3,'go') # insert at index 3 the string 'go'
```

Introduction to Python

Dictionaries allow to store key-value couples



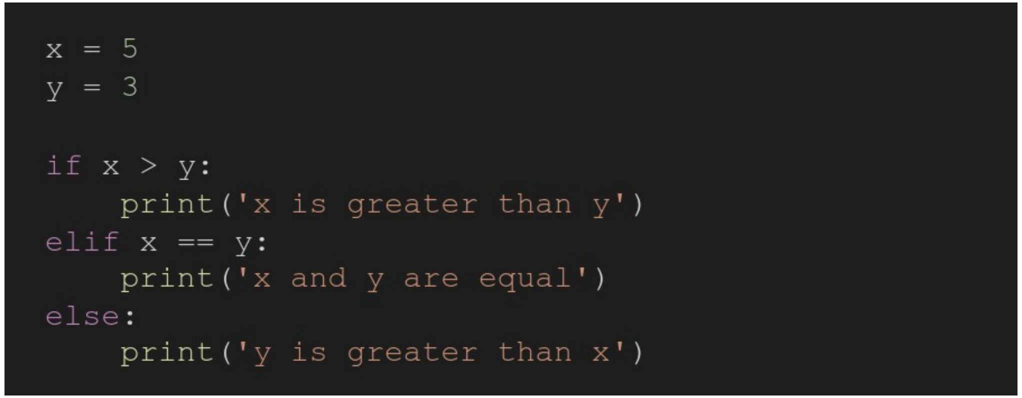
```python
person = {}     # this is a dict
person = dict() # another dict

# adding elements to the dict
person['name'] = 'Filippo'
person['surname'] = ' Aleotti'

# in-line dict creation
person = {
    'name': 'Filippo',
    'surname': 'Aleotti',
    'age': 25
}

name = person['name'] # 'Filippo'
```

Introduction to Python

Statements

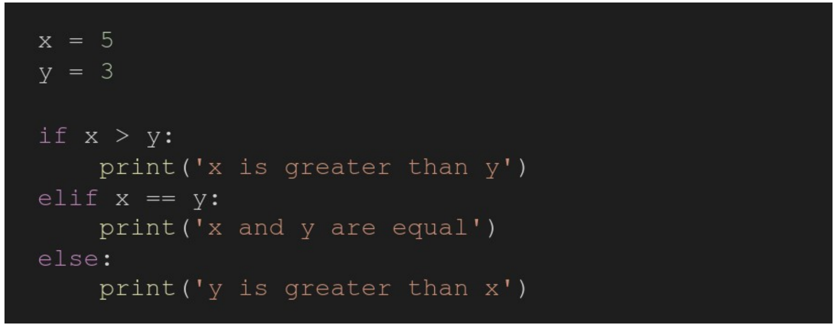
As for other programming languages, Python exposes <u>if</u> statement for conditions and <u>for</u> / <u>while</u> to iterate.

Using if, we can execute a set of instructions given a certain condition



```
x = 5
y = 3

if x > y:
    print('x is greater than y')
elif x == y:
    print('x and y are equal')
else:
    print('y is greater than x')
```

Introduction to Python



```
x = 5
y = 3

if x > y:
    print('x is greater than y')
elif x == y:
    print('x and y are equal')
else:
    print('y greater than x')
```

NOTE: instructions are <u>indented</u> with respect to the condition.

In Python indentation is <u>mandatory</u>: the compiler will throw an exception if not used

You can indent with <u>tabs</u> or <u>spaces</u>, but you <u>cannot use both</u> in the same script

Indentation is the equivalent to <u>wrapping</u> a set of instructions with { } in Java

Introduction to Python

For loop allows for running some instructions N times

```python
counter = 0
for i in range(50):
    counter += 1
print(counter) # 50
```

```python
counter = 0
for i in range(10,40,2):
    counter += 1
print(counter) # 15, since we start from 10 up to 40 with a step of 2
```

We can iterate also over collections

```python
fruits = ['apple', 'strawberry', 'mango', 'grapefruit']
searched_fruit = 'apple'
for fruit in fruits:
    if searched_fruit == fruit:
        print('found it!')
```

```python
countries_people = {
    'cina': 1401199000,
    'india':1387058000,
    'usa': 329472000
}
for country, people in countries_people.items():
    print('{} has {} inhabitants'.format(country, people))
```

Introduction to Python

<u>While</u> loop is used to continue iterating until a given condition is not met

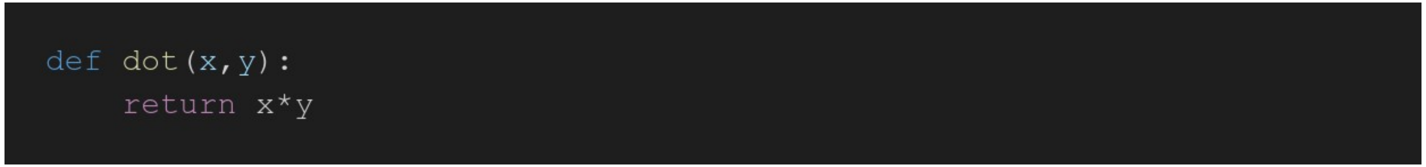
```python
found = False
searched_fruit = 'apple'
index = 0
fruits = ['mango', 'peach', 'pineapple', 'apple', 'strawberry']

while not found and index < len(fruits):
    if fruits[index] == searched_fruit:
        found = True
    else:
        index += 1

result = '{} is at index {}'.format(searched_fruit, index) if found else '{} not found'.format(searched_fruit)
print(result)
```

This script will print “apple is at index 3”

Introduction to Python

In order to improve both the readability and reusability of your code, it is a good practice to encapsulate a set of instruction inside a function. This function may be called both from the same and even also from other scripts.

Functions are defined using the def keyword



```
def dot(x,y):
    return x*y
```

In this case, x and y are arguments of the dot function

Introduction to Python

In Python, we can assign default values to arguments of the function

```python
def pow(x, y=2):
    return x**y

x1 = pow(5)   # x1=25 , y assumes the default value 2
x2 = pow(5,3) # x2=125, y is 3
```

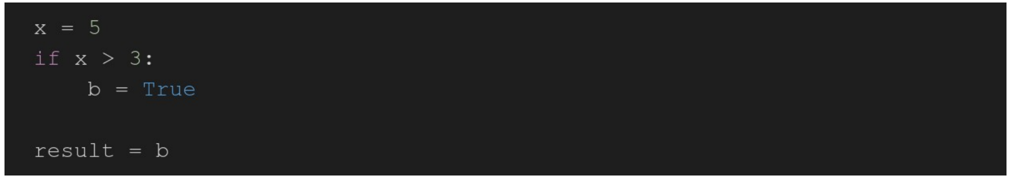
We can also change a specific set of values

```python
def my_transformation(x, alpha=1, beta=5):
    return x**alpha - beta

x1 = my_transformation(5, beta=2) #x1=3, we set beta=3 and alpha=1
```

Scope

The scope of a variable defines its visibility: it depends by where the variable has been defined and influence how we can use that variable

In Python we are not forced to declare always a variable (even it may be a good practice, to improve the readability of the code)

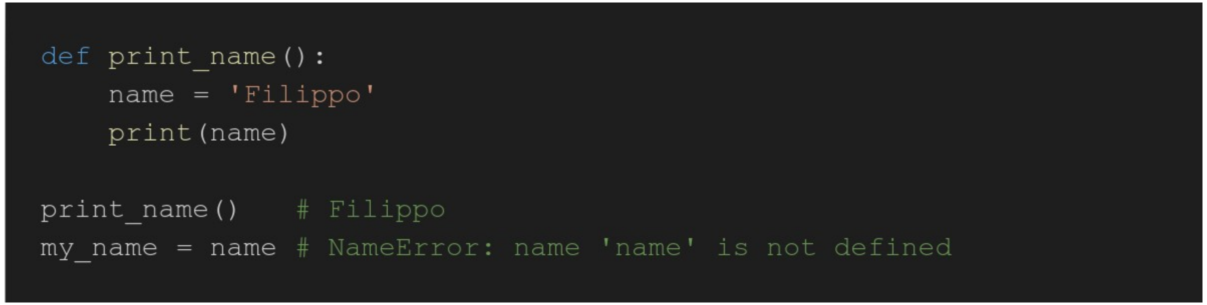


```python
x = 5
if x > 3:
    b = True

result = b
```

In this case b=True since the condition x>3 is verified, but with x=1 the program would raise *NameError: name 'b' is not defined*

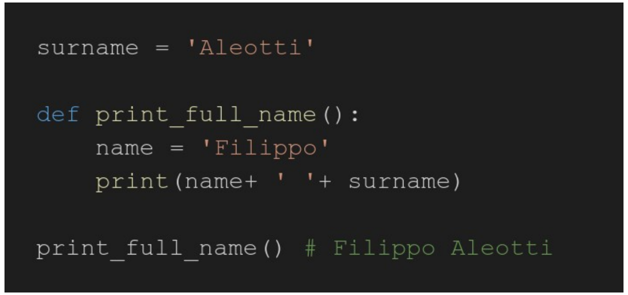
Introduction to Python

All the variables declared inside a function are local, so they are not visible outside that function



```python
def print_name():
    name = 'Filippo'
    print(name)

print_name()    # Filippo
my_name = name # NameError: name 'name' is not defined
```

Variables defined in global scope are visible inside local ones

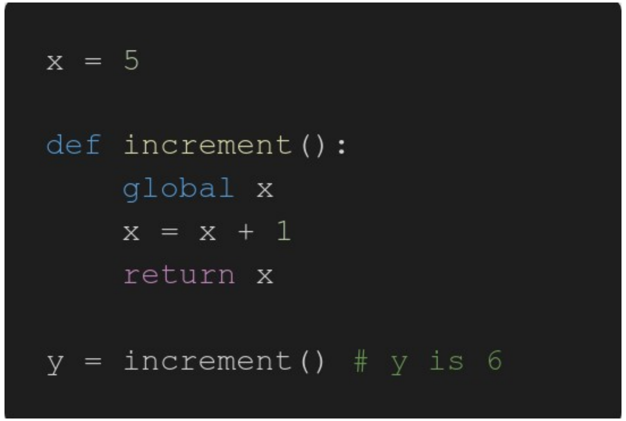


```python
surname = 'Aleotti'

def print_full_name():
    name = 'Filippo'
    print(name+ ' '+ surname)

print_full_name() # Filippo Aleotti
```

Introduction to Python

Global variables are visible but not editable inside a function

Using the global keyword, we can modify global variables as well



```
x = 5

def increment():
    global x
    x = x + 1
    return x

y = increment() # y is 6
```

If we had forgotten the global, the program would have raised *UnboundLocalError: local variable 'x' referenced before assignment*

Objects

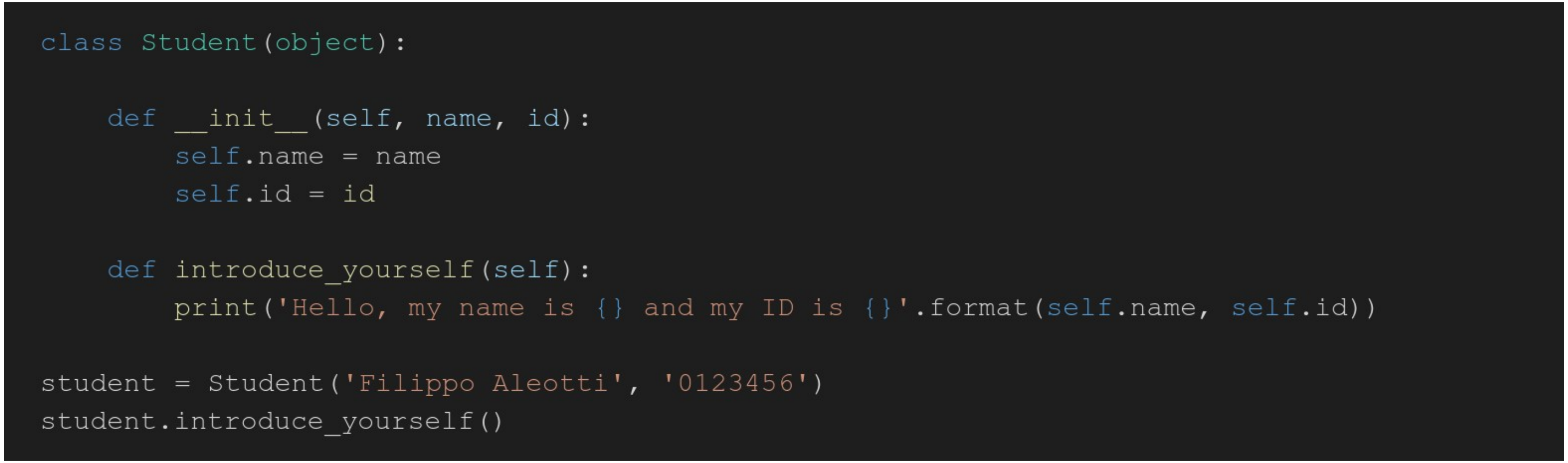
In OOP, objects wrap data and methods to handle a specific element of the domain.

For instance, in our exam-assistant application we can model a student as an object to keep trace of all the properties of a specific student (e.g., his ID) and specify methods valid for all the students

Student will be a class in our application, while the specific student Filippo Aleotti (with name="Filippo Aleotti" and ID="0123456") is an object (i.e., an instance of class **Student**)

Introduction to Python

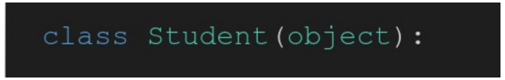
Let’s create the class **Student** and our first object of that class



```python
class Student(object):

    def __init__(self, name, id):
        self.name = name
        self.id = id

    def introduce_yourself(self):
        print('Hello, my name is {} and my ID is {}'.format(self.name, self.id))

student = Student('Filippo Aleotti', '0123456')
student.introduce_yourself()
```

Introduction to Python

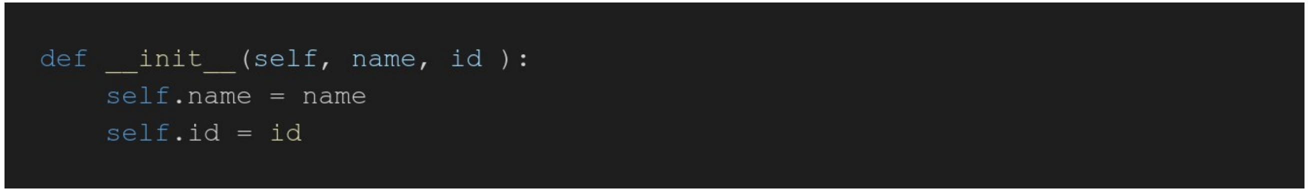
In Python, by default all objects inherit by object



```python
class Student(object):
```

We defined the constructor of our object using the reserved method __init__.

The keyword self is like this operator in Java, allowing to refer to the object itself. In the constructor we set the two properties (name and id) of object



```python
def __init__(self, name, id ):
    self.name = name
    self.id = id
```

Introduction to Python

We defined the method *introduce_yourself*

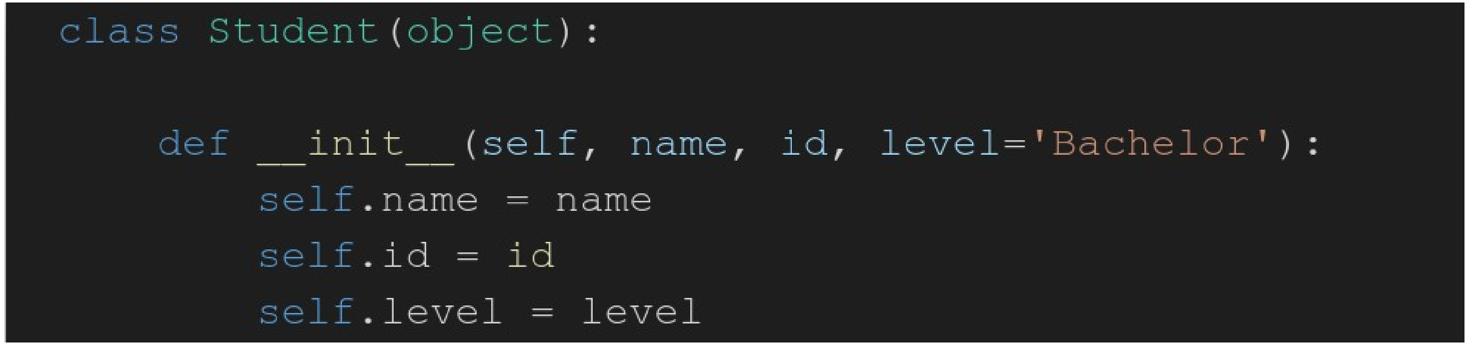
```
def introduce_yourself(self):
    print('Hello, my name is {} and my ID is {}'.format(self.name, self.id))
```

We then created an instance of **class** **Student** and invoked his method *introduce_yourself*

```
student = Student('Filippo Aleotti', '0123456')
student.introduce_yourself()
```

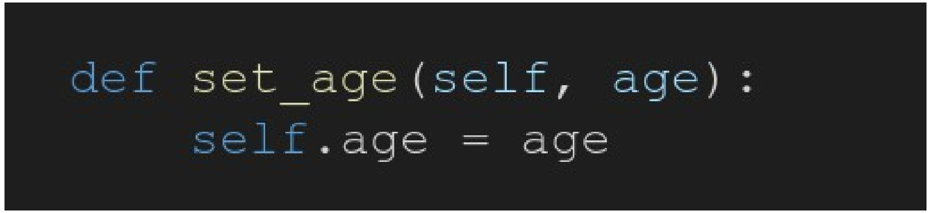
Introduction to Python

In Python you cannot define multiple constructors for your class, but you are free to use default values



```
class Student(object):

    def __init__(self, name, id, level='Bachelor'):
        self.name = name
        self.id = id
        self.level = level
```

We can define setters as well in case of private attributes (self.__age)

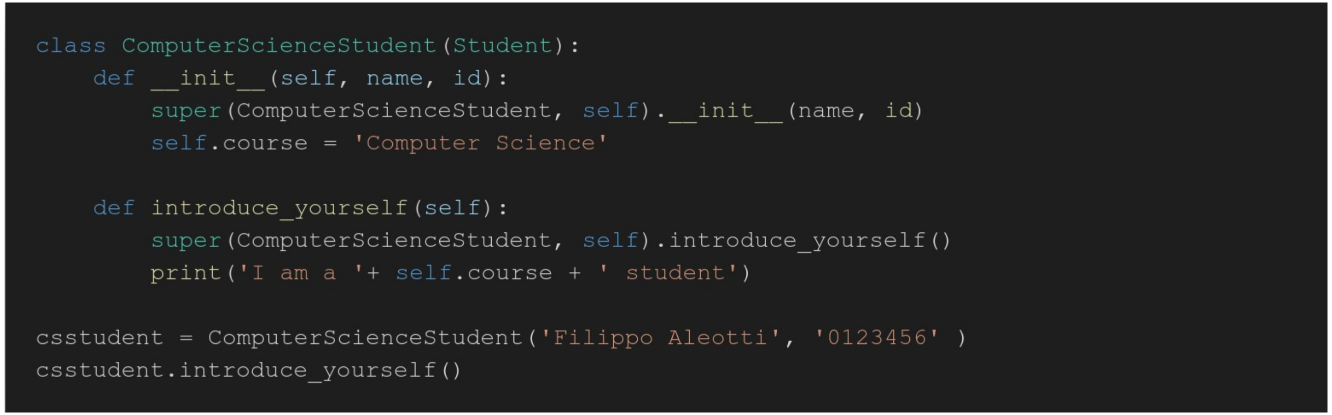


```
def set_age(self, age):
    self.age = age
```

Inheritance

We can extend classes to recycle parent’s methods and properties

We can access parent’s methods using the super keyword



```
class ComputerScienceStudent(Student):
    def __init__(self, name, id):
        super(ComputerScienceStudent, self).__init__(name, id)
        self.course = 'Computer Science'

    def introduce_yourself(self):
        super(ComputerScienceStudent, self).introduce_yourself()
        print('I am a '+ self.course + ' student')

csstudent = ComputerScienceStudent('Filippo Aleotti', '0123456' )
csstudent.introduce_yourself()
```

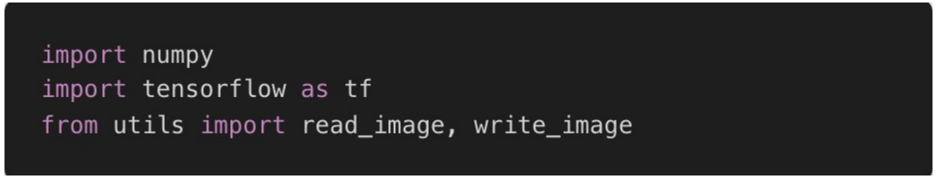
Multiple inheritance is allowed

Introduction to Python

Imports

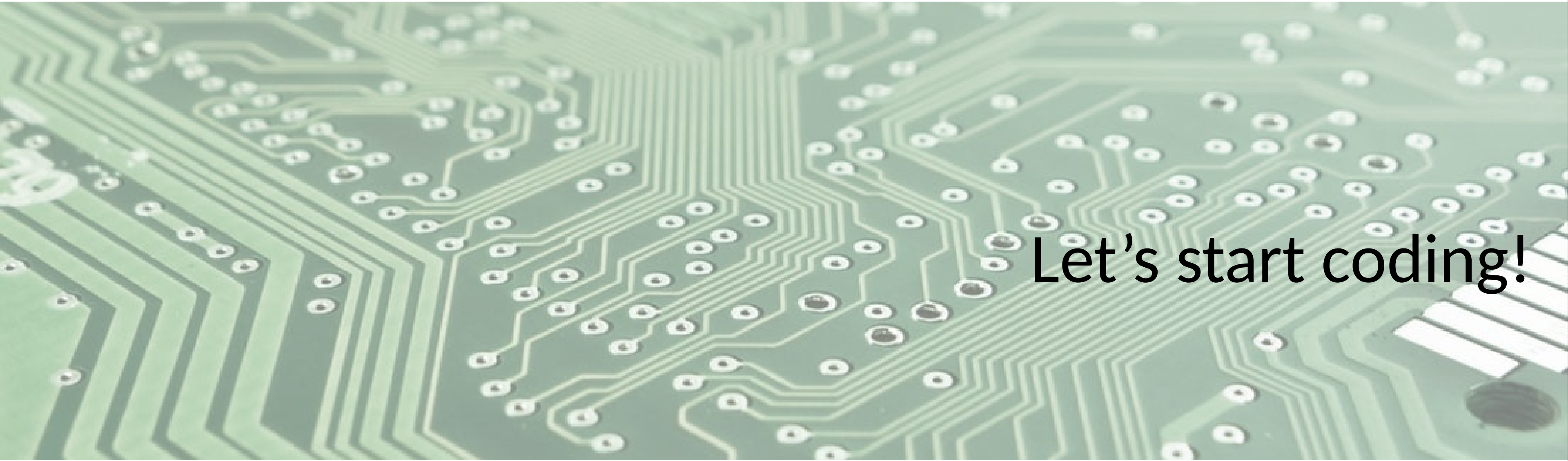
Sometimes we need to import functions or objects created by us in a different file (e.g., utils) or even third-party libraries (e.g., NumPy, TensorFlow)

To do so, Python offers the import keyword



```python
import numpy
import tensorflow as tf
from utils import read_image, write_image
```

NOTE: Third-party libraries need to be installed on your system/environment first. Package manager (Pip, Anaconda etc) can help you to tackle this task

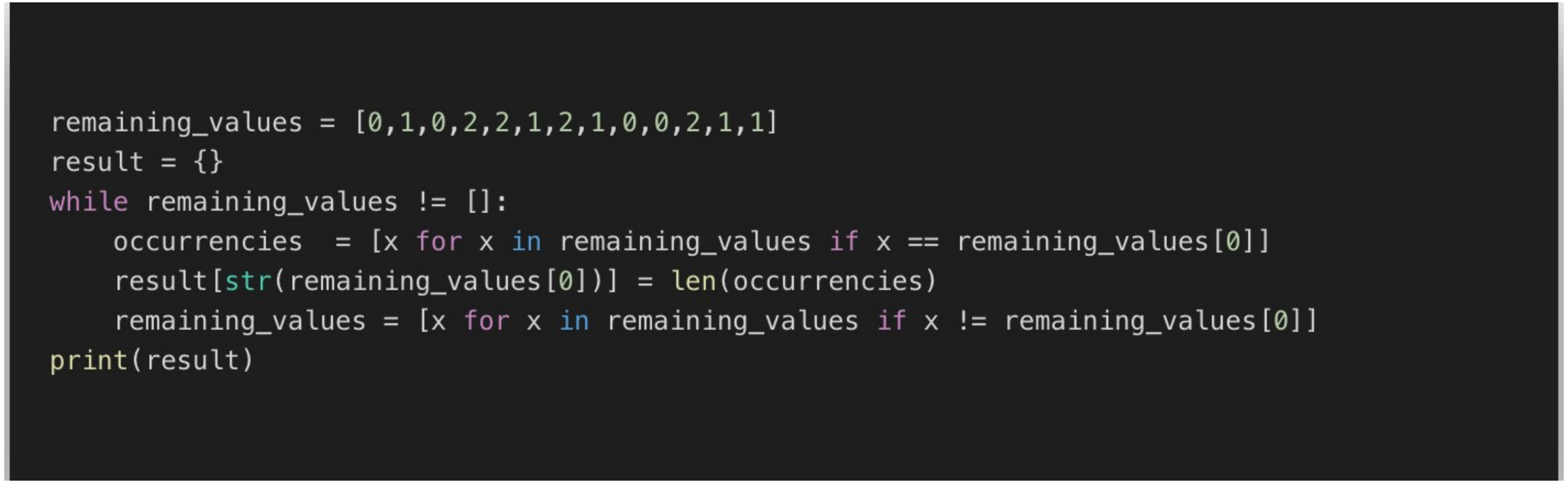


Let’s start coding!

Exercise 1

Given the list [0,1,0,2,2,1,2,1,0,0,2,1,1], store the frequency of each value in a dict, where the key is the value.

Introduction to Python



```
remaining_values = [0,1,0,2,2,1,2,1,0,0,2,1,1]
result = {}
while remaining_values != []:
    occurrencies  = [x for x in remaining_values if x == remaining_values[0]]
    result[str(remaining_values[0])] = len(occurrencies)
    remaining_values = [x for x in remaining_values if x != remaining_values[0]]
print(result)
```

In this solution, we used the list comprehension to iterate over the list with remaining values

It returns a new list, made up by those values that satisfies a given condition

Exercise 2

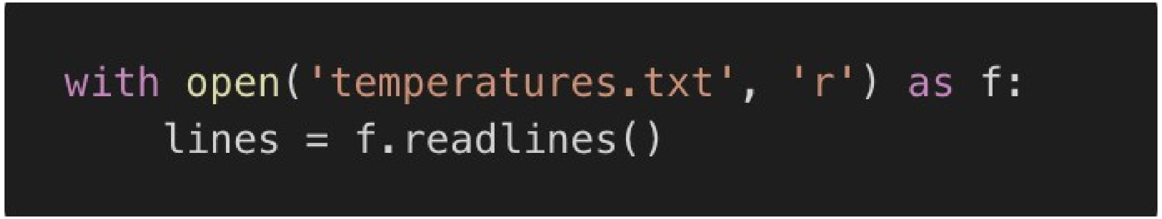
We stored into the file *temperatures.txt* many of temperatures (expressed in Celsius degrees) for a set of places. Each line contains the values, separated by space, for a specific place.

Find the maximum and minimum temperature for each place

Find the global maximum and the minimum temperature

Introduction to Python

In Python, we can manage with files using the **with** statement



```python
with open('temperatures.txt', 'r') as f:
    lines = f.readlines()
```

The with block masks the opening and the closing of the file
In the example we opened the file *temperatures.txt* in *read* modality (‘r’), getting back the file pointer as *f*. Then, we use *f* to store each line of the file into a list

Other modalities are *write* (‘w’), *read-write* (‘rw’), *read binary* (‘rb’) and *write-binary* (‘wb’)

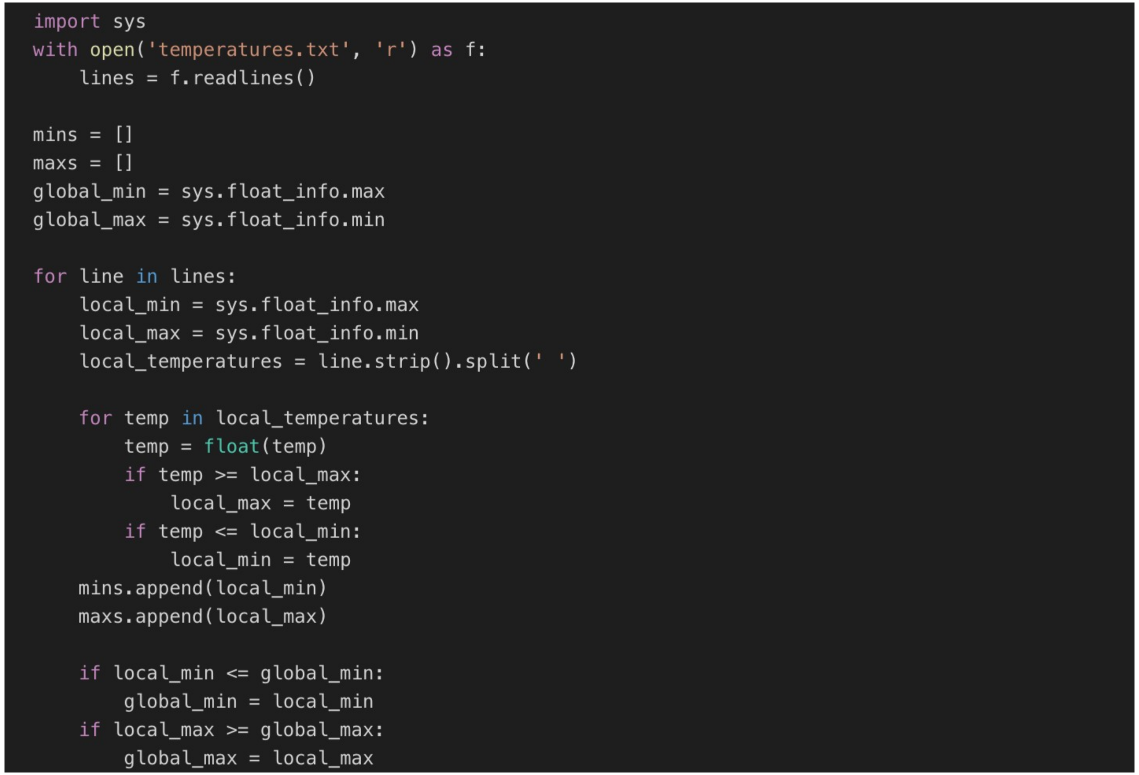
Some hints

sys.float_info.max and *sys.float_info.min* return respectively the maximum and the minimum float values

We need to remove the end-line character. The function *strip()* of string do the task

We read strings from the file, so we need to cast them into float values if we have to perform

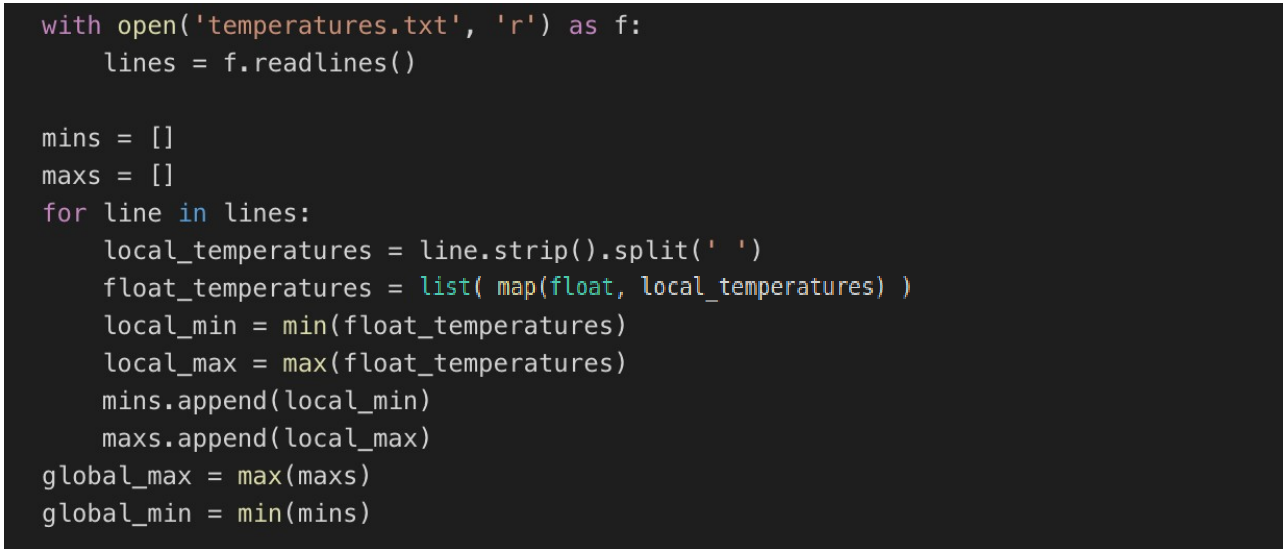
Naïve solution



```python
import sys
with open('temperatures.txt', 'r') as f:
    lines = f.readlines()

mins = []
maxs = []
global_min = sys.float_info.max
global_max = sys.float_info.min

for line in lines:
    local_min = sys.float_info.max
    local_max = sys.float_info.min
    local_temperatures = line.strip().split(' ')

    for temp in local_temperatures:
        temp = float(temp)
        if temp >= local_max:
            local_max = temp
        if temp <= local_min:
            local_min = temp
    mins.append(local_min)
    maxs.append(local_max)

    if local_min <= global_min:
        global_min = local_min
    if local_max >= global_max:
        global_max = local_max
```

Introduction to Python

Advanced solution



```python
with open('temperatures.txt', 'r') as f:
    lines = f.readlines()

mins = []
maxs = []
for line in lines:
    local_temperatures = line.strip().split(' ')
    float_temperatures = list( map(float, local_temperatures) )
    local_min = min(float_temperatures)
    local_max = max(float_temperatures)
    mins.append(local_min)
    maxs.append(local_max)
global_max = max(maxs)
global_min = min(mins)
```

Built-in functions *min* and *max* find the minimum and maximum of a collection
The *map* function applies an operation(in our case, casting) to each element of a collection