Python course for machine learning

The AI Institute

Introduction

- Python is a powerful programming language. It has efficient high level data structures and a simple but effective approach to object-oriented programming.

- It has simple syntax and dynamic typing

- Python is an interpreted language, ideal for scripting and rapid application development in many areas on most platforms.

Plan of the course

The course will be divided in two parts :

- A ***"reading"*** formation, which will give an overview of the different functions of Python and how they're understood by the computer.

- A ***"writing"*** session, in which we will see precisely how to write down these functions, and in which we will be able to make our first programs.

How to download Python interpreter

❖ Let’s install a local Python interpreter called Anaconda. To install it:

➢ Download Anaconda for Windows, Linux or MacOS X (~600MB) from the Anaconda official website https://www.anaconda.com/distribution/#download-section

➢ Install it (~2GB space needed, ~15 minutes)

➢ If you want to check the possibilities of Anaconda, just open the Anaconda Navigator

Python documentation

- On Windows, the manual provided with the interpreter (Start menu / All Programs / Python / Python Manuals) is usually the most convenient way to access the Python documentation.

- Online official documentation: http://www.python.org/doc

- Official tutorial: http://docs.python.org/tut/tut.html
- Largely documented online (Stack Overflow, Quora, …)

Python: Variables

❖ Variables are simply names which point to any value or object:

  ➢ Examples

```
→ a = 13

→ b = 12.5

→ c = “Banana”
```

❖ When you write these commands, the values on the right of the “equal” sign will be stored in the variables named a, b and c.

Python: Variables

- Once a value is saved in a variable, we can call the variable to get the associated value. We will here use the function print.

- To print a constant, write :

  ```
  print("hello, world")
  print(23)
  ```

- To print a variable, write :

  ```
  d = 459
  print(d)
  ```

Python: Variables

- Python is a dynamic language : variable values can be changed at any moment.
  - Example :

```
a = 6
print(a)
b = a
a = 4
print(a)
print(b)
a = a - 3
print(a)
```

Python: Variables

- Each variable has an associated type
  - Example

    → *a = 13* *Integer*

    → *b = 12.5* *Float*

    → *c = “Banana”* *String*

- There are many types in Python, we will see how they can be used.
  To see the type of a variable X, write `type(X)`.

Python: Variables and Types

- Main types in Python :

  Integer `(int)`
  Float (`float`)
  Complex (`complex`)
  Function
  String (`str`)
  List (`list`)
  Tuple (`tuple`)
  Set
  Dictionary
  Boolean (`bool`)

Python: Variable and Types - int, float & complex

Numerical types :

Integer	*eg :* `4, 17, 0, -8...`
Float	*eg :* `5.4, 19.320, 0.3333, 3.0...`
Complex	*eg :* `4 - 5j, 9j, -3 + 1j...`

Python: Variable and Types - int, float & complex

- You can make operations between different numerical variables, using the operators **+**, **-**, * (multiplication), **/** (division), **=**, ** (to the power of)...
  - Example :

```
a = 4
b = 5
c = a + b
print(c)
d = a * b
print(d)
e = d / b
print(a)
f = c - (a+b) / d
print(f)
print(a**2)
```

Python: Variable and Types - int, float & complex

❖ You can make operations between different numerical types. Python will understand and save the result as the right type.

➢ Example :

```
a = 4
b = 3.2
c = 5 + 3j
print(a + b) → 7.2
type(a + b)  → float
print(c - b)  → 1.8 + 3j
type(c - b) → complex
```

TheAIInstitute

Python: Variable and Types - Functions

- Performs a set of instructions when called.
- Can be given a set of parameters
- Two categories :
  - Built-in functions
    - Print(), help(), round()...
  - User-defined functions

Example :

```
a = 7.6
print(type(round(a)))
```

TheAIInstitute

Python: Variable and Types - Functions

❖ How to define a function ?

Pattern :

```
➢  def function_name():
                  ...
           ...
           return(...)
```

Example :

```
def multiply(a,b):
    c = a*b
    return c
```

Python: Variable and Types - Functions

❖ How to define a function ?

Pattern :

```
➢   def function_name():
                …
          ...
          return(...)
```

Example :

```
def multiply(a,b):
    c = a*b
    return c
```

c : temporary local variable

Python: Variable and Types - Functions

❖ Local / Global variables

Example 1 :

```
x = 4
Y = 5
def multiply(a,b):
    c = a*b
    return c
d = multiply(x,y)
print(d)
print(c)
```

Example 2 :

```
c = 4
d = 5
def multiply(a,b):
    c = a*b
    return c
d = multiply(c,d)
print(d)
print(c)
```

Python: Variable and Types - Functions

❖ Local / Global variables

Example 1 :

```
x = 4
Y = 5
def multiply(a,b):
    c = a*b
    return c
d = multiply(x,y)
print(d)   → 20
print(c)   → Error
```

Example 2 :

```
c = 4
d = 5
def multiply(a,b):
    c = a*b
    return c
d = multiply(c,d)
print(d) → 20
print(c)  → 4
```

Python: Variable and Types - String

Strings are used to save characters or chains of character.

❖ 3 syntaxes are using for string constants:

```
➢   string_with_single_quotes = 'prince'
➢   string_with_double_quotes = "prince"
➢   string_with_triple_quotes = """this is a multiline string."""
```

Example :

```
a = “Hello”
B = ‘My dear’
print(a,b)
type(a)
type(b)
```

TheAIInstitute

Python: Variable and Types - String

Strings are used to save characters or chains of character.

- 3 syntaxes are using for string constants:
  - `string_with_single_quotes = 'prince'`
  - `string_with_double_quotes = "prince"`
  - `string_with_triple_quotes = """this is a multiline string."""`

Example :

1. `a = “Hello”`
2. `B = ‘My dear’`
3. `print(a,b)` → **Hello my dear**
4. `type(a)` → **str**
5. `type(b)` → **str**

Variable and Types - String

Concatenation of two strings : use **+**

/ - * ** : Useless on strings

```
a = “Hello”
B = “Hi”
print(a+b)
```

Python provides a lot of functions which are very useful for string processing.

❖ Get the length of a string:

➢
```
len(strings)
```

❖ Convert to uppercase:

➢
```
strings_uppercase = strings.upper()
```

Variable and Types - String

Concatenation of two strings : use **+**

/ - * ** : Useless on strings

```
a = “Hello”
B = “Hi”
print(a+b)    → “HelloHi”
```

Python provides a lot of functions which are very useful for string processing.

- Get the length of a string:
  - `len(strings)`

- Convert to uppercase:
  - `strings_uppercase = strings.upper()`

TheAIInstitute

String Operations Method

- Strip spaces at beginning and end of a string:
  - stripped = a_string.strip()

- Replace a substring inside a string:
  - newstring = a_string.replace('abc', 'def')

- All string methods : https://www.tutorialspoint.com/python/python_strings.htm Note: a string is immutable, all operations create a new string in memory.

Variable and Types - List

- A list is a data structure that contains a series of values.

Example

```
>>> fruits = [“ananas”, “orange”, “pineapple”]

>>> price = [5, 3, 9]

>>> list = [“rice”, 54, “potatoes”, 89]
```

- Python restores the list in the same order it was entered in.

Variable and Types - List

- Access a specific element by index (index starts at zero):

  List = [“a”, 1, “q”, “c”, 9, 10]

  0 1 2 3 4 5

- The indices of a n element list start with 0 and end with n-1.
  - List[0] = ‘a’
  - List[1] = 1
  - List[5] = 10
- A negative integer as index is interpreted as the index starting from the last element.
  - List[-1] = 9
  - List[-4] = 1

Variable and Types - List

- Replacing the ith element :
  - List = [1,2,3,4,5,6]
  - List[3] = 5
  - print(List) → **[1,2,3,5,5,6]**

- Add an element at the end of the list :
  - List.append(10)
  - print(List) → **[1,2,3,5,5,6,10]**

- Get the number of elements in a list :
  - l = [“Jam”, “Bread”, “Table”, Biscuit”]
  - N = len(l)
  - print(N) → **4**

Variable and Types - List

- Remove an element from a list :
    - Given its index :
        - List = [‘A’,’B’,’C’,’D’,’E’]
        - List.pop(2)
        - print(List) → **List = [‘A’,’B’,’D’,’E’]**
    - Given a value :
        - List = [‘A’,’B’,’C’,’D’,’E’]
        - List.remove(‘B’)
        - print(List) → **List = [‘A’,’C’,’D’,’E’]**
- All list operations: http://docs.python.org/lib/typesseq.html

Variable and Types - List

❖ Slicing is extracting a sublist from a list:

Mylist = [‘a’,’b’,’c’,’d’,’e’,’f’,’g’,’h’,’i’,’j’,’k’]

print(mylist[0:3]) → [‘a’,’b’,’c’]

print(mylist[2:4]) → [’c’,’d’]

print(mylist[3:]) → [’d’,’e’,’f’,’g’,’h’,’i’,’j’,’k’]

print(mylist[-2:]) → [’j’,’k’]

print(mylist[:2]) → [‘a’,’b’]

Tuples

- ❖ A tuple is similar to a list but it is a fixed-size, immutable array: once a tuple has been created, its elements may not be changed, removed, appended or inserted.
- ❖ It is declared using parentheses and comma separated values:

```
>>> a_tuple = (1, 2, 3, 'abc', 'def')
```

- ❖ But parentheses are optional:

```
>>> a_tuple = 1, 2, 3, 'abc', 'def'
```

- ❖ Tuples may be seen as “complex constants”.

Dictionaries

- A dictionary is a mapping between indexes and values.
- Indexes can be of almost any type: integers, strings, tuples, objects…

Example:

```
>>> countries = {“us‟ : ”USA‟, “fr‟ : ‟France‟, “uk‟ : ‟United Kingdom‟}

 >>> print(countries[“uk‟])

 >>> countries[“de‟] = “Germany‟
```

Dictionaries

- A dictionary can be seen as a list, but instead of accessing data with an index corresponding to the rank of the object, we use as index the object we want.
- Indexes can be of almost any type: integers, strings, tuples, objects…

Example:

```
>>> countries = {“us“ : ”USA“, “fr“ : “France“, “uk“ : “United Kingdom“}

 >>> print(countries[“uk“]) ➔ “United Kingdom”

 >>> countries[“de“] = “Germany“

 >>> print(countries) ➔ {“us“ : ”USA“, “fr“ : “France“, “uk“ : “United
Kingdom“}
```

TheAI Institute

Conversion

❖ Sometimes, when we manipulate data, we can have some troubles because a variable does not have the type we want.

Ex : x = ‘2’ and we want to multiply this number by 3.

➢ We need to convert a variable into another type.

Some functions allow to do that in the Python base functions.

Conversion

B \ A	Int	Float	Str
Int		B = int(A)	B = int(A)
Float	B = float(A)		B = float(A)
Str	B = str(A)	B = str(A)	

TheAIInstitute

Dictionaries Operations

- ❖ Get a list of all indexes:

```
>>> country_codes = countries.keys()
```

- ❖ Get a list of (index, value) tuples:

```
>>> Countries_list = countries.items()
```

- ❖ Test if a specific index is there:

```
>>> is_uk_there = “uk‟ in countries
```

- ❖ More info: https://www.tutorialspoint.com/python/python_dictionary.htm

Variable and Types - Boolean

- Two values : *True* or *False*
- Boolean are used to answer logical operators. The value is True is the equation is true.
- Examples of logical operators :
  - Equals: `a == b`
  - Not Equals: `a != b`
  - Less than: `a < b`
  - Less than or equal to: `a <= b`
  - Greater than: `a > b`
  - Greater than or equal to: `a >= b`

Variable and Types - Boolean

a = 3
b = 5
c = 7

a < c → True

c == b → False

c - b == b - a → True

- The conditions can also combine several booleans, with the logical operators ***and*** and ***or***. They can be represented in a logical table :

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

Block and Indentation

- Blocks of code are delimited using indentation, either spaces or tabs at the beginning of lines. This is one of the main differences of Python over other languages.
  - Example

```
>>> def function(x,y)
>>>  return x + 2*y
```

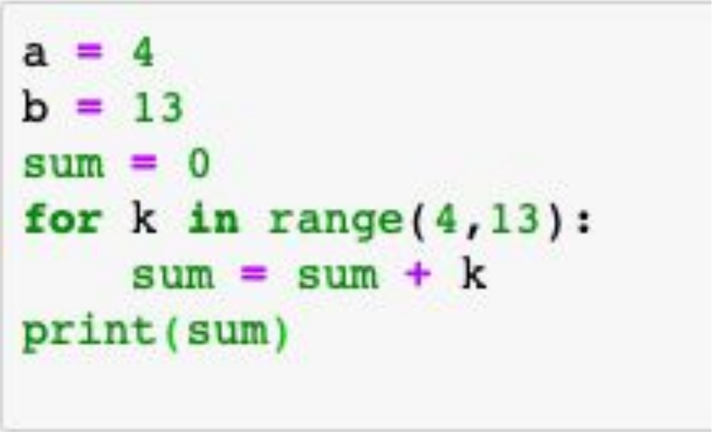
- This indentation has to be respected precisely to run the code, otherwise an error message will be received.

TheAIInstitute

if, for and while

For function :

Repeat a block of code according to a given criteria.

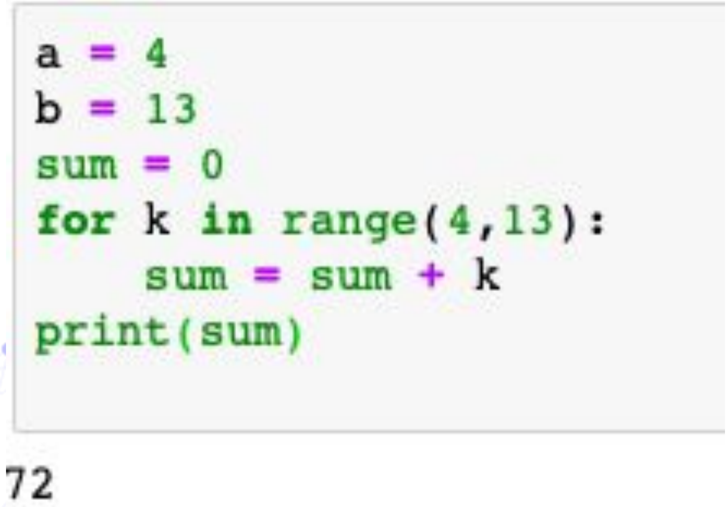


```
a = 4
b = 13
sum = 0
for k in range(4,13):
    sum = sum + k
print(sum)
```

if, for and while

For function :

Repeat a block of code according to a given criteria.



```
a = 4
b = 13
sum = 0
for k in range(4,13):
    sum = sum + k
print(sum)
```

```
72
```

if, for and while

For function :

Executes a set of statements, once for each item in a list, tuple, set etc.

for k in range(m,n) : k takes as value the integers between m included and n excluded.

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
	X = [3,5,2,6]

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
	X = [3,5,2,6]
	Sum = 0

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 3	X = [3,5,2,6]
	Sum = 0

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
➡       Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 3	X = [3,5,2,6]
Y = 9	Sum = 0

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 3	X = [3,5,2,6]
Y = 9	Sum = 9

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 5	X = [3,5,2,6]
Y = 25	Sum = 9

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 5	X = [3,5,2,6]
Y = 25	Sum = 34

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 2	X = [3,5,2,6]
Y = 4	Sum = 34

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 2	X = [3,5,2,6]
Y = 4	Sum = 38

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 6	X = [3,5,2,6]
Y = 36	Sum = 38

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
print(Sum)
```

Local Variables	Global Variables
k = 6	X = [3,5,2,6]
Y = 36	Sum = 74

TheAIInstitute

if, for and while

Example

```
X = [3,5,2,6]
Sum = 0
for k in X :
        Y = k ** 2
        Sum += Y
➡ print(Sum)    → 74
```

Local Variables	Global Variables
	X = [3,5,2,6]
	Sum = 74

If, for and while

If function :

Use a boolean to decide if a block of code is read or not.

Bool = *True* : the block is read
Bool = *False* : the block is skipped

Syntaxe :

```
if bool :
        …
        …
…
```

If/elif/else

Example 1 :

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

If/elif/else

Example :

```
a = 200
b = 33
if b > a:
        print(“b is greater than a")
elif a == b:
        print(“a and b are equal")
else:
        print(“a is greater than b")
```

If/elif/else

Example :

```
a = 200
b = 33
if b > a:
        print("b is greater than a")
elif a == b:
        print("a and b are equal")
else:
        print("a is greater than b")
```

If/elif/else

Example :

```
a = 200
b = 33
if b > a:
        print(“b is greater than a”)
elif a == b:
        print(“a and b are equal”)
else:
        print(“a is greater than b”)
```

If/elif/else

Example :

```
a = 200
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

If/elif/else

Example :

```
a = 200
b = 33
if b > a:
        print(“b is greater than a”)
elif a == b:
        print(“a and b are equal”)
else:
        print(“a is greater than b”)   → “a is greater than b”
```

If/elif/else

Example :

```
a = 20
b = 33
if b > a:
    print("b is greater than a")
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

If/elif/else

Example :

```
a = 20
b = 33
if b > a:
        print(“b is greater than a")
elif a == b:
        print(“a and b are equal")
else:
        print(“a is greater than b")
```

If/elif/else

Example :

```
a = 20
b = 33
if b > a:
        print(“b is greater than a”)
elif a == b:
        print(“a and b are equal”)
else:
        print(“a is greater than b”)
```

If/elif/else

Example :

```
a = 20
b = 33
if b > a:
    ➡print("b is greater than a")   → "b is greater than a"
elif a == b:
    print("a and b are equal")
else:
    print("a is greater than b")
```

If/elif/else

Example :

```
a = 20
b = 33
if b > a:
        print(“b is greater than a”)
elif a == b:
        print(“a and b are equal”)
else:
        print(“a is greater than b”)
```

If/elif/else

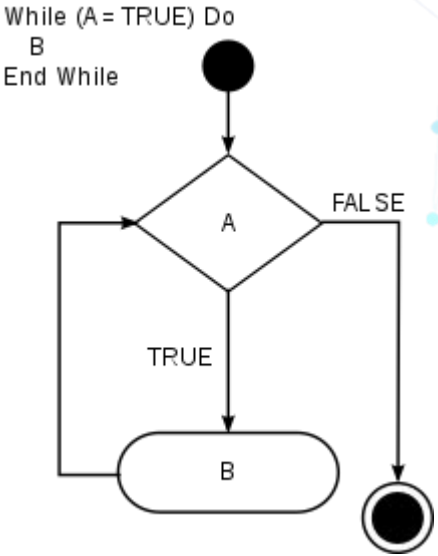
Example :

```
a = 20
b = 33
if b > a:
    print(“b is greater than a”)
elif a == b:
    print(“a and b are equal”)
else:
    print(“a is greater than b”)
```

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```



While loop

❖ *While* loop allows to repeat a set of statements as long as the condition is *True*.

➢ Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 1

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 1

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

        print(i)  → 1

        i += 1
```

i = 1

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - ➢ Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 2

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 2

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)  → 2

    i += 1
```

i = 2

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 3

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

        print(i)

        i += 1
```

i = 3

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)  → 3

    i += 1
```

i = 3

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 4

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 4

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

        print(i)  → 4

        i += 1
```

i = 4

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 5

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

i = 5

While loop

- *While* loop allows to repeat a set of statements as long as the condition is *True*.
  - Exemple: Print all the integers between i and 5.

```
i = 1

while i < 5:

    print(i)

    i += 1
```

- Remember to increment the i, or else the loop will continue forever

While loop: The break statement

- We can stop the loop with the “break” statement even if the while condition is true.
  - Exemple: exit the loop when i is 4

```
i = 1

while i < 8:

    print(i)

    if i == 4:

        break

    i += 1
```

Python: Module

- Modules are Python programs that contain functions that are often reused, also called libraries.

- Python developers have developed many modules that perform a phenomenal amount of tasks. For this reason, always take the reflex to check if some of the code you want to write does not already exist under form of module.
- Most of these modules are already installed in standard Python versions.

Python: Module import

- import give access to all function present in a module
  - *Ex : import* **math**

- Once a library is imported, we use its functions with the command above.
  - **math**.sin(45)

- The name of the imported library can be changed in order to write more easily, by adding the word “as”.

  - Ex : import math as m
    x = m.sin(45)

Python: Module import

- It is also possible to import a function of a library instead of the whole library.
  - Ex : from numpy import sin

    ```
    from numpy import sin
    x = sin(45)
    ```

- Usually, we always start a code with the importation of all the libraries used in the code, instead of importing them in the middle of the code. It gives a better overview of the libraries used in the code.

Main modules

Numpy : Many mathematical function, like cos, sin, pi…

Matplotlib : Graphs, image representation…

Time : Time measurement

Random : Random number generator

Scikit Learn (sklearn) ***:*** Machine Learning algorithms

Pandas : Data manipulation (data frames…)

Keras : Neural Networks tools

NumPy

- The NumPy module allows you to perform calculations on vectors or matrices, element by element, via a new type of object called array.
- The NumPy module is loaded with the command:

```
>>> import numpy
```

- Usually, we import it under the short name ***np*** :

```
>>> import numpy as np
```

NumPy : Arrays

- Arrays are used to store multiple values in one single variable:
  - Example: Create an array containing fruit name:

```
fruits = ["banana", "ananas", "apple"]
```

- An array can hold many values under a single name, and you can access the values by referring to an index number.
- Python does not have built-in support for Arrays, but NumPy does
- You refer to an array element by referring to the *index number*.

NumPy : Arrays

- Arrays are used to store multiple values in one single variable:
  - Example: Create an array containing fruit name:

```
fruits = ["banana", "ananas", "apple"]
```

- An array can hold many values under a single name, and you can access the values by referring to an index number.
- Python does not have built-in support for Arrays, but NumPy does
- You refer to an array element by referring to the *index number*.

➢ **“But what is the difference with a list ?”**

NumPy : Arrays

- Arrays are similar to vectors :

  - Term by term sum
  - Scalar product
  - Matrix product
  - Eigenvectors, eigenvalues…

- Even if their structure looks similar to lists, they work a very different way.

  - No concatenation for arrays
  - No product for lists

NumPy : Arrays

❖ Arrays can be created from lists, using the numpy “array” function.

➢ Ex :

```
import numpy as np
List = [4, 2, -34.2, 36]
Array = np.array(List)
print(Array)
```

❖ A matrix can be defined from a list of lists. Each sublist must have the same dimension.

➢ Ex :

```
import numpy as np
list = [[4,2,9],[23.2, 14, -2],[3 - 7j, -0.3, 4.]]
M = np.array(list)
print(M)
```

NumPy : Arrays

- ❖ Arrays can be created from lists, using the numpy “array” function.
  - ➢ Ex : import numpy as np

```
List = [4, 2, -34.2, 36]
Array = np.array(List)
print(Array) → [ 4.   2.  -34.2  36. ]
```

- ❖ A matrix can be defined from a list of lists. Each sublist must have the same dimension.
  - ➢ Ex : import numpy as np

```
list = [[4,2,9],[23.2, 14, -2],[3 - 7j, -0.3, 4.]]
M = np.array(list)
print(M)
```

NumPy : Arrays

- Arrays can be created from lists, using the numpy “array” function.
  - Ex : import numpy as np

    ```
    List = [4, 2, -34.2, 36]
    Array = np.array(List)
    print(Array)  → [  4.    2.  -34.2  36. ]
    ```

    Note : No comma between variables

- A matrix can be defined from a list of lists. Each sublist must have the same dimension.
  - Ex : import numpy as np

    ```
    list = [[4,2,9],[23.2, 14, -2],[3 - 7j, -0.3, 4.]]
    M = np.array(list)
    print(M)
    ```

NumPy : Arrays

- Arrays can be created from lists, using the numpy “array” function.
  - Ex : import numpy as np

```
List = [4, 2, -34.2, 36]
Array = np.array(List)
print(Array)  → [  4.    2.  -34.2  36. ]
```

Note : No comma between variables

- A matrix can be defined from a list of lists. Each sublist must have the same dimension.
  - Ex : import numpy as np

```
list = [[4,2,9],[23.2, 14, -2],[3 - 7j, -0.3, 4.]]
M = np.array(list)
print(M) →     [[ 4. +0.j  2. +0.j  9. +0.j]
                [23.2+0.j 14. +0.j -2. +0.j]
                [ 3. -7.j -0.3+0.j  4. +0.j]]
```

NumPy : Arrays

❖ Arrays can be created from lists, using the numpy “array” function.

```
➢ Ex : import numpy as np
       List = [4, 2, -34.2, 36]
       Array = np.array(List)
       print(Array) → [  4.    2.  -34.2  36. ]
```

Note : No comma between variables

❖ A matrix can be defined from a list of lists. Each sublist must have the same dimension.

```
➢ Ex : import numpy as np
       list = [[4,2,9],[23.2, 14, -2],[3 - 7j, -0.3, 4.]]
       M = np.array(list)
       print(M) →  [[ 4. +0.j  2. +0.j  9. +0.j]
                    [23.2+0.j 14. +0.j -2. +0.j]
                    [ 3. -7.j -0.3+0.j  4. +0.j]]
```

Note : Every element has been converted to the same type

NumPy : Arrays

❖ Example: Access array element

```
>>> import numpy as np

>>> a = np.array( [2, 4, 6, 8])

>>> print("First element:", a[0])

>>> print("Second element:", a[1])

>>> print("Second last element:", a[-1])

>>> print(a+a)
```

NumPy : Arrays

- ❖ Example: Access array element

```
>>> import numpy as np

>>> a = np.array( [2, 4, 6, 8])

>>> print("First element:", a[0])  → 2

>>> print("Second element:", a[1])  → 4

>>> print("Second last element:", a[-2])  → 6

>>> print(a+a)  → [4   8   12   16]
```

TheAIInstitute

Pandas module

- The pandas module has been designed for data manipulation and data analysis. It is particularly powerful for manipulating structured data in tabular form.
- To load pandas into Python memory, we use the usual import command:

```
>>> import pandas
```

- Pandas is often loaded with the shortened name pd :

```
>>> import pandas as pd
```

Pandas module

- The pandas module has been designed for data manipulation and data analysis. It is particularly powerful for manipulating structured data in tabular form.

- This module is really appreciated by a lot of data scientists because it is very useful in order to visualize data, manipulate it, implement some algorithms…

- It relies mostly on a new type of data, called dataframes.

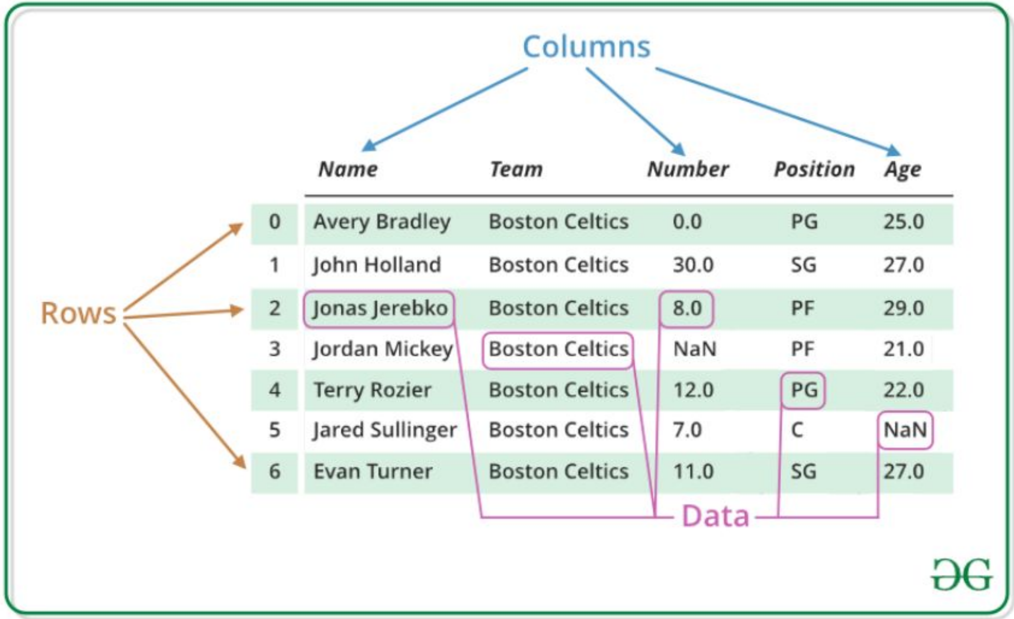
Pandas : Dataframe

- Is a two-dimensional data structure aligned in tabular form in row and column.

- In DataFrame:
  - Column are different on type
  - The size is mutable
  - The rows and columns are labelled
  - We can perform operation on row and column
  - For most databases, each line is a record, and each column corresponds to a feature.

Pandas : Dataframe

- For more info https://www.geeksforgeeks.org/python-pandas-dataframe/



	Name	Team	Number	Position	Age
0	Avery Bradley	Boston Celtics	0.0	PG	25.0
1	John Holland	Boston Celtics	30.0	SG	27.0
2	Jonas Jerebko	Boston Celtics	8.0	PF	29.0
3	Jordan Mickey	Boston Celtics	NaN	PF	21.0
4	Terry Rozier	Boston Celtics	12.0	PG	22.0
5	Jared Sullinger	Boston Celtics	7.0	C	NaN
6	Evan Turner	Boston Celtics	11.0	SG	27.0

Pandas : Dataframe

- A pandas dataframe can be created using the following instruction:

>>> *pandas.DataFrame(data, index, columns, dtype, copy).*

data : Ndarray (structured or homogeneous), Iterable, dict, or DataFrame

index : Index to use for resulting frame.

columns :Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, …, n)

dtype : Data type to force. If None, infer.

copy : Boolean, Copy data from inputs.

Pandas : Dataframe

- A dataframe can also be created from a file which was not made on Python. Actually, almost any kind of database can be imported in Pandas.

- Support formats :
  - txt, csv, JSON, html, Excel, SAS...

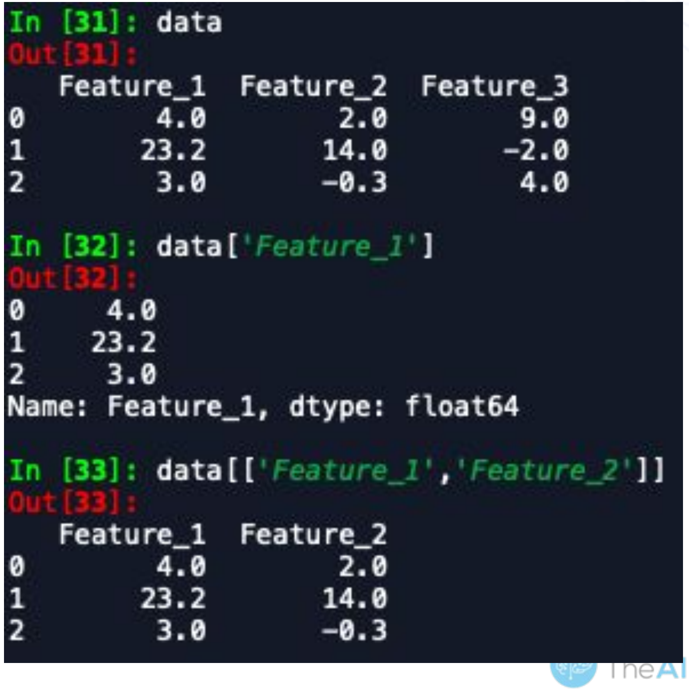
- Specific import functions : read_csv, read_json, read_excel... Pay attention to your input format.

Pandas : Dataframe

A subframe can be created from any dataframe. It can include fewer rows or columns.

➢ Extract some specific columns :

To select some specific columns, dataframes have a similar structure as lists. A column can be selected using [index] at the end of a dataframe name.



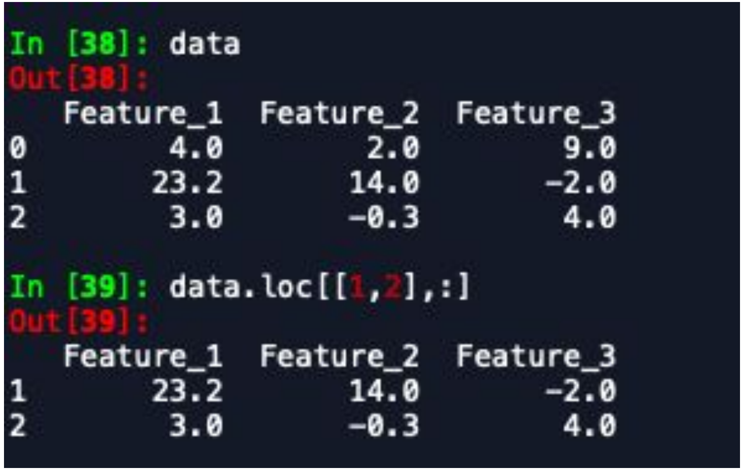
```
In [31]: data
Out[31]:
   Feature_1  Feature_2  Feature_3
0        4.0        2.0        9.0
1       23.2       14.0       -2.0
2        3.0       -0.3        4.0

In [32]: data['Feature_1']
Out[32]:
0     4.0
1    23.2
2     3.0
Name: Feature_1, dtype: float64

In [33]: data[['Feature_1','Feature_2']]
Out[33]:
   Feature_1  Feature_2
0        4.0        2.0
1       23.2       14.0
2        3.0       -0.3
```

Pandas : Dataframe

A subframe can be created from any dataframe. It can include fewer rows or columns.

- ➢ Extract some specific rows :

To select some specific rows, we use the loc[ ] function of dataframes, into which we can indicate the index of the lines we want to keep.



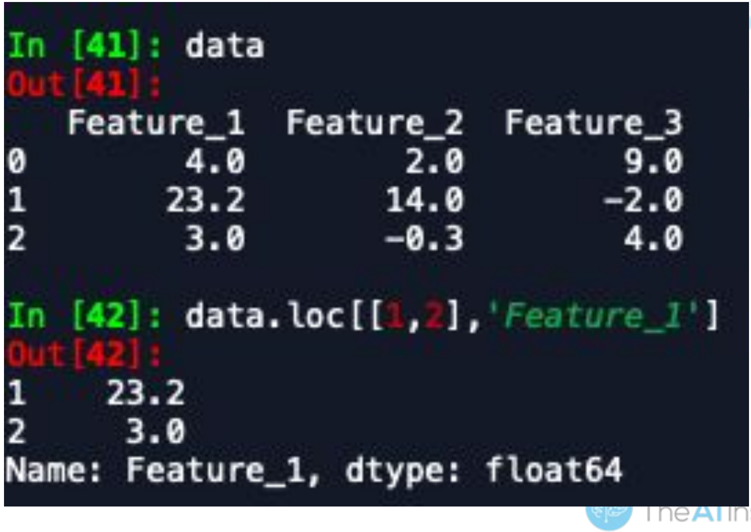
```
In [38]: data
Out[38]:
   Feature_1  Feature_2  Feature_3
0        4.0        2.0        9.0
1       23.2       14.0       -2.0
2        3.0       -0.3        4.0

In [39]: data.loc[[1,2],:]
Out[39]:
   Feature_1  Feature_2  Feature_3
1       23.2       14.0       -2.0
2        3.0       -0.3        4.0
```

Pandas : Dataframe

A subframe can be created from any dataframe. It can include fewer rows or columns.

➢ Extract some specific rows and columns :

An extra parameter can be entered in the loc[ ] function, allowing to select at the same time the rows and features we want to preserve.



```
In [41]: data
Out[41]:
   Feature_1  Feature_2  Feature_3
0        4.0        2.0        9.0
1       23.2       14.0       -2.0
2        3.0       -0.3        4.0

In [42]: data.loc[[1,2],'Feature_1']
Out[42]:
1    23.2
2     3.0
Name: Feature_1, dtype: float64
```

Pandas : Dataframe

❖ More Pandas functions will be described in the ***Writing Python*** part of the course.

❖ Pandas has been developed in order to be compatible with other libraries. Therefore, we can use machine learning algorithms directly on dataframes with libraries such as Scikit Learn.