

CURSO PYTHON PARA DATASCIENCE

• Indice del curso

■ Module 1 - Python basics

- Your first program
- Types
- Expressions and Variables
- String Operations

■ Module 2 - Python Data Structures

- Lists and tuples
- Sets
- Dictionaries

■ Module 3 - Python Programming Fundamentals

- Conditions and branching
- Loops
- Functions
- Objects and Classes

■ Module 4 - Working with Data in Python

- Reading files with open
- Writing files with open
- Loading data with Pandas
- Working with and Saving data with Pandas.

■ Module 5 - Working with Numpy Arrays & Simple APIs

- Numpy 1D Arrays
- Numpy 2D Arrays
- Simple APIs
- API Setup

MODULE 1 - PYTHON BASICS

• Your first program

`print("Hello World")` → Hello World

`print("Hello\nWorld")` → Hello
World

`# this is a comment` → comentario

• Types of objects

`type(1)` → int

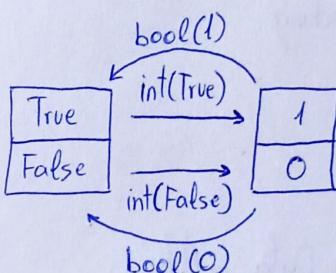
`type(21,213)` → float

`type("Hello Python")` → str (string)

- Cambiar tipo (type casting): `float(2)`: 2.0 `int("1")`: 1
`int(1,1)`: error `int("A")`: error

- Boolean → True / False

`type(True)` → bool
Importante:
mágos



• Expressions and Variables

- Operaciones matemáticas: - + * / (división float) // (división integer)

- Variables:

+ Definición/redefinición de variables: `my_variable = 10`

• String operations

`Name = "Michael Jackson"`

M	i	c	h	a	e	l	J	a	c	k	s	o	n
0	1	2	3	4	5	6	7	8	9	10	11	12	13
-15	-14	-13	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2

`print(Name[2])` → i

`print(Name[-4])` → k

`print(Name[0:4])` → Mich (último número no se escribe)

`print(Name + " is the best")` → Michael Jackson is the best

`print(Name[::2])` → "McaJsn" (cada 2 letras)

`print(Name[0:5:2])` → "Mca" (valores de 0 a 5 cada 2 letras)

`len(Name)` → 15 (tamaño array / n° letras)

`print(3 * Name)` → "Michael Jackson Michael Jackson Michael Jackson"

- Escape sequences:

\n → nueva linea (enter)

\t → tabulador

- String methods:

A = "Michael Jackson"

B = A.upper() → B = "MICHAEL JACKSON"
method

B = A.replace("Michael", "Janet") → B = "Janet Jackson"

A.find("el") → 5 (posición de la primera letra de la secuencia)

A.find("Jack") → 8

A.find("&") → -1 (no está)

MODULE 2 - PYTHON DATA STRUCTURES

• Lists and Tuples ^{→ arrays}

- Tuples → secuencia ordenada

Pueden tener cualquier tipo de variable, su tipo es tuple

tuple1 = ("disco", 10, 1.2)

type(tuple1) = tuple

Igual que en los strings, cada posición tiene asignada un índice, empezando por el 0. (0 -)

+ Inmutabilidad:

Los tuples son inmutables → no puedo cambiar un dato del tuple únicamente (~~tuple[3] = 5~~)

Tengo que cambiar el tuple entero (redefinirlo)

+ Nesting: puedo tener tuples dentro de los tuples

NT = (1, 2, ("pop", "rock"), (3, 4), ("disco", (1, 2)))

NT[2] = ("pop", "rock")

NT[2][1] = "rock"

-Lists: exactamente igual que tuples, solo que son modificables y van entre corchetes
(mutables)

$L = ["Michael Jackson", 10.1, 1928]$

$L.append(["pop", 10]) \rightarrow L = ["Michael Jackson", 10.1, 1982, "pop", 10]$

$L.append(["pop", 10]) \rightarrow L = ["Michael Jackson", 10.1, 1982, ["pop", 10]]$

$del(L[0]) \rightarrow L = [10.1, 1928]$

"hard rock".split() $\rightarrow ["hard", "rock"]$

"A,B,C,D".split(",") $\rightarrow ["A", "B", "C", "D"]$ (separa los elementos separados por comas)

-Mutabilidad

$A = ["hard rock", 10, 1.2]$

$B = A \rightarrow$ si cambio algo de A, cambio algo de B (xq referencia la misma lista)

$B = A[:]$ \rightarrow creo en B una copia de A, asi que si cambio algo en A, no cambio nada en B

help(A) \rightarrow + info

• Sets

-Tipo de colección de datos

-A diferencia de los tuples y las listas, estos no están ordenados

-Tienen elementos únicos \rightarrow en un set solo hay uno de cada elemento particular.

-Crear un set

$Set1 = \{"pop", "rock", "soul", "hard rock", "rock", "R&B", "rock", "disco"\}$

$\rightarrow Set1 = \{"rock", "R&B", "disco", "hard rock", "pop", "soul"\}$

Puedo convertir una lista en set usando set():

$album_list = ["Michael Jackson", "Thriller", "Thriller", 1982]$

$album_set = set(album_list)$

$album_set = \{"Michael Jackson", "Thriller", 1982\}$

- Operaciones con sets

$A = \{\text{"Thriller"}, \text{"Back in Black"}, \text{"AC/DC"}\}$

$A.add(\text{"NSYNC"}) \rightarrow$ Te imaginas lo que hace

$A.remove(\text{"NSYNC"}) \rightarrow$ Lo quito

$\text{"AC/DC"} \text{ in } A \rightarrow \text{True}$

$\text{"Who"} \text{ in } A \rightarrow \text{False}$

Intersección de dos sets \rightarrow set1 & set2
(dentro de un nuevo set)

Unión de sets \rightarrow set1, unión (set2)
(set con todos los elementos de los 2 sets)

- Subsets

set-1 = $\{\text{"AC/DC"}, \text{"Back in Black"}, \text{"Thriller"}\}$

set-2 = $\{\text{"AC/DC"}, \text{"Back in Black"}\}$

set-2.issubset(set-1) \rightarrow True

• Dictionaries

Asigna ^{key} valores a unas "Keys" como si se tratase de los índices de un tuple

$A = \{\text{"Key1": 1, "Key2": 2, "Key3": [3,3,3], "Key4": (4,4,4), ("Key5"): 5}\}$

$A[\text{"Key3"}] = [3,3,3]$

Puedo crear un nuevo elemento en el diccionario: $A[\text{"Key6"}] = 666$
y eliminarlo del ($A[\text{"Key6"}]$)

$\text{"Key1"} \text{ in } A \rightarrow \text{True}$

$\text{"Key7"} \text{ in } A \rightarrow \text{False}$

$A.keys() \rightarrow$ Escribe todas las keys $\rightarrow [\text{"Key 1"}, \text{"Key 2"}, \text{"Key 3"}, \text{"Key 4"}, \text{"Key 5"}]$

$A.values() \rightarrow$ Lo mismo para los valores

"Thriller", "Back in Black"
"AC/DC".

MODULE 3 - PYTHON PROGRAMMING FUNDAMENTALS

• Conditions and Branching

<code>==</code>	igual	}	Comparison operators
<code></></code>	mayor(menor)		
<code></>=</code>	mayor igual		
<code>!=</code>	no igual (menos)		

valen para strings

- Branching

+ The if Statement

no hacen falta los paréntesis

```
if (age > 18)
    print("You can enter")
else:
    print("go see Meat Loaf")
```

→ `elseif(age == 18)`
`print("go see Pink Floyd")`

+ Logic operators

`not(False/True)` → True/False

or, and

• Loops

range(N) → [0, ..., N-1] range(3) → [0, 1, 2]
 range(integer) → sequence range(10, 15) → [10, 11, 12, 13, 14]

- for loops → ejecutan un statement un número determinado de veces

squares	red	yellow	green	purple	blue	→ lista
index	0	1	2	3	4	

Quiero sustituir todos los colores por white →

`for i in range(0, 5)`

`squares[i] = "white"`

`squares = ["red", "yellow", "green"]`

`for square in squares:`

`print(square)`

→	"red"
	"yellow"
	"green"

`for i, square in enumerate(squares):`

`print(square)`

`print(i)`

square	i
red	0
yellow	1
green	2

- While loops → ejecutan un statement siempre que se cumpla una condición

squares = ["orange", "orange", "purple", "blue"]

NewSquares = []

i=0

while (squares[i] == "orange"):

NewSquares.append(squares[i])

i=i+1 (o i+=1)

→ NewSquares = ["orange", "orange"]

• Functions:

def f1(input):

"""add 1 to input"""\n output = input + 1

return output

help(f1) → Help on function f1 in module_main_: f1(input) add 1 to input"

f1(5) = 6

- Funciones

+ len → tamaño de la lista

+ sum → devuelve la suma de todos los números de una lista

- Funciones con parámetros múltiples:

def Mult(a,b)

c = a * b

return c

2 * "Michael Jackson"

Mult(2,3) → 6 ; Mult(2, "Michael Jackson") → "Michael Jackson" "Michael Jackson"

- Funciones sin return (ni input tampoco)

{ def MJ():

print("Michael Jackson")

} MJ() → "Michael Jackson"

{ def NoWork():

pass → Key word para no poner
 cuerpo vacío (comando inútil)

print(NoWork()) → None

def add1(a):

$$b = a + 1$$

print(a, "plus1 equals", b) → add1(2) → $\frac{2}{3}$ plus1 equals 3

return b

• def printStuff(stuff):

for i, s in enumerate(stuff):
 print("Album", i, "Rating is", s)

album_ratings = [10.0, 8.5, 9.5]
printStuff(album_ratings)

→ Album0 Rating is 10
Album1 Rating is 8.5
Album2 Rating is 9.5

def ArtistNames(names)

for name in names:
 print(name)

• Objects and Classes

Types | - integer
 | - float
 | - string
 | - List
 | - Dictionary
 | - Boolean

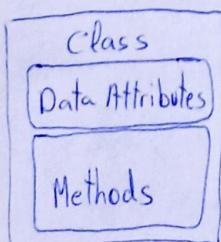
⇒ CADA TIPO ES UN OBJETO

Un objeto es un integer en concreto: dentro del tipo integer puedes tener varios objetos de tipo integer. Lo mismo con el resto de tipos o "classes".

- Methods: cómo interactúas con los datos en un objeto.

.sort()
.reverse()

- Crear un tipo(type) o clase(class)



Class Circle (object): → constructor (método especial para)

def __init__(self, radius, color): initializes class data
 self.radius = radius
 self.color = color

```
def __init__(self, color, height, width):  
    self.height = height  
    self.width = width  
    self.color = color
```

-Create an object of class circle:

Red Circle = Circle(10, "red")
nombre objeto

RedCircle.radius → 10 | RedCircle.color = "blue" → redefino el color del objeto
RedCircle.color → "red"

Para cambiar los datos de un objeto definimos los llamados métodos:

```
class Circle(object):
```

```
def __init__(self, radius, color):
```

$$\text{self-radius} = \text{radius}$$

self-color = color

def add_radius(self, r):

$$\text{self_radius} = \text{self_radius} + r$$

} Método para sumar r al radio

④ Para imprimir el atributo de un objeto no hace falta usar print:

Red Circle. radius → 10

Para usar el método add_radius:

RedCircle.addRadius(2) → RedCircle.radius = 12

MODULE 4 - WORKING WITH DATA IN PYTHON

- Reading files with open

Open \rightarrow función

Nuevo tipo de objeto → file (archivo del directorio)

File l-name → "/resources/data/Example2.txt"

File1.mode → "r"

`File.close()` → cierra el archivo (hay que hacerlo siempre)

Para no tener que cerrar el archivo manualmente se usa with:

with open("Example2.txt", "r") as File1:

file_stuff = File1.read() → copia en file_stuff el contenido del archivo

print(file_stuff) → se cierra el archivo automáticamente

print(File1.closed) → compruebo que está cerrado

print(file_stuff) → no puedo interactuar con el archivo fuera del with, pero sí con la variable en la que he copiado su contenido.

Ejemplo:

file_stuff = This is line 1
This is line 2
This is line 3 → Realmente: file_stuff = This is line1\nThis is line2\nThis is line3

Esto se puede disponer más cómodamente en una lista:

with open("Example2.txt", "r") as File1:

file_stuff = File1.readlines()

print(file_stuff)

→ file_stuff: ["This is line 1\n", "This is line2\n", "This is line3\n"]
file_stuff[0] file_stuff[1] file_stuff[2]

- readline()

with open("Example2.txt", "r") as File1:

file_stuff = File1.readline() → guarda 1ª línea del archivo

print(file_stuff)

file_stuff = File1.readline() → guarda 2ª línea del archivo

print(file_stuff)

with open("Example2.txt", "r") as File1:

file_stuff = File1.readlines(4) → Primeras 4 casillas del archivo ("This")

print(file_stuff)

file_stuff = File1.readlines(5) → Siguientes 5 casillas (" - is l")

print(file_stuff)

• Writing files with open

File1 = open("/resources/data/Example2.txt", "w")

with open("/resources/data/Example2.txt", "w") as File1:

File1.write("This is line A\n")
File1.write("This is line B\n") → This is line A
This is line B

Usando listas:

Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]
with open("/resources/data/Example2.txt", "w") as File1:

for line in Lines:
 File1.write(line)

with open("/resources/data/Example2.txt", "a") as File1:

File1.write("This is line C") → This is line A
This is line B
This is line C

| w → reescribe, o escribe de 0 en un archivo vacío
| a → añade contenido a un archivo en el que ya hay contenido

-Copiar un archivo en otro:

with open("Example1.txt", "r") as readfile:

with open("Example3.txt", "w") as writefile:

for line in readfile:
 writefile.write(line)

• Loading Data with Pandas:

Pandas → library for data analysis → pre-built classes and functions
(Vídeo descargado)

Archivos | ^{CSV}
 | Excel

- Working with and saving data with Pandas

(Video descargado)

MODULE 5: WORKING WITH NUMPY ARRAYS & SIMPLE APIs

- Numpy 1D Arrays

- The basics and Array Creation:

Numpy 1D Array → Lista con todos sus elementos de la misma clase

import numpy as np

a = np.array([0, 1, 2, 3, 4])

type(a) → numpy.ndarray ; a.dtype → dtype("int64")

a.size → 5

a.ndim → 1

a.shape → (5,)

- Indexing and slicing

c = np.array([20, 1, 2, 3, 4])

c[0] = 100

c → array([100, 1, 2, 3, 4]) ¡COMO UNA LISTA!

d = c[1:4]

d → array([1, 2, 3])

c[3:5] = 300, 400

c → array([100, 1, 2, 300, 400])

- Basic operations: u = [1, 0] ; v = [1, 0]

+ Suma (como vectores en mates)

Resta

Si usasemos listas → for n, m in zip(u, v):
z.append(n+m)

| z = u + v (+ fácil)
(con Numpy)

+ Multiplicación

z = 2 * u

+ Producto de dos arrays Numpy (Producto de Hadamard)

$$Z = U \circ V = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \text{ (Resultado = vector/array 1D)}$$

$$Z = U^* V$$

+ Dot product

$$Z = U^T V = 1 \cdot 1 + 0 \cdot 0 = 1$$

Programando: $Z = np.dot(U, V)$

+ Adding constant to an numpy Array

$$U = np.array([1, 2, 3, -1])$$

$$Z = U + 1 ; Z \rightarrow [2, 3, 4, 0]$$

- Universal Functions:

$$a = np.array([1, -1, 1, -1])$$

$$\underline{\underline{mean_a}} = a.mean() \rightarrow \underline{\underline{mean_a}} \rightarrow \frac{1}{4}(1-1+1-1) = 0$$

$$b = np.array([1, -2, 3, 4, 5])$$

$$\underline{\underline{max_b}} = b.max() \rightarrow 5$$

$$np.pi \rightarrow \pi$$

$$x = np.array([0, np.pi/2, np.pi]) \rightarrow [0, \frac{\pi}{2}, \pi]$$

$$y = np.sin(x) \rightarrow [\sin(0), \sin(\frac{\pi}{2}), \sin(\pi)] \rightarrow [0, 1, 0]$$

np.linspace(-2, 2, num=5) eje equiespaciado
inicio final n° espaciados

-2	-1	0	1	2
1	2	3	4	5

util para representar una función

Numpy 2D Arrays

$a = [[11, 12, 13], [21, 22, 23], [31, 32, 33]]$ (List)

$A = \text{np.array}(a) \rightarrow A \rightarrow \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$ (Array)

$A[0][0] \rightarrow A[1][2] \rightarrow A[2][2]$

$A.\text{ndim}: 2 \rightarrow$ list inside a list
 \hookrightarrow nesting $\rightarrow \text{dim}=2$

$A.\text{shape}: (3, 3)$

$A.\text{size}: 9$ (nº elementos)

- Suma de arrays \rightarrow igual que matrices

$$X = \text{np.array}([[1, 0], [0, 1]]) \quad X = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$Y = \text{np.array}([[2, 1], [1, 2]]) \quad Y = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$$

$$Z = X + Y : Z \rightarrow \begin{bmatrix} 3 & 1 \\ 1 & 3 \end{bmatrix}$$

- Multiplicación

$$\text{+ Por una constante: } Z = Z * Y : Z \rightarrow \begin{bmatrix} 4 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\text{+ Por otra matriz } \left| \begin{array}{l} \text{Hadamard: } Z = X * Y : Z \rightarrow \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix} \\ \text{Normal: } Z = \text{np.dot}(X, Y) : Z \rightarrow \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix} \end{array} \right.$$

Simple APIs (Application Program Interface)

- What is an API