

ACÀMICA

TEMA DEL DÍA

# NumPy

NumPy es la primera librería que utilizaremos durante la carrera. Su nombre viene de Numerical Python, y sirve para hacer cálculos de forma eficiente con Python usando arrays.



# Agenda

---

Daily

Explicación: Numpy.

**Break.**

Hands-on training: Numpy.

Cierre.



# Daily



Daily



## Sincronizando...

### Toolbox



¿Cómo te ha ido?  
¿Obstáculos?  
¿Cómo seguimos?

### Challenge



¿Cómo te ha ido?  
¿Obstáculos?  
¿Cómo seguimos?

# Repaso de la toolbox



# Operaciones lógicas

Un tipo importante de operación en programación son las **operaciones lógicas**. Estas pueden realizarse sobre **variables booleanas**.

```
In [27]: variable_1 = True  
         variable_2 = False  
         print(variable_1 or variable_2)
```

True

```
In [28]: print(not(variable_1))
```

False

El resultado es también una **variable booleana**.

A	B	A & B
False	False	False
False	True	False
True	False	False
True	True	True

A	B	A or B
False	False	False
False	True	True
True	False	True
True	True	True

A	A!
False	True
True	False



# CONDICIONALES - if

Los **condicionales** son bloques de código que se ejecutan únicamente si se cumple una condición. El resultado de esta condición debe ser un **Booleano** (True o False). Esto se logra mediante el condicional **if**.

```
[10]: valor = 5
      if valor > 10:
          print('El valor es mayor que 10')
```

5 > 10

False

No se cumple la condición.

```
[11]: valor = 15
      if valor > 10:
          print('El valor es mayor que 10')
```

15 > 10

True

El valor es mayor que 10

Se cumple la condición.





# CONDICIONALES - if / else

Además uno puede agregar un código que se ejecute si la condición no se cumple. Para esto se utiliza el condicional **else**.

```
In [77]: nombre = 'Pedro'

if nombre == 'Juan':
    print('Esta persona se llama Juan')
else:
    print('Esta persona NO se llama Juan')
```

Esta persona NO se llama Juan

```
'Pedro' == 'Pedro'
```

True

```
'Juan' == 'Pedro'
```

False

La comparación entre strings también genera un booleano.

*Nota: Para condicionales usamos doble igual ==, ya que nos reservamos el igual simple = para la asignación de variables.*



## CONDICIONALES - if / elif / else

Además del **if** y el **else**, uno puede agregar más condiciones a través de condicional **elif** (else if). De esta forma se puede agregar un número arbitrario de condiciones.

```
In [80]: edad = 20

if edad < 18:
    print('Esta persona tiene menos de 18 años')
elif edad > 18:
    print('Esta persona tiene mas de 18 años')
else:
    print('Esta persona tiene justo 18 años')
```

Esta persona tiene mas de 18 años



# Cuando Python no alcanza



# Instrucciones incómodas

Recordemos que hay ciertas cosas que a Python no le gustan:

```
[1]: numeros = [0,1,2,3,4,5,6,7,8,9]
      print(numeros + 3)
```

```
-----
TypeError                                Traceback (most recent call last)
<ipython-input-1-8b044cee7884> in <module>
      1 numeros = [0,1,2,3,4,5,6,7,8,9]
----> 2 print(numeros + 3)

TypeError: can only concatenate list (not "int") to list
```

# Instrucciones incómodas

¿Cómo podríamos arreglarlo?

```
[2]: numeros = [0,1,2,3,4,5,6,7,8,9]
    resultados = []
    for numero in numeros:
        resultados.append(numero + 3)
    print(resultados)

[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

# Instrucciones incómodas

¿Cómo podríamos arreglarlo?

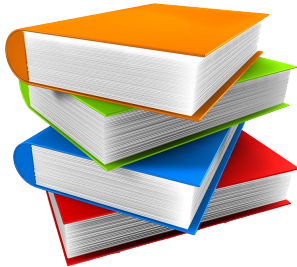
```
[2]: numeros = [0,1,2,3,4,5,6,7,8,9]
    resultados = []
    for numero in numeros:
        resultados.append(numero + 3)
    print(resultados)

[3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

**Pero es muy incómodo...**

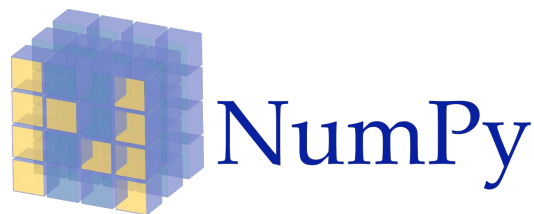
# Librerías

A veces, las estructuras de datos que vienen con Python - y sus funcionalidades asociadas - no son suficientes. Para eso necesitamos usar **Librerías**.



# NumPy

Nuestra primera librería:



- Fundamental para hacer cálculo numérico con Python
- Muy buena [documentación](#)
- Como muchas librerías, trae una estructura de datos propia: los **arrays** o arreglos.



# NumPy: arrays

**array:** a primer orden, es como una **lista**. De hecho, se pueden crear a partir de una lista.

Importamos la librería  
(*numpy*) y le ponemos un  
nombre (*np*)

```
[1]: import numpy as np
```

```
arreglo = np.array([1,2,3,4,5])  
arreglo
```

Es una lista

```
[1]: array([1, 2, 3, 4, 5])
```

```
[2]: print(arreglo)
```

```
[1 2 3 4 5]
```

# NumPy: arrays

Si bien lo creamos a partir de una lista, tiene muchas más funcionalidades:

```
[1]: import numpy as np
```

```
arreglo = np.array([1,2,3,4,5])  
arreglo
```

```
[1]: array([1, 2, 3, 4, 5])
```

```
[2]: print(arreglo)
```

```
[1 2 3 4 5]
```

```
[3]: arreglo + 2
```

```
[3]: array([3, 4, 5, 6, 7])
```

¡Anduvo!

# NumPy: arrays

## Formas de crear arreglos de numpy

- Ya vimos a partir de una lista

```
[1]: import numpy as np  
  
arreglo = np.array([1,2,3,4,5])  
arreglo
```

# NumPy: arrays

## Formas de crear arreglos de numpy

- Ya vimos a partir de una lista
- ¿Qué hace np.arange()?

```
[1]: import numpy as np  
  
arreglo = np.array([1,2,3,4,5])  
arreglo
```

# NumPy: arrays

## Formas de crear arreglos de numpy

- Ya vimos a partir de una lista

```
[1]: import numpy as np

arreglo = np.array([1,2,3,4,5])
arreglo
```

- ¿Qué hace np.arange()? **Arreglo en un rango de valores, de "a saltos".**

```
[4]: arreglo_1 = np.arange(2,9)
arreglo_1

[4]: array([2, 3, 4, 5, 6, 7, 8])
```

```
[6]: arreglo_2 = np.arange(2,9,2)
arreglo_2

[6]: array([2, 4, 6, 8])
```

# NumPy: arrays

## Formas de crear arreglos de numpy

- ¿Qué hace `np.linspace()`?

# NumPy: arrays

## Formas de crear arreglos de numpy

- ¿Qué hace `np.linspace()`? Arreglo equiespaciado

```
[10]: arreglo_3 = np.linspace(2,9,3)  
arreglo_3
```

```
[10]: array([2. , 5.5, 9. ])
```

```
[11]: arreglo_4 = np.linspace(2,9,20)  
arreglo_4
```

```
[11]: array([2.          , 2.36842105, 2.73684211, 3.10526316, 3.47368421,  
            3.84210526, 4.21052632, 4.57894737, 4.94736842, 5.31578947,  
            5.68421053, 6.05263158, 6.42105263, 6.78947368, 7.15789474,  
            7.52631579, 7.89473684, 8.26315789, 8.63157895, 9.          ])
```

Y algunas más que veremos más adelante.

# NumPy: arrays

## Seleccionando elementos de un arreglo:

- Si queremos ver una posición arbitraria:

```
[21]: arreglo = np.arange(2,20,4)  
      arreglo
```

```
[21]: array([ 2,  6, 10, 14, 18])
```

```
[22]: print(arreglo[0], arreglo[2], arreglo[-1], arreglo[-4])  
      2 10 18 6
```



# NumPy: arrays

## Seleccionando elementos de un arreglo:

- Si queremos ver una posición arbitraria:

```
[21]: arreglo = np.arange(2,20,4)  
      arreglo
```

```
[21]: array([ 2,  6, 10, 14, 18])
```

```
[22]: print(arreglo[0], arreglo[2], arreglo[-1], arreglo[-4])  
      2 10 18 6
```

- Y si queremos rangos:

# NumPy: arrays

## Seleccionando elementos de un arreglo:

- Si queremos ver una posición arbitraria:

```
[21]: arreglo = np.arange(2,20,4)
      arreglo
```

```
[21]: array([ 2,  6, 10, 14, 18])
```

```
[22]: print(arreglo[0], arreglo[2], arreglo[-1], arreglo[-4])
      2 10 18 6
```

- Y si queremos rangos:

```
[32]: arreglo = np.arange(0,15)
      arreglo
```

```
[32]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
[33]: arreglo[2:12:2]
```

```
[33]: array([ 2,  4,  6,  8, 10])
```

comienzo

salto

final

# NumPy: arrays

**Seleccionando también podemos asignar:**

```
[34]: arreglo = np.arange(0,15)  
arreglo
```

```
[34]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
[35]: arreglo[2:7] = 25  
arreglo
```

```
[35]: array([ 0,  1, 25, 25, 25, 25, 25,  7,  8,  9, 10, 11, 12, 13, 14])
```

# NumPy: arrays

## Arreglos multidimensionales

“Shape” y “axis” de los arreglos

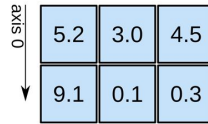
1D array



axis 0 →

shape: (4,)

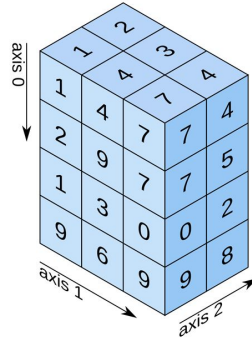
2D array



axis 1 →

shape: (2, 3)

3D array



shape: (4, 3, 2)

No es la forma más cómoda de crearlo

```
[39]: arreglo2d = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
      arreglo2d
```

```
[39]: array([[ 1,  2,  3,  4],
            [ 5,  6,  7,  8],
            [ 0, 10, 11, 12]])
```

```
[40]: arreglo2d.shape
```

```
[40]: (3, 4)
```

filas

columnas

# NumPy: arrays

## Arreglos multidimensionales

```
[42]: arreglo2d = np.arange(100).reshape(10,10)
      arreglo2d
```

```
[42]: array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9],
             [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
             [20, 21, 22, 23, 24, 25, 26, 27, 28, 29],
             [30, 31, 32, 33, 34, 35, 36, 37, 38, 39],
             [40, 41, 42, 43, 44, 45, 46, 47, 48, 49],
             [50, 51, 52, 53, 54, 55, 56, 57, 58, 59],
             [60, 61, 62, 63, 64, 65, 66, 67, 68, 69],
             [70, 71, 72, 73, 74, 75, 76, 77, 78, 79],
             [80, 81, 82, 83, 84, 85, 86, 87, 88, 89],
             [90, 91, 92, 93, 94, 95, 96, 97, 98, 99]])
```

```
[43]: arreglo2d[2:5,::2]
```

```
[43]: array([[20, 22, 24, 26, 28],
             [30, 32, 34, 36, 38],
             [40, 42, 44, 46, 48]])
```

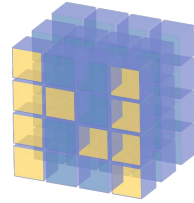
¿Qué está  
haciendo?

A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup sits on a matching white saucer. In the background, a white napkin and a silver fork are visible, though they are out of focus. The overall lighting is soft and even.

**¡BREAK!**

---

Si no instalaste



NumPy

# Numpy: Instalación

1. Activar el ambiente: *"conda activate datascience"*
2. **Instalar** NumPy: *"conda install numpy"*



# Hands-on training



# Trabajamos en el Notebook que descargaste en la Toolbox 03, Sección 2: NumPy

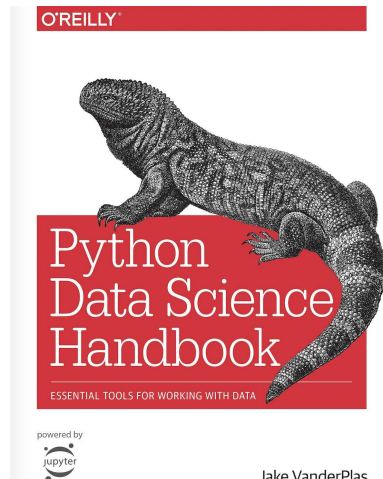


# Recursos




# NumPy

- Su [documentación](#) es muy buena, y cuenta con un [tutorial introductorio](#).
- [Python Data Science Handbook](#) - Capítulo 2, “Introduction to Numpy”.



# Recomendaciones para programar

- 1) Comentar el código en voz alta ayuda a aprender y a entender lo que estás haciendo.
- 2) No tengas miedo de hacer, romper y arreglar.
- 3) La frustración es una buena señal (“Get things done”).
- 4) Pedir la opinión de tus compañeros/as y Leads/as sobre tu código.
- 5) Busca crecer en comunidad (Medium, Github, Slack Stackoverflow, etc).
- 6) Pide ayuda a tu mejor amigo:



Google Search

I'm Feeling Lucky

Search by voice



# Para la próxima

---

- Termina el notebook de hoy
- Lee la Toolbox 04
- Resuelve el Challenge

ACÀMICA