

FUNDACIÓN
YPF

Módulo II | Clase 3

Python para Data
Science: Python
Intermedio -
Introducción a Numpy





¿Ponemos a grabar el
taller?

¿Qué vamos a ver hoy?



- Condicional IF
- Bucle FOR
- Funciones y Métodos
- String Formatting

- Introducción a Numpy

¿Por qué NumPy?

Arrays y sus características

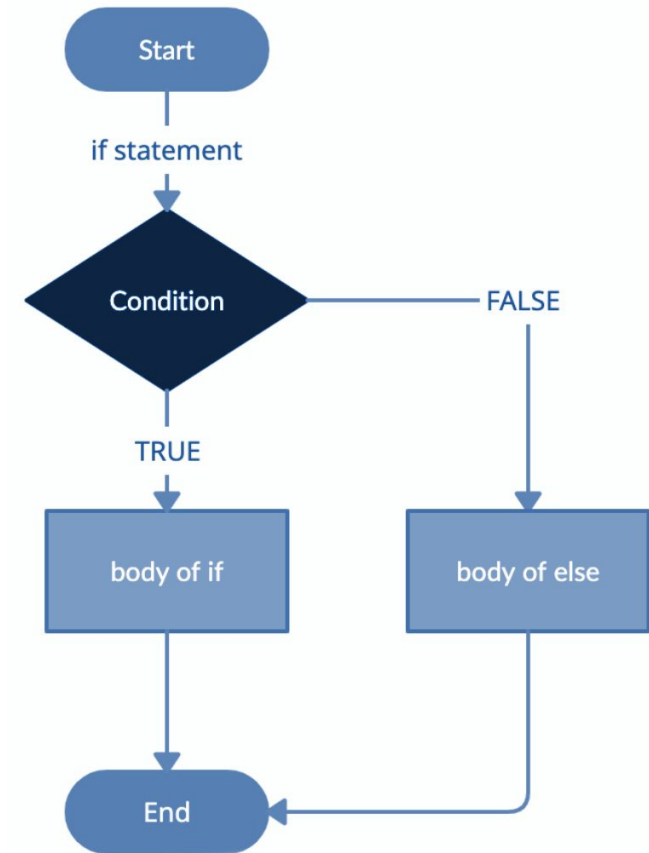
Funciones matemáticas en
Arrays



Estructuras de control de flujo

Control de Flujo

Muchas veces creamos y codeamos un programa que necesita chequear condiciones y analizar qué camino seguir y/o cuántas veces. Para esto, se utilizan las **estructuras de control de flujo**.



Extraído de toppr.com


Condicionales



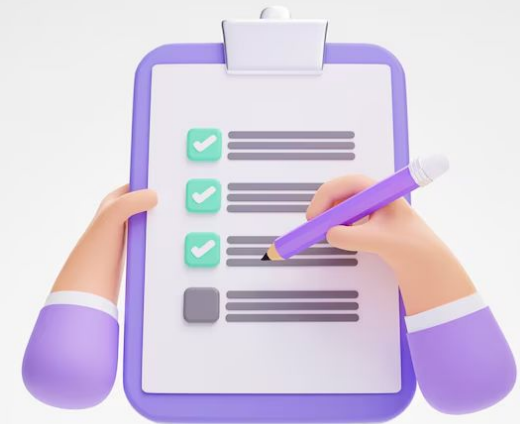
Los **condicionales** permiten comprobar condiciones y modifican el comportamiento de un programa ejecutando un fragmento de código u otro alternativo, dependiendo del **valor de verdad** de la condición.

Sentencia IF

Esta sentencia comprueba si la **condición** establecida es verdadera (**True**) y ejecuta el bloque de código que se encuentra **dentro** del if.



```
if numero_1 == 2:  
    print('Es el numero 2')
```



Sentencia IF-else


Comprueba si la **condición** es **verdadera** (True) y ejecuta el bloque de código dentro del **if**, si **no lo es** (False), ejecuta el bloque de código dentro del **else**.



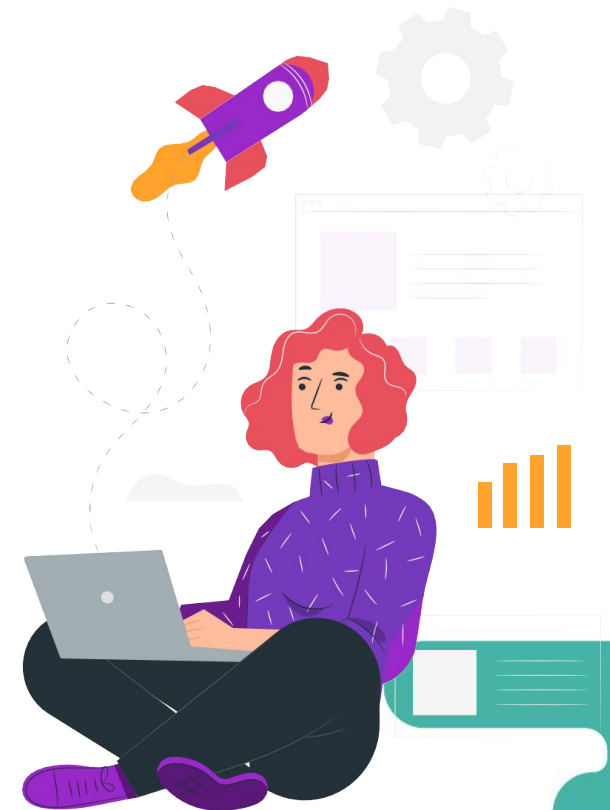
```
if numero_1 == 2:  
    print('Es el numero 2')  
else:  
    print('No es el numero 2')
```


Sentencia IF-elif

Comprueba si la **condición** es **verdadera** (True), de no serlo comprueba la condición presente en **elif** y ejecutará uno u el otro bloque de código. Si ambas son **False** el bloque dentro del **else**.




```
if numero_1 == 2:
    print('Es el numero 2')
elif numero_1 == 3:
    print('Es el numero 3')
else:
    print('No es el numero 2 ni 3')
```



Sentencia IF anidadas

Comprueba paso a paso **condiciones a evaluar**, cada condición anidada **depende** del valor de verdad de la **anterior**.

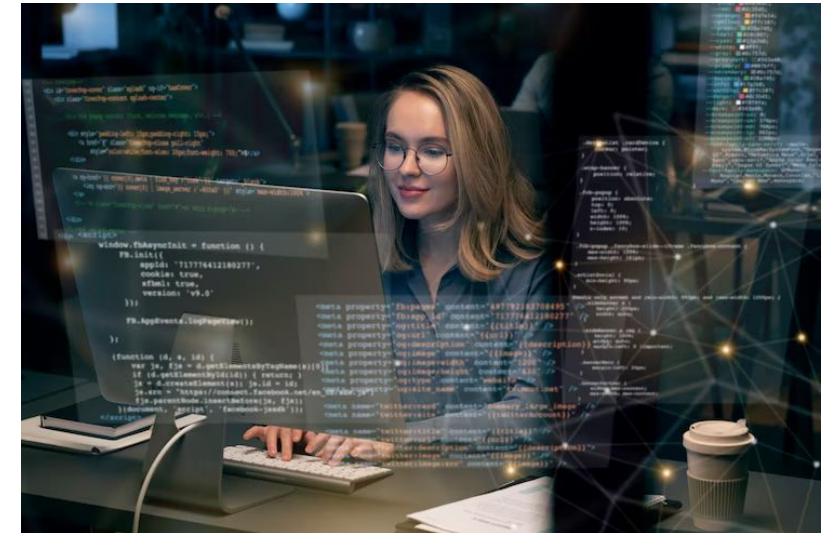


```
if numero_1 != 2:
    if numero_1 > 1:
        print('No es 2 y es mayor a 1')
    else:
        print('No es 2 y es menor a 1')
else:
    print('Es el numero 2')
```



Iterativas

Las sentencias **iterativas** o **bucles** permiten ejecutar un mismo fragmento de código una cierta cantidad de veces. La cantidad de iteraciones depende del valor de verdad de una determinada condición o del tamaño de objetos iterables.



Bucle FOR

Este bucle recorre **elemento a elemento** un objeto iterable y en cada iteración ejecuta el bloque de código establecido.



```
lista = [1, 2, 3]
for i in lista:
    print(i)
```





Funciones y Métodos

Funciones

Una función es un bloque de código que se ejecuta como una unidad funcional, tiene un **nombre específico** y puede ser invocado desde otra parte del código o aplicación tantas veces como se desee.



Funciones en Python

Nombre Argumentos

```
def par(a):  
    if a % 2 == 0:  
        return "Es par"  
    else:  
        return "Es impar"
```

Devuelve un valor y corta la
evaluación de la función

Funciones en Python

nombre: cumple con los mismos requisitos que los identificadores de las variables.

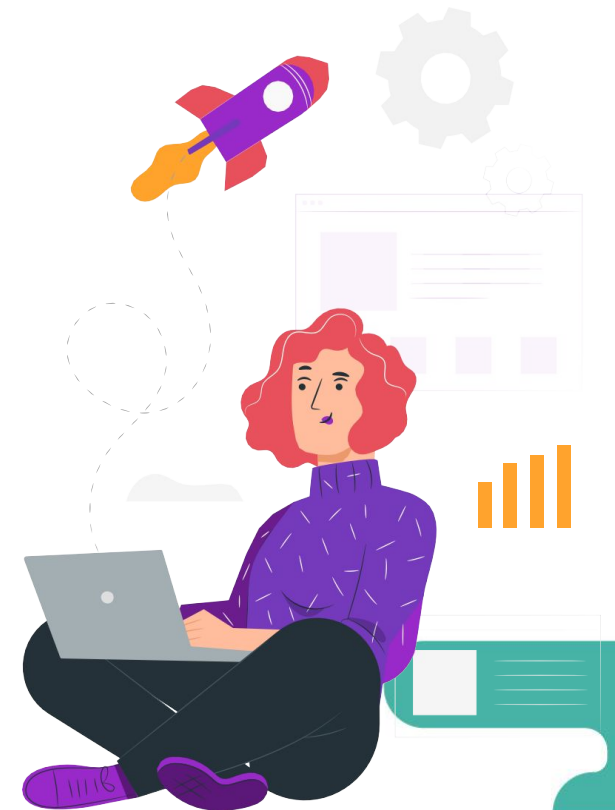
parámetros: no son obligatorios*

valor de retorno: no es obligatorio*

* Una función puede o no tener parámetros (datos de entrada) y puede o no tener un valor de retorno (datos de salida).

Pero:

- si tiene parámetros en su definición, los debe tener en su llamada.
- si tiene valor de retorno, para imprimir éste, hay que imprimir la llamada a la función.



Funciones en Python

```
def par(a):  
    if a % 2 == 0:  
        return "Es par"  
    else:  
        return "Es impar"
```

```
def par_alt(a):  
    if a % 2 == 0:  
        print("Es par")  
    else:  
        print("Es impar")
```

```
def say_hi():  
    print('Hi')
```

Sin valor de retorno

Sin parámetros



Parámetros

Por defecto

Si al ser llamada la función no tiene parámetros incluidos, no da error y simplemente se toman como valores los por defecto.

```
def par(a=2):  
    if a % 2 == 0:  
        return "Es par"
```

Indefinidos

No se conoce cuántos parámetros son necesarios cuando la función sea llamada. Se le asigna un número indeterminado de parámetros

```
def sumatoria(*args):  
    suma = 0  
    for arg in args:  
        suma += arg  
    return suma
```

Parámetros vs. Argumentos

Parámetro es el identificador que se crea al definir una función, y **argumento** es el valor que se pasa como parámetro al llamar la función.



Parámetros vs. Argumentos

Parámetro

```
def par(a):  
    if a % 2 == 0:  
        return "Es par"  
    else:  
        return "Es impar"
```

```
a = 4  
par(a)
```

Argumento

Scope

Las variables que se utilizan pertenecen a un ámbito en particular depende donde estén definidas.

Global

```
numero = 120
```

```
def suma():
```

```
    numero = 25
```

```
    print(numero)
```

Local

Métodos

Un método es una función pero la diferencia es que está definido **dentro de cierta clase**.

Un método, como la función, cumple determinada tarea al ser ejecutado.





Descanso

Nos vemos en 10 minutos



String Formatting

String Formatting

En Python, existen herramientas para dar formato a las cadenas de texto o string. Particularmente, si queremos insertar el contenido de variables en strings o manipularlos.

- `str.format()`
- f-strings
- Template Class



str.format()

“frase {}".format(a)

```
#str format basico
print("Maria tiene una prima que se llama {} y una amiga que se llama {}".format('Clara', 'Sabrina'))

#asignandoselo a una variable y reorganizando los placeholders
frase = "Maria tiene una prima que se llama {1} y una amiga que se llama {0}"

#imprimo
print(frase.format('Clara', 'Sabrina'))
```

Maria tiene una prima que se llama Clara y una amiga que se llama Sabrina

Maria tiene una prima que se llama Sabrina y una amiga que se llama Clara

str.format()

“frase {a[prima]}”.format(a=familia)

```
#llamando un diccionario
familia = {
    'prima': 'Clara',
    'amiga': 'Sabrina'
}

#imprimo
print("Maria tiene una prima que se llama {a[prima]} y una amiga que se llama {a[amiga]}".format(a=familia))
```

Maria tiene una prima que se llama Clara y una amiga que se llama Sabrina

f-strings

f“frase {a}”

```
prima = 'Clara'  
amiga = 'Sabrina'  
  
print(f"Maria tiene una prima que se llama {prima} y una amiga que se llama {amiga}")
```

Maria tiene una prima que se llama Clara y una amiga que se llama Sabrina

f-strings

f“frase {dict['key']}”

```
#llamando un diccionario
familia = {
    'prima': 'Clara',
    'amiga': 'Sabrina'
}

print(f"Maria tiene una prima que se llama {familia['prima']} y una amiga que se llama {familia['amiga']}")
```

Maria tiene una prima que se llama Clara y una amiga que se llama Sabrina

Template Class

```
from string import Template
```

```
prima = 'Clara'  
amiga = 'Sabrina'
```

```
frase = Template("Maria tiene una prima que se llama $prima y una amiga que se llama $amiga")
```

```
frase.substitute(prima=prima, amiga=amiga)
```

```
'Maria tiene una prima que se llama Clara y una amiga que se llama Sabrina'
```



Actividad práctica:

Funciones y Control de Flujo
Notebook 2



Trabajamos en salas

Trabajamos en salas de zoom

Funciones y control de flujo

En los grupos establecidos, empezamos a resolver los ejercicios planteados en la Notebook 2.



20 minutos de actividad.



Sección práctica

Resolvemos algunos ejercicios
de Notebook 2

Funciones y Control de Flujo

Resolvemos algunos puntos de la Notebook 2

Demostraremos los conceptos vistos en la primera parte de la clase.



Desafío 4

Para la siguiente clase:

Como tarea, en los grupos establecidos, resuelvan los ejercicios propuestos en la Notebook 3



¿Alguna consulta?

Numerical Python

Librería de Python especializada en el **cálculo numérico** y el **análisis de datos**, particularmente para un gran volumen de datos.

Incorpora una nueva estructura de datos, los **arrays**, que permite representar colecciones de datos de un mismo tipo en varias dimensiones.



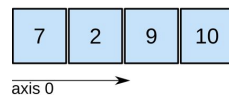
¿Por qué NumPy?

NumPy ofrece una manera de realizar operaciones eficientes sobre los datos: **mayor velocidad** y **menor espacio**.

Además introduce los **arrays multidimensionales** (ndarray), en lugar de listas de listas.

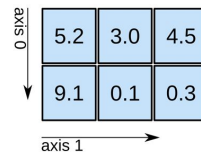


1D array



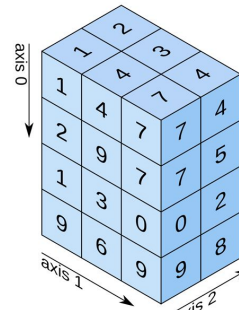
shape: (4,)

2D array



shape: (2, 3)

3D array




shape: (4, 3, 2)

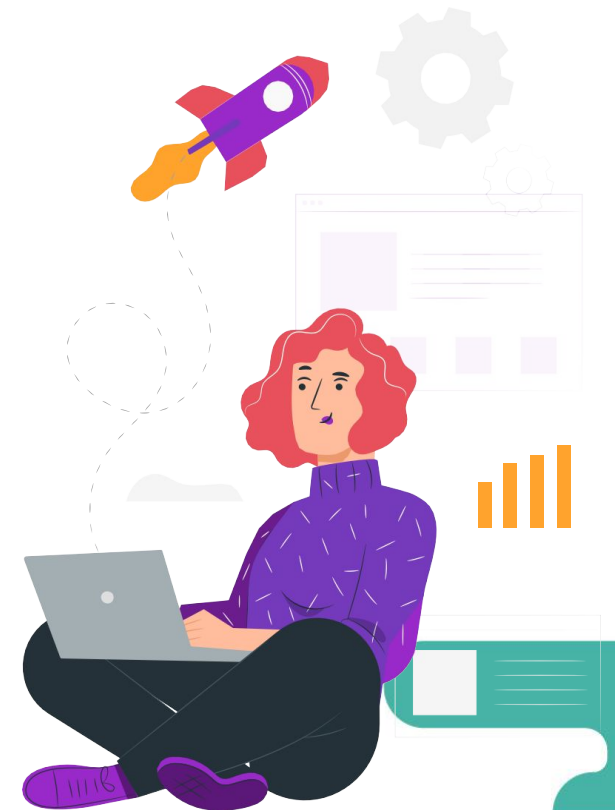


¿Por qué NumPy?

Podemos seleccionar **elementos** usando su **posición** en el array o vector



```
      0  1  2  
array([[10, 20, 30], 0  
       [ 9, 29, 33], 1  
       [ 0,  2,  3]]) 2
```





Sección práctica

**Aprendemos como funciona
NumPy con la Notebook 4**



Descanso

Nos vemos en 5 minutos



Actividad práctica:

Introducción a NumPy
Notebook 5



Trabajamos en salas

Trabajamos en salas de zoom

Introducción a Numpy

En los grupos establecidos, empezamos a resolver los ejercicios planteados en la Notebook 5.

 *30 minutos de actividad*

Preparando la primera pre-entrega

¡En la clase 5 deberán realizar la primera pre-entrega! Vayan poniendo al día...

Deberán entregar los desafíos resueltos en la notebook 5 y 7. Los mismos serán subidos como repositorio a su cuenta de Github.

Una vez completos **presentarán lo trabajado entregando el link al de Github en el foro del aula virtual**





¿Alguna consulta?

FUNDACIÓN
YPF

¡Muchas gracias!

