



# INTRODUCCIÓN A PYTHON

M. en T.A. Angel Hernández

Correo: [amhrdz.1001@gmail.com](mailto:amhrdz.1001@gmail.com)

Linkedin: Angel Moisés Hernández Ponce

Telegram: @amhrdz.1001

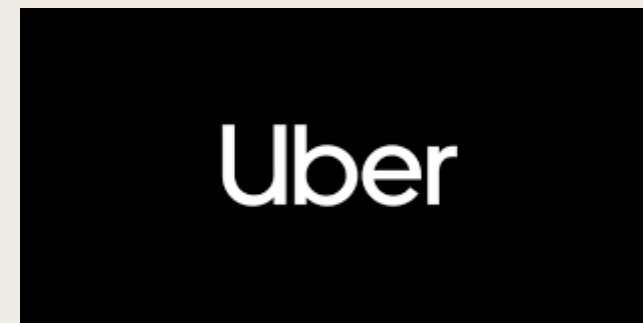
# Contenido

1. ¿Qué es Python?
2. ¿Cómo instalar y usar Python?
  1. *Python IDE / CMD / Terminal*
  2. *Jupyter Notebook y Anaconda*
  3. *Google Colabs*
3. Introducción a Google Colabs
4. Tipos de objetos y estructura de datos
  1. *Números*
  2. *Strings*
  3. *Listas*
  4. *Tuplas*
  5. *Diccionarios*
  6. *Booleanos*
5. Operadores de comparación
6. Declaraciones (statements) en Python
  - *Sintaxis*
  - *If y elif*
  - *Ciclos for*
  - *Ciclos while*
  - *Iterando a través de listas*
7. Expandiendo Python con librerías
8. Funciones y definiciones
  - *Estructura*
  - *Palabras clave*
  - *Cómo invocarlas*

# ¿QUÉ ES PYTHON?

- Python es un lenguaje de programación no interpretado cuya filosofía se basa en una sintaxis ligera que favorezca un código legible.
- Es de código libre
- Corre en todos los sistemas operativos (Windows, MacOS y Linux).
- Es fácil de aprender.
- Puede expandirse a través de librerías.

# ¿Quién usa Python?



# ¿Para qué se usa Python?

- Python se puede usar para las siguientes aplicaciones:
  - *Machine learning*
  - *Desarrollo de interfaces gráficos de usuario*
  - *Framework para páginas web como Django*
  - *Procesamiento digital de imágenes*
  - *Computación científica*



**PUTS A ; IN PYTHON**

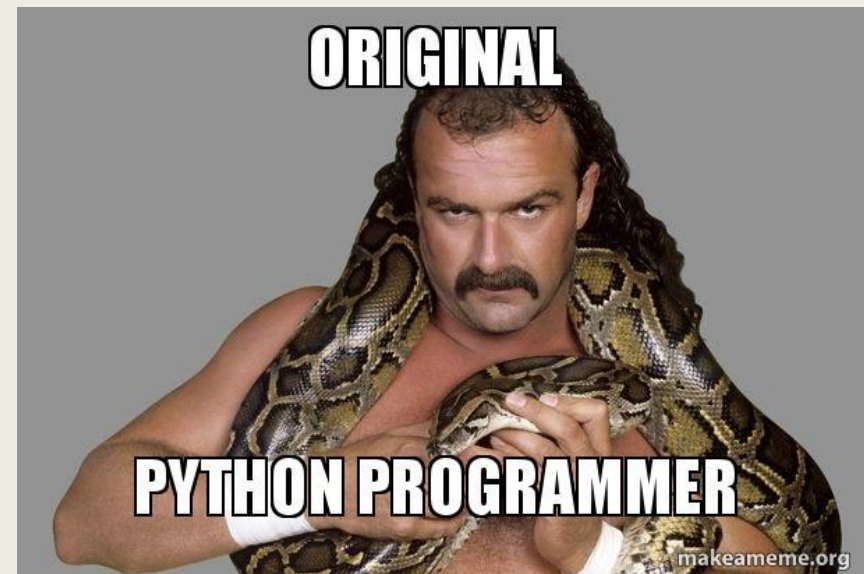


**When you switch  
from C++ to Python**



**ORIGINAL**

**PYTHON PROGRAMMER**



**SI HARRY POTTER  
HABLA PARCEL...**



# C++



# UNIX SHELL



# PYTHON



# JAVA



# LATEX



# HTML



# ENSAMBLADOR



# C





# Python vs the world

**C**

```
#include
```

```
int main(void)
{
    puts("Hola mundo");
}
```

**C++**

```
#include
```

```
int main()
{
    std::cout << "Hola mundo!";
    return 0;
}
```

# Python vs the world

## Java

```
import javax.swing.JFrame;
import javax.swing.JLabel;
{
    public static void main(String[] args) {
        JFrame frame = new JFrame(); //Creating frame
        frame.setTitle("Hi!");
        frame.add(new JLabel("Hola mundo!"));
        frame.pack();
        frame.setLocationRelativeTo(null);
        frame.setVisible(true);
    }
}
```

## C#

```
using System;
class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Hello, world!");
    }
}
```

# Python vs the world

```
print('Hola mundo')
```

# Un poco de filosofía...

- Bello es mejor que feo.
- Simple es mejor que complejo.
- Lo práctico gana a lo puro.
- Si la idea es difícil de explicar, entonces es una mala idea.
- Si la idea es fácil de explicar, entonces es una buena idea.
- Ahora es mejor que nunca.



# CONOCIENDO PYTHON

# Instalando Python

---

## De forma nativa

- Pueden instalar Python en su PC buscando el instalador en su página web.
- Ya viene preinstalado en Linux y MacOS.
- Interfaz poco amigable.

## En un entorno virtual

- Permite instalar distintas versiones y librerías.
- Entornos específicos para proyectos específicos.
- Mejor control de librerías.
- La forma más común de instalarlo es a través de Jupyter Notebooks y Anaconda.
- Interfaz sencilla y permite añadir texto complementario.



Google Colabs

# Google Colabs

- Colaboratory es una herramienta de investigación para la educación y la exploración del aprendizaje automático.
- Funciona a modo de bloc de notas de Jupyter que se puede usar sin ninguna configuración previa.
- Ya no hay que instalar algo.
- Viene con bastantes librerías externas precargadas.
- Funciona y se guarda en la nube, a través de Google Drive.



Nombre del archivo

Monitor de recursos

The screenshot shows the JupyterLab interface. At the top, the file name 'Untitled0.ipynb' is displayed, with a grey arrow pointing to it from the label 'Nombre del archivo'. To the right, the 'Monitor de recursos' (Resource Monitor) is visible, showing RAM and Disk usage, with a yellow arrow pointing to it from the label 'Monitor de recursos'. Below the file name, there is a menu bar with options: Archivo, Editar, Ver, Insertar, Entorno de ejecución, Herramientas, and Ayuda. On the right side of the menu bar, there are icons for 'Comentar' (Comment), 'Compartir' (Share), and a user profile icon. Below the menu bar, there are tabs for '+ Código' (Code) and '+ Texto' (Text). The main area contains three code cells. The first cell has the code '[1] 1+1' and the output '2'. The second cell has the code '[2] print('Hola mundo')' and the output 'Hola mundo'. The third cell has the code '[3] print('En serio esto es gratis?')' and the output 'En serio esto es gratis?'. A grey arrow points from the label 'Lineas de código' (Code lines) to the code area of the first cell. Another grey arrow points from the label 'Resultados' (Results) to the output area of the first cell. Below the code cells, there is a text cell with the text 'Esto es una linea de texto, aquí no corre ningún código Como en la mayoría de editores de texto podemos dar formato al texto como **negritas** o *itálicas* entre otros'. A green arrow points from the label 'Celda de texto' (Text cell) to this text cell. At the bottom right of the text cell, there is a toolbar with icons for undo, redo, link, comment, settings, delete, and a menu icon.

co Untitled0.ipynb ☆

Archivo Editar Ver Insertar Entorno de ejecución Herramientas Ayuda

+ Código + Texto

Comentar Compartir

✓ RAM Disco Editen

[1] 1+1

2

[2] print('Hola mundo')

Hola mundo

[3] print('En serio esto es gratis?')

En serio esto es gratis?

[ ]

Esto es una linea de texto, aquí no corre ningún código Como en la mayoría de editores de texto podemos dar formato al texto como **negritas** o *itálicas* entre otros

↑ ↓ ↪ ↻ ⚙️ 🗑️ ⋮

Celda de texto



# OBJETOS Y ESTRUCTURA DE DATOS



# ASIGNACIÓN DE VARIABLES

# Qué es una variable

- En Python, como la mayoría de los lenguajes de programación, se puede asignar un valor arbitrario a una variable.
- Técnicamente hablando, una variable es un espacio reservado de memoria que guarda su valor asignado; este valor puede ser llamado desde cualquier parte del programa para su uso o manipulación.
- Las variables pueden contener cualquier tipo de objeto o estructuras.
- En Python no es necesario indicar qué tipo de dato contendrá una variable.



# Algunas restricciones para las variables

- No pueden comenzar con números.
  - *2x, 20pesos, 1.5litros*
- No puede haber espacios en el nombre, es recomendable usar guion (-) o guion bajo (\_) en su lugar.
- No puede contener ninguno de estos símbolos: “”<>/\?|()@\$%&\*+,-
- No se pueden utilizar palabras clave como: for, while, if, else, help, or, and.

And	As	Assert	Break	Class
Continue	Def	Del	Elif	Else
Except	False	Finally	For	From
Global	If	Import	In	Is
Lambda	None	Nonlocal	Not	Or
Pass	Raise	Return	True	Try
While	With	yield		

## Palabras clave

# Otra cosa más...

Python usa una filosofía de tecleado dinámico, por lo que los valores de las variables pueden ser reasignados sobre la marcha.

Tener cuidado de repetir los nombres de una variable, puede ser que pierdan información valiosa.

```
>>> x = 5
>>> y = 10
>>> c = x + y
>>> print(c)
15
>>> x = 10
>>> print(c)
15
>>> c = x + y
>>> print(c)
20
```

# Tipos de datos

Nombre	Tipo	Descripción	Ejemplo
Enteros	int	Cualquier número entero positivo o negativo	3, 1000, -2
Flotantes	float	Números con punto decimal	5.8, 0.00001, 1.0
Texto/ Caracteres	str	Secuencia de caracteres, pueden ser números, letras o ambos	"Hola", "Python", '100'
Listas	list	Secuencia de objetos, puede contener números, strings o ambos	[10, 'hola', 100.5, '25']
Diccionarios	dict	Secuencia de objetos, no ordenada, dados en pares	{"c1":"valor", "nombre":"Juan"}
Tuplas	tup	Secuencia de objetos ordenada no inmutable	(10, "hola", 100.5, '25')
Sets	set	Colección de datos no ordenada y no indexada	{"fruta","comida","platos"}
Booleano	bool	Valores lógicos	True & False

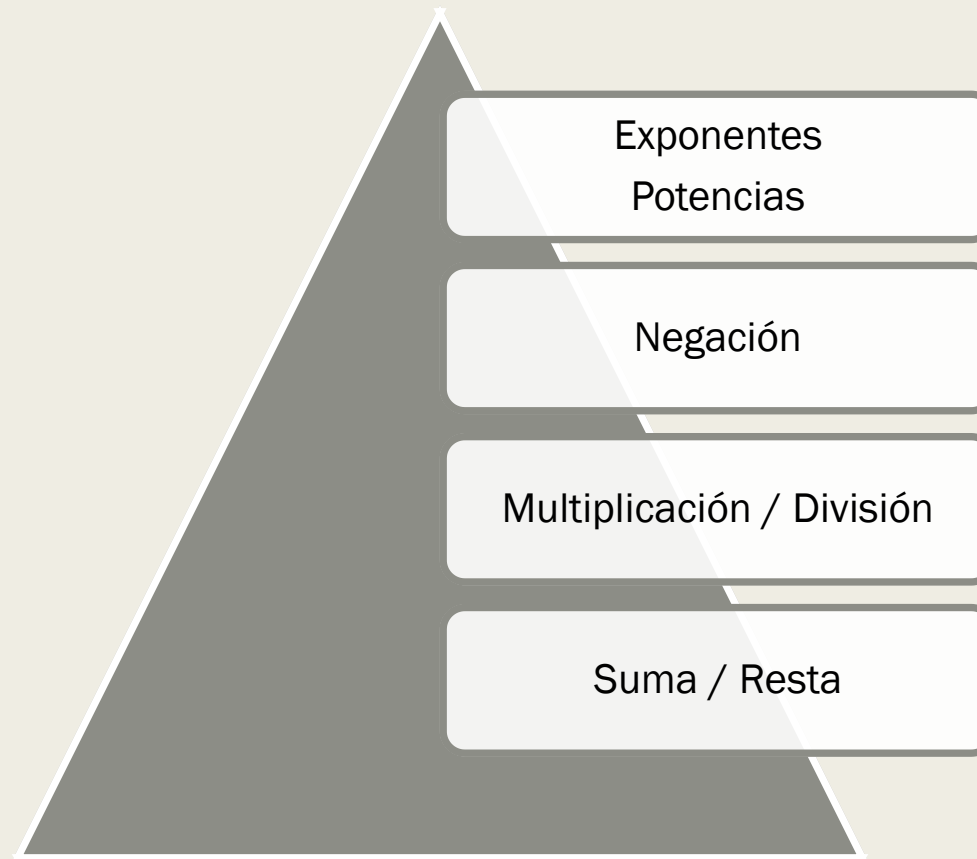


# Números

- Puede ser cualquier número.
- Los números con decimales son considerados flotantes (**float**).
- Se puede realizar cualquier operación aritmética.
- En otras palabras, Python puede funcionar como una calculadora.

Operador	Nombre	Ejemplo
+	Suma	1 + 1
-	Resta	3 - 2
*	Multiplicación	3 * 4
/	División	10 / 4
%	Módulo	20 % 5
**	Potencia	10**3
//	División entera ( <i>floor división</i> )	15 // 2

# Jerarquía de operaciones



```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.json'), 'a')
39             self.file.seek(0)
40             self.fingerprints.update(e.request for e in self.requests)
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFUTRA_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

Práctica 1: Usando Python como calculadora

## Sintaxis básica de Python

```
32 self.fingerprints = set()
33 self.logdups = True
34 self.debug = debug
35 self.logger = logging.getLogger(__name__)
36 if path:
37     self.file = open(os.path.join(path, 'requests.log'),
38                     'a')
39     self.file.seek(0)
40     self.fingerprints.update(self.request)
41
42 @classmethod
43 def from_settings(cls, settings):
44     debug = settings.getbool('SUPERLINT_DEBUG')
45     return cls(job_dir(settings), debug)
46
47 def request_seen(self, request):
48     fp = self.request_fingerprint(request)
49     if fp in self.fingerprints:
50         return True
51     self.fingerprints.add(fp)
52     if self.file:
53         self.file.write(fp + os.linesep)
```

# Las reglas de Python

- Al igual que en los lenguajes humanos, los lenguajes de programación tienen sus propias reglas.
- Estas reglas se debe seguir si desean que sus programas se ejecuten sin errores.
- Por suerte estas reglas son pocas y muy sencillas.

# Strings

- Las strings (**str**) se componen de cualquier carácter (ya sea número, letra o símbolo) siempre y cuando estén entre “” o ‘’.
- Para visualizar una string se puede usar la función *print*, o también se puede asignar a una variable.
  - *print('Hola mundo')*
  - *msg = 'hola mundo'*
- Si se quiere ingresar una string multilínea se debe usar tres comillas dobles.
  - *msg = """El veloz murciélago hindú comía cardillo y kiwi"""*
  - *Si la string que se utilizará contiene un apostrofe (') se debe usar las comillas ("").*
    - *print("I don't do that")*



# Operaciones con strings

- Podemos realizar algunas operaciones con el tipo de dato string.
- Las únicas operaciones válidas son la suma y la multiplicación.
- Sin embargo, hay otras formas de manejar a las strings.

```
>> a = 'hola'
>> a+a
holahola'
>> a*2
```

# Dando formato a nuestras strings

- Ya vimos como trabajar con las strings, sin embargo, el texto que ingresamos se despliega tal como está.
- Esto es funcional más no presentable, por suerte, existe una forma de dar formato a las strings.
- La forma más sencilla de hacer esto es usando *placeholders*.
- Un *placeholder* se declara con el operador modulo (%), este símbolo indicará a Python que en esa posición habrá un objeto.

Operador	Tipo de dato
%d	Enteros
%f	Flotantes
%b	Binarios
%o	Octal
%x	Octal Hexadecimal
%s	Strings
%e	Números exponenciales

# Manipulación de strings

- En ciertas ocasiones puede resultar conveniente obtener un solo carácter o una parte de la string, por ejemplo. Obtener la primera letra o valor.
- Para este caso se utilizan las operaciones de indexación (indexing) y slicing.
  - Indexación: tomar solo un carácter de la string.
  - Slicing: tomar una subsecuencia de caracteres de la string.
- La cuenta inicia desde 0, no desde 1.
- También se puede consultar si existe un carácter en la string.
- Nota: los espacios cuentan.

```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.log'), 'a')
39             self.file.seek(0)
40             self.fingerprints.update(re.findall(r'(?P<ip>[0-9.]+)', self.file.read()))
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFUTURA_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```



# ARRAYS



# Listas

- Una lista es una colección de objetos que es ordenada y puede cambiar el orden de sus elementos.
- Las listas pueden tener todos los tipos de datos.
- Se puede realizar operaciones aritméticas con ellas como la suma o multiplicación.

```
>>> lst = [1, 2, 3, 4.5, 5.5]
>>> lst
[1, 2, 3, 4.5, 5.5]
>>> lst2 = ['a', 1, 'b', 2, 'c', 3]
>>> lst2
['a', 1, 'b', 2, 'c', 3]
>>> l_bool = [True, False, True, True]
>>> l_bool
[True, False, True, True]
>>> n_lst = lst + lst2 + l_bool
>>> n_lst
[1, 2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True]
>>> lst[1]
2
>>> n_lst[5]
'a'
>>> lst[3] = 4
>>> lst
[1, 2, 3, 4, 5.5]
```



# Manipulando Listas

- Los elementos de las listas se pueden agregar, remover o modificar.
- Estas operaciones se ejecutan mediante los **métodos** de Python.
- La sintaxis es:  
`variable.método()`

Método	Descripción
<code>append()</code>	Agrega un elemento al final de la lista
<code>clear()</code>	Quita todos los elementos de la lista
<code>copy()</code>	Regresa una copia de la lista
<code>remove()</code>	Quita el primer elemento del valor especificado
<code>pop()</code>	Quita el elemento en el índice especificado

```
>>> n_lst.append("agregado")
>>> n_lst
[1, 2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True, 'agregado']
>>> n_lst.remove(1)
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True, 'agregado']
>>> n_lst.pop()
'agregado'
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True, True]
>>> n_lst.pop(-1)
True
>>> n_lst
[2, 3, 4.5, 5.5, 'a', 1, 'b', 2, 'c', 3, True, False, True]
```

```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.log'), 'a')
39             self.file.seek(0)
40             self.fingerprints.update(self.get_fingerprints())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFILTER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

# Tuplas

- Una **tupla** es una colección de datos que están ordenados y **no son intercambiables**.
- Las tuplas son declaradas con paréntesis ().
- Se pueden acceder a los elementos al igual que las listas.

```
>>> tupla = ("manzana", "fresa", "kiwi", "pera")
>>> print(tupla)
('manzana', 'fresa', 'kiwi', 'pera')
>>> print(tupla[1]) # Mostramos el segundo elemento
fresa
>>> print(tupla[-1]) # Desplegamos el último elemento
pera
>>> print(tupla[1:4])
('fresa', 'kiwi', 'pera')
>>> "manzana" in tupla
True
>>> "limon" in tupla
False
>>> len(tupla) # Muestra la longitud de la tupla
4
```

# Diccionarios

- Es una colección de datos que no tiene un orden pero si un **identificador**.
- Sus elementos pueden **cambiar** y ser **indexados**.
- Se declaran mediante laves { }.
- Para cada elemento **debe** existir una **key** indicando qué elemento se está haciendo referencia.
- Se pueden manipular al igual que las listas.

```
1 # Declaramos un diccionario
2 orquesta = {
3     "vientos": "flauta",
4     "metales": "trompeta",
5     "cuerdas": "violin",
6     "percusiones": "tambor"
7 }
```

```
1 # Diccionario con valores repetidos
2 planetas = {
3     "rocosos": "mercurio, venus, tierra, marte",
4     "gaseosos": "neptuno, jupiter, urano, saturno",
5     "anillados": "saturno, urano, neptuno, jupiter"
6 }
7
8 print(planetas)
```



```
1 ### Referenciaremos los valores de cada diccionario
2 print(orquesta["metales"])
3 print(planetas["rocosos"])
```



```
trompeta
mercurio, venus, tierra, marte
```

```
[ ] 1 ### También se pueden crear diccionarios que contengan otros diccionarios
```

```
2
```

```
3 info1 = {
```

```
4     |     "artista": "acdc", "genero": "rock"
```

```
5 }
```

```
6
```

```
7 info2 = {
```

```
8     |     "artista": "abba", "genero": "pop"
```

```
9 }
```

```
10
```

```
11 info3 = {
```

```
12     |     "artista": "daft punk", "genero": "electronica"
```

```
13 }
```

```
14
```

```
15 musica = {
```

```
16     |     "artista 1": info1,
```

```
17     |     "artista 2": info2,
```

```
18     |     "artista 3": info3
```

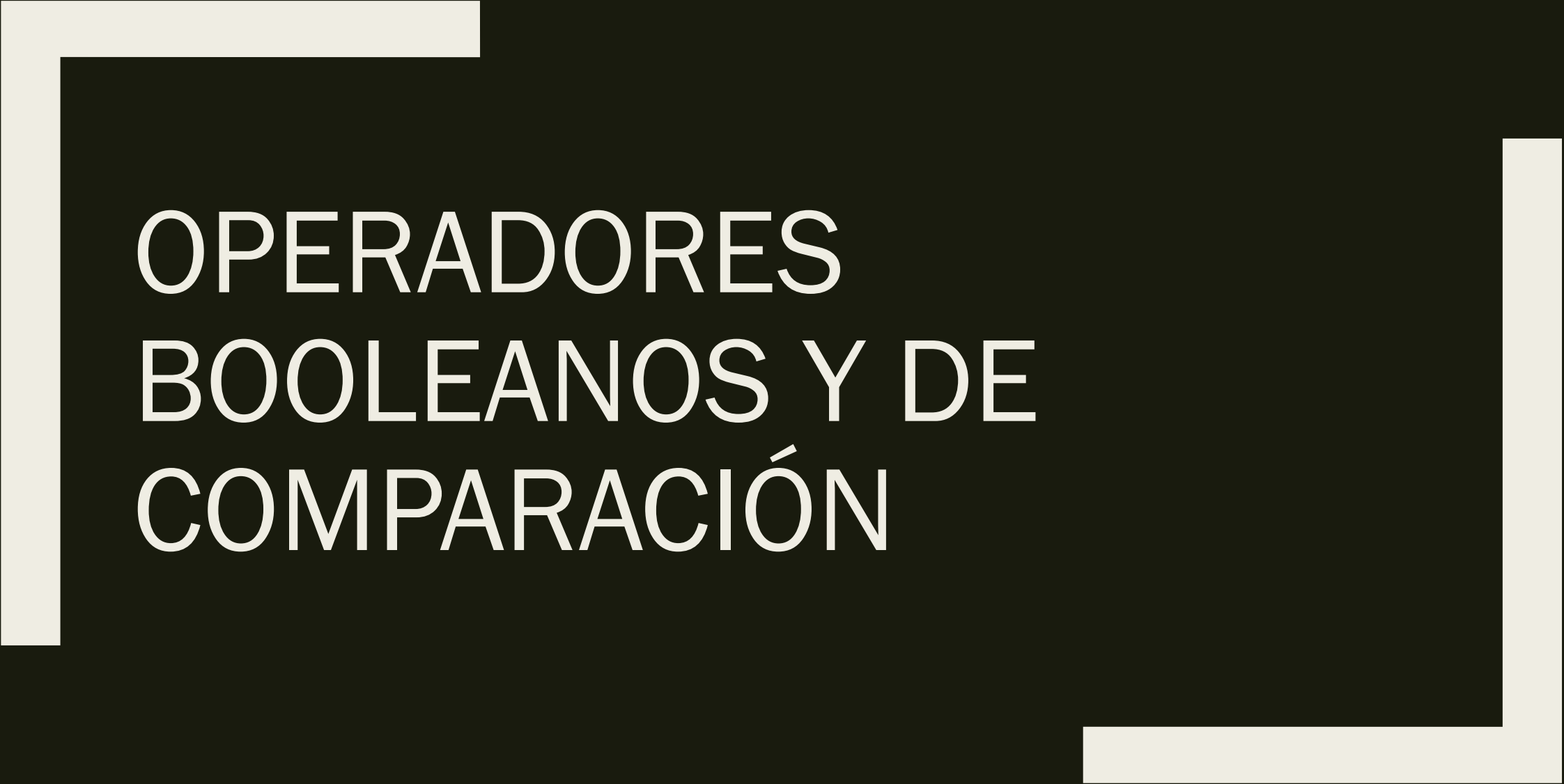
```
19 }
```

```
20
```

```
21 print(musica)
```

```
{'artista 1': {'artista': 'acdc', 'genero': 'rock'}, 'artista 2': {'artista': 'abba', 'genero': 'pop'}, 'artista 3': {'artista': 'daft punk', 'genero': 'electronica'}}
```





# OPERADORES BOOLEANOS Y DE COMPARACIÓN

# Operadores de comparación

- Sirven para evaluar si dos valores o sentencias son iguales, mayores, menores o distintos.
- Pueden establecer relaciones para manipular el flujo de un programa.
- Pueden ser aplicados a distintos tipos de datos.
- El resultado siempre será del tipo **booleano**.

Operador	Nombre	Ejemplo
==	Es igual a	x == y
!=	No es igual. Es distinto de	x != y
>	Mayor que	x > y
<	Menor que	x < y
>=	Mayor o igual que	x >= y
<=	Menor o igual que	x <= y

# Operadores booleanos

- Los operadores booleanos vienen desde el álgebra booleana.
- Agrupan términos en combinaciones lógicas.
- En esencia sólo son tres: **and**, **or** y **not**.
- El resultado de una operación booleana puede regresar dos valores: **True** o **False**.

# Tablas de verdad

Compuerta Or

Entrada		Salida
A	B	S
0	0	0
0	1	1
1	0	1
1	1	1

Compuerta And

Entrada		Salida
A	B	S
0	0	0
0	1	0
1	0	0
1	1	1

```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.json'), 'w')
39             self.file.seek(0)
40             self.fingerprints.update(re.findall(r'"[a-zA-Z0-9_]+"', self.file.read()))
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFILTER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

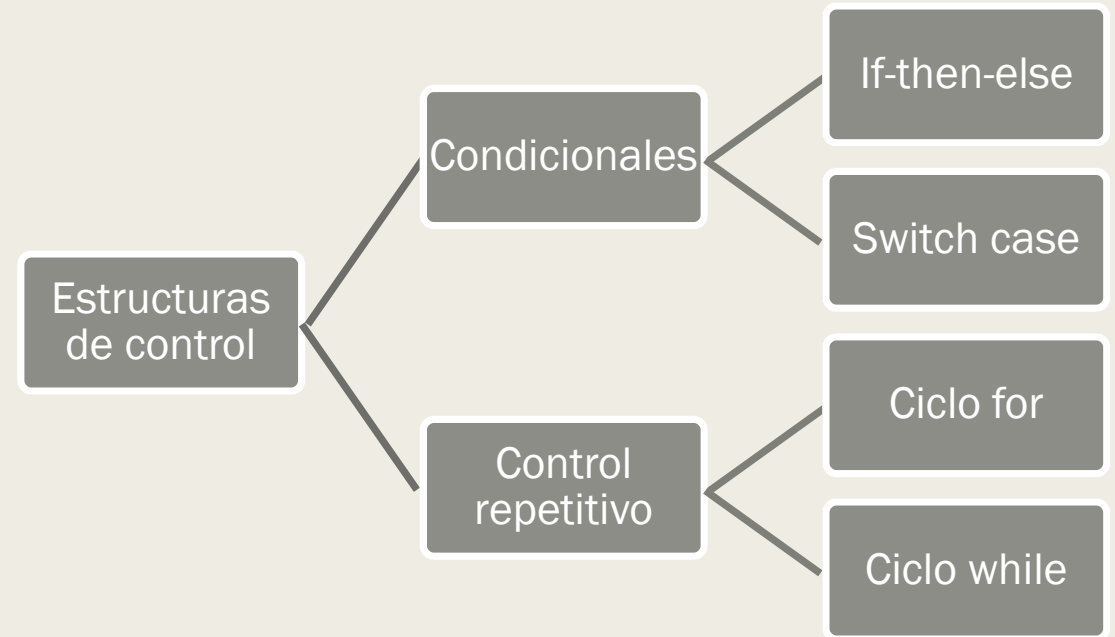


Examen sorpresa!

# ESTRUCTURAS DE CONTROL

# Condicionales

- En programación, se conoce como **estructuras de control** a las herramientas que permiten modificar el flujo de trabajo del programa.
- Son útiles para dirigir al programa hacia donde queramos.
- Se pueden dividir en dos grupos: **condicionales** y **control repetitivo**.





Palabra clave *if*

Condición a evaluar

Si no escriben los :



```
>>> if var < 200:  
    print('Afirmativo')
```

Afirmativo

Resultado a desplegar

Instrucciones a ejecutar dentro  
de la condición

# If & Elif

- Python soporta todos los operadores de comparación.
- Hay que realizar la indentación después de la sentencia.
- Se puede usar la palabra clave **elif** para reevaluar una condición. Esto se interpretaría como *si la condición previa no se cumple, entonces intentar esto*.
- La palabra clave **else** indica realizar una acción si la condición no se cumplió.

```
[1] a = 33  
    b = 33
```

```
[2] if b > a:  
    print("b es mayor que a")  
    elif a == b:  
    print("a y b son iguales").
```

```
↳ a y b son iguales
```

Lógica de programación hijos!!!



# Control repetitivo

---

## Ciclo *for*

- Se utiliza para iterar sobre una secuencia de datos.
- A diferencia del *while* éste ciclo itera sobre todos los elementos de la colección, no mientras la condición sea verdadera.
- Se pueden ejecutar líneas de código mientras el ciclo esté activo.
- Si se quiere iterar con números enteros se debe añadir la palabra clave **range()** antes del número.

## Ciclo *while*

- Este ciclo se ejecuta siempre y cuando se cumpla la condición.
- Se puede interrumpir usando el comando **break**.
- Se debe agregar o quitar un valor a la variable contadora.

¿Qué pasa si no agregamos la variable contadora?









Recuerden amigos, si no escriben bien la sintaxis les  
marcará error

Hasta la próxima!!!

```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.json'), 'w')
39             self.file.seek(0)
40             self.fingerprints.update(self._get_fingerprints())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFILTER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

## Práctica 5: Ciclo *while*



```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.json'), 'w')
39             self.file.seek(0)
40             self.fingerprints.update(self._get_fingerprints())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFILTER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```

# EXPANDIENDO PYTHON CON LIBRERÍAS

# Cómo usar e instalar librerías

- Python puede usar una gran cantidad de librerías con las cuales puede adquirir más flexibilidad y poder de procesamiento.
- En nuestro caso, Google Colabs cuenta con bastantes librerías precargadas.
- Para instalar una librería se utiliza el comando ***pip*** y para conocer qué librerías tenemos instaladas se utiliza la instrucción ***pip list***.
- Para indicar a Python que queremos utilizar una o varias librerías es necesario usar la palabra clave ***import*** seguida del nombre de la misma.





# Numpy

- Librería especializada para cómputo numérico científico y avanzado.
- Ideal para trabajar con matrices y arreglos de datos (arrays).
- Manejo de operaciones de álgebra lineal, matricial, transformadas de Fourier, entre otras.
- Su manejo (e incluso algunas funciones) es similar al software Matlab.

# Matplotlib

- Colección de comandos para realizar gráficas elegantes y entendibles.
- Se pueden hacer gráficas de barras, histogramas, de pastel, entre otras.
- Se pueden editar aspectos como los ejes, la escala, el título de la gráfica, color, leyendas.

# Pandas

- Es una biblioteca de software especializada en manejo de base de datos y su análisis.
- Ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales.
- Trabaja principalmente con objetos conocidos como DataFrames, que permiten almacenar y manipular los datos contenidos a modo de tablas.
- Permite crear bases de datos desde cero hasta importar archivos (hojas de cálculo, blocs de notas, csv) completos para su trabajo.
- Permite crear bases de datos alternas a la original.

# SkLearn

- Es una herramienta para realizar análisis de datos y algunas técnicas de machine learning.
- También ofrece formas de evaluar modelos de aprendizaje automático.
- Sirve para preprocesamiento de datos y aplicar normalizaciones.
- También tiene precargadas algunas bases de datos para utilizarlas.



A large, white, L-shaped decorative frame is positioned on the left and bottom edges of the slide, framing the central text.

# FUNCIONES, DEFINICIONES Y MÉTODOS

# Funciones propias de Python

- Python trae incluidas varias funciones que pueden ser útiles.
- Para usar estas funciones debemos seguir la siguiente sintaxis:
  - *funcion(variable)*

# Algunos ejemplos...

Nombre	Funcionamiento	Ejemplo
abs()	Devuelve el valor absoluto de cualquier número	abs(x)
bin(x)	Convierte un entero a binario añadiendo un prefijo '0b'	bin(x)
enumerate()	Devuelve un objeto enumerado.	enumerate(x)
len()	Devuelve la longitud (número de elementos) de un objeto.	len(x)
max(), min(x)	Devuelve el valor máximo o mínimo de un objeto	max(x), min(x)
print()	Despliega la información contenida en los paréntesis	print(x)
range()	Crea un secuencia de números. Se puede indicar el inicio, el fin y los incrementos.	range(x,y) range(x,y,z)

# Funciones personalizadas (Scripts)

- Un **script** son varias líneas de código que se ejecutan **una sola vez** al ser llamadas.
- Un script puede contener una o varias **funciones**.
- Una **función** siempre regresa un **resultado**.
- Para declarar un función se usa la palabra clave **def**.

Palabra clave **def**

Nombre de la función

Parámetros

```
[19] def my_function():  
      print("Hola mundo desde una función")
```

Instrucciones de la función

```
[20] my_function()
```

```
☞ Hola mundo desde una función
```

# Parámetros

- Se puede pasar información a la función a través de **parámetros**.
- Son especificados después del nombre de la función, dentro de los **paréntesis**.
- También se puede asignar un parámetro **default**.

```
[24] def my_function(fname):  
      print(fname + " Hernández")
```

```
[25] my_function("Angel")  
      my_function("Moisés")  
      my_function("Tomás")  
      my_function("Luis")
```

```
☞ Angel Hernández  
   Moisés Hernández  
   Tomás Hernández  
   Luis Hernández
```

```
[26] def nacionalidad(pais= "México"):  
      print("Yo soy de " + pais)
```

```
[27] nacionalidad("EEUU")  
      nacionalidad("Francia")  
      nacionalidad("Japón")  
      nacionalidad()
```

```
☞ Yo soy de EEUU  
   Yo soy de Francia  
   Yo soy de Japón  
   Yo soy de México
```

# \*args

- Es una **sintaxis especial** que nos permite pasar cualquier número de parámetros a una función.
- Por convención se suele usar \*args, pero puede ser reemplazada por cualquier palabra, siempre y cuando esté después de un **asterisco(\*)**.

```
>>> def myfunc(*args):  
    print(args)  
  
>>> myfunc('hola', 'mundo', 'estoy', 'vivo', 'muajajaja')  
( 'hola', 'mundo', 'estoy', 'vivo', 'muajajaja')  
>>> def myfunc2(*args):  
    for arg in args:  
        print(arg)  
  
>>> myfunc2('hola', 'mundo', 'estoy', 'vivo', 'muajajaja')  
hola  
mundo  
estoy  
vivo  
muajajaja  
>>> def suma(*args):  
    print(sum(args))  
  
>>> suma(1,2,3,4,5,6,7,8,9)  
45
```



# La palabra clave return()

```
1 x = 100
2 y = 300
3
4 def suma1(x,y):
5 |   print('El resultado de la suma es: ', x+y)
6
7 def suma2(x,y):
8 |   s = x + y
9 |   return(s)

1 suma1(x,y)

El resultado de la suma es:  400

1 suma2(x,y)

400
```

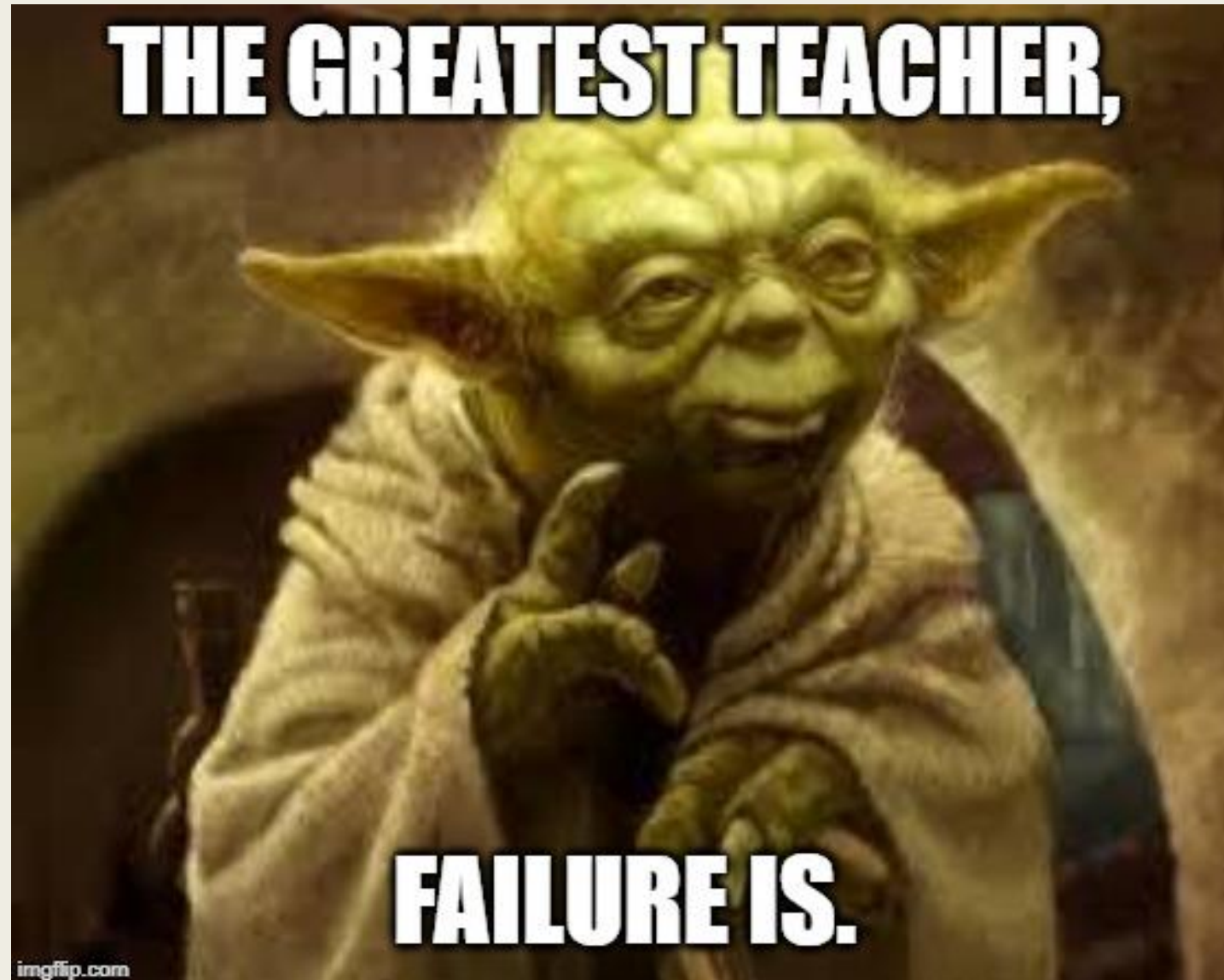
# Algunos tips...

- Definan muy bien y antes de comenzar qué hará exactamente su función.
- Procuren dar siempre una descripción del funcionamiento y ejemplos; nunca sabremos quién podría leer nuestro código.
- Elijan un nombre acorde al funcionamiento de la función.
- Si su función hará dos o mas cosas distintas, mejor divídanlas. **Dividan y vencerán.**

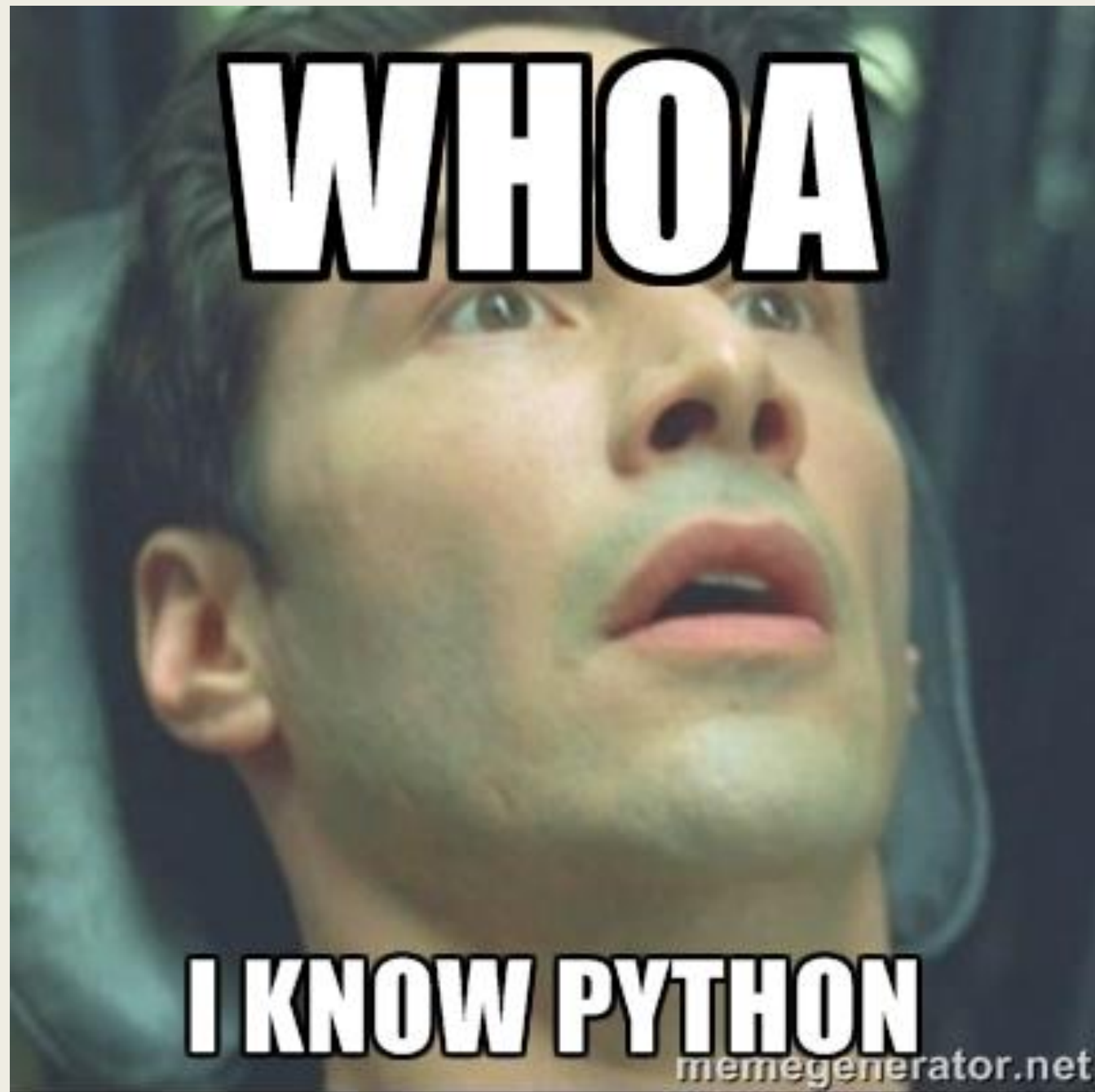
```

31     def __init__(self, path):
32         self.file = None
33         self.fingerprints = set()
34         self.logdups = True
35         self.debug = debug
36         self.logger = logging.getLogger(__name__)
37         if path:
38             self.file = open(os.path.join(path, 'requests.json'), 'w')
39             self.file.seek(0)
40             self.fingerprints.update(self._get_fingerprints())
41
42     @classmethod
43     def from_settings(cls, settings):
44         debug = settings.getbool('SUPERFILTER_DEBUG')
45         return cls(job_dir(settings), debug)
46
47     def request_seen(self, request):
48         fp = self.request_fingerprint(request)
49         if fp in self.fingerprints:
50             return True
51         self.fingerprints.add(fp)
52         if self.file:
53             self.file.write(fp + os.linesep)
54
55     def request_fingerprint(self, request):
56         return request_fingerprint(request)

```







# Bibliografía

- From zero to hero, José Portilla, Udemy, 2019.
- Learning to program: the fundamentals, University of Toronto, Coursera, 2016.
- Python Tutorial, w3schools.com
- Python, python.org