

CONQUER **BLOCKS**

PYTHON

VARIABLES, TIPOS DE DATOS Y OPERACIONES BASICAS

CONQUERBLOCKS



QUE ES PYTHON

Lenguaje creado en los años 80 por Guido van Rossum en el Centro para las Matemáticas y la Informática CWI de los países bajos

Multiplataforma: Unix, Linux, MacOS, Windows

Multiparadigma (alto nivel) : permite programación orientada a objetos, programación estructurada y programación funcional.

Sintaxis compacta, sencilla e intuitiva, con una curva de aprendizaje mínima y una potente librería de funciones y clases.

Un programa de Python no se compila si no que se ejecuta directamente (usa un intérprete). Eso permite hacer cosas que son imposibles en otros lenguajes como ejecutar instrucciones de manera interactiva, crear funciones al vuelo mientras un programa se ejecuta, interpretar un string como código Python y ejecutarlo etc.



LINEA DE COMANDOS VS SCRIPTS

Línea de comandos de Python en la terminal

```
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ conda activate conque
(base) (ConquerB) MacBook-Pro-2:Python tu_nombre_de_usuario$ python
Python 3.9.15 (main, Nov 24 2022, 08:29:02)
[Clang 14.0.6 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 'Esto es una linea de comandos de python'
'Esto es una linea de comandos de python'
>>> 3+2
5
>>> exit()■
```



LINEA DE COMANDOS VS SCRIPTS

Línea de comandos de Python en la terminal

```
(base) MacBook-Pro-2:Python tu_nombre_de_usuario$ conda activate conque
(base) (ConquerB) MacBook-Pro-2:Python tu_nombre_de_usuario$ python
Python 3.9.15 (main, Nov 24 2022, 08:29:02)
[Clang 14.0.6 ] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 'Esto es una linea de comandos de python'
'Esto es una linea de comandos de python'
>>> 3+2
5
>>> exit()■
```

suma.py

```
suma.py
1 print(3+2)
```

TERMINAL

```
PROBLEMAS SALIDA TERMINAL CONSOLA DE DEPURACIÓN

The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`
For more details, please visit https://support.apple.com/kb/H
● (base) MacBook-Pro-2:Python tu_nombre_de_usuario $ conda act
● (ConquerB) MacBook-Pro-2:Python tu_nombre_de_usuario $ /User
tu_nombre_de_usuario/Desktop/Máster Conquer Blocks/MODULOS/PyUini/
5
○ (ConquerB) MacBook-Pro-2:Python $
```

Python Scripts



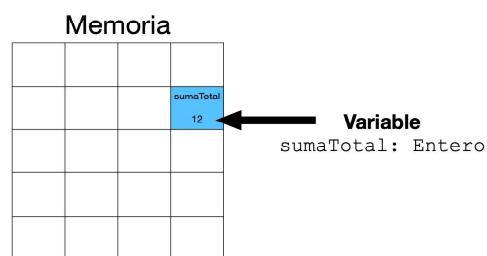
VARIABLES

DEFINICIÓN: Espacio reservado en memoria que tiene asignado un identificador



VARIABLES

DEFINICIÓN: Espacio reservado en memoria que tiene asignado un identificador





VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria			
sumaTotal			
	precio		terminado
	nota		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria			
sumaTotal			
	precio		terminado
	nota		

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria			
sumaTotal			
12			
	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria			
sumaTotal			
	precio		terminado
	nota		

En PYTHON **no declaramos** las variables.
Las variables se inicializan directamente

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria			
sumaTotal			
12	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria			
sumaTotal			
	precio		terminado
	nota		

En PYTHON **no declaramos** las variables.
Las variables se inicializan directamente

```
numero_entero = 42
numero_decimal = 12.5
texto = 'hola'
variable_logica = True
```

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria			
sumaTotal			
12	precio		terminado
	20.5		Falso
	nota		
	"Hola"		



VARIABLES

DECLARACIÓN

```
Definir sumaTotal Como Entero
Definir precio Como Real
Definir nota Como Texto
Definir terminado Como Logica
```

Memoria			
sumaTotal			
	precio		terminado
	nota		

En PYTHON no declaramos las variables.
Las variables se inicializan directamente

```
numero_entero = 42
numero_decimal = 12.5
texto = 'hola'
variable_logica = True
```

INICIALIZACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria			
sumaTotal			
12	precio		terminado
	20.5		Falso
	nota		
	"Hola"		

INICIALIZACIÓN EXPLÍCITA

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```



VARIABLES

MODIFICACIÓN

```
sumaTotal = 12
precio = 20.5
nota = "Hola"
terminado = Falso
```

Memoria			
sumaTotal			
12	precio		terminado
	20.5		Falso
	nota		
	"Hola"		

MODIFICACION EN PYTHON

```
numero_entero = 42
numero_decimal = 12.5
texto = 'hola'
variable_logica = True
```

```
sumaTotal = sumaTotal + 4
precio = 3.4 + 4.6
nota = "Adios"
terminado = Verdadero
```

Memoria			
sumaTotal			
16	precio		terminado
	8.0		Verdadero
	nota		
	"Adios"		

```
numero_entero = numero_entero + 4
numero_decimal = 12.5 + 4.6
texto = 'adios'
variable_logica = False
```



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

my_variable_1 ✓

_my_variable_1 ✓

1_my_variable ✗



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

1_my_variable ✗

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras my variable ✗



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

1_my_variable ✗

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras my variable ✗

No se deben usar palabras que están asociadas ya a funciones internas de python

por ejemplo: print ✗



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

1_my_variable ✗

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras

my variable ✗

No se deben usar palabras que están asociadas ya a funciones internas de python

por ejemplo: print ✗

Los nombres de variables deben ser cortos pero descriptivos

nombre >> n

nombre_estudiante >> n_e

tamaño_nombre >> tamaño_del_nombre_de_las_personas



VARIABLES

SINTAXIS PERMITIDA AL NOMBRAR VARIABLES:

Los nombres pueden contener solo letras, números y barras bajas

my_variable_1 ✓

_my_variable_1 ✓

Pueden comenzar por una letra o una barra baja pero NUNCA por un número

1_my_variable ✗

Los espacios no están permitidos, pero se pueden usar barras bajas para separar las palabras

my variable ✗

No se deben usar palabras que están asociadas ya a funciones internas de python

por ejemplo: print ✗

Los nombres de variables deben ser cortos pero descriptivos

nombre >> n

nombre_estudiante >> n_e

tamaño_nombre >> tamaño_del_nombre_de_las_personas

Cuidado al usar la ele minúscula 'l', la i mayúscula 'I' y la letra o mayúscula 'O'. Al leer el código el usuario puede confundirlos con unos y ceros '1', '0'



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10

```
x, y, z = 10,20,30  
print(x,y,z)  
✓ 0.1s
```

10 20 30



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10

```
x, y, z = 'texto 1', 'texto 2', 'texto2'  
print(x,y,z)  
✓ 0.2s
```

texto 1 texto 2 texto2

```
x, y, z = 10,20,30  
print(x,y,z)  
✓ 0.1s
```

10 20 30



VARIABLES - ASIGNACIÓN MÚLTIPLE

```
x = y = z = 10  
print(x,y,z)
```

10 10 10

```
x, y, z = 'texto 1', 'texto 2', 'texto2'  
print(x,y,z)  
✓ 0.2s
```

texto 1 texto 2 texto2

```
x, y, z = 10,20,30  
print(x,y,z)  
✓ 0.1s
```

10 20 30

```
x, y, z = 'texto 1', 120.3, 42  
print(x,y,z)  
✓ 0.2s
```

texto 1 120.3 42



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegra de conocerte', nombre)
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegra de conocerte', nombre)
```

```
¿Cómo te llamas?
Elena
Me alegra de conocerte Elena
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas?
Elena
Me alegro de conocerte Elena
```

```
nombre = input('¿Cómo te llamas? ')
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas? Elena
Me alegro de conocerte Elena
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
print('¿Cómo te llamas?')
nombre = input()
print('Me alegro de conocerte', nombre)
```

OUTPUT

```
¿Cómo te llamas?
Elena
Me alegro de conocerte Elena
```

```
nombre = input('¿Cómo te llamas? ')
print('Me alegro de conocerte', nombre)
```

```
¿Cómo te llamas? Elena
Me alegro de conocerte Elena
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'días')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'días')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "<file>", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = float(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Entonces has vivido aproximadamente 9125.0 dias
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "<file>", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = float(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Entonces has vivido aproximadamente 9125.0 dias
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Entonces has vivido aproximadamente 9125 dias
```



VARIABLES - PEDIR VALORES

La función input() permite obtener **texto** escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'dias')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'dias')
```

```
¿Cúantos años tienes? 25.5
Traceback (most recent call last):
  File "/Users/ tu_nombre_de_usuario /Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 17, in <module>
    numero = int(input('¿Cúantos años tienes? '))
ValueError: invalid literal for int() with base 10: '25.5'
```



VARIABLES - PEDIR VALORES

La función input() permite obtener texto escrito por teclado.

```
numero = input('¿Cúantos años tienes? ')
print('Entonces has vivido aproximadamente', 365.0*numero, 'días')
```

```
¿Cúantos años tienes? 25
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 15, in <module>
    print('Entonces has vivido aproximadamente', 365.0*numero)
TypeError: can't multiply sequence by non-int of type 'float'
```

```
numero = int(input('¿Cúantos años tienes? '))
print('Entonces has vivido aproximadamente', 365*numero, 'días')
```

```
¿Cúantos años tienes? 25.5
Traceback (most recent call last):
  File "/Users/tu_nombre_de_usuario/Desktop/Master Conquer Blocks/MODULOS/Python/Clase 1/teoria_clase1.py", line 17, in <module>
    numero = int(input('¿Cúantos años tienes? '))
ValueError: invalid literal for int() with base 10: '25.5'
```



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

```
numero_entero = 42
numero_decimal = float(numero_entero)
print(numero_decimal)
✓ 0.1s
```

42.0



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

```
numero_entero = 42
numero_decimal = float(numero_entero)
print(numero_decimal)
✓ 0.1s
```

42.0

La función `type()` nos devuelve el tipo de dato con el que estamos trabajando

```
numero_entero = 42
numero_decimal = float(numero_entero)
numero_texto = str(numero_entero)
print(type(numero_entero), type(numero_decimal), type(numero_texto))
✓ 0.3s
```

<class 'int'> <class 'float'> <class 'str'>



VARIABLES - TIPOS

Las funciones `int()`, `float()`, `str()` y `bool()` son funciones de tipo. Pueden convertir un tipo de dato en otro tipo de dato

```
numero_entero = int(42)
numero_decimal = float(12.5)
texto = str('hola')
variable_logica = bool(True)
```

```
numero_entero = 42
numero_decimal = float(numero_entero)
print(numero_decimal)
✓ 0.1s
42.0
```

La función `type()` nos devuelve el tipo de dato con el que estamos trabajando

```
numero_entero = 42
numero_decimal = float(numero_entero)
numero_texto = str(numero_entero)
print(type(numero_entero), type(numero_decimal), type(numero_texto))
✓ 0.3s
<class 'int'> <class 'float'> <class 'str'>
```

```
boolean_0 = bool(0)
boolean_1 = bool(1)
boolean_42 = bool(42)
boolean_float = bool(45.3)
boolean_texto = bool('hola')
boolean_texto_vacio = bool('')
print(boolean_0, boolean_1, boolean_42, boolean_float, boolean_texto)
print(boolean_texto, boolean_texto_vacio)
] ✓ 0.3s
False True True True True
True False
```



VARIABLES - ERRORES TIPICOS

```
variable = 'esta es mi variable'
print[variable]
] ✘ 0.6s
-----
NameError                                                 Traceback (most recent call last)
Cell In[12], line 2
      1 variable = 'esta es mi variable'
----> 2 print(variable)

NameError: name 'varable' is not defined
```



CONSTANTES

CONSTANTE = VARIABLE

En algunos lenguajes de programación estas variables son inmutables

En Python las constantes **no existen como tal**:

Una constante será una variable que no variaremos a lo largo de nuestro código



STRINGS

VARIABLES DE TIPO TEXTO

Son útiles para muchísimos propósitos:

- representar nombres de usuario y contraseñas
- direcciones de email
- mensajes de error
- links

...



STRINGS O CADENAS DE CARACTERES

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = 'El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"'
print(string1)
print(string2)
print(string3)
' 0.2s
to es un texto
to tambien es un texto
otro dia le dije a mi amigo, "Python es mi lenguaje favorito"
```



STRINGS O CADENAS DE CARACTERES

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = "El otro dia le dije a mi amigo, "Python es mi lenguaje favorito"
print(string1)
print(string2)
print(string3)
^ 0.2s

to es un texto
to tambien es un texto
otro dia lc dije a mi amigo, "Python cs mi lcnguajc favorito"
```

string5 = esto pretende ser un texto
print(string5)

✖ 0.5s

Cell In[26], line 1
string5 = esto pretende ser un texto
^
SyntaxError: invalid syntax



STRINGS

VARIABLES DE TIPO TEXTO

```
string1 = "Esto es un texto"
string2 = 'Esto tambien es un texto'
string3 = 'El otro dia le dije a mi amigo, "Python es mi lenguaje"'
print(string1)
print(string2)
print(string3)
```

✓ 0.2s

Esto es un texto
Esto tambien es un texto
El otro dia le dije a mi amigo, "Python es mi lenguaje"

```
string4 = ''
Este texto es completamente inventado:
Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.
...
print(string4)
```

✓ 0.1s

Este texto es completamente inventado:

✓ Lorem ipsum dolor sit amet, consectetur adipiscing elit,
sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation
ullamco laboris nisi ut aliquip ex ea commodo consequat.



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

```
title()  
~~~~~  
nombre = 'juan gomez'  
print(nombre.title())  
✓ 0.6s  
Juan Gomez
```



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

<pre>title() ~~~~~ nombre = 'juan gomez' print(nombre.title()) ✓ 0.6s Juan Gomez</pre>	<pre>upper() ~~~~~ nombre = 'juan gomez' print(nombre.upper()) ✓ 0.6s JUAN GOMEZ</pre>
--	--



STRINGS

Existen **funciones internas** (o prefabricadas) en python para realizar acciones sobre variables o **objetos** que sean de **tipo** texto que puede sernos útiles.

Cambio de mayúscula a minúscula:

title()	upper()	lower()
<pre>nombre = 'juan gomez' print(nombre.title()) ✓ 0.6s</pre> Juan Gomez	<pre>nombre = 'juan gomez' print(nombre.upper()) ✓ 0.6s</pre> JUAN GOMEZ	<pre>nombre = 'jUAn goMeZ' print(nombre.lower()) ✓ 0.2s</pre> juan gomez



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '  
>>> nombre  
'python '  
>>> nombre.rstrip()  
'python'  
>>> nombre  
'python '  
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> █
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> █
```

left strip - lstrip():

```
>>> nombre = ' python'
>>> nombre.lstrip()
'python'
>>> █
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> █
```



STRINGS

Eliminar espacios en blanco:

right strip - rstrip():

```
>>> nombre = 'python '
>>> nombre
'python '
>>> nombre.rstrip()
'python'
>>> nombre
'python '
>>> 
```

```
>>> nombre = nombre.rstrip()
>>> nombre
'python'
>>> 
```

left strip - lstrip():

```
>>> nombre = ' python'
>>> nombre.lstrip()
'python'
>>> 
```

strip():

```
>>> nombre = ' python '
>>> nombre
' python '
>>> nombre.strip()
'python'
>>> 
```



STRINGS

Sustituir partes del string:

replace():

```
>>> string = 'Hola.Mundo'
>>> print(string.replace(".", " "))
Hola Mundo
>>> 
```



STRINGS

Sustituir partes del string:

replace():

```
>>> string = 'Hola.Mundo'
>>> print(string.replace(".", " "))
Hola Mundo
>>> █
```

Encontrar un string dentro de otro string:

find():

```
>>> string = 'Hola Mundo'
>>> print(string.find('Hol'))
0
>>> print(string.find('do'))
8
>>> print(string.find('hey'))
-1
>>> █
```



STRINGS

Comprueba que el string empieza de cierta forma :

startswith():

```
>>> string = 'Hola Mundo'
>>> print(string.startswith('Hol'))
True
>>> print(string.startswith('Mun'))
False
>>> █
```

Comprueba que el string termina de cierta forma :

endswith():

```
>>> string = 'Hola Mundo'
>>> print(string.endswith('do'))
True
>>> print(string.endswith('Ho'))
False
>>> █
```



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'
apellido = 'gomez'
nombre_completo = nombre + apellido
print(nombre_completo)
✓ 0.3s
juangomez
```



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'
apellido = 'gomez'
nombre_completo = nombre +(" ") + apellido
print(nombre_completo)
✓ 0.3s
juan gomez
```



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print(nombre_completo)
```

✓ 0.3s

juan gomez

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print("¡Hola, " + nombre_completo.title() + "!")
```

✓ 0.6s

¡Hola, Juan Gomez!



STRINGS

Combinar y concatenar texto:

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print(nombre_completo)
```

✓ 0.3s

juan gomez

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
print("¡Hola, " + nombre_completo.title() + "!")
```

✓ 0.6s

¡Hola, Juan Gomez!

```
nombre = 'juan'  
apellido = 'gomez'  
nombre_completo = nombre + " " + apellido  
mensaje = "¡Hola, " + nombre_completo.title() + "!"  
print(mensaje)
```

✓ 0.2s

¡Hola, Juan Gomez!



STRINGS

Tabs y saltos de linea:

Tab - \t :

```
>>> print("Python")
Python
>>> print("\tPython")
      Python
```



STRINGS

Tabs y saltos de linea:

Tab - \t :

```
>>> print("Python")
Python
>>> print("\tPython")
      Python
```

Salto de linea - \n :

```
>>> print("Lenguajes:\nPython\nJavaScript\nSolidity")
Lenguajes:
Python
JavaScript
Solidity
>>> █
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en 0



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> █
```

Los índices comienzan en 0

O extraer partes del mismo:

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> █
```

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> █
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> 
```

Los índices comienzan en **0**

O extraer partes del mismo: de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> 
```

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> 
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'  
>>> print(nombre[0])  
J  
>>> 
```

Los índices comienzan en **0**

O extraer partes del mismo: de **0** a **4**

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[0:5])  
YoSoy  
>>> 
```

```
>>> usuario = 'YoSoyJuan'  
>>> print(usuario[5:9])  
Juan  
>>> 
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'
>>> print(nombre[0])
J
>>> 
```

Los índices comienzan en **0**

Revertir un string:

```
>>> cadena = 'abcde'
>>> print(cadena[::-1])
edcba
>>> 
```

O extraer partes del mismo:

de **0** a **4**

de **5** a **8**

```
>>> usuario = 'YoSoyJuan'
>>> print(usuario[0:5])
YoSoy
>>> 
```

```
>>> usuario = 'YoSoyJuan'
>>> print(usuario[5:9])
Juan
>>> 
```



STRINGS

Podemos acceder a los componentes del string mediante sus índices:

```
>>> nombre = 'Juan'
>>> print(nombre[0])
J
>>> 
```

Los índices comienzan en **0**

Revertir un string:

```
>>> cadena = 'abcde'
>>> print(cadena[::-1])
edcba
>>> 
```

O extraer partes del mismo:

de **0** a **4**

de **5** a **8**

```
>>> usuario = 'YoSoyJuan'
>>> print(usuario[0:5])
YoSoy
>>> 
```

```
>>> usuario = 'YoSoyJuan'
>>> print(usuario[5:9])
Juan
>>> 
```

Consultar el tamaño de un string:

```
>>> cadena = 'abcde'
>>> print(len(cadena))
5
>>> 
```



STRINGS - ERRORES TÍPICOS

```
>>> mensaje = "Jean le Rond d'Alembert fue un gran matematico"
>>> print(mensaje)
Jean le Rond d'Alembert fue un gran matematico
>>> █
```

```
>>> mensaje = 'Jean le Rond d'Alembert fue un gran matematico'
      File "<stdin>", line 1
        mensaje = 'Jean le Rond d'Alembert fue un gran matematico'
                           ^
SyntaxError: invalid syntax
```

Syntax Error significa que el interprete no reconoce esta parte del código como código válido de python



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> █
```

Suma (+), resta (-),
multiplicación (*),
división (/)



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> ■
```

3 ↑ 2

```
>>> 3**2
9
>>> 3**3
27
>>> 10**6
1000000
>>> ■
```

potencia (**)

Suma (+), resta(-),
multiplicación (*),
división (/)



NÚMEROS

Enteros o Integers:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> ■
```

3 ↑ 2

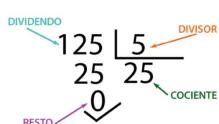
```
>>> 3**2
9
>>> 3**3
27
>>> 10**6
1000000
>>> ■
```

potencia (**)

Suma (+), resta(-),
multiplicación (*),
división (/)

```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> ■
```

modulo o resto (%)





NÚMEROS

Enteros o Integers:

```
>>> 2 + 3
5
>>> 3 - 2
1
>>> 2 * 3
6
>>> 3 / 2
1.5
>>> [REDACTED]
```

Suma (+), resta(-), multiplicación (*), división (/)

3 ↑ 2

```
>>> 3**2
9
>>> 3**3
27
>>> 10**6
1000000
>>> [REDACTED]
```

potencia (**)

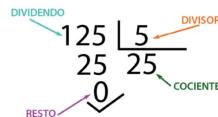
```
>>> 4 % 3
1
>>> 5 % 3
2
>>> 6 % 3
0
>>> [REDACTED]
```

modulo o resto (%)

```
>>> 2 + 3*4
14
>>> (2 + 3) * 4
20
>>> [REDACTED]
```

Python sigue el orden matemático de las operaciones:

1. contenido de los paréntesis
2. exponentes
3. multiplicación y división
4. suma y resta



NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2
0.4
>>> 2 * 0.1
0.2
>>> 2 * 0.2
0.4
>>> 0.2 + 0.5
0.7
```



NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4  
>>> 0.2 + 0.5  
0.7
```

```
>>> 0.2 + 0.1  
0.30000000000000004  
>>> 3 * 0.1  
0.30000000000000004  
>>> █
```



NÚMEROS

Floats o decimales:

```
>>> 0.2 + 0.2  
0.4  
>>> 2 * 0.1  
0.2  
>>> 2 * 0.2  
0.4  
>>> 0.2 + 0.5  
0.7
```

```
>>> 0.2 + 0.1  
0.30000000000000004  
>>> 3 * 0.1  
0.30000000000000004  
>>> █
```



Python intenta darte tantos valores tras la coma como le es posible. Intenta trabajar con la mayor precisión posible.

El origen de esta imprecisión está en como los ordenadores están forzados a representar los números de manera interna.

Ocurre en todos los lenguajes de programación.



COMBINAR NUMEROS Y STRINGS

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + numero_dias + 'dias'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```



COMBINAR NUMEROS Y STRINGS

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + numero_dias + 'dias'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

```
>>> numero_dias = 365
>>> mensaje = 'El año tiene ' + str(numero_dias) + ' dias'
>>> print(mensaje)
El año tiene 365 dias
>>> █
```



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

OBJETIVO:

- Explicar que debe hacer el código
- Explicar cómo funciona sus distintos segmentos



Especialmente importante en trabajos colaborativos

o al reutilizar código antiguo



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

OBJETIVO:

- Explicar que debe hacer el código
- Explicar cómo funciona sus distintos segmentos



Especialmente importante en trabajos colaborativos

o al reutilizar código antiguo

SINTAXIS:

la almohadilla # indica que esa linea es un comentario

```

# Esto es un comentario
# Puedo escribir lo que quiera aqui
# y el interprete lo ignorará
print('hey!')
✓ 0.7s
hey!

```



COMENTARIOS

PARTES DEL SCRIPT IGNORADAS POR EL INTERPRETE - NO SE EJECUTAN

TRUCO:

Los strings que no están asociados a una variable son ignorados por el intérprete.

```

"Esto es un string"
'Esto también es un string'
...
Y esto
también
lo
es
...
print('hey!')
✓ 0.3s
hey!

```

SINTAXIS:

la almohadilla # indica que esa linea es un comentario

```

# Esto es un comentario
# Puedo escribir lo que quiera aqui
# y el interprete lo ignorará
print('hey!')
✓ 0.7s
hey!

```



RESUMEN

1. El uso de variables y su nomenclatura
2. Strings y como trabajar con sus funciones y métodos
3. Como trabajar con números y operaciones aritméticas
4. Combinar números y strings
5. Leer valores de entrada
6. Conversión entre tipos de datos
7. Uso de los comentarios

CONQUER
BLOCKS

CONQUER
BLOCKS

PYTHON

TEST CONDICIONALES, IF STATEMENT Y SWITCH-CASE

CONQUER BLOCKS



REPASO CLASE ANTERIOR

1. El uso de variables y su nomenclatura
2. Strings y como trabajar con sus funciones y métodos
3. Como trabajar con números y operaciones aritméticas
4. Combinar números y strings
5. Leer valores de entrada
6. Conversión entre tipos de datos
7. Uso de los comentarios



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'Fede'  
False  
>>> █
```

Asignamos a la variable nombre_usuario un string con el símbolo ` = ' Juan'.

Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un ` == '.



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'Fede'  
False  
>>> █
```

Asignamos a la variable nombre_usuario un string con el símbolo ` = ' Juan'.

Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un ` == '.



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'Fede'  
False  
>>> █
```

Asignamos a la variable nombre_usuario un string con el símbolo `=`
Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un `==`
Efectivamente nombre_usuario es igual a Juan luego por tanto esa comparación es VERDADERA o TRUE



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'Fede'  
False  
>>> █
```

Asignamos a la variable nombre_usuario un string con el símbolo `=`
Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un `==`
Efectivamente nombre_usuario es igual a Juan luego por tanto esa comparación es VERDADERA o TRUE
nombre_usuario no está asociado al valor Fede, luego el resultado es FALSO o FALSE



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario == 'Juan'
True
>>> nombre_usuario == 'Fede'
False
>>> ■
```

Asignamos a la variable nombre_usuario un string con el símbolo ` = ' .
Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un ` == ' .
Efectivamente nombre_usuario es igual a Juan luego por tanto esa comparación es VERDADERA o TRUE
nombre_usuario no está asociado al valor Fede, luego el resultado es FALSO o FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario != 'Juan'
False
>>> nombre_usuario != 'Fede'
True
>>> ■
```



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario == 'Juan'
True
>>> nombre_usuario == 'Fede'
False
>>> ■
```

Asignamos a la variable nombre_usuario un string con el símbolo ` = ' .
Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un ` == ' .
Efectivamente nombre_usuario es igual a Juan luego por tanto esa comparación es VERDADERA o TRUE
nombre_usuario no está asociado al valor Fede, luego el resultado es FALSO o FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario != 'Juan'
False
>>> nombre_usuario != 'Fede'
True
>>> ■
```

Comparamos si la variable nombre_usuario es DIFERENTE al string Juan usando un ` != ' .
Como nombre_usuario *sí que esta asociado* a la cadena Juan nos da como resultado un FALSE



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario == 'Juan'
True
>>> nombre_usuario == 'Fede'
False
>>> 
```

Asignamos a la variable nombre_usuario un string con el símbolo `=`
 Comparamos si la variable nombre_usuario es IGUAL al string Juan usando un `==`
 Efectivamente nombre_usuario es igual a Juan luego por tanto esa comparación es VERDADERA o TRUE
 nombre_usuario no está asociado al valor Fede, luego el resultado es FALSO o FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario != 'Juan'
False
>>> nombre_usuario != 'Fede'
True
>>> 
```

Comparamos si la variable nombre_usuario es DIFERENTE al string Juan usando un `!=`
 Como nombre_usuario *sí que esta asociado* a la cadena Juan nos da como resultado un FALSE
 Si preguntamos si nombre_usuario es diferente a Fede nos dará un VERDADERO o TRUE



TEST CONDICIONALES

COMPRUEBAN SI ALGO ES CIERTO O FALSO, TRUE/FALSE

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario == 'Juan'
True
>>> nombre_usuario == 'Fede'
False
>>> 
```

```
>>> nombre_usuario = 'Juan'
>>> nombre_usuario != 'Juan'
False
>>> nombre_usuario != 'Fede'
True
>>> 
```

```
>>> nombre = 'Fede'
>>> 'e' in nombre
True
>>> 'a' in nombre
False
>>> 'de' in nombre
True
>>> 
```

También podemos comprobar si cierto string está contenido en otro string



TEST CONDICIONALES

STRINGS: MAYÚSCULAS Y MINÚSCULAS

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'juan'  
False  
>>> nombre_usuario.lower() == 'juan'  
True  
>>> █
```

La condición es sensible a las mayúsculas y las minúsculas (*case sensitive*)



TEST CONDICIONALES

STRINGS: MAYÚSCULAS Y MINÚSCULAS

```
>>> nombre_usuario = 'Juan'  
>>> nombre_usuario == 'Juan'  
True  
>>> nombre_usuario == 'juan'  
False  
>>> nombre_usuario.lower() == 'juan'  
True  
>>> █
```

La condición es sensible a las mayúsculas y las minúsculas (*case sensitive*)

Hacemos la comparación insensible a la forma de nombre_usuario (*case insensitive*)



TEST CONDICIONALES

COMPARACIONES NUMÉRICAS

```
>>> edad = 23
>>> edad == 23
True
>>> edad != 23
False
>>> █
```

```
>>> edad > 24
False
>>> edad < 24
True
>>> edad < 23
False
>>> edad <= 23
True
>>> █
```

Comprobar igualdad o desigualdad



TEST CONDICIONALES

COMPARACIONES NUMÉRICAS

```
>>> edad = 23
>>> edad == 23
True
>>> edad != 23
False
>>> █
```

```
>>> edad > 24
False
>>> edad < 24
True
>>> edad < 23
False
>>> edad <= 23
True
>>> █
```

Comprobar igualdad o desigualdad

Comprobar si es *mayor que* otro numero



TEST CONDICIONALES

COMPARACIONES NUMÉRICAS

```
>>> edad = 23
>>> edad == 23
True
>>> edad != 23
False
>>> █
```

Comprobar igualdad o desigualdad

```
>>> edad > 24
False
>>> edad < 24
True
>>> edad < 23
False
>>> edad <= 23
True
>>> █
```

Comprobar si es *mayor que* otro numero

Comprobar si es *menor que* otro numero



TEST CONDICIONALES

COMPARACIONES NUMÉRICAS

```
>>> edad = 23
>>> edad == 23
True
>>> edad != 23
False
>>> █
```

Comprobar igualdad o desigualdad

```
>>> edad > 24
False
>>> edad < 24
True
>>> edad < 23
False
>>> edad <= 23
True
>>> █
```

Comprobar si es *mayor que* otro numero

Comprobar si es *menor que* otro numero

Comprobar si es *menor o igual que* otro numero



TEST CONDICIONALES

COMPARACIONES NUMÉRICAS

```
>>> edad = 23
>>> edad == 23
True
>>> edad != 23
False
>>> █
```

Comprobar igualdad o desigualdad

```
>>> edad > 24
False
>>> edad < 24
True
>>> edad < 23
False
>>> edad >= 23
True
>>> █
```

Comprobar si es *mayor que* otro numero

Comprobar si es *menor que* otro numero

Comprobar si es *mayor o igual que* otro numero



TESTEAR CONDICIONES MÚLTIPLES

EL USO DE AND Y OR

Muchas veces puede ser útil comprobar si varias condiciones se cumplen simultáneamente.

En estos casos usaremos la palabra *AND*



TESTEAR CONDICIONES MÚLTIPLES

EL USO DE AND Y OR

Muchas veces puede ser útil comprobar si varias condiciones se cumplen simultáneamente.

Ejemplo: Buscamos a una persona que se llama Juan y que sabemos que es tenga 21 años o más.

En estos casos usaremos la palabra *AND*

```
>>> nombre_usuario = 'Juan'  
>>> edad = 23  
>>> (nombre_usuario == 'Juan') and (edad >= 21)  
True  
>>>
```



TESTEAR CONDICIONES MÚLTIPLES

EL USO DE AND Y OR

Muchas veces puede ser útil comprobar si varias condiciones se cumplen simultáneamente.

Ejemplo: Buscamos a una persona que se llama Juan y que sabemos que es tenga 21 años o más.

En estos casos usaremos la palabra *AND*

```
>>> nombre_usuario = 'Juan'  
>>> edad = 16  
>>> (nombre_usuario == 'Juan') and (edad >= 21)  
False  
>>>
```



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

En otras ocasiones nos basta si una de varias condiciones se cumple.

En estos casos usaremos la palabra *OR*



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

En otras ocasiones nos basta si una de varias condiciones se cumple.

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de un mayor de edad

En estos casos usaremos la palabra *OR*

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> (edad_juan > 18) or (edad_lucas > 18)
True
>>> edad_juan = 17
>>> edad_lucas = 16
>>> (edad_juan > 18) or (edad_lucas > 18)
False
```



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

En otras ocasiones nos basta si una de varias condiciones se cumple.

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de un mayor de edad

En estos casos usaremos la palabra *OR*

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> (edad_juan > 18) or (edad_lucas > 18)
True
>>> edad_juan = 17
>>> edad_lucas = 16
>>> (edad_juan > 18) or (edad_lucas > 18)
False
```

Con que uno de los dos lo cumpla será TRUE



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

En otras ocasiones nos basta si una de varias condiciones se cumple.

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de un mayor de edad

En estos casos usaremos la palabra *OR*

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> (edad_juan > 18) and (edad_lucas > 18)
False
>>> edad_juan = 17
>>> edad_lucas = 16
>>> (edad_juan > 18) or (edad_lucas > 18)
False
```

Con el AND ambos deben cumplir la condición



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

En otras ocasiones nos basta si una de varias condiciones se cumple.

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de un mayor de edad

En estos casos usaremos la palabra *OR*

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> (edad_juan > 18) or (edad_lucas > 18)
True
>>> edad_juan = 17
>>> edad_lucas = 16
>>> (edad_juan > 18) or (edad_lucas > 18)
False
```

Como ninguno de los dos lo cumple será FALSE



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

Podemos concatenar tantas condiciones como queramos:



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

Podemos concatenar tantas condiciones como queramos:

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> edad_jose = 28
>>> (edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)
False
```



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

Podemos concatenar tantas condiciones como queramos:

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> edad_jose = 28
>>> (edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)
False
```

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de alguien mayor de 25





TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

Podemos concatenar tantas condiciones como queramos:

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> edad_jose = 28
>>> (edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)
False
```

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de alguien mayor de 25

```
>>> ((edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)) or ((edad_juan > 25) or (edad_lucas > 25) or (edad_jose > 25))
True
>>> █
```



TESTEAR CONDICIONES MÚLTIPLES

El USO DE AND Y OR

Podemos concatenar tantas condiciones como queramos:

```
>>> edad_juan = 17
>>> edad_lucas = 22
>>> edad_jose = 28
>>> (edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)
False
```

Ejemplo: Hay un local en el que solo puede entrar gente mayor de edad o menores acompañados de alguien mayor de 25

```
>>> ((edad_juan > 18) and (edad_lucas > 18) and (edad_jose > 18)) or ((edad_juan > 25) or (edad_lucas > 25) or (edad_jose > 25))
True
>>> █
```



LISTA DE TEST CONDICIONALES

Igualdades	-----	h == 10	
Desigualdades	-----	h != 10	Concatenación de condicionales:
Mayor que	-----	h > 0	and (y) : todas las condiciones deben cumplirse
Mayor o igual que	-----	h >= 0	or (o): una de las condiciones debe cumplirse
Menor que	-----	h < 0	
Menor o igual que	-----	h <= 0	



EXPRESIONES BOOLEANAS

TAN SOLO ES OTRA MANERA DE LLAMAR A LOS TEST CONDICIONALES

Un valor booleano puede ser solo VERDADERO (TRUE) o FALSO (FALSE)

```
>>> proceso_activo = True  
>>> permiso_edicion = False
```

Estos valores pueden usarse para:

- Tener información de si un proceso se está ejecutando
 - Saber si un usuario tiene ciertos permisos sobre un archivo
- ...
- ...



EXPRESIONES BOOLEANAS

TAN SOLO ES OTRA MANERA DE LLAMAR A LOS *TEST CONDICIONALES*

Un valor booleano puede ser solo VERDADERO (TRUE) o FALSO (FALSE)

```
>>> proceso_activo = True  
>>> permiso_edicion = False
```

Nos serán útiles para realizar un seguimiento del estado de nuestro programa

Estos valores pueden usarse para:

- Tener información de si un proceso se está ejecutando
- Saber si un usuario tiene ciertos permisos sobre un archivo

...

...



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

If Statement en Python:



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```



IF STATEMENTS = EXPRESIONES IF

Expresion Si-Entonces de PSEINT:

```
Si expresion_logica Entonces
    ejecucion_codigo_1
SiNo
    ejecucion_codigo_2
FinSi
```

If Statement en Python:

```
if conditional_1:
    ejecucion_de_codigo_1
elif condicional_2:
    ejecucion_de_codigo_2
else:
    ejecucion_de_codigo_3
```

La INDENTACION es necesaria en Python



IF STATEMENTS = EXPRESIONES IF

PSEINT

Python



IF STATEMENTS = EXPRESIONES IF

PSEINT

Python

```
//Definición e inicialización
//de variable precio
Definir precio Como Entero
precio = 0
//Leemos el nuevo dato y se
//almacena en "precio"
Leer precio
//Condicional SI-ENTONCES
Si precio < 20 Entonces
    Escribir "El precio es menor a 20. Es: ", precio
SiNo
    Escribir "El precio es mayor o igual a 20. Es: ", precio
FinSi
```



IF STATEMENTS = EXPRESIONES IF

PSEINT

Python

```
//Definición e inicialización
//de variable precio
Definir precio Como Entero
precio = 0
//Leemos el nuevo dato y se
//almacena en "precio"
Leer precio
//Condicional SI-ENTONCES
Si precio < 20 Entonces
    Escribir "El precio es menor a 20. Es: ", precio
SiNo
    Escribir "El precio es mayor o igual a 20. Es: ", precio
FinSi
```

```
#Leemos el nuevo dato
#almacena en precio
precio = int(input())
# condicional IF STATEMENT
if precio < 20:
    print('El precio es menor a 20. Es ', precio)
else:
    print('El precio es mayor a 20. Es ', precio)
```



IF STATEMENTS ANIDADOS

PSEINT

Python



IF STATEMENTS ANIDADOS

PSEINT

Python

```
Si expresion_logica_1 Entonces
    Si expresion_logica_2 Entonces
        ejecucion_codigo_1
    SiNo
        ejecucion_codigo_2
    FinSi
SiNo
    Si expresion_logica_3 Entonces
        ejecucion_codigo_3
    SiNo
        ejecucion_codigo_4
    FinSi
FinSi
```



IF STATEMENTS ANIDADOS

PSEINT

```
Si expresion_logica_1 Entonces
    Si expresion_logica_2 Entonces
        ejecucion_codigo_1
    SiNo
        ejecucion_codigo_2
    FinSi
SiNo
    Si expresion_logica_3 Entonces
        ejecucion_codigo_3
    SiNo
        ejecucion_codigo_4
    FinSi
FinSi
```

Python

```
if condicional_1:
    if condicional_a:
        ejecucion_de_codigo_a
    else:
        ejecucion_de_codigo_b
else:
    if condicional_c:
        ejecucion_de_codigo_c
    elif condicional_d:
        ejecucion_de_codigo_d
    else:
        ejecucion_de_codigo_e
```



IF STATEMENTS ANIDADOS

PSEINT

Python



IF STATEMENTS ANIDADOS

PSEINT

```
//Definición e inicialización
Definir hasPagado Como Logica
Definir precio Como Entero
hasPagado = Falso
precio = 0
//Leemos los datos
Leer hasPagado
Leer precio
//Condicional SI-ENTONCES anidado
Si hasPagado = Falso Entonces
    Escribir "No has pagado aún"
    Si precio ≤ 20 Entonces
        Escribir "Tienes que pagar menos de 20 euros"
    SiNo
        Escribir "Tienes que pagar más de 20 euros"
    FinSi
SiNo
    Escribir "Ya has pagado"
FinSi
```

Python

```
# Leemos los datos
# que necesitamos de entrada
hasPagado = input()
precio = float(input())
#condicional IF anidado
if hasPagado == "False":
    print("No has pagado aún")
    if precio <= 20:
        print("Tienes que pagar menos de 20 euros")
    else:
        print("Tienes que pagar más de 20 euros")
else:
    print("Ya has pagado")
```



IF STATEMENTS ANIDADOS

PSEINT

```
//Definición e inicialización
Definir hasPagado Como Logica
Definir precio Como Entero
hasPagado = Falso
precio = 0
//Leemos los datos
Leer hasPagado
Leer precio
//Condicional SI-ENTONCES anidado
Si hasPagado = Falso Entonces
    Escribir "No has pagado aún"
    Si precio ≤ 20 Entonces
        Escribir "Tienes que pagar menos de 20 euros"
    SiNo
        Escribir "Tienes que pagar más de 20 euros"
    FinSi
SiNo
    Escribir "Ya has pagado"
FinSi
```

Python



ESTRUCTURA SEGÚN O SWITCH-CASE

PSEINT

Python



ESTRUCTURA SEGÚN O SWITCH-CASE

PSEINT

Python

```
Definir nombre Como Texto
nombre = ""

Escribir "Introduce tu nombre"
Ler nombre

Segun nombre Hacer
    "Juan":
        Escribir "Bienvenido Juan!"
    "Maria":
        Escribir "Bienvenida Maria!"
    "Pepa":
        Escribir "Bienvenida Pepa!"
De otro Modo:
    Escribir "Bienvenido, seas quien seas!"
FinSegun
```



ESTRUCTURA SEGÚN O SWITCH-CASE

PSEINT

```

Definir nombre Como Texto
nombre = ""

Escribir "Introduce tu nombre"
Leer nombre

Segun nombre Hacer
    "Juan":
        Escribir "Bienvenido Juan!"
    "Maria":
        Escribir "Bienvenida Maria!"
    "Pepa":
        Escribir "Bienvenida Pepa!"
De otro Modo:
    Escribir "Bienvenido, seas quien seas!"
FinSegun

```

Python

En Python no existe la estructura switch-case (según)

Podemos imitar su funcionamiento con el if-statement



ESTRUCTURA SEGÚN O SWITCH-CASE

PSEINT

```

Definir nombre Como Texto
nombre = ""

Escribir "Introduce tu nombre"
Leer nombre

Segun nombre Hacer
    "Juan":
        Escribir "Bienvenido Juan!"
    "Maria":
        Escribir "Bienvenida Maria!"
    "Pepa":
        Escribir "Bienvenida Pepa!"
De otro Modo:
    Escribir "Bienvenido, seas quien seas!"
FinSegun

```

Python

```

# leemos el nombre
nombre = input('Introduce tu nombre')

# usamos condicional IF
if nombre == 'Juan':
    print('¡Bienvenido, Juan!')
elif nombre == 'Maria':
    print('¡Bienvenida, Maria!')
elif nombre == 'Pepa':
    print('¡Bienvenida, Pepa!')
else:
    print('¡Bienvenido, seas quien seas!')

```



REPASO

1. Test condicionales simples
2. Test condicionales múltiples
3. Expresión If simple
4. Expresión If anidada
5. Sustitución del switch case por expresiones If

CONQUER
BLOCKS

CONQUER
BLOCKS

PYTHON

AMA 01.02.2023

CONQUER**BLOCKS**



PYTHON DOCS

Link a la tabla de contenidos: <https://docs.python.org/es/3.9/contents.html>

Metodos para integers: <https://docs.python.org/es/3.9/library/stdtypes.html#additional-methods-on-integer-types>

Metodos para floats: <https://docs.python.org/es/3.9/library/stdtypes.html#additional-methods-on-float>

Metodos para strings: <https://docs.python.org/es/3.9/library/stdtypes.html#string-methods>



REPOSITORIOS DE GITHUB INTERESANTES

30-Days-Of-Python: <https://github.com/Asabeneh/30-Days-Of-Python> (Principiante)

Learn-python: https://github.com/trekhleb/learn-python/tree/master/src/control_flow (Principiante)

30-Seconds-Of-Python: <https://github.com/30-seconds/30-seconds-of-python/tree/master/snippets> (Principiante)

También en javascript y react!

Practical-Python: <https://github.com/dabeaz-course/practical-python/blob/master/Notes/Contents.md> (Nivel Intermedio)

Geek-Python: <https://github.com/geekcomputers/Python> (Nivel Intermedio)

The Algorithm: <https://github.com/TheAlgorithms/Python> (Nivel Intermedio) También en javascript!



¿ES CORRECTO LLAMAR A LAS VARIABLES DE NUESTRO PROGRAMA IGUAL QUE A LOS ARGUMENTOS DE NUESTRA FUNCIÓN?

script de python:

```
def funcion_suma(numero_1, numero_2):
    suma = numero_1 + numero_2
    result(suma)
```

```
variable_1 = 30
variable_2 = 40
resultado = funcion_suma(variable_1, variable_2)
print(resultado)
```



¿ES CORRECTO LLAMAR A LAS VARIABLES DE NUESTRO PROGRAMA IGUAL QUE A LOS ARGUMENTOS DE NUESTRA FUNCIÓN?

```
def funcion_suma(numero_1, numero_2):
    suma = numero_1 + numero_2
    result(suma)
```



Una función es un bloque de código reutilizable —> Los nombres deben ser lo más generales posibles dentro del objetivo que tenga la función.

script de python:

```
numero_1 = 30
numero_2 = 40
resultado = funcion_suma(numero_1, numero_2)
print(resultado)
```



Los nombres deben adecuarse al caso particular del programa.



¿ES CORRECTO LLAMAR A LAS VARIABLES DE NUESTRO PROGRAMA IGUAL QUE A LOS ARGUMENTOS DE NUESTRA FUNCIÓN?

```
def funcion_suma(sumando_1, sumando_2):
    suma = sumando_1 + sumando_2
    result(suma)
```



Una función es un bloque de código reutilizable —> Los nombres deben ser lo más generales posibles dentro del objetivo que tenga la función.

script de python —> calcula ingresos:

```
ingreso_1 = 30
ingreso_2 = 40
ingresos_totales = funcion_suma(ingreso_1, ingreso_2)
print(resultado)
```



Los nombres deben adecuarse al caso particular del programa.

CONQUER
BLOCKS



PYTHON

LISTAS Y ESTRUCTURAS ITERATIVAS (PARTE 1)

CONQUER BLOCKS



REPASO CLASE ANTERIOR

1. Test condicionales simples
2. Test condicionales múltiples
3. Expresión If simple
4. Expresión If anidada
5. Sustitución del switch case por expresiones If



LISTAS

¿QUÉ ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

Lista1 = ["a", "b", "c", "d", "e", "f"]

Lista2 = [1, 2, 3, 4, 5, 6]

Lista3 = [1, 3, 7, 2, 5, 1, 13]

Lista4 = [1, "a", "b", 3, 7, "e"]



LISTAS

¿QUÉ ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

Lista1 = ["a", "b", "c", "d", "e", "f"]

Lista2 = [1, 2, 3, 4, 5, 6]

Lista3 = [1, 3, 7, 2, 5, 1, 13]

Lista4 = [1, "a", "b", 3, 7, "e"]



LISTAS

¿QUE ES UNA LISTA?

Una Lista es una colección de objetos en un orden particular.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones)
✓ 0.1s
['bote', 'yate', 'velero', 'catamarán']
```



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones[0])
✓ 0.7s
bote
```



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones[0])
✓ 0.7s
bote
```

¡Es un string!



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(type(embarcaciones))
print(type(embarcaciones[0]))
✓ 0.8s
<class 'list'>
<class 'str'>
```



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones[0].title())
```

✓ 0.6s

Bote



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones[1], embarcaciones[2])
```

✓ 0.8s

yate velero



LISTAS

ACCEDER A LOS ELEMENTOS DE UNA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
print(embarcaciones[-1])
✓ 0.1s
catamarán
```



LISTAS

USAR UN ELEMENTO DE LA LISTA

Se trata de una colección ordenada así que podemos acceder a los elementos de la colección diciéndole a Python el índice del elemento al que queremos acceder.

```
embarcaciones = ['bote', 'yate', 'velero', 'catamarán']
mensaje = 'Si me comprase una embarcación, elegiría un ' + embarcaciones[-1].title()
print(mensaje)
✓ 0.9s
Si me comprase una embarcación, elegiría un Catamarán
```



LISTAS

MODIFICAR ELEMENTOS

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coches[2] = 'mercedes'
print(coches)

✓ 0.6s

['bmw', 'audi', 'seat']
['bmw', 'audi', 'mercedes']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Append() nos permite añadir elementos a una lista en la última posición.

APPEND()

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coches.append('mercedes')
print(coches)

✓ 0.1s

['bmw', 'audi', 'seat']
['bmw', 'audi', 'seat', 'mercedes']
```

Las listas son objetos dinámicos. Se les puede asignar más memoria de manera dinámica.



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Append() nos permite añadir elementos a una lista de manera dinámica.

```
coches = []
coches.append('bmw')
coches.append('audi')
coches.append('mercedes')
print(coches)

✓ 0.9s

['bmw', 'audi', 'mercedes']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Insert() nos permite añadir elementos a una lista en cualquier posición

INSERT()

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coches.insert(0, 'mercedes')
print(coches)

✓ 0.1s

['bmw', 'audi', 'seat']
['mercedes', 'bmw', 'audi', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

AÑADIR ELEMENTOS: Insert() nos permite añadir elementos a una lista en cualquier posición

INSERT()

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coches.insert(1, 'mercedes')
print(coches)

✓ 0.8s

['bmw', 'audi', 'seat']
['bmw', 'mercedes', 'audi', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coches.pop()
print(coches)

✓ 0.1s

['bmw', 'audi', 'seat']
['bmw', 'audi']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
coches = ['bmw', 'audi', 'seat']
print(coches)
coche_eliminado = coches.pop()
print(coches)
print(coche_eliminado)

✓ 0.1s
['bmw', 'audi', 'seat']
['bmw', 'audi']
seat
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

POP()

```
coches = ['bmw', 'audi', 'seat']
coche_eliminado = coches.pop(1)
print(coche_eliminado)
print(coches)
```

✓ 0.1s

audi

```
['bmw', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

REMOVE()

```
coches = ['bmw', 'audi', 'seat']
coches.remove('audi')
print(coches)
```

✓ 0.7s

```
['bmw', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

REMOVE()

```
coches = ['bmw', 'audi', 'seat', 'audi']
```

```
coches.remove('audi')
```

```
print(coches)
```

✓ 0.1s

```
['bmw', 'seat', 'audi']
```

Elimina la primera aparición del elemento indicado



LISTAS

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

DEL - Palabra Clave (Keyword)

```
coches = ['bmw', 'audi', 'seat']
del coches[1]
print(coches)

✓ 0.1s

['bmw', 'seat']
```



LISTAS

BORRAR ELEMENTOS: Igual que podemos insertar elementos en una lista, también podemos borrarlos.

DEL - Palabra Clave (Keyword)

```
coches = ['bmw', 'audi', 'seat']
del coches
print(coches)

✖ 0.8s

NameError: name 'coches' is not defined
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *permanente*

SORT()

<pre>coches = ['bmw', 'audi', 'seat'] coches.sort() print(coches)</pre>	<pre>coches = [23, 11, 5] coches.sort() print(coches)</pre>
<p>✓ 0.7s</p> <pre>['audi', 'bmw', 'seat']</pre>	<p>✓ 0.6s</p> <pre>[5, 11, 23]</pre>



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *permanente*

SORT()

<pre>coches = ['bmw', 'audi', 'seat'] coches.sort() print(coches)</pre>	<pre>coches = ['23', '11', '5'] coches.sort() print(coches)</pre>
<p>✓ 0.7s</p> <pre>['audi', 'bmw', 'seat']</pre>	<p>✓ 0.9s</p> <pre>['11', '23', '5']</pre>



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *temporal*

SORTED()

```
coches = ['bmw', 'audi', 'seat']
print(sorted(coches))

✓ 0.9s

['audi', 'bmw', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos ordenar elementos por orden alfabético o por número de manera *temporal*

SORTED()

```
coches = ['bmw', 'audi', 'seat']
autos = sorted(coches)
print(autos)
print(sorted(coches))

✓ 0.1s

['audi', 'bmw', 'seat']
['audi', 'bmw', 'seat']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

ORDENAR ELEMENTOS: Podemos imprimir la lista en orden inverso.

REVERSE()

```
coches = ['bmw', 'audi', 'seat']
coches.reverse()
print(coches)
```

✓ 0.1s

```
['seat', 'audi', 'bmw']
```



LISTAS

LAS LISTAS TAMBIÉN TIENEN ASOCIADAS UNAS FUNCIONES INTERNAS O MÉTODOS EN PYTHON

LONGITUD DE UNA LISTA

LEN()

```
coches = ['bmw', 'audi', 'seat']
print(len(coches))
```

✓ 0.5s

3



LISTAS

DEBUGGING

```
coches = ['bmw', 'audi', 'seat']
print(coches[3])

⊗ 0.9s

-----
IndexError                                Traceback (most recent call last)
Cell In[43], line 2
      1 coches = ['bmw', 'audi', 'seat']
----> 2 print(coches[3])

IndexError: list index out of range
```



LISTAS

DEBUGGING

```
coches = []
print(coches[0])

⊗ 0.1s

-----
IndexError                                Traceback (most recent call last)
Cell In[45], line 2
      1 coches = []
----> 2 print(coches[0])

IndexError: list index out of range
```



LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si un elemento está en la lista

```
coches = ['bmw', 'audi', 'seat']
coche_elegido = 'audi'
if coche_elegido in coches:
    print('Has escogido un', coche_elegido )
else:
    print('No tenemos esa marca de coche')
✓ 0.1s
```

Has escogido un audi



LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si un elemento está en la lista

```
coches = ['bmw', 'audi', 'seat']
coche_elegido = 'mercedes'
if coche_elegido in coches:
    print('Has escogido un', coche_elegido )
else:
    print('No tenemos esa marca de coche')
✓ 0.9s
```

No tenemos esa marca de coche



LISTAS Y ESTRUCTURA CONDICIONAL

Podemos comprobar si la lista tiene contenido

```
coches = []
if coches:
    print('La lista tiene el siguiente contenido ', coches)
else:
    print('La lista está vacía')
✓ 0.1s
```

La lista está vacía



LISTAS Y ESTRUCTURA CONDICIONAL

```
coches = ['seat']
if coches:
    print('La lista tiene el siguiente contenido ', coches)
else:
    print('La lista está vacía')
✓ 0.3s
```

La lista tiene el siguiente contenido ['seat']



ESTRUCTURAS ITERATIVAS

¿QUÉ ES UNA ESTRUCTURA ITERATIVA O BUCLE?

Es una secuencia de instrucciones que se ejecuta repetidamente mientras se cumple cierta condición establecida previamente.

ESTRUCTURAS ITERATIVAS EN PYTHON



WHILE	= MIENTRAS
FOR	= PARA



ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Mientras expr_logica Hacer
    ejecucion_codigo
FinMientras

Mientras (expr_logica_1) OP_LOG (expr_logica_2) Hacer
    ejecucion_codigo
FinMientras
```

PYTHON

```
while expr_logica:
    ejecucion_codigo
while (expr_logica_1) and/or (expr_logica_2)
    ejecucion_codigo
```

Para entrar en el ciclo WHILE debe cumplirse la condición de la expresión lógica.

Pero debe haber alguna opción para que la expresión no se cumpla y se salga del bucle.



Debe cambiarse el valor de la expresión lógica



ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Definir temporizador Como Entero
temporizador = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Mientras (temporizador > 0) Hacer
    Escribir "Quedan ", temporizador, " segundos"
    Esperar 1 Segundos
    temporizador = temporizador - 1
FinMientras
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input(''Introduce el número de
segundos del temporizador''))

print('Comienza el temporizador')

while (temporizador > 0):
    print('Quedan ', temporizador, ' segundos')
    time.sleep(1)
    temporizador = temporizador - 1

print('El temporizador ha finalizado!')
```



ESTRUCTURAS ITERATIVAS

BUCLE MIENTRAS O WHILE

PSEINT

```
Definir temporizador Como Entero
temporizador = 0

Escribir "Introduce el número de
Leer temporizador

Escribir "Comienza el temporizador"
Mientras (temporizador > 0) Hace
    Escribir "Quedan ", temporizad
    Esperar 1 Segundos
    temporizador = temporizador
FinMientras
Escribir "El temporizador ha finalizado!!"
```

Introduce el número de
segundos del temporizador 3
Comienza el temporizador
Quedan 3 segundos
Quedan 2 segundos
Quedan 1 segundos
iEl temporizador ha finalizado!

PYTHON

```
import time
temporizador = int(input(''Introduce el número de
segundos del temporizador''))

temporizador)

while (temporizador > 0):
    print('Quedan ', temporizador, ' segundos')
    temporizador = temporizador - 1

print('El temporizador ha finalizado!')
```



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Para i = valor_inicial Hasta valor_final Con Paso incr_decr Hacer
    ejecucion_codigo
FinPara
```

PYTHON

```
for i in range(valor_inicial, valor_final, paso):
    ejecucion_codigo
```

El número de repeticiones viene dado por el valor de la variable i

No es necesario incluir una instrucción para cambiar la condición del bucle.



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!"
```

PYTHON

```
import time
temporizador = int(input('''Introduce el número de
segundos del temporizador'''))

print('Comienza el temporizador')

for i in range(0, temporizador):
    print('Quedan ', temporizador, ' segundos')
    time.sleep(1)
    temporizador = temporizador - 1

print('El temporizador ha finalizado!')
```



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de segundos del temporizador'))

print('Comienza el temporizador')

for i in range(temporizador,0,-1):
    print('Quedan ', i, ' segundos')
    time.sleep(1)

print('El temporizador ha finalizado!')
```



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador"
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
import time
temporizador = int(input('Introduce el número de segundos del temporizador'))

Comienza el temporizador
Quedan 3 segundos
Quedan 2 segundos
Quedan 1 segundos
iEl temporizador ha finalizado!

temporizador = temporizador - 1
print('El temporizador ha finalizado!')
```



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
for i in range(0,3):
    print(i)
✓ 0.9s
```

0
1 i va de 0 a 2
2



ESTRUCTURAS ITERATIVAS

BUCLE FOR O PARA

PSEINT

```
Definir i, temporizador Como Entero
temporizador = 0
i = 0

Escribir "Introduce el número de segundos del temporizador"
Leer temporizador

Escribir "Comienza el temporizador..."
Para i = temporizador Hasta 1 Con paso -1 Hacer
    Escribir "Quedan ", i, " segundos"
    Esperar 1 Segundos
FinPara
Escribir "El temporizador ha finalizado!!"
```

PYTHON

```
for i in range(1,3):
    print(i)
✓ 0.7s
```

1 i va de 1 a 2
2



ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Exit point

```
while condicion
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Exit point



ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Exit point

```
for i in range(5):
    if i == 3:
        break
    print(i)

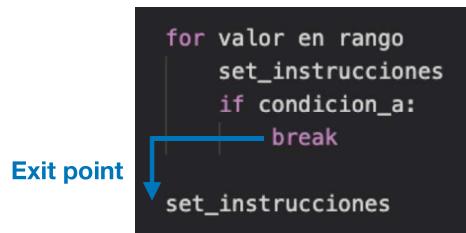
print("Estamos fuera del bucle")
✓ 0.2s
```

0
1
2
Estamos fuera del bucle



ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *BREAK*



Exit point

```

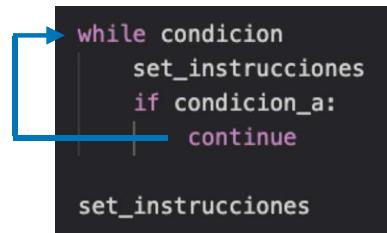
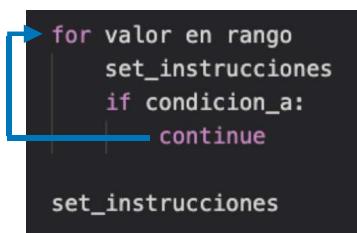
for i in range(5):
    if i == 3:
        print("Estoy dentro del if")
        break
    print("Estoy dentro del if y",
          "despues del break")
    print(i)
print("Estamos fuera del bucle")
  
```

0
1
2
Estoy dentro del if
Estamos fuera del bucle



ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: *CONTINUE*





ESTRUCTURAS ITERATIVAS

INTERRUMPIR UN BUCLE: CONTINUE

```

for valor en rango
    set_instrucciones
    if condicion_a:
        continue

    set_instrucciones

```

```

for i in range(5):
    if i == 3:
        print("Estoy dentro del if")
        continue
        print("Estoy dentro del if y",
              "despues del break")
        print(i)

    print("Estamos fuera del bucle")
    ✓ 0.1s

0
1
2
Estoy dentro del if
4
Estamos fuera del bucle

```



ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

→ Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

Exit point

```

for valor en rango
    set_instrucciones
    if condicion_a:
        break

    set_instrucciones

```

```

for valor en rango
    set_instrucciones
    if condicion_a:
        continue

    set_instrucciones

```



ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

- Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.

```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
    set_instrucciones
```



ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

- Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.

Casos de uso:

Para crear comprobaciones rápidas al comienzo del programa → actual como pre-condiciones.

```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
    set_instrucciones
```



ESTRUCTURAS ITERATIVAS

CONTINUE y BREAK

Exit point

```
for valor en rango
    set_instrucciones
    if condicion_a:
        break
    set_instrucciones
```

Son estructuras muy útiles y cómodas a la hora de escribir el código pero hay que intentar no abusar de ellas.

- Crear muchos *exit points* complica la lectura y el flujo del código, es preferible que se priorice salir del bucle de manera natural.

La base fundamental de la **programación estructurada** es tener unos puntos de entrada (entry points) y puntos de salida (exit points) bien definidos.

Casos de uso:

Para crear comprobaciones rápidas al comienzo del programa → actual como pre-condiciones.

Regla de estilo a seguir:

Echa un ojo al código. Piensa si hay alguna manera de escribirlo sin usar *break* o *continue* y que mantenga el código sencillo y elegante. Si ves que esto no es posible entonces usa *break* o *continue*.

```
for valor en rango
    set_instrucciones
    if condicion_a:
        continue
    set_instrucciones
```

**CONQUER
BLOCKS**



PYTHON

LISTAS Y ESTRUCTURAS ITERATIVAS (PARTE 2)



REPASO CLASE ANTERIOR

1. Listas como una colección dinámica de elementos o objetos

- Acceder a los elementos
- Añadir elementos
- Borrar elementos

2. Estructuras iterativas o bucles

- Bucle while
- Bucle for
- Uso y contexto de *break* y *continue*



LISTAS Y ESTRUCTURAS ITERATIVAS

RECORRER UNA LISTA CON UN BUCLE

```
for objeto in lista:  
|   print(objeto)
```

```
coches = ['bmw', 'audi', 'seat']  
for coche in coches:  
|   print(coche)
```

✓ 0.1s

```
bmw  
audi  
seat
```



LISTAS Y ESTRUCTURAS ITERATIVAS

RECORRER UNA LISTA CON UN BUCLE

```
for objeto in lista:  
|   print(objeto)
```

```
coches = ['bmw', 'audi', 'seat']  
for coche in coches:  
|   print(coche)
```

✓ 0.1s

```
bmw  
audi  
seat
```

```
for i in range(0,len(lista)):  
|   print(lista[i])
```

```
coches = ['bmw', 'audi', 'seat']  
for i in range(0,len(coches)):  
|   print(coches[i])
```

✓ 0.7s

```
bmw  
audi  
seat
```



LISTAS Y ESTRUCTURAS ITERATIVAS

RECORRER UNA LISTA CON UN BUCLE

```
coches = ['bmw', 'audi', 'seat']
for i in range(0,len(coches)):
    print('este coche es un ',coches[i].title())
print('Lista de coches terminada')

✓ 0.1s

este coche es un  Bmw
este coche es un  Audi
este coche es un  Seat
Lista de coches terminada
```



LISTAS Y ESTRUCTURAS ITERATIVAS

RECORRER UNA LISTA CON UN BUCLE

```
coches = ['bmw', 'audi', 'seat']
for i in range(0,len(coches)):
    print('este coche es un ',coches[i].title())
print('Lista de coches terminada')

✓ 0.1s

este coche es un  Bmw
este coche es un  Audi
este coche es un  Seat
Lista de coches terminada
```

DENTRO DEL BUCLE

FUERA DEL BUCLE



LISTAS Y ESTRUCTURAS ITERATIVAS

RECORRER UNA LISTA CON UN BUCLE

```
coches = ['bmw', 'audi', 'seat']
for i in range(0,len(coches)):
    print('este coche es un ',coches[i].title())
    print('Lista de coches terminada')
✓ 0.4s

este coche es un  Bmw
Lista de coches terminada
este coche es un  Audi
Lista de coches terminada
este coche es un  Seat
Lista de coches terminada
```

DENTRO DEL BUCLE

DENTRO DEL BUCLE



LISTAS Y ESTRUCTURAS ITERATIVAS

DEBUGGING

```
coches = ['bmw', 'audi', 'seat']
for i in range(0,len(coches)):
    print('este coche es un ',coches[i].title())

⊗ 0.7s

Cell In[66], line 3
    print('este coche es un ',coches[i].title())
    ^
IndentationError: expected an indented block
```

FALTA DE SANGÍA TRAS EL LA DECLARACIÓN DEL FOR



LISTAS Y ESTRUCTURAS ITERATIVAS

DEBUGGING

```
coches = ['bmw', 'audi', 'seat']
for i in range(0,len(coches))
|   print('este coche es un ',coches[i].title())
```

⊗ 0.8s

```
Cell In[69], line 2
for i in range(0,len(coches))
^
SyntaxError: invalid syntax
```

ERROR DE SINTAXIS. FALTAN LOS DOS PUNTOS TRAS EL FOR



LISTAS Y ESTRUCTURAS ITERATIVAS

DEBUGGING

```
print('Aqui no hay sangría')
|   print('Aquí hay sangría')
```

⊗ 0.1s

```
Cell In[67], line 2
|   print('Aquí hay sangría')
^
```

```
IndentationError: unexpected indent
```

SANGRÍA INESPERADA/
INJUSTIFICADA



LISTAS NUMÉRICAS

```
numeros = list(range(1,6))
print(numeros)
```

✓ 0.1s

[1, 2, 3, 4, 5]

```
numeros_cuadrados = []
for valor in range(1,11): # irá del 0 al 10
    cuadrado = valor**2
    numeros_cuadrados.append(cuadrado)
print(numeros_cuadrados)
```

✓ 0.9s

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

```
numeros_pares = list(range(2,11,2))
print(numeros_pares)
```

✓ 0.7s

[2, 4, 6, 8, 10]

Podemos usar la función `range()` para crear listas numéricas



LISTAS NUMÉRICAS – COMPRESIÓN

Declaración extendida

```
numeros_cuadrados = []
for valor in range(1,11): # irá del 0 al 10
    cuadrado = valor**2
    numeros_cuadrados.append(cuadrado)
print(numeros_cuadrados)
```

✓ 0.9s

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

Declaración comprimida

```
numeros_cuadrados = [valor**2 for valor in range(1,11)]
print(numeros_cuadrados)
```

✓ 0.4s

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]



LISTAS NUMÉRICAS

En Python existen funciones específicas para tratar con listas numéricas.

```
>>> digitos = [1,2,3,4,5,6,7,8,9,0]
>>> min(digitos)
0
>>> max(digitos)
9
>>> sum(digitos)
45
```

Encuentra el mínimo de una lista de números
Encuentra el máximo de una lista de números
Encuentra la suma de una lista de números



PARTES DE UNA LISTA

Porción de una lista

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

✓ 0.8s

```
[3, 4, 5]
```



PARTES DE UNA LISTA

Porción de una lista

```
          0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
[3, 4, 5]
```



PARTES DE UNA LISTA

Porción de una lista

```
          0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
[3, 4, 5]
```

```
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[0:3]
equipo_B = jugadores[3:6]

print(equipo_A)
print(equipo_B)
✓ 0.1s
['Alejandro', 'Felipe', 'Samuel']
['Juan Marcos', 'Lucas', 'David']
```



PARTES DE UNA LISTA

Porción de una lista

```
          0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
[3, 4, 5]

jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[:3]
equipo_B = jugadores[0:3]

print(equipo_A)
print(equipo_B)
✓ 0.2s
['Alejandro', 'Felipe', 'Samuel']
['Alejandro', 'Felipe', 'Samuel']
```



PARTES DE UNA LISTA

Porción de una lista

```
          0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
[3, 4, 5]

jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[3:]
equipo_B = jugadores[3:6]

print(equipo_A)
print(equipo_B)
✓ 0.1s
['Juan Marcos', 'Lucas', 'David']
['Juan Marcos', 'Lucas', 'David']
```



PARTES DE UNA LISTA

Porción de una lista

```
0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
```

[3, 4, 5]

```
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[3:6]
equipo_B = jugadores[-3:]

print(equipo_A)
print(equipo_B)
✓ 0.1s
```

['Juan Marcos', 'Lucas', 'David']
['Juan Marcos', 'Lucas', 'David']



PARTES DE UNA LISTA

Porción de una lista

```
0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
```

[3, 4, 5]

```
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[3:6]
equipo_B = jugadores[-3:]

print(equipo_A)
print(equipo_B)
✓ 0.1s
```

['Juan Marcos', 'Lucas', 'David']
['Juan Marcos', 'Lucas', 'David']



PARTES DE UNA LISTA

Porción de una lista

```
          0 1 2 3 4 5 6 7 8 9
digitos = [1,2,3,4,5,6,7,8,9,0]
algunos_digitos = digitos[2:5]
print(algunos_digitos)
✓ 0.8s
[3, 4, 5]

          -6   -5   -4   -3   -2   -1
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']
equipo_A = jugadores[3:6]
equipo_B = jugadores[-3:]

print(equipo_A)
print(equipo_B)
✓ 0.1s
['Juan Marcos', 'Lucas', 'David']
['Juan Marcos', 'Lucas', 'David']
```



RECORRER UNA PARTE DE UNA LISTA

```
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']

print('Estos son los jugadores del equipo A:')
for jugador in jugadores[0:3]:
| print(jugador)
✓ 0.1s

Estos son los jugadores del equipo A:
Alejandro
Felipe
Samuel
```



RECORRER UNA PARTE DE UNA LISTA

```
jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']

print('Estos son los jugadores del equipo A:')
for jugador in jugadores[0:3]:
    print(jugador)
✓ 0.1s

jugadores = ['Alejandro', 'Felipe', 'Samuel', 'Juan Marcos', 'Lucas', 'David']

print('Estos son los jugadores del equipo A:')
for i in range(len(jugadores[0:3])):
    print(jugadores[i])
✓ 0.2s

Estos son los jugadores del equipo A:
Alejandro
Felipe
Samuel
```



COPiar UNA LISTA

Podemos crear nuevas listas a partir de listas ya existentes

```
mi_comida = ['pizza', 'carne', 'tarta de queso']
comida_invitado = mi_comida
print(mi_comida)
print(comida_invitado)
✓ 0.1s

['pizza', 'carne', 'tarta de queso']
['pizza', 'carne', 'tarta de queso']
```



COPIAR UNA LISTA

Podemos crear nuevas listas a partir de listas ya existentes

```
mi_comida = ['pizza', 'carne', 'tarta de queso']
comida_invitado = mi_comida
comida_invitado.append('helado')
print(mi_comida)
print(comida_invitado)

✓ 0.2s

['pizza', 'carne', 'tarta de queso', 'helado']
['pizza', 'carne', 'tarta de queso', 'helado']
```



COPIAR UNA LISTA

Podemos crear nuevas listas a partir de listas ya existentes

```
mi_comida = ['pizza', 'carne', 'tarta de queso']
comida_invitado = mi_comida
comida_invitado.append('helado')
print(mi_comida)
print(comida_invitado)

✓ 0.2s

['pizza', 'carne', 'tarta de queso', 'helado']
['pizza', 'carne', 'tarta de queso', 'helado']
```

MEMORIA

mi_comida

[.....]



COPIAR UNA LISTA

Podemos crear nuevas listas a partir de listas ya existentes

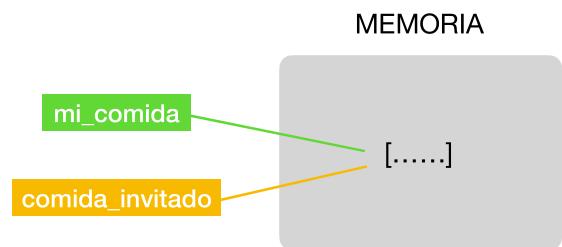
```

mi_comida = ['pizza', 'carne', 'tarta de queso']
comida_invitado = mi_comida
comida_invitado.append('helado')
print(mi_comida)
print(comida_invitado)

✓ 0.2s

['pizza', 'carne', 'tarta de queso', 'helado']
['pizza', 'carne', 'tarta de queso', 'helado']

```



COPIAR UNA LISTA

Podemos crear nuevas listas a partir de listas ya existentes

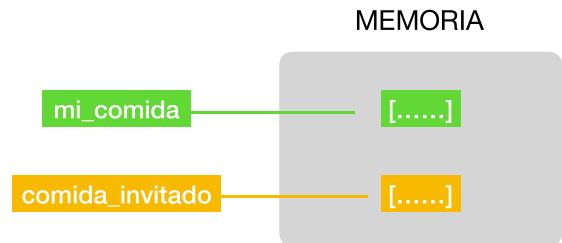
```

mi_comida = ['pizza', 'carne', 'tarta de queso']
comida_invitado = mi_comida[:]
comida_invitado.append('helado')
print(mi_comida)
print(comida_invitado)

✓ 0.1s

['pizza', 'carne', 'tarta de queso']
['pizza', 'carne', 'tarta de queso', 'helado']

```





LISTAS ANIDADAS

Las listas son colecciones de objetos, pero a su vez son un objeto también. Igual que una lista puede contener objetos como strings o números también puede contener otras listas:

```
datos_alumnos = [['David', 27], ['Jose', 22], ['Lucas',23]]  
print(type(datos_alumnos))  
  
✓ 0.9s  
  
<class 'list'>
```



LISTAS ANIDADAS

Las listas son colecciones de objetos, pero a su vez son un objeto también. Igual que una lista puede contener objetos como strings o números también puede contener otras listas:

```
datos_alumnos = [['David', 27], ['Jose', 22], ['Lucas',23]]  
print(type(datos_alumnos))  
print(datos_alumnos[0])  
print(type(datos_alumnos[0]))  
  
✓ 0.5s  
  
<class 'list'>  
['David', 27]  
<class 'list'>
```



LISTAS ANIDADAS

Las listas son colecciones de objetos, pero a su vez son un objeto también. Igual que una lista puede contener objetos como strings o números también puede contener otras listas:

```
datos_alumnos = [['David', 27], ['Jose', 22], ['Lucas',23]]
print(type(datos_alumnos))
print(datos_alumnos[0][0])
print(datos_alumnos[0][1])
print(type(datos_alumnos[0][0]))
```

✓ 0.6s

```
<class 'list'>
David
27
<class 'str'>
```



LISTAS ANIDADAS

Las listas son colecciones de objetos, pero a su vez son un objeto también. Igual que una lista puede contener objetos como strings o números también puede contener otras listas:

```
datos_alumnos = [['David', 27], ['Jose', 22], ['Lucas',23]]
print(type(datos_alumnos))
print(datos_alumnos[0:2])
print(type(datos_alumnos[0]))
```

✓ 0.1s

```
<class 'list'>
[['David', 27], ['Jose', 22]]
<class 'str'>
```



REPASO

Listas = colecciones de objetos

- Acceder a los elementos
- Usar los elementos
- Modificar los elementos
- Añadir elementos
- Borrar elementos
- Ordenar elementos dentro de la lista
- Longitud de la lista
- Listas numéricas + funciones específicas
- Acceder a partes de una lista
- Copiar listas y partes de listas
- Anidamiento de listas

Estructura condicional + Lista → Comprobar existencia de elementos



REPASO

Bucles For y While

- Funcionamiento general
- Uso de *range*
- Usos de *break*
- Usos de *continue*

Estructura iterativa + Lista → Recorrer los elementos

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

ARRAYS, MODULOS Y LIBRERÍAS

CONQUERBLOCKS



QUÉ ES UN MÓDULO

Módulo: Archivo con extensión .py que contiene código de Python que puede importarse dentro de otro archivo de Python.



QUÉ ES UN MÓDULO

Módulo: Archivo con extensión .py que contiene código de Python que puede importarse dentro de otro archivo de Python.

```

constantes.py
1 # Este es un modulo donde guardamos todas
2 # las constantes que podamos necesitar
3
4 """ Constantes """
5 pi = 3.14
6 tau = 6.28
7 e_euler = 2.71
8 G_gauss = 0.83

imprimir_cte.py
1 # --- importamos el modulo de constantes ---
2 import constantes
3
4 print(constantes.pi)

```

TERMINAL

```

(cblocks) MacBook-Pro-4:Prueba Elena$ /Users/Elena/miniconda3/envs/cblocks/bin/python "/Users/Elena/Desktop/Master Conquer/Clase4/Prueba/imprimir_cte.py"
3.14
(cblocks) MacBook-Pro-4:Prueba Elena$ 

```



QUÉ ES UN MÓDULO

Módulo: Archivo con extensión .py que contiene código de Python que puede importarse dentro de otro archivo de Python.

```

constantes.py
1 # Este es un modulo donde guardamos todas
2 # las constantes que podamos necesitar
3
4 """ Constantes """
5 pi = 3.14
6 tau = 6.28
7 e_euler = 2.71
8 G_gauss = 0.83

imprimir_cte.py
1 # --- importamos el modulo de constantes ---
2 import constantes as cte
3
4 print(cte.pi)

```

TERMINAL

```

(cblocks) MacBook-Pro-4:Prueba Elena$ /Users/Elena/miniconda3/envs/cblocks/bin/python "/Users/Elena/Desktop/Master Conquer/Clase4/Prueba/imprimir_cte.py"
3.14
(cblocks) MacBook-Pro-4:Prueba Elena$ 

```



QUÉ ES UN MÓDULO

Módulo: Archivo con extensión .py que contiene código de Python que puede importarse dentro de otro archivo de Python.

```

EXPLORADOR ... constantes.py ...
PRUEBA > __pycache__ < constantes.py > ...
area_circulo.py < constantes.py > ...
constantes.py < constantes.py > ...
imprimir_cte.py < constantes.py > ...

# Este es un modulo donde guardamos todas
# las constantes que podamos necesitar
''' Constantes '''
pi = 3.14
tau = 6.28
e_euler = 2.71
G_gauss = 0.83

# En este script calculamos el area de un circulo
# --- importamos el modulo de constantes ---
import constantes
radio = 12.1 # radio del circulo
area = constantes.pi * radio**2 # calculamo el area
# usando la constante
# de nuestro modulo
print("Area circulo 1:", area) # imprimimos por pantalla

# --- importamos el modulo de constantes ---
import constantes as cte # importamos modulo
# constantest como cte
radio = 12.1 # radio del circulo
area = cte.pi * radio**2 # calculamo el area
# usando la constante
# de nuestro modulo
print("Area circulo 2:", area) # imprimimos por pantalla

```



QUÉ ES UN MÓDULO

Módulo: Archivo con extensión .py que contiene código de Python que puede importarse dentro de otro archivo de Python.

```

EXPLORADOR ... constantes.py ...
PRUEBA > __pycache__ < constantes.py > ...
area_circulo.py < constantes.py > ...
constantes.py < constantes.py > ...
imprimir_cte.py < constantes.py > ...

# Este es un modulo donde guardamos todas
# las constantes que podamos necesitar
''' Constantes '''
pi = 3.14
tau = 6.28
e_euler = 2.71
G_gauss = 0.83

# En este script calculamos el area de un circulo
# --- importamos el modulo de constantes ---
import constantes
radio = 12.1 # radio del circulo
area = constantes.pi * radio**2 # calculamo el area
# usando la constante
# de nuestro modulo
print("Area circulo 1:", area) # imprimimos por pantalla

# --- importamos el modulo de constantes ---
import constantes as cte # importamos modulo
# constantest como cte
radio = 12.1 # radio del circulo
area = cte.pi * radio**2 # calculamo el area
# usando la constante
# de nuestro modulo
print("Area circulo 2:", area) # imprimimos por pantalla

```

(cblocks) MacBook-Pro-4:Prueba Elena\$ /Users/Elena/Clase4/Prueba/area_circulo.py"
Area circulo 1: 459.7274
Area circulo 2: 459.7274
(cblocks) MacBook-Pro-4:Prueba Elena\$



PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

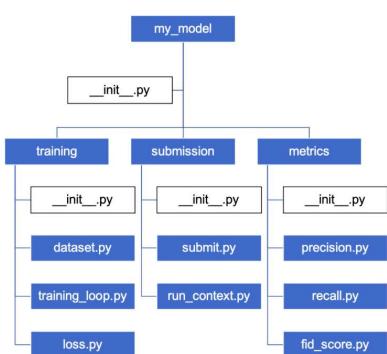
Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos



PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos

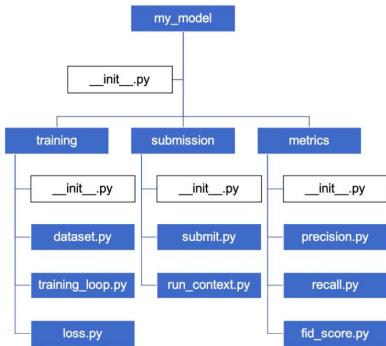




PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos



Library/Librería: Es un término general para referirse a una pieza de código reutilizable.

Normalmente una librería a contiene una colección de módulos y paquetes.

Ejemplos:

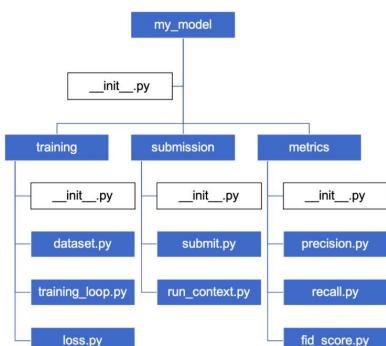
Numpy
Matplotlib
Pytorch
Pandas
...



PAQUETES Y LIBRERIAS

Package/Paquete: Un directorio o carpeta con una colección de módulos

Podemos tener paquetes y subpaquetes con distintos módulos en cada uno de ellos



Library/Librería: Es un término general para referirse a una pieza de código reutilizable.

Normalmente una librería a contiene una colección de módulos y paquetes.

Ejemplos:

Numpy
Matplotlib
Pytorch
Pandas
...

Una librería es una colección de paquetes
Un paquete es una colección de módulos



ARRAYS

¿Qué es un Array?

Son contenedores que son capaces de guardar más de un objeto al mismo tiempo.
Colección ordenada de elementos/objetos.



ARRAYS

¿Qué es un Array?

Son contenedores que son capaces de guardar más de un objeto al mismo tiempo.
Colección ordenada de elementos/objetos.

Lista	Array
Es una estructura in-built	Es una estructura que es necesario importar usando <code>array</code> o <code>numpy</code>
Se crea usando corchetes []	Se crea usando la función <code>array()</code>
Puede contener elementos de distintos tipos	No puede contener elementos de distinto tipo
El anidamiento de listas o estructuras de distinta dimensión es posible	Debe contener elementos del mismo tamaño
No se pueden aplicar operaciones aritméticas	Se pueden aplicar operaciones aritméticas directamente
Consumo más memoria	Consumo comparativamente menos memoria
Mayor flexibilidad a la hora de modificar datos	Menor flexibilidad a la hora de modificar datos



ARRAYS

¿Qué es un Array?

Son contenedores que son capaces de guardar más de un objeto al mismo tiempo.
Colección ordenada de elementos/objetos.

Lista	Array
Es una estructura in-built	Es una estructura que es necesario importar usando <i>array</i> o <i>numpy</i>
Se crea usando corchetes []	Se crea usando la función <i>array()</i>
Puede contener elementos de distintos tipos	No puede contener elementos de distinto tipo
El anidamiento de estructuras de distinta dimensión es posible	Debe contener elementos del mismo tamaño
No se pueden aplicar operaciones aritméticas	Se pueden aplicar operaciones aritméticas directamente
Consumo más memoria	Consumo comparativamente menos memoria
Mayor flexibilidad a la hora de modificar datos	Menor flexibilidad a la hora de modificar datos



ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

Array

Se pueden aplicar operaciones aritméticas directamente

```

my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
✓ 0.0s
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```



ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
✓ 0.0s
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = []

for i in range(0,len(my_list)):
    nueva_lista.append(my_list[i] * 2)

print(nueva_lista)
✓ 0.0s
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Array

Se pueden aplicar operaciones aritméticas directamente



ARRAYS

Lista

No se pueden aplicar operaciones aritméticas

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = my_list * 2
print(nueva_lista)
✓ 0.0s
[1, 2, 3, 4, 5, 6, 7, 8, 9, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
my_list = [1,2,3,4,5,6,7,8,9]
nueva_lista = []

for i in range(0,len(my_list)):
    nueva_lista.append(my_list[i] * 2)

print(nueva_lista)
✓ 0.0s
[2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Array

Se pueden aplicar operaciones aritméticas directamente

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
nuevo_array = my_array * 2
print(nuevo_array)
✓ 0.0s
[ 2  4  6  8 10 12 14 16 18]
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
nuevo_array = my_array / 2
print(nuevo_array)
✓ 0.2s
[0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5]
```



ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la librería numpy



ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la librería numpy

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```



ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la librería numpy

```
my_array = array([1,2,3,4,5,6,7,8,9])
⊗ 0.2s
NameError: name 'array' is not defined
Traceback (most recent call last)
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])
NameError: name 'array' is not defined
+ Código
```



ARRAYS

Lista

Es una estructura in-built

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Necesitamos usar el modulo array o la librería numpy

```
my_array = array([1,2,3,4,5,6,7,8,9])
⊗ 0.2s
NameError: name 'array' is not defined
Traceback (most recent call last)
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])
NameError: name 'array' is not defined
+ Código
```

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```



ARRAYS

Lista

Se crea usando corchetes

```
lista = [1,2,3,4,5,6,7]
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

```
lista = list([1,2,3,4,5,6,7])
print(type(lista))
print(lista)
✓ 0.0s
<class 'list'>
[1, 2, 3, 4, 5, 6, 7]
```

Array

Se crea usando la función `array()` (sea del modulo array o de la librería numpy)

```
my_array = array([1,2,3,4,5,6,7,8,9])
⊗ 0.2s
NameError
Cell In[1], line 1
----> 1 my_array = array([1,2,3,4,5,6,7,8,9])
NameError: name 'array' is not defined
+ Código
```

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```



ARRAYS

Lista

Puede contener elementos de distinto tipo

Array

No puede contener elementos de distinto tipo

```
lista = ['string', 3, 7.8, True]
print(type(lista))
print(type(lista[0]), type(lista[1]))
✓ 0.0s
<class 'list'>
<class 'str'> <class 'int'>
```



ARRAYS

Lista

Puede contener elementos de distinto tipo

```

lista = ['string', 3, 7.8, True]
print(type(lista))
print(type(lista[0]), type(lista[1]))
✓ 0.0s

<class 'list'>
<class 'str'> <class 'int'>

```

Array

No puede contener elementos de distinto tipo

```

import array
my_array = array.array('i', ['string', 3, 7.8, True])
print(type(my_array[0]))

TypeError: Traceback (most recent call last)
Cell In[28], line 2
      1 import array
      2 my_array = array.array('i', ['string', 3, 7.8, True])
      3 print(type(my_array[0]))
      4
      5 TypeError: an integer is required (got type str)

import numpy
my_array = numpy.array(['string', 3, 7.8, True])
print(my_array)
print(type(my_array[2]))
✓ 0.0s

['string' '3' '7.8' 'True']
<class 'numpy.str_'>

```



ARRAYS

Lista

Anidamiento de listas

```

lista_de_listas = [ ["manzana", "pera", "cereza"], ["string"], ["bmw", "mercedes"] ]
print(type(lista_de_listas))
✓ 0.0s

<class 'list'>

lista_de_listas = [ ["manzana", "pera", "cereza"],
                    | ["string"],
                    | ["bmw", "mercedes"] ]

print(type(lista_de_listas))
✓ 0.0s

<class 'list'>

```

Array

Debe contener elementos del mismo tamaño

```

import numpy
my_array = numpy.array(["manzana", "pera", "cereza"],
                      ["string"],
                      ["bmw", "mercedes"])

print(my_array)
print(type(my_array))

✓ 0.0s

TypeError: Traceback (most recent call last)
Cell In[39], line 2
      1 import numpy
      2 my_array = numpy.array(["manzana", "pera", "cereza"],
      3                         ["string"],
      4                         ["bmw", "mercedes"])
      5 print(my_array)
      6 print(type(my_array))

TypeError: array() takes from 1 to 2 positional arguments but 3 were given

```



¿CUANDO USAMOS UN ARRAY Y CUANDO UNA LISTA?

Lista

- ✓ Nos permite ordenar elementos
- ✓ Es una estructura fácilmente mutable y flexible
- ✓ No necesitamos importar ningún módulo
- Necesita más memoria
- No se pueden realizar operaciones aritméticas
- ➡ Guardar una secuencia pequeña de elementos
- ➡ No queremos realizar operaciones matemáticas

Array

- ✓ Necesita menos memoria
- ✓ Se pueden realizar operaciones aritméticas fácilmente
- Los elementos de un array deben ser del mismo tipo
- Es una estructura algo menos flexible
- ➡ Guardar una secuencia grandes de elementos
- ➡ Queremos realizar operaciones matemáticas con los datos guardados



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```

import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))

✓ 0.0s
<class 'array.array'>

```

importar modulo array llamándolo arr



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

importar modulo `array` llamándolo `arr`
del módulo `array` (al que llamamos `arr` a partir de
ahora) usamos la función `array()`



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

importar modulo `array` llamándolo `arr`
del módulo `array` (al que llamamos `arr` a partir de
ahora) usamos la función `array()`

indicamos que tipo de objeto va a contener el
array. Es este caso enteros simples.



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

importar modulo array llamándolo arr
del módulo array (al que llamamos arr a partir de ahora) usamos la función array()
indicamos que tipo de objeto va a contener el array. Es este caso enteros simples.

Contenido del array



MODULO ARRAY

Viene por defecto al instalar Python.

Es un modulo básico para tratar con Arrays.

Sintaxis:

```
import array as arr
my_array = arr.array('i', [1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.0s
<class 'array.array'>
```

Este modulo define un tipo de objeto que representa un arreglo de valores básicos: caracteres, números enteros y de punto flotante. Los arreglos son tipos de secuencias que se comportan de forma similar a las listas, a excepción que el tipo de objeto guardado es definido. El tipo es especificado al momento de crear el objeto mediante *type code*, que es un carácter simple. Se definen los siguientes tipos:

Código de tipo	Tipo C	Tipo Python	Tamaño mínimo en bytes	Notas
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	wchar_t	Carácter unicode	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

<https://docs.python.org/es/3/library/array.html>

signed (con signo) = tipo entero de 32 bits que contiene un numero en el rango [-2147483648, 2147483648].

unsigned (sin signo) = tipo entero de 32 bits que contiene un numero en el rango [0, 4294967295].



LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*



LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*

del módulo *numpy* (al que llamamos *np* a partir de ahora) usamos la función *array()*



LIBRERIA NUMPY

No viene pre-instalada

Es una librería mucho más potente para el tratamiento de arrays.

Sintaxis:

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```

importar modulo *numpy* llamándolo *np*
del módulo *numpy* (al que llamamos *np* a partir de ahora) usamos la función *array()*

no necesitamos explicitar el tipo de dato como en el caso del modulo in-built array



INSTALAR NUMPY

Instalación...

Activamos nuestro environment de trabajo:

```
(base) MacBook-Pro-4:Prueba Elena$ conda activate cblocks
(cblocks) MacBook-Pro-4:Prueba Elena$ █
```

Dentro del environment de trabajo instalamos numpy:

Podemos usar conda...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ conda install numpy
```

O también pip...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ pip install numpy
```



REPASO

- 1) Qué es un módulo
- 2) Qué es un package/paquete
- 3) Qué es una library/librería
- 4) Qué es un Array y las diferencias con una Lista
- 5) Cómo importar los módulos y librerías relacionadas con Arrays y su sintaxis

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

TRABAJANDO CON ARRAYS (PARTE 1)

CONQUERBLOCKS



CLASE ANTERIOR

- 1) Qué es un módulo
- 2) Qué es un package/paquete
- 3) Qué es una library/librería
- 4) Qué es un Array y las diferencias con una Lista
- 5) Cómo importar los módulos y librerías relacionadas con Arrays y su sintaxis



INSTALAR NUMPY

Instalación...

Activamos nuestro environment de trabajo:

```
(base) MacBook-Pro-4:Prueba Elena$ conda activate cblocks
(cblocks) MacBook-Pro-4:Prueba Elena$
```

Dentro del environment de trabajo instalamos numpy:

Podemos usar conda...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ conda install numpy
```

O también pip...

```
(cblocks) MacBook-Pro-4:Prueba Elena$ pip install numpy
```

```
import numpy as np
my_array = np.array([1,2,3,4,5,6,7,8,9])
print(type(my_array))
✓ 0.3s
<class 'numpy.ndarray'>
```



DE LISTA A ARRAY

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(lista_a_array)
✓ 0.1s
```

[1 2 3]

Parecen lo mismo...
Tienen prácticamente el mismo propósito...

¿Por qué convertir una lista en un array?

```
import numpy as np
# de lista a array
my_list = [1,2,3]
lista_a_array = np.array(my_list)
print(lista_a_array)
✓ 0.0s
```

[1 2 3]

Eficiencia en memoria



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

String



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

String

True / False

1 / 0



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

String

True / False

1 / 0

2 opciones



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

String

True / False

26 letras minúsculas

26 letras mayúsculas

1 / 0

10 dígitos

10+ signos de puntuación

2 opciones



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

True / False

1 / 0

2 opciones

String

26 letras minúsculas

26 letras mayúsculas

10 dígitos

10+ signos de puntuación

70+ opciones

¡para un carácter!



DE LISTA A ARRAY

En una lista podemos guardar todo tipo de datos...

Boolean

True / False

26 letras minúsculas

26 letras mayúsculas

Esta flexibilidad de las listas es al mismo tiempo una locura en términos de manejo de memoria

String

10+ signos de puntuación

2 opciones

70+ opciones

¡para un carácter!



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

Entero de 64 bits



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

¿Necesitamos 64 bits para representar los números 1, 2 y 3?

Entero de 64 bits



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

¿Necesitamos 64 bits para representar los números 1, 2 y 3?

decimal

binario

Entero de 64 bits



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

¿Necesitamos 64 bits para representar los números 1, 2 y 3?

decimal

1

binario

0	1
---	---

Entero de 64 bits



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

¿Necesitamos 64 bits para representar los números 1, 2 y 3?

decimal

1

binario

0	1
---	---

2

1	0
---	---

Entero de 64 bits



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

Entero de 64 bits

¿Necesitamos 64 bits para representar los números 1, 2 y 3?

decimal

1
2
3

binario

0	1
1	0
1	1

En este caso necesitamos un máximo de **2 bits**



DE LISTA A ARRAY

Los Arrays están pensados para guardar un único tipo de dato

Y no solo en términos de si se trata de enteros o decimales, si no en términos del *número de bits*

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3])
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int64'>
```

Entero de 64 bits

```
import numpy as np
# de lista a array
lista_a_array = np.array([1,2,3], dtype = np.int8)
print(type(lista_a_array[0]))
✓ 0.0s
<class 'numpy.int8'>
```

Entero de 8 bits → Consumimos menos memoria



ARRAYS MULTIDIMENSIONALES

Hasta ahora hemos tratado con arrays unidimensionales → Todos los datos están en una linea

¿Y si queremos lineas y columnas?

ARRAY BIDIMENSIONAL

```
# Array bidimensional 2D
lista_a_array = np.array([[1,2,3],[4,5,6]], dtype = np.int8)
print(lista_a_array)

✓ 0.0s
[[1 2 3] | 2 columnas
 [4 5 6]] _____
3 filas
```



ARRAYS MULTIDIMENSIONALES

Hasta ahora hemos tratado con arrays unidimensionales → Todos los datos están en una linea

¿Y si queremos lineas y columnas?

ARRAY BIDIMENSIONAL

```
# Array bidimensional 2D
lista_a_array = np.array([[1,2,3],[4,5,6]], dtype = np.int8)
print(lista_a_array.shape)

✓ 0.0s
(2, 3)
| / \
| 2 filas 3 columnas
```



ARRAYS MULTIDIMENSIONALES

Hasta ahora hemos tratado con arrays unidimensionales → Todos los datos están en una linea

¿Y si queremos lineas y columnas?

ARRAY TRIDIMENSIONAL

```
# Array tridimensional 3D
lista_a_array = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]]], dtype = np.int8)
print(lista_a_array)
print(lista_a_array.shape)

✓ 0.0s
[[[ 1  2  3]
 [ 4  5  6]

 [[ 7  8  9]
 [10 11 12]]]
(2, 2, 3)
```



ARRAYS MULTIDIMENSIONALES

Hasta ahora hemos tratado con arrays unidimensionales → Todos los datos están en una linea

¿Y si queremos lineas y columnas?

ARRAY TRIDIMENSIONAL

```
# Array tridimensional 3D
lista_a_array = np.array([[1,2,3],[4,5,6],[7,8,9],[10,11,12]]], dtype = np.int8)
print(lista_a_array)
print(lista_a_array.ndim)

✓ 0.0s
[[[ 1  2  3]
 [ 4  5  6]

 [[ 7  8  9]
 [10 11 12]]]
3
```



ARRAYS MULTIDIMENSIONALES

Podemos redimensionar el array con reshape():

```
# Array bidimensional shape 2,3
array_1 = np.array([[1,2,3],[4,5,6]], dtype = np.int8)
print("array_1 shape:", array_1.shape)
print(array_1)
array_2 = array_1.reshape((3,2))
print("array_2 shape:", array_2.shape)
print(array_2)

✓ 0.0s
array_1 shape: (2, 3)
[[1 2 3]
 [4 5 6]]
array_2 shape: (3, 2)
[[1 2]
 [3 4]
 [5 6]]
```



ARRAYS MULTIDIMENSIONALES

Podemos redimensionar el array con reshape():

```
# Array bidimensional shape 2,3
array_1 = np.array([[1,2,3],[4,5,6]], dtype = np.int8)
print("array_1 shape:", array_1.shape)
print(array_1)
array_2 = array_1.reshape(6,1)
print("array_2 shape:", array_2.shape)
print(array_2)

✓ 0.0s
array_1 shape: (2, 3)
[[1 2 3]
 [4 5 6]]
array_2 shape: (6, 1)
[[1]
 [2]
 [3]
 [4]
 [5]
 [6]]
```



ARRAYS MULTIDIMENSIONALES

Podemos redimensionar el array con reshape():

```
# Array bidimensional shape 2,3
array_1 = np.array([[1,2,3],[4,5,6]], dtype = np.int8)
print("array_1 shape:", array_1.shape, "dim", array_1.ndim)
print(array_1)
array_2 = array_1.reshape(6)
print("array_2 shape:", array_2.shape, "dim", array_2.ndim)
print(array_2)

✓ 0.0s

array_1 shape: (2, 3) dim 2
[[1 2 3]
 [4 5 6]]
array_2 shape: (6,) dim 1
[1 2 3 4 5 6]
```

Hemos convertido un array de dimension 2 en un array de dimension 1



CREACIÓN DE ARRAYS SIN USAR LISTAS

Podemos crear arrays con numpy.arange():

```
import numpy as np
my_array = np.arange(8)
print(my_array)
print(type(my_array))

✓ 0.0s

[0 1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```

```
import numpy as np
my_array = np.array([0,1,2,3,4,5,6,7])
print(my_array)
print(type(my_array))

✓ 0.0s

[0 1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```



CREACIÓN DE ARRAYS SIN USAR LISTAS

Podemos crear arrays con numpy.arange():

The code block shows two examples of creating arrays. The first example uses `np.arange(8)`, which is highlighted with a red circle. A red arrow points from the text "Stop Value" to this circled number. The second example uses `np.array([0,1,2,3,4,5,6,7])`. Both examples include print statements for the array and its type.

```
import numpy as np
my_array = np.arange(8)
print(my_array)
print(type(my_array))

✓ 0.0s

[0 1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```

```
import numpy as np
my_array = np.array([0,1,2,3,4,5,6,7])
print(my_array)
print(type(my_array))

✓ 0.0s

[0 1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```



CREACIÓN DE ARRAYS SIN USAR LISTAS

Podemos crear arrays con numpy.arange():

The code block shows an example of creating an array using `np.arange(1,8)`. The start value (1) and stop value (8) are highlighted with red circles. A red arrow points from the text "Start Value" to the circled 1, and another arrow points from "Stop Value" to the circled 8. The output shows the array [1 2 3 4 5 6 7] and its type <class 'numpy.ndarray'>.

```
import numpy as np
my_array = np.arange(1,8)
print(my_array)
print(type(my_array))

✓ 0.0s

[1 2 3 4 5 6 7]
<class 'numpy.ndarray'>
```



CREACIÓN DE ARRAYS SIN USAR LISTAS

Podemos crear arrays con numpy.arange():

```

import numpy as np
my_array = np.arange(1,8)
print(my_array)
print(type(my_array))

✓ 0.0s
[1 2 3 4 5 6 7]
<class 'numpy.ndarray'>

```

Stop Value

Start Value

¿Y si solo quisiese números impares?



CREACIÓN DE ARRAYS SIN USAR LISTAS

Podemos crear arrays con numpy.arange():

```

import numpy as np
my_array = np.arange(1,8)
print(my_array)
print(type(my_array))

✓ 0.0s
[1 2 3 4 5 6 7]
<class 'numpy.ndarray'>

```

Stop Value

Start Value

¿Y si solo quisiese números impares?

```

import numpy as np
my_array = np.arange(1,8,2)
print(my_array)
print(type(my_array))

✓ 0.0s
[1 3 5 7]
<class 'numpy.ndarray'>

```

Step



CREACIÓN DE ARRAYS SIN USAR LISTAS

También podemos trabajar con decimales:

```
import numpy as np
my_array = np.arange(1,8,0.5)
print(my_array)
print(type(my_array))
✓ 0.0s
[1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5 6.  6.5 7.  7.5]
<class 'numpy.ndarray'>
```



CREACIÓN DE ARRAYS SIN USAR LISTAS

Y con números negativos:

```
import numpy as np
my_array = np.arange(-1,8,0.5)
print(my_array)
print(type(my_array))
✓ 0.0s
[-1. -0.5 0.  0.5 1.  1.5 2.  2.5 3.  3.5 4.  4.5 5.  5.5
 6.  6.5 7.  7.5]
<class 'numpy.ndarray'>
```



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.zeros((2,3))
print(array_vacio)

✓ 0.0s
[[0. 0. 0.]
 [0. 0. 0.]]
```

```
# array vacio
array_vacio = np.ones((2,3))
print(array_vacio)

✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]
```



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.empty((2,3))
print(array_vacio)

✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]
```



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.empty((2,3))
print(array_vacio)
```

✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]

```
# array vacio
array_vacio = np.empty((4,3))
print(array_vacio)
```

✓ 0.0s
[[0.0000000e+000 2.16385932e-314 5.92878775e-323]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]]



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.empty((2,3))
print(array_vacio)
```

✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]

```
# array vacio
array_vacio = np.empty((4,3))
print(array_vacio)
```

✓ 0.0s
[[0.0000000e+000 2.16385932e-314 5.92878775e-323]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]
 [0.0000000e+000 0.0000000e+000 0.0000000e+000]]

```
# array vacio
array_vacio = np.empty((1,3))
print(array_vacio)
```

✓ 0.0s
[[4.9e-324 9.9e-324 1.5e-323]]



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.empty((2,3))
print(array_vacio)
✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]

# array vacio
array_vacio = np.empty((4,3))
print(array_vacio)
✓ 0.0s
[[0.0000000e+000 2.16385932e-314 5.92878775e-323]
 [0.0000000e+000 0.000000e+000 0.000000e+000]
 [0.0000000e+000 0.000000e+000 0.000000e+000]
 [0.0000000e+000 0.000000e+000 0.000000e+000]]

# array vacio
array_vacio = np.empty((3,3))
print(array_vacio)
✓ 0.0s
[[ 6.17779239e-31 -1.23555848e-30  3.08889620e-31]
 [-1.23555848e-30  2.68733969e-30 -8.34001973e-31]
 [ 3.08889620e-31 -8.34001973e-31  4.78778910e-31]]

# array vacio
array_vacio = np.empty((1,3))
print(array_vacio)
✓ 0.0s
[[4.9e-324 9.9e-324 1.5e-323]]
```



CREACIÓN DE ARRAYS “VACÍOS”

A veces no sabemos de antemano que valores va a contener el array.
En estos casos puede interesarnos crear un array “vacío”

```
# array vacio
array_vacio = np.empty((2,3))
print(array_vacio)
✓ 0.0s
[[1. 1. 1.]
 [1. 1. 1.]]

# array vacio
array_vacio = np.empty((4,3))
print(array_vacio)
✓ 0.0s
[[0.0000000e+000 2.16385932e-314 5.92878775e-323]
 [0.0000000e+000 0.000000e+000 0.000000e+000]
 [0.0000000e+000 0.000000e+000 0.000000e+000]
 [0.0000000e+000 0.000000e+000 0.000000e+000]]

# array vacio
array_vacio = np.empty((3,3))
print(array_vacio)
✓ 0.0s
[[ 6.17779239e-31 -1.23555848e-30  3.08889620e-31]
 [-1.23555848e-30  2.68733969e-30 -8.34001973e-31]
 [ 3.08889620e-31 -8.34001973e-31  4.78778910e-31]]

# array vacio
array_vacio = np.empty((1,3))
print(array_vacio)
✓ 0.0s
[[4.9e-324 9.9e-324 1.5e-323]]
```

Cuidado al crear un array con np.empty()



CREACIÓN DE ARRAYS “UNIDAD”

Podemos usar numpy.eye() para crear arrays de ceros y unos:

```
eye_array = np.eye(3)
print(eye_array)
✓ 0.0s
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

```
eye_array = np.eye(3, k=-1)
print(eye_array)
✓ 0.0s
[[0. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
```

```
eye_array = np.eye(3, k=1)
print(eye_array)
✓ 0.0s
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```



MANIPULACION DE ARRAYS

Una vez creado el array podemos reasignar sus valores:

(Podemos hacerlo con todos los arrays, los creamos con np.array(), np.arange() o np.eye())

```
eye_array = np.eye(3, k=1)
print(eye_array)
✓ 0.0s
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
print(eye_array)
✓ 0.0s
[[2. 1. 2.]
 [2. 2. 1.]
 [2. 2. 2.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
print(eye_array)
✓ 0.0s
[[2. 9. 2.]
 [2. 2. 9.]
 [2. 2. 2.]]
```



MANIPULACION DE ARRAYS

Una vez creado el array podemos reasignar sus valores:

(Podemos hacerlo con todos los arrays, los creamos con np.array(), np.arange() o np.eye())

```
eye_array = np.eye(3, k=1)
eye_array[0] = 2
print(eye_array)

✓ 0.0s

[[2. 2. 2.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

Sustituye la fila 0

```
eye_array = np.eye(3, k=1)
eye_array[:2] = 2
print(eye_array)

✓ 0.0s

[[2. 2. 2.]
 [2. 2. 2.]
 [0. 0. 0.]]
```

Sustituye todas las filas hasta la 2

```
eye_array = np.eye(3, k=1)
eye_array[1:] = 2
print(eye_array)

✓ 0.0s

[[0. 1. 0.]
 [2. 2. 2.]
 [2. 2. 2.]]
```

Sustituye desde la fila 1 hasta la última



MANIPULACION DE ARRAYS

Una vez creado el array podemos reasignar sus valores:

(Podemos hacerlo con todos los arrays, los creamos con np.array(), np.arange() o np.eye())

```
eye_array = np.eye(3, k=1)
eye_array[1:, 2:] = 2
print(eye_array)

✓ 0.0s

[[0. 1. 0.]
 [0. 0. 2.]
 [0. 0. 2.]]
```

Sustituye desde la **fila 1** hasta la última y desde la **columna 2** hasta la última



ORDENAR EL CONTENIDO DE LOS ARRAYS

Podemos ordenar el contenido usando numpy.sort():

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
print(eye_array)

✓ 0.0s

[[2. 9. 2.]
 [4. 4. 9.]
 [4. 4. 2.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
sorted_array = np.sort(eye_array)
print(sorted_array)

✓ 0.0s

[[2. 2. 9.]
 [4. 4. 9.]
 [2. 4. 4.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
sorted_array = np.sort(eye_array, axis=0)
print(sorted_array)

✓ 0.0s

[[2. 4. 2.]
 [4. 4. 2.]
 [4. 9. 9.]]
```



ORDENAR EL CONTENIDO DE LOS ARRAYS

Podemos ordenar el contenido usando numpy.sort():

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
print(eye_array)

✓ 0.0s

[[2. 9. 2.]
 [4. 4. 9.]
 [4. 4. 2.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
sorted_array = np.sort(eye_array)
print(sorted_array)

✓ 0.0s

[[2. 2. 9.]
 [4. 4. 9.]
 [2. 4. 4.]]
```

```
eye_array = np.eye(3, k=1)
eye_array[eye_array == 0] = 2
eye_array[eye_array < 2] = 9
eye_array[1:, :2] = 4
sorted_array = np.sort(eye_array, axis = -1)
print(sorted_array)

✓ 0.0s

[[2. 2. 9.]
 [4. 4. 9.]
 [2. 4. 4.]]
```



ODERNAR EL CONTENIDO DE LOS ARRAYS

También podemos indicar que algoritmo queremos usar para ordenar el array:

```
sorted_array = np.sort(eye_array, axis = 0, kind = 'quicksort')
print(sorted_array)
Por defecto
✓ 0.0s
[[2. 4. 2.]
 [4. 4. 2.]
 [4. 9. 9.]]
```

Distintos tipos de algoritmos de ordenamiento pueden ser mas o menos rápidos dependiendo del array y los datos con los que estemos tratando

(Al nivel al que estamos trabajando ahora eso aún no lo vamos a notar)



ODERNAR EL CONTENIDO DE LOS ARRAYS

También podemos indicar que algoritmo queremos usar para ordenar el array:

```
sorted_array = np.sort(eye_array, axis = 0, kind = 'quicksort')
print(sorted_array)
Por defecto
✓ 0.0s
[[2. 4. 2.]
 sorted_array = np.sort(eye_array, axis = 0, kind = 'heapsort')
print(sorted_array)
✓ 0.0s
[[2. 4. 2.]
 [4. 4. 2.]
 [4. 9. 9.]]
```

Distintos tipos de algoritmos de ordenamiento pueden ser mas o menos rápidos dependiendo del array y los datos con los que estemos tratando

(Al nivel al que estamos trabajando ahora eso aún no lo vamos a notar)



ORDENAR EL CONTENIDO DE LOS ARRAYS

También podemos indicar que algoritmo queremos usar para ordenar el array:

```

sorted_array = np.sort(eye_array, axis = 0, kind = 'quicksort')
print(sorted_array)
    Por defecto
✓ 0.0s
[[2. 4. 2.]
 sorted_array = np.sort(eye_array, axis = 0, kind = 'heapsort')
print(sorted_array)
✓ 0.0s
[[2. 4. 2.]
 sorted_array = np.sort(eye_array, axis = 0, kind = 'mergesort')
print(sorted_array)
✓ 0.0s
[[2. 4. 2.]
[4. 4. 2.]
[4. 9. 9.]]

```

Distintos tipos de algoritmos de ordenamiento pueden ser mas o menos rápidos dependiendo del array y los datos con los que estemos tratando

(Al nivel al que estamos trabajando ahora eso aún no lo vamos a notar)



COPiar ARRAYS

Hay dos opciones, `view()` y `copy()`:

`view():`

```

array_view = sorted_array.view()
array_view[:] = 5
print(array_view)
print(sorted_array)
✓ 0.0s
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]
 [[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]]

```

`copy():`

```

array_copy = sorted_array.copy()
array_copy[:] = 5
print(array_copy)
print(sorted_array)
✓ 0.0s
[[5. 5. 5.]
 [5. 5. 5.]
 [5. 5. 5.]]
 [[2. 4. 2.]
 [4. 4. 2.]
 [4. 9. 9.]]]

```



COPiar ARRAYS

Hay dos opciones, `view()` y `copy()`:

`view():`

```
array_view = sorted_array.view()  
array_view[:] = 5  
print(array_view)  
print(sorted_array)
```

`view()` afecta también al array original

```
[5. 5. 5.]  
[5. 5. 5.]  
[5. 5. 5.]  
[[5. 5. 5.]  
 [5. 5. 5.]  
 [5. 5. 5.]]
```

`copy():`

```
array_copy = sorted_array.copy()  
array_copy[:] = 5  
print(array_copy)  
print(sorted_array)
```

`copy()` crea un array independiente

```
[[5. 5. 5.]  
 [5. 5. 5.]  
 [5. 5. 5.]]  
[[2. 4. 2.]  
 [4. 4. 2.]  
 [4. 9. 9.]]
```



REPASO

- 1) Convertir Listas en Arrays
- 2) Multidimensionalidad en los Arrays
- 3) Crear Arrays sin usar Listas
- 4) Crear Arrays unidad
- 5) Reasignar el contenido de los Arrays
- 5) Ordenar el contenido de los Arrays

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

TRABAJANDO CON ARRAYS (PARTE 2)

CONQUERBLOCKS



CLASE ANTERIOR

- 1) Convertir Listas en Arrays
- 2) Multidimensionalidad en los Arrays
- 3) Crear Arrays sin usar Listas
- 4) Crear Arrays unidad
- 5) Reasignar el contenido de los Arrays
- 5) Ordenar el contenido de los Arrays



CREACION DE UN ARRAY

```
import numpy as np  
  
a = np.zeros((3,3), dtype = np.int64)  
a[:] = 2  
print(a)  
✓ 0.0s  
[[2 2 2]  
 [2 2 2]  
 [2 2 2]]
```

```
import numpy as np  
  
b = np.arange(1,10)  
print(b)  
b = b.reshape((3,3))  
print(b)  
✓ 0.0s  
[1 2 3 4 5 6 7 8 9]  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]
```



CREACION DE UN ARRAY

```
import numpy as np  
  
a = np.zeros((3,3), dtype = np.int64)  
a[:] = 2  
print(a)  
✓ 0.0s  
[[2 2 2]  
 [2 2 2]  
 [2 2 2]]
```

```
✓ import numpy as np  
  
b = np.arange(1,10).reshape((3,3))  
print(b)  
✓ 0.0s  
[[1 2 3]  
 [4 5 6]  
 [7 8 9]]
```



LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)

✓ 0.0s
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a += 6
print(a)

✓ 0.0s
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```



LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)

✓ 0.0s
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a -= 6
print(a)

✓ 0.0s
[[-4 -4 -4]
 [-4 -4 -4]
 [-4 -4 -4]]
```



LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)

✓ 0.0s
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a *= 6
print(a)

✓ 0.0s
[[12 12 12]
 [12 12 12]
 [12 12 12]]
```



LLENAR ARRAYS DE VALORES

fill()

operador de asignación

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)

✓ 0.0s
[[8 8 8]
 [8 8 8]
 [8 8 8]]
```

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a /= 6
print(a)

⊗ 0.2s
UFuncTypeError                                     Traceback (most recent call last)
Cell In[10], line 4
      2 a = np.zeros((3,3), dtype = np.int64)
      3 a[:] = 2
      4 a /= 6
      5 print(a)

UFuncTypeError: Cannot cast ufunc 'divide' output from dtype('float64') to dtype('int64') with casting rule 'same_kind'
```



LLENAR ARRAYS DE VALORES

fill()

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
a.fill(8)
print(a)
```

✓ 0.0s
[[8 8 8]
[8 8 8]
[8 8 8]]

operador de asignación

```
import numpy as np
a = np.zeros((3,3))
a[:] = 2
a /= 6
print(a)
```

✓ 0.0s
[[0.33333333 0.33333333 0.33333333]
[0.33333333 0.33333333 0.33333333]
[0.33333333 0.33333333 0.33333333]]



SUMAR ELEMENTOS DE UN ARRAY

sum()

sum(0)

sum(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum()
print(b)
print(suma_array)
```

✓ 0.0s
[[1 2 3]
[4 5 6]
[7 8 9]]
45

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(axis = 0)
print(b)
print(suma_array)
```

✓ 0.0s
[[1 2 3]
[4 5 6]
[7 8 9]]
[12 15 18]

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(axis = 1)
print(b)
print(suma_array)
```

✓ 0.0s
[[1 2 3]
[4 5 6]
[7 8 9]]
[6 15 24]

suma de todos los elementos del array

suma de las columnas del array

suma de las filas del array



SUMAR ELEMENTOS DE UN ARRAY

sum()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum()
print(b)
print(suma_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
45
```

suma de todos los elementos del array

sum(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(0)
print(b)
print(suma_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[12 15 18]
```

suma de las columnas del array

sum(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
suma_array = b.sum(1)
print(b)
print(suma_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 6 15 24]
```

suma de las filas del array



MULTIPLICAR ELEMENTOS DE UN ARRAY

prod()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod()
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
362880
```

multiplica todos los elementos del array

prod(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(axis = 0)
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 28  80 162]
```

multiplica las columnas del array

prod(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(axis = 1)
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 6 120 504]
```

multiplica las filas del array



MULTIPLICAR ELEMENTOS DE UN ARRAY

prod()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod()
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
362880
```

multiplica todos los elementos del array

prod(0)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(0)
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 28  80 162]
```

multiplica las columnas del array

prod(1)

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
prod_array = b.prod(1)
print(b)
print(prod_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[ 6 120 504]
```

multiplica las filas del array



MÁXIMO MÍNIMO Y VALOR MEDIO DE LOS ELEMENTOS DE UN ARRAY

mean()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.mean()
print(b)
print(mean_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
5.0
```

max()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.max()
print(b)
print(mean_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
9
```

valor medio de todos los elementos del array

máximo de todos los elementos del array

min()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
mean_array = b.min()
print(b)
print(mean_array)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
1
```

mínimo de todos los elementos del array



OBTENER INDICES DE MÁXIMO Y MÍNIMO DE LOS ELEMENTOS DE UN ARRAY

argmin()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
min_array_id = b.argmin()
print(b)
print(min_array_id)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
0
```

argmax()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
max_array_id = b.argmax()
print(b)
print(max_array_id)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
8
```



APLANAR UN ARRAY

Pasar de líneas y columnas a tener todo en una línea

reshape() + size

flatten()

ravel()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.reshape(b.size)
print(b)
print(array_plano)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3 4 5 6 7 8 9]
```

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.flatten()
print(b)
print(array_plano)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3 4 5 6 7 8 9]
```

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_plano = b.ravel()
print(b)
print(array_plano)

✓ 0.0s
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3 4 5 6 7 8 9]
```

Crea una copy() del array

Crea una view() del array



TRANSPOSICIÓN DE UN ARRAY

Intercambiar filas y columnas:

swapaxes()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
array_traspn = np.swapaxes(b, 0, 1)
print(b)
print(array_traspn)

✓ 0.0s

[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```

transpose()

```
import numpy as np

b = np.arange(1,10).reshape((3,3))
print(b)
array_traspn = b.transpose(1, 0)
print(array_traspn)

✓ 0.0s

[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[1 4 7]
 [2 5 8]
 [3 6 9]]
```



OPERACIONES CON MATRICES

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----')
resultado_suma = a + b
print(resultado_suma)

✓ 0.0s

[[2 2 2]
 [2 2 2]
 [2 2 2]]      suma de
[[1 2 3]          matrices
 [4 5 6]
 [7 8 9]]

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

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----')
resultado_resta = a - b
print(resultado_resta)

✓ 0.0s

[[2 2 2]
 [2 2 2]
 [2 2 2]]      resta de
[[1 2 3]          matrices
 [4 5 6]
 [7 8 9]]

-----
```

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----')
resultado = (a + b - 2*a)/4
print(resultado)

✓ 0.0s

[[2 2 2]
 [2 2 2]
 [2 2 2]]      combinación
[[1 2 3]          de
 [4 5 6]
 [7 8 9]]          operaciones

-----
```



MULTIPLICACIÓN MATRICIAL

Importante para IA y ML

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \times \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$



MULTIPLICACIÓN MATRICIAL

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \times \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

```

import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:, :] = 2
b = np.arange(1,10).reshape((3,3))
print(a)
print(b)
print('-----')
# multiplicacion de matrices
matrix_multi = np.matmul(a, b)
print(matrix_multi)
print(a*b)
]
    ✓ 0.0s

```

[[2 2 2]
[2 2 2]
[2 2 2]]

[[1 2 3]
[4 5 6]
[7 8 9]]

[[24 30 36]
[24 30 36]
[24 30 36]]
[[2 4 6]
[8 10 12]
[14 16 18]]



MULTIPLICACIÓN MATRICIAL

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \times \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = np.matmul(a, b)
print(matrix_multi)

✓ 0.0s
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = a.dot(b)
print(matrix_multi)

✓ 0.0s
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```

```
import numpy as np
a = np.zeros((3,3), dtype = np.int64)
a[:] = 2
b = np.arange(1,10).reshape((3,3))

# multiplicacion de matrices
matrix_multi = a @ b
print(matrix_multi)

✓ 0.0s
[[24 30 36]
 [24 30 36]
 [24 30 36]]
```



NUMPY DOCS

<https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html>

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

TUPLAS

CONQUERBLOCKS



REPASO

Estructuras de datos:

1. Listas → in - built

2. Arrays → importar modulo



QUE VEREMOS HOY

Estructuras de datos in - built:

1. Listas

2. Tuplas

3. Sets

4. Diccionarios



TUPLAS

Definición: Las tuplas son listas *inmutables*

- No permiten añadir, eliminar o mover elementos
(append, remove...)
- Permiten extraer porciones pero eso da como resultado una tupla nueva.
- Permiten comprobar si un elemento se encuentra en la tupla.



TUPLAS

- ✓ Más rápidas que las listas
- ✓ Ocupan menos espacio (mayor optimización)
- ✓ Pueden usarse como llaves de un diccionario.



Si necesitamos guardar varios elementos pero en el futuro solo queremos recorrerlos para verlos, entonces nos conviene usar tuplas



SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
✓ 0.0s
<class 'tuple'>
```

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
✓ 0.0s
<class 'list'>
```



SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla  
mi_tupla_1 = ("fruta", 45, True)  
print(type(mi_tupla_1))  
✓ 0.0s  
<class 'tuple'>
```

```
# sintaxis de una lista  
mi_lista_1 = ["fruta", 45, True]  
print(type(mi_lista_1))  
✓ 0.0s  
<class 'list'>
```

```
# sintaxis de una tupla  
mi_tupla_2 = "fruta", 45, True  
print(type(mi_tupla_2))  
✓ 0.0s  
<class 'tuple'>
```



SINTAXIS BASICA DE UNA TUPLA

```
# sintaxis de una tupla  
mi_tupla_1 = ("fruta", 45, True)  
print(type(mi_tupla_1))  
✓ 0.0s  
<class 'tuple'>
```

```
# sintaxis de una lista  
mi_lista_1 = ["fruta", 45, True]  
print(type(mi_lista_1))  
✓ 0.0s  
<class 'list'>
```

```
# sintaxis de una tupla  
mi_tupla_2 = "fruta", 45, True  
print(type(mi_tupla_2))  
✓ 0.0s  
<class 'tuple'>
```

```
#acceder a elementos de un tupla  
mi_tupla_1 = ("fruta", 45, True)  
print(mi_tupla_1[1])  
✓ 0.0s  
45
```



INMUTABILIDAD

```
# mutabilidad de una lista
mi_lista = [1, 2, 3]
mi_lista[0] = 4 # reasignacion
print(mi_lista)

✓ 0.0s

[4, 2, 3]
```

```
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla[0] = 4 # intento de reasignacion
⊗ 0.1s

TypeError
Cell In[15], line 3
    1 # inmutabilidad de una tupla
    2 mi_tupla = (1, 2, 3)
    ----> 3 mi_tupla[0] = 4

Traceback (most recent call last)
TypeError: 'tuple' object does not support item assignment
```



INMUTABILIDAD

```
# mutabilidad de una lista
mi_lista = [1, 2, 3]
mi_lista[0] = 4 # reasignacion
print(mi_lista)

✓ 0.0s

[4, 2, 3]
```

```
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla[0] = 4 # intento de reasignacion
⊗ 0.1s

TypeError
Cell In[15], line 3
    1 # inmutabilidad de una tupla
    2 mi_tupla = (1, 2, 3)
    ----> 3 mi_tupla[0] = 4

Traceback (most recent call last)
TypeError: 'tuple' object does not support item assignment
```



INMUTABILIDAD

```
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.append(4)

⊗ 0.0s

-----
AttributeError                               Traceback (most recent call
Cell In[19], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```



INMUTABILIDAD

```
# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.insert(0,4)

⊗ 0.0s

-----
AttributeError                               Traceback (most recent call
Cell In[20], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.insert(0,4)

AttributeError: 'tuple' object has no attribute 'insert'

# inmutabilidad de una tupla
mi_tupla = (1, 2, 3)
mi_tupla.append(4)

⊗ 0.0s

-----
AttributeError                               Traceback (most recent call
Cell In[19], line 3
      1 # inmutabilidad de una tupla
      2 mi_tupla = (1, 2, 3)
----> 3 mi_tupla.append(4)

AttributeError: 'tuple' object has no attribute 'append'
```



OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:**TIEMPO DE CREACIÓN:**

OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:**TIEMPO DE CREACIÓN:**

```
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2,"hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')

✓ 0.0s
120 bytes
80 bytes
```



OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

```
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2,"hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')
✓ 0.0s
120 bytes
80 bytes
```

TIEMPO DE CREACIÓN:

```
# tiempo de creacion list vs tupla
import timeit
print(timeit.timeit(stmt="[0,1,2,3,4,5]", number = 10000000))
print(timeit.timeit(stmt="(0,1,2,3,4,5)", number = 10000000))
✓ 0.3s
0.24537658299993836
0.037102917000083835
```



OPTIMIZACIÓN LISTA VS TUPLA

ESPACIO EN MEMORIA:

```
# espacio en memoria lista vs tupla
import sys
mi_lista = [0,1,2, "hola", True]
mi_tupla = (0,1,2,"hola", True)
print(sys.getsizeof(mi_lista),'bytes')
print(sys.getsizeof(mi_tupla),'bytes')
✓ 0.0s
120 bytes
80 bytes
```

TIEMPO DE CREACIÓN:

X1.5

```
# tiempo de creacion list vs tupla
import timeit
print(timeit.timeit(stmt="[0,1,2,3,4,5]", number = 10000000))
print(timeit.timeit(stmt="(0,1,2,3,4,5)", number = 10000000))
✓ 0.3s
0.24537658299993836
0.037102917000083835
```

X8



LISTA VS ARRAY VS TUPLA

	LISTAS	ARRAYS	TUPLAS
Mutabilidad	Mutable	Mutable	Inmutable
Acceso a elementos	Por índice o slicing	Por índice o slicing	Por índice o slicing
Tamaño de la lista	Dinámico	Fijo	Fijo
Tipo de elementos	Puede contener diferentes tipos	Todos los elementos son del mismo tipo	Puede contener diferentes tipos
Eficiencia	No tan eficiente como los arrays o tuplas	Más eficiente que las listas	Más eficiente que las listas
Uso principal	Cuando se requiere modificar la lista con frecuencia	Cuando se necesita eficiencia en la manipulación de elementos del mismo tipo	Cuando se necesitan elementos inmutables y eficientes



PASAR DE LISTAS A TUPLAS Y VICEVERSA

```
# pasar de lista a tupla
mi_lista = [0,1,2, "hola", True]
print("mi_lista es de tipo...", type(mi_lista))
mi_tupla = tuple(mi_lista)
print("mi_tupla es de tipo...", type(mi_tupla))
print(mi_tupla)

✓ 0.0s

mi_lista es de tipo... <class 'list'>
mi_tupla es de tipo... <class 'tuple'>
(0, 1, 2, 'hola', True)
```

```
# pasar de tupla a lista
mi_tupla = (0,1,2, "hola", True)
print("mi_tupla es de tipo...", type(mi_tupla))
mi_lista = list(mi_tupla)
print("mi_lista es de tipo...", type(mi_lista))
print(mi_lista)

✓ 0.0s

mi_tupla es de tipo... <class 'tuple'>
mi_lista es de tipo... <class 'list'>
[0, 1, 2, 'hola', True]
```



PASAR DE LISTAS A TUPLAS Y VICEVERSA

```
# pasar de lista a tupla
mi_lista = [0,1,2, "hola", True]
print("mi_lista es de tipo...", type(mi_lista))
mi_tupla = tuple(mi_lista)
print("mi_tupla es de tipo...", type(mi_tupla))
print(mi_tupla)

✓ 0.0s
mi_lista es de tipo... <class 'list'>
mi_tupla es de tipo... <class 'tuple'>
(0, 1, 2, 'hola', True)
```

```
# pasar de tupla a lista
mi_tupla = (0,1,2, "hola", True)
print("mi_tupla es de tipo...", type(mi_tupla))
mi_lista = list(mi_tupla)
print("mi_lista es de tipo...", type(mi_lista))
print(mi_lista)

✓ 0.0s
mi_tupla es de tipo... <class 'tuple'>
mi_lista es de tipo... <class 'list'>
[0, 1, 2, 'hola', True]
```



TRABAJANDO CON TUPLAS

ACCEDER A ELEMENTOS:

```
# acceder a elementos de un tupla
mi_tupla_1 = ("fruta", 45, True)
print(mi_tupla_1[1])

✓ 0.0s
45
```

SLICING:

```
# slicing
mi_tupla = (1, 2, 3, 4, 5)
subtupla = mi_tupla[1:3]
print(subtupla) # (2, 3)

✓ 0.0s
(2, 3)
```



TRABAJANDO CON TUPLAS

COMPROBAR LA EXISTENCIA DE ELEMENTOS:

```
# comprobar si un elemento esta en la tupla  
mi_tupla_1 = ("fruta", 45, True)  
print("fruta" in mi_tupla_1)  
✓ 0.0s  
True
```

```
# comprobar si un elemento esta en la tupla  
mi_tupla_1 = ("fruta", 45, True)  
print(100 in mi_tupla_1)  
✓ 0.0s  
False
```



TRABAJANDO CON TUPLAS

NUMERO DE APARICIONES:

NUMERO DE ELEMENTOS:

```
# longitud de la tupla  
mi_tupla = ("fruta", 45, True)  
longitud = len(mi_tupla)  
print(longitud) # 3  
✓ 0.0s  
3
```

```
# numero de apariciones  
mi_tupla = ("fruta", 45, True)  
num_apariciones = mi_tupla.count(45)  
print(num_apariciones)  
✓ 0.0s  
1
```



TRABAJANDO CON TUPLAS

NUMERO DE APARICIONES:

NUMERO DE ELEMENTOS:

```
# longitud de la tupla
mi_tupla = ("fruta", 45, True)
longitud = len(mi_tupla)
print(longitud) # 3
```

✓ 0.0s

3

```
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count(45)
print(num_apariciones)
```

✓ 0.0s

1

```
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count("perro")
print(num_apariciones)
```

✓ 0.0s

0



TRABAJANDO CON TUPLAS

NUMERO DE APARICIONES:

NUMERO DE ELEMENTOS:

```
# longitud de la tupla
mi_tupla = ("fruta", 45, True)
longitud = len(mi_tupla)
print(longitud) # 3
```

✓ 0.0s

3

```
# numero de apariciones
mi_tupla = ("fruta", 45, True)
num_apariciones = mi_tupla.count(45)
```

```
mi_tupla = (1, 2, 3, 3, 4, 5)
num_apariciones = mi_tupla.count(3)
print(num_apariciones) # 3
```

✓ 0.0s

3

```
print(num_apariciones)
```

✓ 0.0s

0



TRABAJANDO CON TUPLAS

ÍNDICE DE UN ELEMENTO:

```
# indice de un elemento
mi_tupla = ("fruta", 45, True)
indice = mi_tupla.index(45)
print(indice) # 2
```

✓ 0.0s

1

MÁXIMOS Y MÍNIMOS:

```
# maximos y minimos
mi_tupla = (3, 1, 4, 1, 5, 9, 2, 6, 5)
maximo = max(mi_tupla)
minimo = min(mi_tupla)
print(maximo, minimo) # 9
```

✓ 0.0s

9 1



TRABAJANDO CON TUPLAS

ORDENAR ELEMENTOS:

RETORNA UNA LISTA!

```
mi_tupla = (3, 1, 4, 1, 5, 9, 2, 6, 5)
mi_lista_ordenada = sorted(mi_tupla)
print(mi_lista_ordenada)
print(type(mi_lista_ordenada))

✓ 0.0s
```

[1, 1, 2, 3, 4, 5, 5, 6, 9]
<class 'list'>

ORDENAR ELEMENTOS DE MANERA INVERSA:

RETORNA UN OBJETO DE TIPO “REVERSED”...

```
mi_tupla = (1, 2, 3, 4, 5)
tupla_inversa = reversed(mi_tupla)
print(tupla_inversa)
print(type(tupla_inversa))

✓ 0.0s
```

<reversed object at 0x10a0537c0>
<class 'reversed'>



TRABAJANDO CON TUPLAS

ORDENAR ELEMENTOS:

RETORNA UNA LISTA!

```
mi_tupla = (3, 1, 4, 1, 5, 9, 2, 6, 5)
mi_tupla_ordenada = tuple(sorted(mi_tupla))
print(mi_tupla_ordenada)
print(type(mi_tupla_ordenada))

✓ 0.0s
(1, 1, 2, 3, 4, 5, 5, 6, 9)
<class 'tuple'>
```

ORDENAR ELEMENTOS DE MANERA INVERSA:

RETORNA UN OBJETO DE TIPO “REVERSED”...

```
mi_tupla = (1, 2, 3, 4, 5)
tupla_inversa = tuple(reversed(mi_tupla))
print(tupla_inversa)
print(type(tupla_inversa))

✓ 0.0s
(5, 4, 3, 2, 1)
<class 'tuple'>
```



TRABAJANDO CON TUPLAS

COMBINAR TUPLAS:

ACCEDER A LOS ELEMENTOS DE UNA TUPLA DE TUPLAS:

```
# combinar tuplas formando una tupla de tuplas
tupla1 = (1, 2, 3)
tupla2 = ('a', 'b', 'c')
tupla_combinada = tuple(zip(tupla1, tupla2))
print(tupla_combinada)

✓ 0.0s
((1, 'a'), (2, 'b'), (3, 'c'))
```

```
# acceder a los elementos de una tupla de tuplas
```

```
mi_tupla = ((1, 'a'), (2, 'b'), (3, 'c'))
```

```
print(mi_tupla[0][0])
```

```
print(mi_tupla[1][1])
```

```
print(mi_tupla[2][0]) |
```

```
✓ 0.0s
```

```
1
```

```
b
```

```
3
```



TRABAJANDO CON TUPLAS

SLICING DE UNA TUPLA DE TUPLAS:

```
mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una tupla interior
tupla_interior = mi_tupla[1]
print(tupla_interior) # (4, 5, 6)

✓ 0.0s
(4, 5, 6)

mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una porción de la tupla de tuplas
porcion_tupla = mi_tupla[0:2]
print(porcion_tupla) # ((1, 2, 3), (4, 5, 6))

✓ 0.0s
((1, 2, 3), (4, 5, 6))
```

```
mi_tupla = ((1, 2, 3), (4, 5, 6), (7, 8, 9))

# Slicing de una porción de una tupla interior
porcion_interior = mi_tupla[2][0:2]
print(porcion_interior) # (7, 8)

✓ 0.0s
(7, 8)
```



TUPLA UNITARIA

```
mi_tupla = (1)
print(type(mi_tupla))

✓ 0.0s

<class 'int'>
1
```

```
mi_tupla = (1,) ↑
print(type(mi_tupla))
print(mi_tupla)

✓ 0.0s

<class 'tuple'>
(1,)
```



EMPAQUETAMIENTO Y DESEMPAQUETAMIENTO

```
# empaquetamiento
mi_tupla = "fruta", 45, True
print(mi_tupla)

✓ 0.0s
('fruta', 45, True)
```

```
# desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero, booleano = mi_tupla
print(string)
print(entero)
print(booleano)

✓ 0.0s

fruta
45
True
```



DESEMPAQUETAMIENTO: POSIBLES ERRORES

```
# errores en el desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero = mi_tupla

✗ 0.0s

ValueError                                     Traceback (most
Cell In[24], line 3
      1 # errores en el desempaquetamiento
      2 mi_tupla = ("fruta", 45, True)
----> 3 string, entero = mi_tupla

ValueError: too many values to unpack (expected 2)
```

```
# errores en el desempaquetamiento
mi_tupla = ("fruta", 45, True)
string, entero, booleano, otra_variable = mi_tupla

✗ 0.0s

ValueError                                     Traceback (most recent ca
Cell In[25], line 3
      1 # errores en el desempaquetamiento
      2 mi_tupla = ("fruta", 45, True)
----> 3 string, entero, booleano, otra_variable = mi_tupla

ValueError: not enough values to unpack (expected 4, got 3)
```



REPASO

- 1) Diferencias entre array, lista y tupla
- 2) Inmutabilidad de una tupla
- 3) Performance tupla vs lista
- 4) Bases del trabajo con tuplas (acceso a elementos, métodos y funciones, empaquetamiento y desempaquetamiento...)
- 5) Trabajo con tuplas de tuplas (acceso a elementos y slicing)

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

SETS

CONQUERBLOCKS



REPASO

Estructuras de datos:

1. Listas → in - built
2. Arrays → importar modulo
3. Tuplas → in - built



QUE VEREMOS HOY

Estructuras de datos in - built:

1. Listas
2. Tuplas
3. Sets
4. Diccionarios



SETS

Definición: Colecciones *no ordenadas* de elementos *únicos* e *inmutables*.

- ➡ Los elementos no llevan un índice asociado
- ➡ No podemos reasignar valores a los elementos del set
- ➡ Podemos añadir y borrar elementos
- ➡ Los elementos son únicos, no hay duplicados



SINTAXIS BASICA DE UN CONJUNTO/SET

Set:

```
# sintaxis de un set  
mi_set= {'fruta', 45, True}  
print(mi_set)  
✓ 0.0s  
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla  
mi_tupla_1 = ("fruta", 45, True)  
print(type(mi_tupla_1))  
✓ 0.0s  
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista  
mi_lista_1 = ["fruta", 45, True]  
print(type(mi_lista_1))  
✓ 0.0s  
<class 'list'>
```

```
# Crear set vacío  
mi_set = set()  
print(type(mi_set))  
✓ 0.0s  
<class 'set'>
```

CUIDADO →

```
# Crear set vacío  
mi_set = {}  
print(type(mi_set))  
✓ 0.0s  
<class 'dict'>
```



AUSENCIA DE ORDENAMIENTO

```
# sintaxis de un set  
set_frutas = {'manzana', 'naranja', 'plátano'}  
print(set_frutas)  
✓ 0.0s  
{'naranja', 'plátano', 'manzana'}
```

El orden del contenido no se preserva.



AUSENCIA DE ORDENAMIENTO

```
# sintaxis de un set
set_frutas = {'manzana', 'naranja', 'plátano'}
print(set_frutas)
✓ 0.0s
{'naranja', 'plátano', 'manzana'}
```

El orden del contenido no se preserva.

```
# no existen los indices
set_frutas = {'manzana', 'naranja', 'plátano'}
print(set_frutas[0])
⊗ 0.2s
```

```
TypeError                                     Traceback (most recent call last)
Cell In[43], line 3
      1 # no existen los indices
      2 set_frutas = {'manzana', 'naranja', 'plátano'}
----> 3 print(set_frutas[0])

TypeError: 'set' object is not subscriptable
```

Los elementos no llevan asignados un índice:



INMUTABILIDAD

```
# no existen los indices
set_frutas = {'manzana', 'naranja', 'plátano'}
set_frutas[0] = "pera"
⊗ 0.0s
```

```
TypeError                                     Traceback (most recent call last)
Cell In[44], line 3
      1 # no existen los indices
      2 set_frutas = {'manzana', 'naranja', 'plátano'}
----> 3 set_frutas[0] = "pera"

TypeError: 'set' object does not support item assignment
```

No podemos reasignar valores



UNICIDAD

```
# los elementos son únicos
set_frutas = {'manzana', 'manzana', 'naranja', 'plátano'}
print(set_frutas)

✓ 0.0s
{'naranja', 'plátano', 'manzana'}
```

En un set todos los elementos son *únicos*.
No hay repeticiones.



COMPROBAR PERTENENCIA

Sets:

```
# comprobar pertenencia
frutas = {'manzana', 'naranja', 'plátano'}
print('manzana' in frutas) # True
print('fresa' in frutas) # False
```

✓ 0.0s
True
False

Listas:

```
# comprobar pertenencia en listas
frutas = ['manzana', 'naranja', 'plátano']
print('manzana' in frutas) # True
print('fresa' in frutas) # False
```

✓ 0.0s
True
False

Las pruebas de pertenencia son mucho **más eficientes** en sets que en listas



COMPROBAR PERTENENCIA

Las pruebas de pertenencia son mucho **más eficientes** en **sets** que en **listas**...

¿Por qué?

Los elementos en una lista tienen asociado un índice



Para comprobar la pertenencia se recorren todos los elementos de la lista hasta encontrar o no el coincidente

Los elementos en un set no tiene un indice si no un *hash*

(Un set es una hash table o tabla de hash)



Python comprueba el bucket correspondiente a ese set y ve si esta lleno o no.

El hash es único para cada elemento, de manera que ese elemento siempre va a estar guardado en el mismo lugar dentro de ese set (en le mismo “bucket”)



PROPIEDADES: LISTA VS ARRAY VS TUPLA VS SETS

Característica	Lista	Array	Tupla	Conjunto (Set)
Definición	Una colección de elementos ordenados y modificables	Un conjunto de elementos homogéneos ordenados y modificables	Una colección de elementos ordenados e inmutables	Una colección de elementos únicos e inmutables
Sintaxis	mi_lista = [1, 2, 3]	mi_array = np.array([1, 2, 3])	mi_tupla = (1, 2, 3)	mi_set = {1, 2, 3}
Índices	Sí	Sí	Sí	No
Modificable	Sí	Sí	No	Sí
Homogeneidad	No	Sí	No	No
Tamaño fijo	No	Sí	Sí	No
Únicos	No	No	No	Sí
Iterables	Sí	Sí	Sí	Sí



AÑADIR Y BORRAR ELEMENTOS

```
# Añadir elementos
frutas = {'manzana', 'naranja', 'plátano'}
frutas.add('fresa')
print(frutas)
✓ 0.0s
{'naranja', 'plátano', 'fresa', 'manzana'}
```

Podemos añadir elementos usando el método `add()`



AÑADIR Y BORRAR ELEMENTOS

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('naranja')
print(frutas)
✓ 0.0s
{'plátano', 'manzana'}
```

Podemos usar los métodos `remove()` o `discard()` para borrar elementos de un set.

```
# usar discard para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.discard('naranja')
print(frutas)
✓ 0.0s
{'plátano', 'manzana'}
```



AÑADIR Y BORRAR ELEMENTOS

La diferencia entre remove() y discard():

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('fresa')
print(frutas)
⊗ 0.0s

KeyError                                     Traceback (most recent call last)
Cell In[52], line 3
      1 # usar remove para eliminar un elemento
      2 frutas = {'manzana', 'naranja', 'plátano'}
----> 3 frutas.remove('fresa')
      4 print(frutas)

KeyError: 'fresa'
```

Si intentamos borrar un elemento que no existe en el set remove() nos devuelve un **error**



AÑADIR Y BORRAR ELEMENTOS

La diferencia entre remove() y discard():

```
# usar remove para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.remove('fresa')
print(frutas)
⊗ 0.0s

KeyError                                     Traceback (most recent call last)
Cell In[52], line 3
      1 # usar remove para eliminar un elemento
      2 frutas = {'manzana', 'naranja', 'plátano'}

# usar discard para eliminar un elemento
frutas = {'manzana', 'naranja', 'plátano'}
frutas.discard('fresa')
print(frutas)
✓ 0.0s

{'naranja', 'plátano', 'manzana'}
```

Si intentamos borrar un elemento que no existe en el set remove() nos devuelve un **error**

Mientras que discard() simplemente no hace **nada**



PASAR DE LISTAS A SETS Y VICEVERSA

```
# pasamos de lista a set
mi_lista = ['manzana', 'naranja', 'plátano']
print(type(mi_lista))
mi_set = set(mi_lista)
print(type(mi_set))
print(mi_set)
✓ 0.0s
<class 'list'>
<class 'set'>
{'naranja', 'plátano', 'manzana'}
```

```
# pasamos de set a lista
mi_set = {'manzana', 'naranja', 'plátano'}
print(type(mi_set))
mi_lista = list(mi_set)
print(type(mi_lista))
print(mi_lista)
✓ 0.0s
<class 'set'>
<class 'list'>
['naranja', 'plátano', 'manzana']
```



PASAR DE LISTAS A SETS Y VICEVERSA

```
# pasamos de lista a set
mi_lista = ['manzana', 'naranja', 'plátano']
print(type(mi_lista))
mi_set = set(mi_lista)
print(type(mi_set))
print(mi_set)
✓ 0.0s
<class 'list'>
<class 'set'>
{'naranja', 'plátano', 'manzana'}
```

```
# pasamos de set a lista
mi_set = {'manzana', 'naranja', 'plátano'}
print(type(mi_set))
mi_lista = list(mi_set)
print(type(mi_lista))
print(mi_lista)
✓ 0.0s
<class 'set'>
<class 'list'>
['naranja', 'plátano', 'manzana']
```



TRABAJANDO CON SETS - EJEMPLO

CREAR UNA LISTA NUEVA ELIMINANDO DUPLICADOS:

```
# Eliminar duplicados en una lista
lista_alumnos = ["Pedro", "Lucas", "Juan", "Lucas"]
set_alumnos = set(lista_alumnos)
lista_alumnos_unico = list(set_alumnos)
print(lista_alumnos_unico)

✓ 0.0s

['Pedro', 'Lucas', 'Juan']
```

```
# Eliminar duplicados en una lista
lista_alumnos = ["Pedro", "Lucas", "Juan", "Lucas"]
set_alumnos = set(lista_alumnos)
print(set_alumnos)

✓ 0.0s

{'Pedro', 'Lucas', 'Juan'}
```



TRABAJANDO CON SETS

UNION DE CONJUNTOS:

```
# Union de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 | set2)
print(set1.union(set2))

✓ 0.0s

{1, 2, 3, 4, 5}
{1, 2, 3, 4, 5}
```



TRABAJANDO CON SETS

INTERSECCION DE CONJUNTOS:

```
# Intersección de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 & set2)
print(set1.intersection(set2))

✓ 0.0s
{3}
{3}
```



TRABAJANDO CON SETS

DIFERENCIA DE CONJUNTOS:

```
# Diferencia de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 - set2)
print(set1.difference(set2))

✓ 0.0s
{1, 2}
{1, 2}
```



TRABAJANDO CON SETS

DIFERENCIA SIMETRICA DE CONJUNTOS:

```
# Diferencia simétrica de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 ^ set2)
print(set1.symmetric_difference(set2))

✓ 0.0s
{1, 2, 4, 5}
{1, 2, 4, 5}
```



TRABAJANDO CON SETS

DIFERENCIA SIMETRICA DE CONJUNTOS:

```
# Diferencia simétrica de conjuntos
set1 = {1, 2, 3}
set2 = {3, 4, 5}
print(set1 ^ set2)
print(set1.symmetric_difference(set2))

✓ 0.0s
{1, 2, 4, 5}
{1, 2, 4, 5}
```



REPASO

- 1) Que es un set y su sintaxis
- 2) Propiedades de un set
- 3) Indices y hash
- 4) Añadir y borrar elementos de un set
- 5) Trabajar con sets y listas
- 6) Operaciones con sets (union, intersección y diferencias)

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

DICCIONARIOS (PARTE 1)

CONQUERBLOCKS



REPASO

Estructuras de datos:

1. Listas → in - built
2. Arrays → importar modulo
3. Tuplas → in - built
4. Sets → in - built



QUE VEREMOS HOY

Estructuras de datos in - built:

1. Listas
2. Tuplas
3. Sets
4. Diccionarios



DICCIONARIOS

Definición: Colecciones *no ordenadas* de pares *clave-valor*

- ➔ Los elementos no llevan un índice asociado si no una clave

Podemos pensar en un diccionario real, donde buscamos una palabra y encontramos su definición



SINTAXIS BASICA DE UN DICCIONARIO

Set:

```
# sintaxis de un set
mi_set= {'fruta', 45, True}
print(mi_set)
✓ 0.0s
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
✓ 0.0s
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
✓ 0.0s
<class 'list'>
```

Diccionario:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3}
<class 'dict'>
```



SINTAXIS BASICA DE UN DICCIONARIO

Set:

```
# sintaxis de un set
mi_set= {'fruta', 45, True}
print(mi_set)
✓ 0.0s
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
✓ 0.0s
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
✓ 0.0s
<class 'list'>
```

Diccionario:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3}
<class 'dict'>
```

Clave



SINTAXIS BASICA DE UN DICCIONARIO

Set:

```
# sintaxis de un set
mi_set= {'fruta', 45, True}
print(mi_set)
✓ 0.0s
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
✓ 0.0s
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
✓ 0.0s
<class 'list'>
```

Diccionario:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3}
<class 'dict'>
```

Clave : Valor



SINTAXIS BASICA DE UN DICCIONARIO

Set:

```
# sintaxis de un set
mi_set= {'fruta', 45, True}
print(mi_set)
✓ 0.0s
{'fruta', 45, True}
```

Tupla:

```
# sintaxis de una tupla
mi_tupla_1 = ("fruta", 45, True)
print(type(mi_tupla_1))
✓ 0.0s
<class 'tuple'>
```

Lista:

```
# sintaxis de una lista
mi_lista_1 = ["fruta", 45, True]
print(type(mi_lista_1))
✓ 0.0s
<class 'list'>
```

Diccionario:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3}
<class 'dict'>
```

Clave : Valor

```
mi_diccionario = {}
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s
{} Diccionario vacío
<class 'dict'>
```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Accedemos a los valores a través de las llaves:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['manzana'])
✓ 0.0s
1
```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Accedemos a los valores a través de las llaves:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['manzana'])
✓ 0.0s
1
```

Podemos modificar los valores:

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario['manzana'] = 4
print(mi_diccionario)
✓ 0.0s
{'manzana': 4, 'plátano': 2, 'naranja': 3}
```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Accedemos a los valores a través de las llaves:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['manzana'])
✓ 0.0s
1

```

Podemos modificar los valores:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario['manzana'] = 4
print(mi_diccionario)
✓ 0.0s
{'manzana': 4, 'plátano': 2, 'naranja': 3}

```

Agregar pares clave-valor:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario['pera'] = 4
print(mi_diccionario)
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3, 'pera': 4}

```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Accedemos a los valores a través de las llaves:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['manzana'])
✓ 0.0s
1

```

Podemos modificar los valores:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario['manzana'] = 4
print(mi_diccionario)
✓ 0.0s
{'manzana': 4, 'plátano': 2, 'naranja': 3}

```

Agregar pares clave-valor:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario['pera'] = 4
print(mi_diccionario)
✓ 0.0s
{'manzana': 1, 'plátano': 2, 'naranja': 3, 'pera': 4}

```

Eliminar pares clave-valor:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
del mi_diccionario['plátano']
print(mi_diccionario)
✓ 0.0s
{'manzana': 1, 'naranja': 3}

```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Eliminar pares clave-valor:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
del mi_diccionario
print(mi_diccionario)
⊗ 0.2s      Cuidado al usar del:

NameError: name 'mi_diccionario' is not defined

```



TRABAJAR CON ELEMENTOS DE UN DICCIONARIO

Eliminar pares clave-valor:

```

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
del mi_diccionario
print(mi_diccionario)
⊗ 0.2s      Cuidado al usar del:

NameError: name 'mi_diccionario' is not defined

```

Diccionarios vacíos:

```

mi_diccionario = {}
mi_diccionario['manzana'] = 1
mi_diccionario['platano'] = 2
mi_diccionario['naranja'] = 3
print(mi_diccionario)
print(type(mi_diccionario))
✓ 0.0s

{'manzana': 1, 'platano': 2, 'naranja': 3}
<class 'dict'>

```



CASOS DE USO COMUNES:

Diccionario con informaciones diversas sobre un objeto:

```
persona = {"edad": 23, "estado civil": "casado", "DNI": "78656427A"}  
edad = persona["edad"]  
print(edad)  
✓ 0.0s  
23
```



CASOS DE USO COMUNES:

Diccionario con informaciones diversas sobre un objeto:

```
persona = {"edad": 23, "estado civil": "casado", "DNI": "78656427A"}  
edad = persona["edad"]  
print(edad)  
✓ 0.0s  
23
```

Diccionario con un tipo de información sobre varios objetos:

```
lenguaje_programacion = { "Paolo": "Python", "Lucas": "C", "Dani": "Solidity"}  
lenguaje_dani = lenguaje_programacion["Dani"]  
print(lenguaje_dani)  
✓ 0.0s  
Solidity
```



BUENAS PRACTICAS:

Diccionario con informaciones diversas sobre un objeto:

```
persona = {
    "edad": 23,
    "estado civil": "casado",
    "DNI": "78656427A",
}
edad = persona["edad"]
print(edad)
```

Buenas prácticas

✓ 0.0s
23

Diccionario con un tipo de información sobre varios objetos:

```
lenguaje_programacion = {
    "Paolo": "Python",
    "Lucas": "C",
    "Dani": "Solidity",
}
lenguaje_dani = lenguaje_programacion["Dani"]
print(lenguaje_dani)
```

✓ 0.0s
Solidity



MÉTODOS DE DICCIONARIOS:

Keys():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario.keys())
dict_keys(['manzana', 'plátano', 'naranja'])
```

Devuelve una lista de todas las claves en un diccionario



MÉTODOS DE DICCIONARIOS:

Values():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario.values())
✓ 0.0s
dict_values([1, 2, 3])
```

Devuelve una lista de todos los valores en un diccionario.



MÉTODOS DE DICCIONARIOS:

Items():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario.items())
✓ 0.0s
dict_items([('manzana', 1), ('plátano', 2), ('naranja', 3)])
```

Devuelve una lista de todos los pares en un diccionario.



MÉTODOS DE DICCIONARIOS:

Get():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario.get('manzana'))
print(mi_diccionario.get('pera'))
✓ 0.0s
1
None
```

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['manzana'])
✓ 0.0s
1

mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario['pera'])
⊗ 0.0s
-----
KeyError                                     Traceback (most recent call last)
Cell In[28], line 2
      1 mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
      2 print(mi_diccionario['pera'])

KeyError: 'pera'
```

Devuelve el valor de una clave en un diccionario. Si la clave no existe, devuelve un valor predeterminado



MÉTODOS DE DICCIONARIOS:

Get():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
print(mi_diccionario.get('manzana'))
print(mi_diccionario.get('pera',0))
✓ 0.0s
1
0
```

Devuelve el valor de una clave en un diccionario. Si la clave no existe, devuelve un valor predeterminado



MÉTODOS DE DICCIONARIOS:

Pop():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
valor = mi_diccionario.pop('manzana')
print(mi_diccionario)
print(valor)

✓ 0.0s
{'plátano': 2, 'naranja': 3}
1
```

Elimina y devuelve el valor de una clave en un diccionario



MÉTODOS DE DICCIONARIOS:

Clear():

```
mi_diccionario = {'manzana': 1, 'plátano': 2, 'naranja': 3}
mi_diccionario.clear()
print(mi_diccionario)
```

✓ 0.0s

```
{}
```

Elimina todos los pares clave-valor de un diccionario



DE TUPLAS A DICCIONARIOS:

Sintaxis:

```
mis_tuplas = [(key1, value1), (key2, value2), (key3, value3)]  
mi_dict = dict(mis_tuplas)
```

→ Lista de tuplas



DE TUPLAS A DICCIONARIOS:

Sintaxis:

```
mis_tuplas = [(key1, value1), (key2, value2), (key3, value3)]  
mi_dict = dict(mis_tuplas)
```

→ Lista de tuplas

Ejemplo:

```
eventos = [('2022-01-01', 100),  
           ('2022-02-14', 50),  
           ('2022-03-17', 75),  
           ]  
event_dict = dict(eventos)  
print(event_dict)  
✓ 0.0s  
{'2022-01-01': 100, '2022-02-14': 50, '2022-03-17': 75}
```

Lista de tuplas con fecha de eventos y numero de asistentes convertida a un diccionario

IMPORTANTE: Si hay dos tuplas con la misma clave en la lista de tuplas, solo se conservará la última tupla. Es decir, el valor de la clave correspondiente será el valor de la última tupla en la lista.



DE TUPLAS A DICCIONARIOS:

Sintaxis:

```
mis_tuplas = [(key1, value1), (key2, value2), (key3, value3)]  
mi_dict = dict(mis_tuplas)
```

→ Lista de tuplas

Ejemplo:

```
eventos = [('2022-01-01', 100),  
           ('2022-02-14', 50),  
           ('2022-03-17', 75),  
           ('2022-03-17', 15),  
          ]  
event_dict = dict(eventos)  
print(event_dict)  
  
✓ 0.0s  
{'2022-01-01': 100, '2022-02-14': 50, '2022-03-17': 15}
```

Lista de tuplas con fecha de eventos y numero de asistentes convertida a un diccionario

IMPORTANTE: Si hay dos tuplas con la misma clave en la lista de tuplas, solo se conservará la última tupla. Es decir, el valor de la clave correspondiente será el valor de la última tupla en la lista.



DE LISTAS A DICCIONARIOS:

Sintaxis:

```
keys = ['key1', 'key2', 'key3']  
values = [value1, value2, value3]  
my_dict = dict(zip(keys, values))
```



DE LISTAS A DICCIONARIOS:

Sintaxis:

```
keys = ['key1', 'key2', 'key3']
values = [value1, value2, value3]
my_dict = dict(zip(keys, values))
```

Ejemplo:

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
notas_dict = dict(zip(materias, notas))
print(notas_dict)
✓ 0.0s
{'matemáticas': 8.5, 'historia': 7.0, 'ciencias': 9.0}
```



DE LISTAS A DICCIONARIOS:

Sintaxis:

```
keys = ['key1', 'key2', 'key3']
values = [value1, value2, value3]
my_dict = dict(zip(keys, values))
```

Ejemplo:

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
notas_dict = dict(zip(materias, notas))
print(notas_dict)
✓ 0.0s
{'matemáticas': 8.5, 'historia': 7.0, 'ciencias': 9.0}
```

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
lista_tuplas = list(zip(materias, notas))
print(lista_tuplas)
✓ 0.0s
[('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0)]
```

Aquí, la función `zip()` combina las dos listas `materias` y `notas` en una lista de tuplas de elementos correspondientes, y luego el constructor de diccionarios `dict()` crea un diccionario a partir de esta lista de tuplas.



DE LISTAS A DICCIONARIOS:

Sintaxis:

```
keys = ['key1', 'key2', 'key3']
values = [value1, value2, value3]
my_dict = dict(zip(keys, values))
```

Ejemplo:

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
notas_dict = dict(zip(materias, notas))
print(notas_dict)

✓ 0.0s
{'matemáticas': 8.5, 'historia': 7.0, 'ciencias': 9.0}
```

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
lista_tuplas = list(zip(materias, notas))
print(lista_tuplas)

✓ 0.0s
[('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0)]
```

Aquí, la función `zip()` combina las dos listas `materias` y `notas` en una lista de tuplas de elementos correspondientes, y luego el constructor de diccionarios `dict()` crea un diccionario a partir de esta lista de tuplas.

IMPORTANTE: Ten en cuenta que las dos listas deben tener la misma longitud para poder crear un diccionario de esta manera. Si las dos listas tienen longitudes diferentes, la función `zip()` solo tomará los elementos hasta la longitud de la lista más corta.



DE LISTAS A DICCIONARIOS:

Sintaxis:

```
keys = ['key1', 'key2', 'key3']
values = [value1, value2, value3]
my_dict = dict(zip(keys, values))
```

Ejemplo:

```
materias = ['matemáticas', 'historia', 'ciencias', 'física']
notas = [8.5, 7.0, 9.0]
lista_tuplas = list(zip(materias, notas))
print(lista_tuplas)

✓ 0.0s
[('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0)]
```

```
materias = ['matemáticas', 'historia', 'ciencias']
notas = [8.5, 7.0, 9.0]
lista_tuplas = list(zip(materias, notas))
print(lista_tuplas)

✓ 0.0s
[('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0)]
```

Aquí, la función `zip()` combina las dos listas `materias` y `notas` en una lista de tuplas de elementos correspondientes, y luego el constructor de diccionarios `dict()` crea un diccionario a partir de esta lista de tuplas.

IMPORTANTE: Ten en cuenta que las dos listas deben tener la misma longitud para poder crear un diccionario de esta manera. Si las dos listas tienen longitudes diferentes, la función `zip()` solo tomará los elementos hasta la longitud de la lista más corta.



DE SETS A DICCIONARIOS:

Sintaxis:

```
my_set = {('key1', value1), ('key2', value2), ('key3', value3)}
```

Set de tuplas

Ejemplo:

```
notas_set = {('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0)}
notas_dict = dict(notas_set)
print(notas_dict)
✓ 0.0s
{'ciencias': 9.0, 'matemáticas': 8.5, 'historia': 7.0}
```

Aquí, la función `zip()` combina las dos listas *materias* y *notas* en una lista de tuplas de elementos correspondientes, y luego el constructor de diccionarios `dict()` crea un diccionario a partir de esta lista de tuplas.

IMPORTANTE: Si hay dos tuplas con la misma clave en la lista de tuplas, solo se conservará la última tupla. Es decir, el valor de la clave correspondiente será el valor de una de las tuplas del set.



DE SETS A DICCIONARIOS:

Sintaxis:

```
my_set = {('key1', value1), ('key2', value2), ('key3', value3)}
```

Set de tuplas

Ejemplo:

```
notas_set = {('matemáticas', 8.5), ('historia', 7.0), ('ciencias', 9.0), ('ciencias', 8.0)}
notas_dict = dict(notas_set)
print(notas_dict)
✓ 0.0s
{'ciencias': 8.0, 'matemáticas': 8.5, 'historia': 7.0}
```

Aquí, la función `zip()` combina las dos listas *materias* y *notas* en una lista de tuplas de elementos correspondientes, y luego el constructor de diccionarios `dict()` crea un diccionario a partir de esta lista de tuplas.

IMPORTANTE: Si hay dos tuplas con la misma clave en la lista de tuplas, solo se conservará la última tupla. Es decir, el valor de la clave correspondiente será el valor de una de las tuplas del set.



REPASO

- 1) Que es un diccionario y su sintaxis
- 2) Manipular pares clave-valor
(añadir, reasignar, eliminar...)
- 3) Casos de uso
- 4) Métodos aplicables a diccionarios
- 5) Relación entre listas/tuplas/sets y diccionarios

CONQUER
BLOCKS

**CONQUER
BLOCKS**

PYTHON

DICCIONARIOS (PARTE 2)

CONQUERBLOCKS



HASTA AHORA...

- 1) Que es un diccionario y su sintaxis
- 2) Manipular pares clave-valor
(añadir, reasignar, eliminar...)
- 3) Casos de uso
- 4) Métodos aplicables a diccionarios
- 5) Relación entre listas/tuplas/sets y diccionarios



RECORRER PARES DE UN DICCIONARIO

```
● # Diccionario con informacion de un usuario
# en una pagina web
user_0 = {
    'username': 'ef3rmi',
    'nombre': 'Enrique',
    'apellido': 'Gomez',
}

for clave, valor in user_0.items():
    print("Clave:", clave)
    print("Valor:", valor)
    print("")
```

✓ 0.0s

```
Clave: username
Valor: ef3rmi

Clave: nombre
Valor: Enrique

Clave: apellido
Valor: Gomez
```



RECORRER PARES DE UN DICCIONARIO

```
● # Diccionario con informacion de un usuario
# en una pagina web
user_0 = {
    'username': 'ef3rmi',
    'nombre': 'Enrique',
    'apellido': 'Gomez',
}

for clave, valor in user_0.items():
    print("Clave:", clave)
    print("Valor:", valor)
    print("")
```

✓ 0.0s

```
Clave: username
Valor: ef3rmi

Clave: nombre
Valor: Enrique

Clave: apellido
Valor: Gomez
```

```
# Diccionario con informacion de un usuario
# en una pagina web
user_0 = {
    'username': 'ef3rmi',
    'nombre': 'Enrique',
    'apellido': 'Gomez',
}

print(user_0.items()) # Lista de tuplas
```

✓ 0.0s

```
dict_items([('username', 'ef3rmi'), ('nombre', 'Enrique'), ('apellido', 'Gomez')])
```



RECORRER PARES DE UN DICCIONARIO

```
# Diccionario con informacion de un usuario
# en una pagina web
user_0 = {
    'username': 'ef3rmi',
    'nombre': 'Enrique',
    'apellido': 'Gomez',
}

for clave, valor in user_0.items():
    print("Clave:", clave)
    print("Valor:", valor)
    print("")

0.0s

Clave: username
Valor: ef3rmi

Clave: nombre
Valor: Enrique

Clave: apellido
Valor: Gomez
```

```
# Diccionario con informacion de un usuario
# en una pagina web
user_0 = {
    'username': 'ef3rmi',
    'nombre': 'Enrique',
    'apellido': 'Gomez',
}

for tupla in user_0.items():

    print(tupla)
    clave, valor = tupla
    print(clave, valor)

0.0s

('username', 'ef3rmi')
username ef3rmi
('nombre', 'Enrique')
nombre Enrique
('apellido', 'Gomez')
apellido Gomez
```



RECORRER PARES DE UN DICCIONARIO

```
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre, lenguaje in programadores.items():
    print("El programador " + nombre.title() + " domina el lenguaje " +
          lenguaje.title() + ",")

0.0s

El programador Juan domina el lenguaje Python.
El programador Sara domina el lenguaje C.
El programador Eduardo domina el lenguaje Solidity.
El programador Felipe domina el lenguaje Python.
```



RECORRER PARES DE UN DICCIONARIO

```
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre, lenguaje in programadores.items():
    print("El programador " + nombre.title() + " domina el lenguaje " +
          lenguaje.title() + ".")
✓ 0.0s

El programador Juan domina el lenguaje Python.
El programador Sara domina el lenguaje C.
El programador Eduardo domina el lenguaje Solidity.
El programador Felipe domina el lenguaje Python.
```



RECORRER CLAVES DE UN DICCIONARIO

Explicito

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores.keys():
    print(nombre)
✓ 0.0s

juan
sara
eduardo
felipe
```



RECORRER CLAVES DE UN DICCIONARIO

Explicito

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores.keys():
    print(nombre)

✓ 0.0s

juan
sara
eduardo
felipe
```

Implicito

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores:
    print(nombre)

✓ 0.0s

juan
sara
eduardo
felipe
```



RECORRER CLAVES DE UN DICCIONARIO

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores.keys():
    print(nombre)

✓ 0.0s

juan
sara
eduardo
felipe
```

```
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in sorted(programadores.keys()):
    print(nombre)

✓ 0.0s

eduardo
felipe
juan
sara
```



RECORRER CLAVES DE UN DICCIONARIO

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores.keys():
    print(nombre)

✓ 0.0s

juan
sara
eduardo
felipe
```

```
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre, lenguaje in sorted(programadores.items()):
    print(nombre, lenguaje)

✓ 0.0s

eduardo solidity
felipe python
juan python
sara c
```



RECORRER CLAVES DE UN DICCIONARIO

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre in programadores:
    print(nombre, programadores[nombre])

✓ 0.0s

juan python
sara c
eduardo solidity
felipe python
```

```
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for nombre, lenguaje in sorted(programadores.items()):
    print(nombre, lenguaje)

✓ 0.0s

eduardo solidity
felipe python
juan python
sara c
```



RECORRER VALORES DE UN DICCIONARIO

Valores

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for lenguaje in programadores.values():
    print(lenguaje)
] ✓ 0.0s
python
c
solidity
python
```

Valores únicos

```
# Programadore y lenguaje de programacion
programadores = {
    'juan': 'python',
    'sara': 'c',
    'eduardo': 'solidity',
    'felipe': 'python',
}

for lenguaje in set(programadores.values()):
    print(lenguaje)
] ✓ 0.0s
solidity
c
python
```



ANIDAMIENTO — LISTA DE DICCIONARIOS

Tenemos una serie de usuarios. Para cada usuario tenemos un diccionario con la información de esta forma...

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

print(usuario_0)
] ✓ 0.0s
{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}
```



ANIDAMIENTO — LISTA DE DICCIONARIOS

Tenemos una serie de usuarios. Para cada usuario tenemos un diccionario con la información de esta forma...

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

print(usuario_0)
✓ 0.0s
{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}
```

Para guardar la información de todos los usuarios podemos crear una lista de diccionarios, donde cada diccionario guarda la información de cada uno de los usuarios:

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

usuario_1 = {
    'username': 'jane_doe',
    'nacionalidad': 'Canada',
    'puntuacion': 92,
}

usuario_2 = {
    'username': 'bob_smith',
    'nacionalidad': 'UK',
    'puntuacion': 78,
}

usuarios = [usuario_0, usuario_1, usuario_2]

print(usuarios)
✓ 0.0s
[{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}, {'username': 'jane_doe', 'nacionalidad': 'Canada', 'puntuacion': 92}, {'username': 'bob_smith', 'nacionalidad': 'UK', 'puntuacion': 78}]
```



ANIDAMIENTO — LISTA DE DICCIONARIOS

Tenemos una serie de usuarios. Para cada usuario tenemos un diccionario con la información de esta forma...

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

print(usuario_0)
✓ 0.0s
{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}
```

Recorrer elementos de una lista de diccionarios:

```
usuario3_puntuacion = usuarios[2]['puntuacion']
print(usuario3_puntuacion)
✓ 0.0s
78
```

Para guardar la información de todos los usuarios podemos crear una lista de diccionarios, donde cada diccionario guarda la información de cada uno de los usuarios:

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

usuario_1 = {
    'username': 'jane_doe',
    'nacionalidad': 'Canada',
    'puntuacion': 92,
}

usuario_2 = {
    'username': 'bob_smith',
    'nacionalidad': 'UK',
    'puntuacion': 78,
}

usuarios = [usuario_0, usuario_1, usuario_2]

print(usuarios)
✓ 0.0s
[{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}, {'username': 'jane_doe', 'nacionalidad': 'Canada', 'puntuacion': 92}, {'username': 'bob_smith', 'nacionalidad': 'UK', 'puntuacion': 78}]
```



ANIDAMIENTO — LISTA DE DICCIONARIOS

Tenemos una serie de usuarios. Para cada usuario tenemos un diccionario con la información de esta forma...

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

print(usuario_0)
✓ 0.0s
{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}
```

Recorrer elementos de una lista de diccionarios:

```
usuario3_puntuacion = usuarios[2]['puntuacion']
print(usuario3_puntuacion)
```

Para guardar la información de todos los usuarios podemos crear una lista de diccionarios, donde cada diccionario guarda la información de cada uno de los usuarios:

```
usuario_0 = {
    'username': 'john_doe',
    'nacionalidad': 'USA',
    'puntuacion': 85,
}

usuario_1 = {
    'username': 'jane_doe',
    'nacionalidad': 'Canada',
    'puntuacion': 92,
}

usuario_2 = {
    'username': 'bob_smith',
    'nacionalidad': 'UK',
    'puntuacion': 78,
}

usuarios = [usuario_0, usuario_1, usuario_2]

print(usuarios)
✓ 0.0s
[{'username': 'john_doe', 'nacionalidad': 'USA', 'puntuacion': 85}, {'username': 'jane_doe', 'nacionalidad': 'Canada', 'puntuacion': 92}, {'username': 'bob_smith', 'nacionalidad': 'UK', 'puntuacion': 78}]
```



ANIDAMIENTO — LISTAS EN DICCIONARIOS

Pedidos en un restaurante:

```
# Datos de un pedido de pizza
pizza = {
    'masa': 'fina',
    'ingredientes': ["aceitunas", "champiñones"],
}

# Resumen del pedido
print("Has pedido una pizza de masa " + pizza['masa'] +
      " con los siguientes ingredientes:")
for ingrediente in pizza['ingredientes']:
    print(ingrediente)
✓ 0.0s
Has pedido una pizza de masa fina con los siguientes ingredientes:
aceitunas
champiñones
```

Trabajadores en una compañía:

```
# Datos de trabajadores
programadores = {
    'juan': ['python', 'c++'],
    'sara': ['c', 'rust'],
    'eduardo': ['solidity', 'fortran'],
    'felipe': ['python', 'fortran', 'R'],
}
for nombre, lenguajes in programadores.items():
    print("\n" + nombre.title() + " sabe usar los lenguajes:")
    for lenguaje in lenguajes:
        print(lenguaje.title())
✓ 0.0s
Juan sabe usar los lenguajes:
Python
C++
Sara sabe usar los lenguajes:
C
Rust
Eduardo sabe usar los lenguajes:
Solidity
Fortran
Felipe sabe usar los lenguajes:
Python
Fortran
R
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

Sintaxis:

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : {
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

Sintaxis:

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : {
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

Sintaxis:

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : {
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

Sintaxis:

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        nombre: 'lucas',
        apellido: 'vene',
        ubicacion: 'paris',
    },
    'crodriguez' : {
        nombre: 'carlos',
        apellido: 'rodriguez',
        ubicacion: 'madrid',
    },
    'tbauer': {
        nombre: 'thomas',
        apellido: 'bauer',
        ubicacion: 'berlin',
    }
}
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

Sintaxis:

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : {
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

```
# diccionario con ususarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : {
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```

```
for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['nombre'] + " " + user_info["apellido"]
    ubicacion = user_info["ubicacion"]

    print("\tNombre completo: " + full_name.title())
    print("\tUbicacion: " + ubicacion.title())
    ✓ 0.0s

Username: lvene
    Nombre completo: Lucas Vene
    Ubicacion: Paris

Username: crodriguez
    Nombre completo: Carlos Rodriguez
    Ubicacion: Madrid

Username: tbauer
    Nombre completo: Thomas Bauer
    Ubicacion: Berlin
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

```
# diccionario con usuarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : { Atención: Hacer esto puede complicar el código innecesariamente
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```

```
for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['nombre'] + " " + user_info["apellido"]
    ubicacion = user_info["ubicacion"]

    print("\tNombre completo: " + full_name.title())
    print("\tUbicacion: " + ubicacion.title())
```

```
Username: lvene
    Nombre completo: Lucas Vene
    Ubicacion: Paris

Username: crodriguez
    Nombre completo: Carlos Rodriguez
    Ubicacion: Madrid

Username: tbauer
    Nombre completo: Thomas Bauer
    Ubicacion: Berlin
```



ANIDAMIENTO — DICCIONARIOS EN DICCIONARIOS

```
# diccionario con usuarios de una pagina web
users = {
    'lvene': {
        'nombre': 'lucas',
        'apellido': 'vene',
        'ubicacion': 'paris',
    },
    'crodriguez' : { Atención: Hacer esto puede complicar el código innecesariamente
        'nombre': 'carlos',
        'apellido': 'rodriguez',
        'ubicacion': 'madrid',
    },
    'tbauer': {
        'nombre': 'thomas',
        'apellido': 'bauer',
        'ubicacion': 'berlin',
    }
}
```

```
for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['nombre'] + " " + user_info["apellido"]
    ubicacion = user_info["ubicacion"]

    print("\tNombre completo: " + full_name.title())
    print("\tUbicacion: " + ubicacion.title())
```

Consejo: Mantener los diccionarios internos con la misma estructura

```
Nombre completo: Lucas Vene
Ubicacion: Paris

Username: crodriguez
    Nombre completo: Carlos Rodriguez
    Ubicacion: Madrid

Username: tbauer
    Nombre completo: Thomas Bauer
    Ubicacion: Berlin
```



REPASO

- 1) Bucles y diccionarios (recorrer claves, valores y pares)

- 2) Anidamiento en diccionarios
(listas en diccionarios / diccionarios en listas / diccionarios en diccionarios)

CONQUER
BLOCKS

AMA Python

AMA 06-04-2024

- Objetivo: switch

```
texto = input("Que lenguaje de programacion te gustaria aprender?")
match texto:
    case "JavaScript":
        print("Aprendamos JS")
    case "Python":
        print("Aprendamos Python")           I
    case "PHP":
        print("Aprendamos PHP")
    case "Java":
        print("Aprendamos Java")
```

AMA 13-04-2024

- Objetivo: compresión de listas

```
[2] cap_lenguajes = []
    for lenguaje in lenguajes:
        cap_lenguajes.append(lenguaje.capitalize())
    cap_lenguajes

['Python', 'Javascript', 'C++']

[11] multiplos = []
    for numero in range(1,100):
        if numero % 5 == 0:
            multiplos.append(numero)
    multiplos

['Python', 'Javascript', ...]

[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]

[12] multiplos_comp = [numero for numero in range(1,100) if numero%5 == 0]
    multiplos_comp

[5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95]
```

AMA 13-04-2024

- Objetivo: compresión de listas

```
[13] coordenadas = []
    for x in range(0,6):
        for y in range(0,11):
            coordenadas.append((x,y))
    coordenadas

[14] coordenadas = [(x,y) for x in range(1,6)
    coordenadas
```

▼ Diccionarios

```
dobles = { str(numero): numero *2 for numero in range(1,11)}
doubles

{'1': 2,
 '2': 4,
 '3': 6,
 '4': 8,
 '5': 10,
 '6': 12,
 '7': 14,
 '8': 16,
 '9': 18,
 '10': 20}
```

AMA 13-04-2024

- Objetivo: compresión de listas

```
palabras = ["Manzana", "Banana", "123abc", "Perro", "Gato", "Elefante", "456def", "Abeja", "Pájaro"]
# Lista numero 1 - palabras que empiezen con vocal
vocal = [palabra.lower() for palabra in palabras if palabra[0].lower() in "aeiou"]
# Lista numero 2 - palabras que empiezen con consonante
consonante = [palabra.lower() for palabra in palabras if palabra[0].lower() not in "aeiou" and palabra[0].lower().isalpha()]
# Lista numero 3 - palabras que empiezen con numero
numeros = [palabra.lower() for palabra in palabras if palabra[0].isdigit()]
print()

personas = {
    "Ana": 25,
    "Juan": 30,
    "123Maria": 40,
    "Pedro": 35,
    "678Carlos": 45,
    "Elena": 28,
    "999Luisa": 50
}

vocales = {nombre.lower(): edad for nombre,edad in personas.items() if nombre[0].lower() in "aeiou"}
consonantes= {nombre.lower():edad for nombre,edad in personas.items() if nombre[0].lower() not in "aeiou" and nombre[0].isalpha()}
numeros = {nombre.lower():edad for nombre,edad in personas.items() if nombre[0].isdigit()}
print(f"""
Diccionario con nombres y edad cuyo nombre empiecen con una vocal: {vocales}
Diccionario con nombres y edad cuyo nombre empiecen con una consonante: {consonantes}
Diccionario con nombres y edad cuyo nombre empiecen con una numero: {numeros}
""")
```

AMA 03-05-2024

- Errores de Python

1. SyntaxError -> errores de sintaxis (lenguaje propio de python)
2. NameError -> Variable no fue definida
3. TypeError -> Nos referimos a una variable del tipo incorrecto
4. IndexError -> Cuando intentamos acceder a un indice FUERA DE RANGO
5. KeyError -> (diccionarios) intentamos acceder a una llave que no existe
6. ValueError -> Cuando recibimos un argumento de tipo correcto pero no del valor correcto
7. AttributeError -> Intentamos acceder a un atributo que no existe en un obj.
8. ImportError -> cuando el modulo a importar no existe
9. FileNotFoundError -> Intentamos abrir un directorio o fichero en especifico , pero no lo encontramos
10. ZeroDivisionError -> Cuando intentamos dividir un numero entre 0

AMA 03-05-2024

- Errores de Python

1. SyntaxError -> errores de sintaxis (lenguaje propio de python)

```
▶ x = 10
if x=10:
    print( "El valor de x es igual a 10 ")

→ File "<ipython-input-8-9dcb8f8f9371>", line 2
    if x=10:
        ^
SyntaxError: invalid syntax. Maybe you meant '==' or '!=' instead of '='?
```

AMA 03-05-2024

- Errores de Python

2. NameError -> Variable no fue definida

```
▶ # NameError
print(y)

→ -----
NameError                                 Traceback (most recent call last)
<ipython-input-10-5ac28001d210> in <cell line: 2>()
      1 # NameError
----> 2 print(y)

NameError: name 'y' is not defined
```

AMA 03-05-2024

- Errores de Python

3. `TypeError` -> Nos referimos a una variable del tipo incorrecto

```
# TypeError
resultado = 5 +"2"

[1]: 
  File "", line 2
    resultado = 5 +"2"
               ^
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

AMA 03-05-2024

- Errores de Python

10. ZeroDivisionError -> Cuando intentamos dividir un numero entre 0

```
division = 10/0
-----
ZeroDivisionError                                 Traceback (most recent call last)
<ipython-input-22-0c12ad8bda7b> in <cell line: 1>()
      1 division = 10/0
----> 1 division = 10/0

ZeroDivisionError: division by zero
```

AMA 03-05-2024

- Errores de Python

10. ZeroDivisionError -> Cuando intentamos dividir un numero entre 0

```
division = 10/0
-----
ZeroDivisionError                                Traceback (most recent call last)
<ipython-input-22-0c12ad8bda7b> in <cell line: 1>()
      1 division = 10/0
      2
      3 ZeroDivisionError: division by zero
      4     try:
      5         division = 10/2
      6     except ZeroDivisionError:
      7         print( "Hey! no puedes dividir entre cero " )
      8     else:
      9         print( division )
```

AMA 03-05-2024

- Error genérico general

```
# Generico error
x=0
def function():
    if x<2:
        raise Exception("Error ")

try:
    function()
except Exception as e :
    print( "Se produjo un error: ",e)
```

AMA 03-05-2024

- Error genérico personalizado

```
# Personalizadas
class MiError(Exception):
    def __init__(self,mensaje):
        self.mensaje=mensaje

def funcion_personalizada():
    raise MiError("Este es un error personalizado ")

try:
    funcion_personalizada()
except MiError as e:
    print( "Hay un error de tipo: ",e.mensaje) | I

[62] # class
class ErrorBooleano(Exception):
    def __init__(self,mensaje):
        self.mensaje=mensaje

def fubncion(apagado):
    if apagado==False:
        raise ErrorBooleano("Porfavor prende la luz")
try:
    fubncion(True)
except ErrorBooleano as e:
    print(e)
else:
    print("Gracias por tener la luz encendida")
```

Hay un error de tipo: Este es un error personalizado

Gracias por tener la luz encendida

AMA 01-06-2024

- Importar módulos

AMA
 paquete
 > __pycache__
 ✚ _init_.py
 ✚ modulo_1.py
 ✚ modulo_2.py
 ✚ main.py

AMA > paquete > ✚ modulo_1.py > ⚡ funcion_modulo1

```
1  def funcion_modulo1():
2      print('Hola soy el modulo 1')
```

AMA > paquete > ✚ modulo_2.py > ⚡ funcion_modulo2

```
1  def funcion_modulo2():
2      print('Hola soy el modulo 2')
```

AMA > ✚ main.py

```
1  from paquete import modulo_1,modulo_2
2  modulo_1.funcion_modulo1()
3  modulo_2.funcion_modulo2()
```