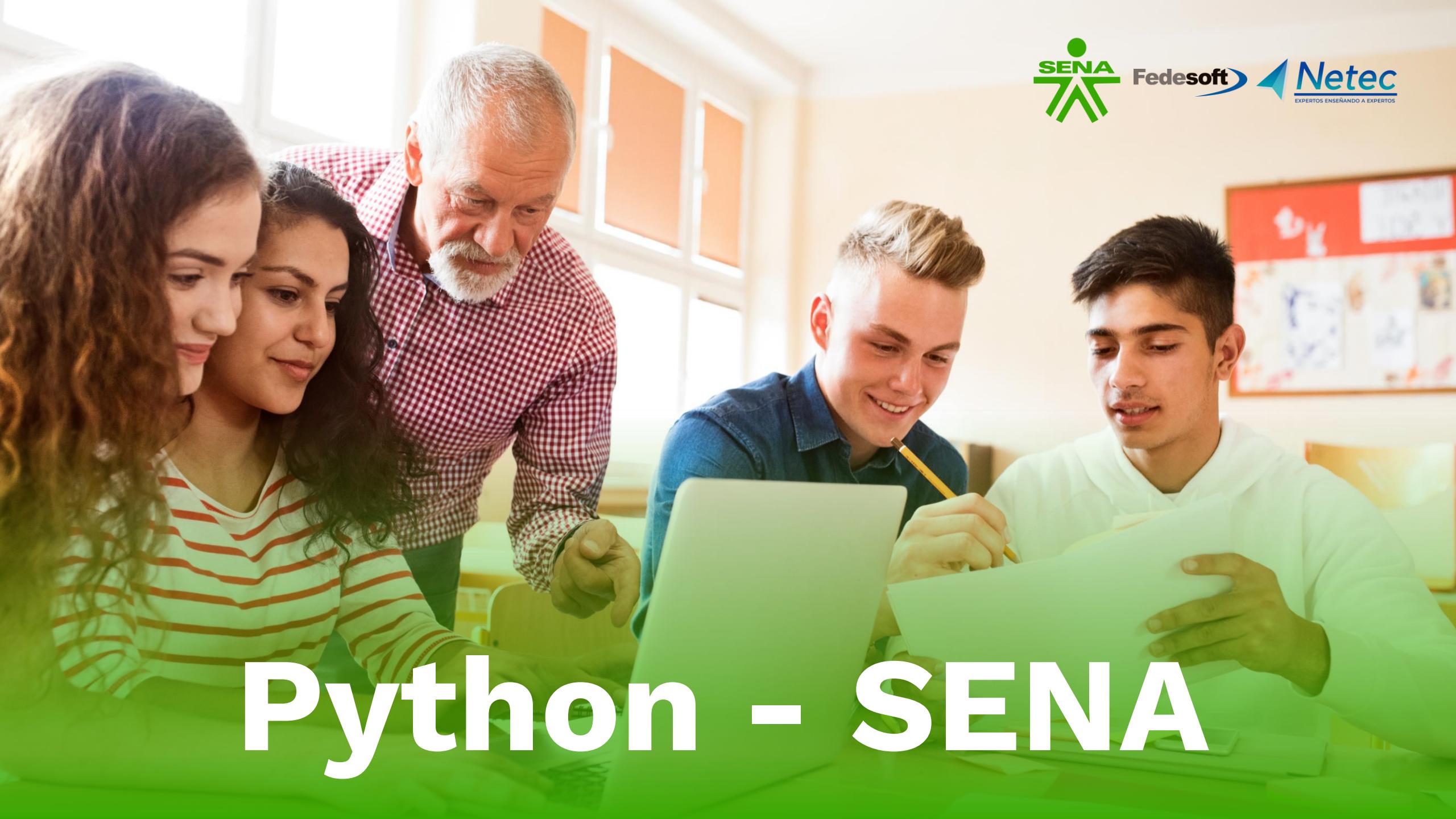


Formación Continua Especializada

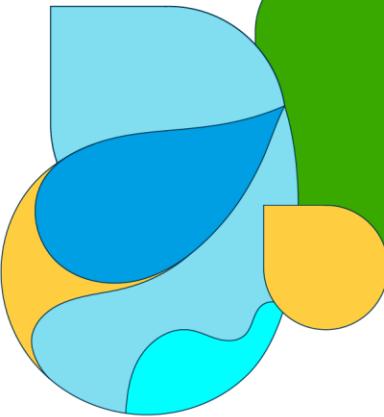
CONVOCATORIA
DG 0001 - 2023



Las acciones de formación ejecutadas en el marco de la convocatoria
DG 0001 - 2023 son gratuitas para los trabajadores beneficiarios



Python - SENA

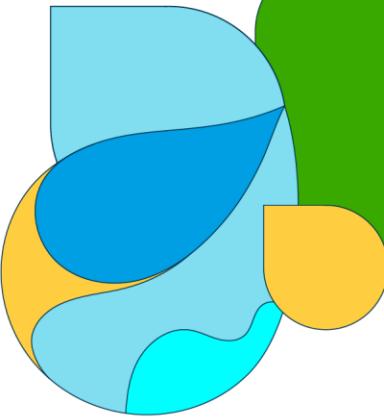


Propiedad intelectual

Material didáctico preparado por la empresa Global K, S.A. de C.V., Registrado en Derechos de Autor.

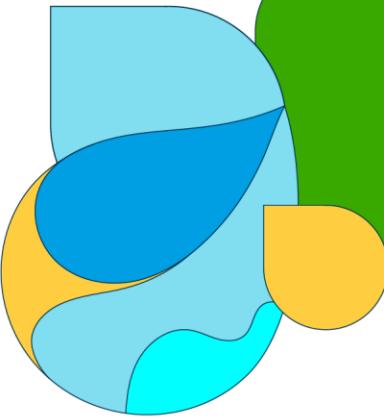
Todos los contenidos de este Sitio (Incluyendo, pero no limitado a, texto, logotipos, contenido, fotografías, audio, botones, nombres comerciales y video) están sujetos a derechos de propiedad por las leyes de Derechos de Autor de la empresa Global K, S.A. de C.V.

Queda prohibido copiar, reproducir, distribuir, publicar, transmitir, difundir, o en cualquier modo explotar cualquier parte de este documento sin la autorización previa por escrito de Global K, S.A. de C.V. o de los titulares correspondientes.



Temario

- **UT 1.** Introducción al lenguaje Python
- **UT 2.** Tipos, operadores, expresiones y E/S y control de flujos
- **UT 3.** Colecciones y funciones
- **UT 4.** Introducción al uso aplicado de Python para el desarrollo
- **UT 5.** Python Essentials para el desarrollo de software
- **UT 6.** Módulos y Paquetes de phyton para el desarrollo de software
- **UT 7.** Cadenas y colecciones para el desarrollo de software
- **UT 8.** Excepciones



Temario

- **UT 9.** La programación orientada a objetos usando python
- **UT 10.** Archivos, directorios y closures 6 virtuales
- **UT 11.** Bases de datos y Archivo CSV
- **UT 12.** Librerías para Ciencia de Datos: NumPy
- **UT 13.** Librerías para Ciencia de Datos: Pandas
- **UT 14.** Librerías para Ciencia de Datos: Matplotlib
- **UT 15.** Librerías para Ciencia de Datos: SciPy
- **UT 16.** Librerías para Ciencia de Datos: Scikit-learn



Presentación del grupo

- ¿Cuál es su nombre?
- ¿Experiencia acerca del desarrollo?
- ¿Qué tecnología/idea/software le ha impresionado?
- ¿Cuáles son sus expectativas respecto al curso?

Unidad temática 1

Introducción al lenguaje Python y conceptos básicos

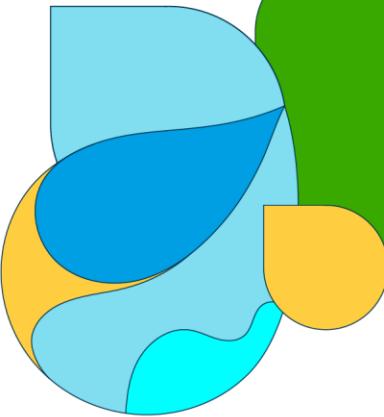
Objetivos:

- Identificar las principales características de Python.
- Listar los tipos de aplicaciones más comunes en las que se usa Python.
- Instalar las herramientas de desarrollo Python usadas en el curso.

1.1 Introducción

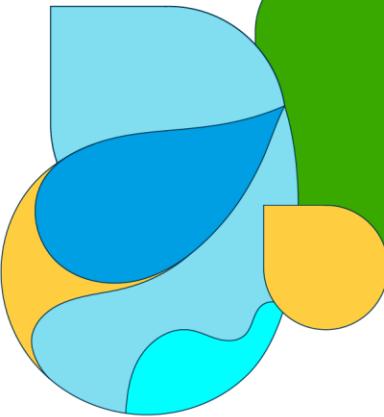
Objetivos:

- Identificar las principales características de Python.
- Listar los tipos de aplicaciones más comunes en las que se usa Python.
- Instalar las herramientas de desarrollo Python usadas en el curso.



Introducción

- Python fue diseñado por Guido van Rossum en 1991.
- Sintaxis de alto nivel.
- Interpretado.
- Python no se basa en la sintaxis de Lenguaje C
 - Como son: Java, C#, Objective-C, PHP y JavaScript.
- Fácil de usar.
- Multiparadigma.



Introducción

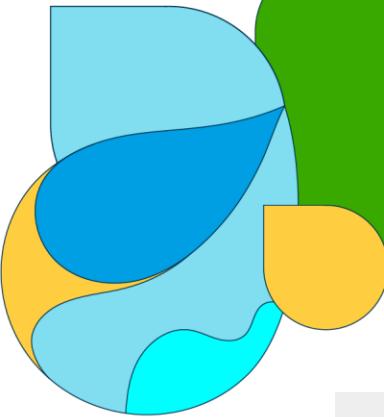
- Orientado a Objetos, todo en Python es un objeto.
 - Pero incorpora aspectos de la programación funcional, procedural e imperativa.
- En el curso se usará Python 3.
 - El soporte de Python 2 finalizó oficialmente en 2020.

Posicionamiento | IEEE Spectrum

Rank	Language	Type	Score
1	Python	🌐💻⚙️	100.0
2	Java	🌐📱💻	96.3
3	C	📱💻⚙️	94.4
4	C++	📱💻⚙️	87.5
5	R	💻	81.5
6	JavaScript	🌐	79.4

Posicionamiento | TIOBE

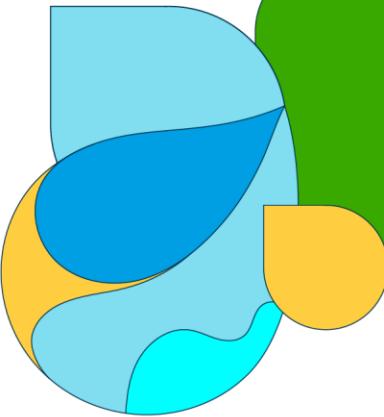
Oct 2021	Oct 2020	Change	Programming Language	Ratings	Change
1	3	▲	 Python	11.27%	-0.00%
2	1	▼	 C	11.16%	-5.79%
3	2	▼	 Java	10.46%	-2.11%
4	4		 C++	7.50%	+0.57%
5	5		 C#	5.26%	+1.10%



Posicionamiento | PYPL

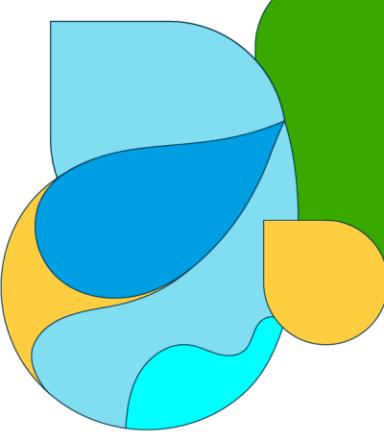
Worldwide, Oct 2021 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Python	29.66 %	-2.1 %
2		Java	17.18 %	+0.8 %
3		JavaScript	8.81 %	+0.4 %
4		C#	7.3 %	+1.1 %
5	↑	C/C++	6.48 %	+0.7 %



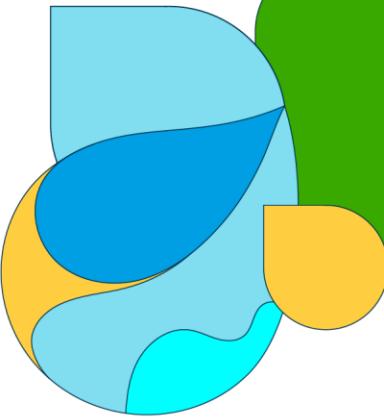
Tipos de Aplicaciones

- Desarrollo Web.
- Ciencia de Datos.
- Aprendizaje Automático: Scikit-Learn & Tensor-Flow.
- Análisis y visualización de Datos con Python: Matplotlib & SeaBorn.
- Scripting.
- Aplicaciones integradas.
- Gaming y desarrollo de videojuegos.



Documentación

- [Python 3.10.0 documentation](#)
- [Python 2.7.17 documentation](#)
- [Beginner's Guide to Python](#)
- [Ecosistema de software de código abierto basado en Python para matemáticas, ciencias e ingeniería.](#)
- [Conferencias y reuniones de grupos de usuarios.](#)



Entorno de Desarrollo



The screenshot shows the Python.org website's download page for Windows. At the top, the Python logo and the word "python" are displayed. Below the logo is a navigation bar with four tabs: "About", "Downloads", "Documentation", and "Community". The "Downloads" tab is currently selected. A large yellow button with the text "Download Python 3.10.0" is prominently featured. Below this button, text encourages users to look for Python versions for different operating systems, mentioning "Windows", "Linux/UNIX", "macOS", and "Other". It also provides links for "Prereleases" and "Docker images". Finally, it mentions "Python 2.7" releases.

python™

About Downloads Documentation Community

Download the latest version for Windows

Download Python 3.10.0

Looking for Python with a different OS? Python for [Windows](#),
[Linux/UNIX](#), [macOS](#), [Other](#)

Want to help test development versions of Python? [Prereleases](#),
[Docker images](#)

Looking for Python 2.7? See below for specific releases

Entorno de Desarrollo | PATH



Agregar la carpeta
a la variable de
ambiente PATH

Entorno de Desarrollo | Versión

```
Administrator: Command Prompt

C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>python --version
Python 3.10.0

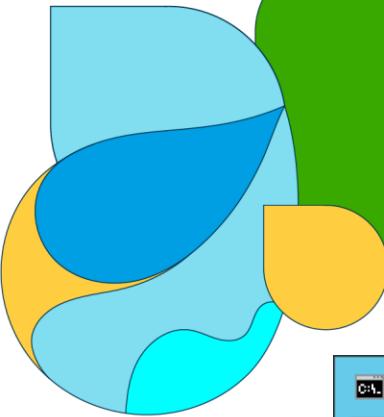
C:\Users\Administrator>python -V
Python 3.10.0

C:\Users\Administrator>py --version
Python 3.10.0

C:\Users\Administrator>py -V
Python 3.10.0

C:\Users\Administrator>
```

Entorno de Desarrollo | Consola



```
Administrator: Command Prompt - python
C:\Users\Administrator>
C:\Users\Administrator>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print("Hola Mundo")
Hola Mundo
>>>
>>> import sys
>>> print(sys.version)
3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)]
>>>
>>>
```

Entorno de Desarrollo | VSC

The image shows a composite view of the Visual Studio Code website and its interface.

Visual Studio Code Website:

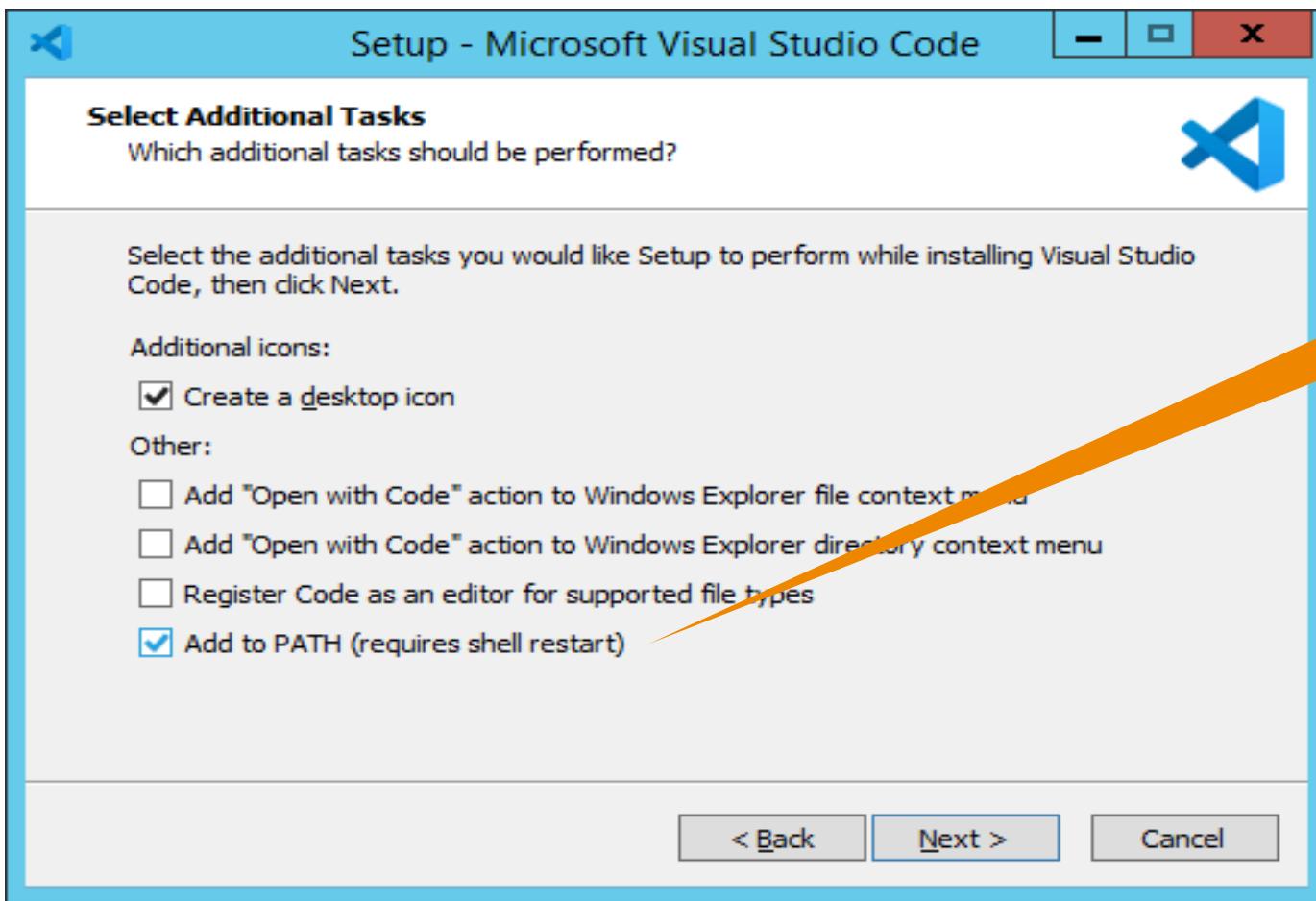
- Header:** Visual Studio Code, Docs, Updates, Blog, API, Extensions, FAQ, Search Docs, Download.
- Middle Content:** Text: "Version 1.42 is now available! Read about the new features and fixes from January." Below it, a large graphic reads: "Code editing. Redefined. Free. Built on open source. Runs everywhere." A download section for Windows, macOS, and Linux is shown.
- Bottom Content:** Text: "You can now view create-react-app in the browser." Local: http://localhost:3000/

Visual Studio Code Interface:

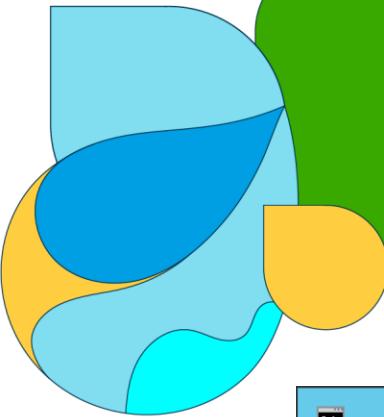
- Left Sidebar:** Extensions Marketplace showing installed extensions like Python, GitLens, C/C++, ESLint, Debugger for Chrome, Language Support, vscode-icons, and Vetur.
- Central Area:** Code editor showing a service worker file with code related to service workers and navigation.
- Bottom:** Terminal window showing "node" and the URL "http://localhost:3000/".

<https://code.visualstudio.com/>

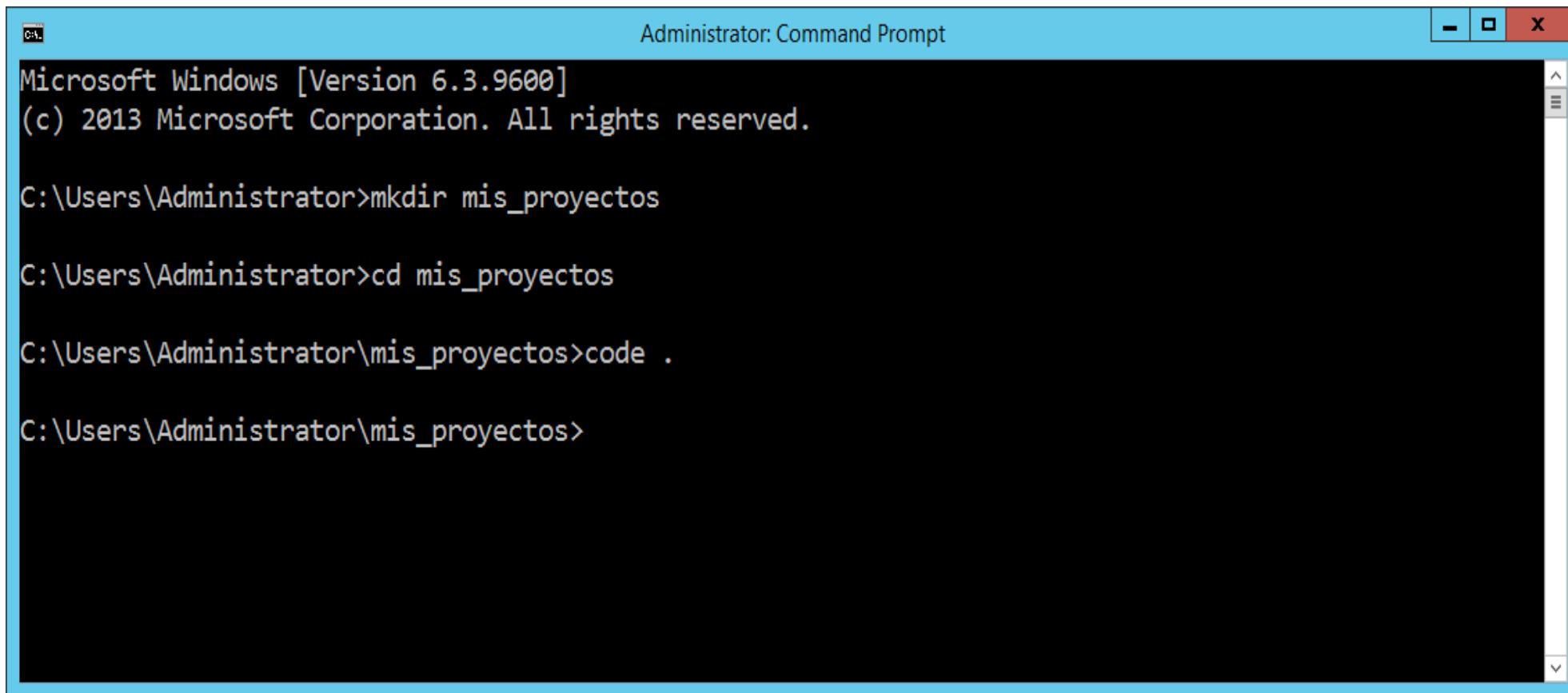
Entorno de Desarrollo | VSC



Agregar la carpeta
a la variable de
ambiente PATH



Entorno de Desarrollo | code



```
Administrator: Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>mkdir mis_proyectos

C:\Users\Administrator>cd mis_proyectos

C:\Users\Administrator\mis_proyectos>code .

C:\Users\Administrator\mis_proyectos>
```

Entorno de Desarrollo | VSC

The diagram illustrates the Visual Studio Code (VSC) interface with numbered callouts:

- 1**: Explorer icon in the left sidebar.
- 2**: Line numbers in the code editor.
- 3**: Terminal output in the bottom panel.
- 4**: File menu in the top bar.
- 5**: Python icon in the left sidebar.
- 6**: Grid icon in the left sidebar.

Code Editor Content:

```
File Edit Selection View Go Debug Terminal Help Saludo.py - mis_proyectos - Visual Studio Code [Administrator]
EXPLORER Saludo.py X
OPEN EDITORS Saludo.py > ...
MIS_PROYECTOS Saludo.py
1 import sys
2 def saludo(mensaje):
3     print(mensaje)
4     print(sys.version)
5
6 saludo("Hola Mundo Python")
```

Terminal Output:

```
C:\Users\Administrator\mis_proyectos>python Saludo.py
Hola Mundo Python
3.5.4 (v3.5.4:3f56838, Aug  8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)]
C:\Users\Administrator\mis_proyectos>
```

Python 3.8.1 64-bit ① 0 △ 0 Ln 8, Col 28 Spaces: 4 UTF-8 CRLF Python ⚙ 1

Resumen

- Python es un lenguaje de programación, interpretado de código abierto.
- Python es un lenguaje multiparadigma, principalmente orientado a objetos de alto nivel.
- La sintaxis de Python hace énfasis en la legibilidad del código, lo cual facilita el mantenimiento y depuración.
- Python ofrece la potencia y la flexibilidad de los lenguajes compilados con una curva suave de aprendizaje.
- Python es un lenguaje versátil usado en varios campos diferentes, desde aplicaciones móviles, videojuegos, scripting, ciencia de datos entre otros.
- PyPI es un repositorio de módulos Python de terceros aportados para la comunidad.

Resumen (Cont.)

- Sus características multiplataforma permiten que este pueda ser ejecutado en diferentes sistemas operativos.
- CPython, Jython, IronPython y PyPy son implementaciones de Python.
- CPython, implementación de Python en Lenguaje C.
- JPython, implementación de Python en Java.
- IronPython, implementación de Python en C#.
- PyPy, implementación de Python en Python.

1.2 Conceptos Básicos

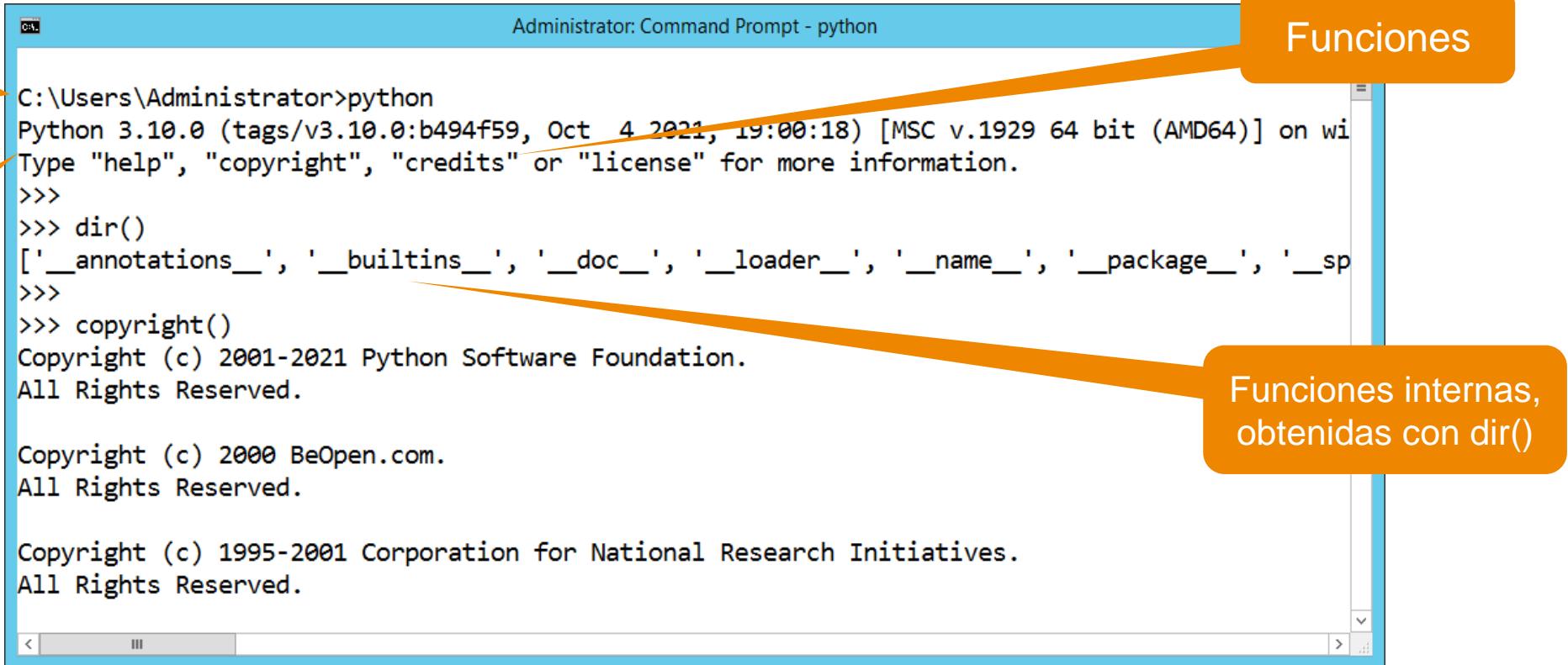
Objetivos:

- Escribir sentencias de código en la consola de Python.
- Conocer los tipos de comentarios de Python.
- Identificar el término de tipado dinámico.
- Conocer los tipos de asignación que tiene Python.
- Listar los tipos de datos básicos con los que cuenta Python.

Introducción | Consola Python

Comando:
python o py

Versión



Administrator: Command Prompt - python

```
C:\>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on wi
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> dir()
['__annotations__', '__builtins__', '__doc__', '__loader__', '__name__', '__package__', '__sp
>>>
>>> copyright()
Copyright (c) 2001-2021 Python Software Foundation.
All Rights Reserved.

Copyright (c) 2000 BeOpen.com.
All Rights Reserved.

Copyright (c) 1995-2001 Corporation for National Research Initiatives.
All Rights Reserved.
```

Funciones

Funciones internas,
obtenidas con dir()

Comentario de una sola línea

```
"""
```

Comentarios multilínea

línea 1

línea 2

```
"""
```

```
'''
```

Comentarios multilínea

línea 1

línea 2

```
'''
```

Tipos de Datos

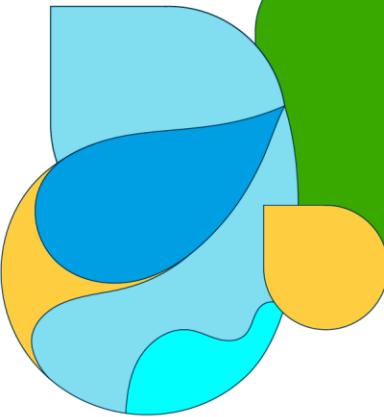
int

complex

float

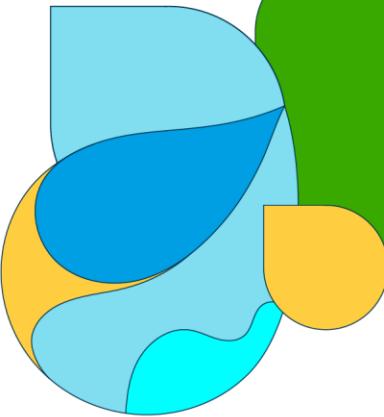
str

bool



Tipos de Datos (Cont.)

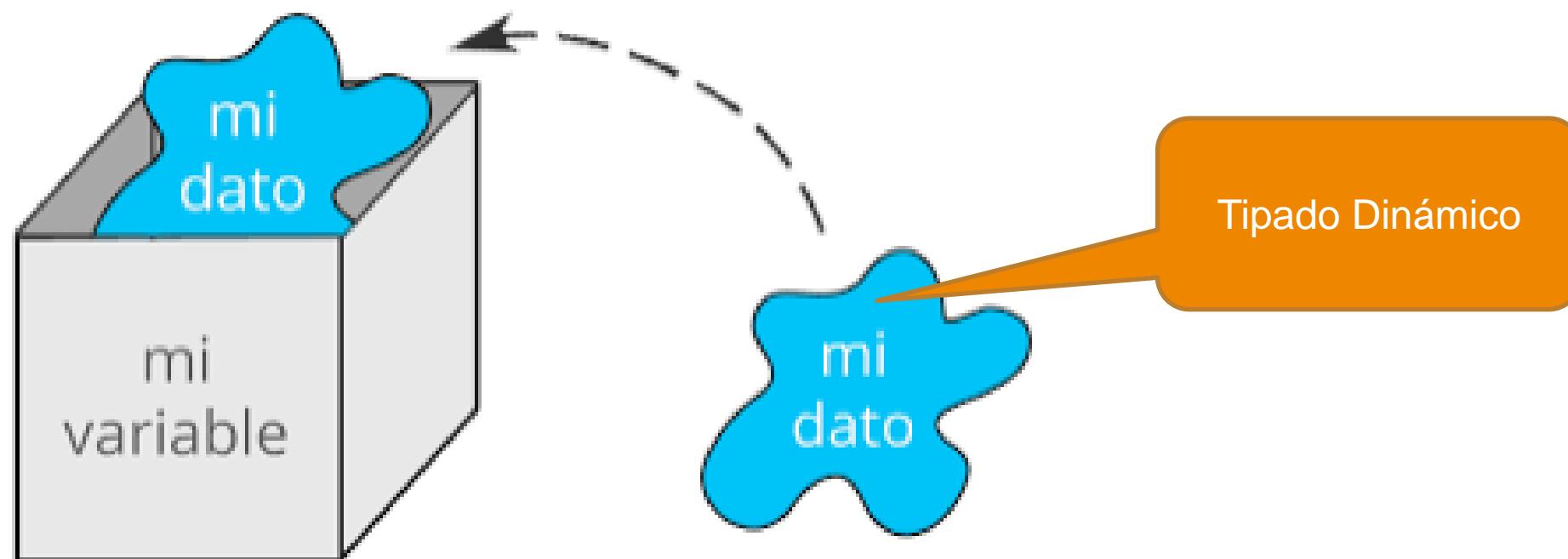
boolean	bool	True o False.
integer	int	-1, 0, 1, 5, 0xffee, 0o647, 0b0101
float	float	3.1416, 0.10, 10e-1, 100E-2
string	str	'Hola', "Hola", 'Python "es un lenguaje" '
complex	complex	10+5j, 5j

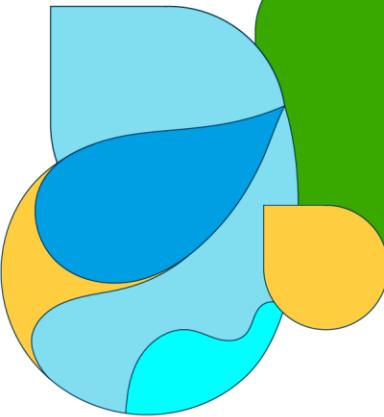


Literales

int	-3, -2, -1, 0, 1, 2, 3
float	3.5, math.pi, 0.16
str	"", "Hola", "Netec México", “Hola\n\rMundo”
None	None podría ser considerada una literal, y no una palabra reservada.
bool	True y False , son consideradas literales booleanas y/o palabras reservadas.

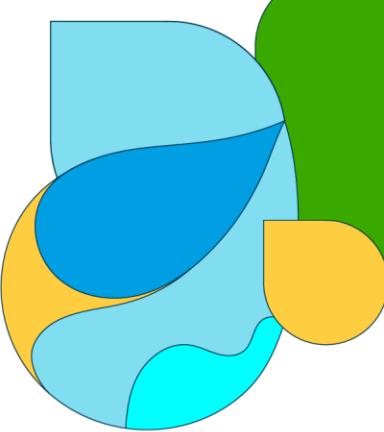
Variables





Palabras Reservadas | Keywords

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	



Expresiones

Una expresión es una combinación de valores, variables y operadores.

$$2 + 3$$

$$a = 5$$

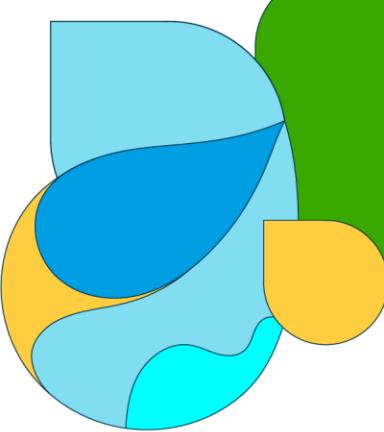
$$b = a + 2 + 3$$

Expresiones (Cont.)

```
Administrator: Command Prompt

C:\Users\Administrator>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 1 + 1
2
>>> 10 + 4 + 3*2
20
>>>
>>> "Hola" + " " + "Mundo" + 'Python'
'Hola MundoPython'
>>>
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> exit()

C:\Users\Administrator>
```



Operadores

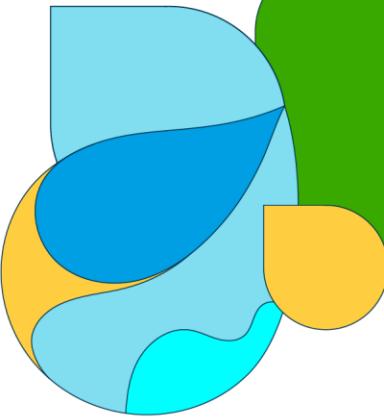
```
>>> help('symbols')
```

**
*/%
+ -
=

```
10 + 30  
minutos - 1  
hora*60 + minutos  
minutos / 60  
2 ** 4  
(1 + 9) * (10 - 3)
```

Operadores | Ejemplos

```
Administrator: Command Prompt - python
0
>>> hora=1
>>> minutos=60
>>>
>>> 10 + 30;  minutos - 1;  hora*60 + minutos;  minutos/60 ; 2**4; (1+9)*(10-3)
40
59
120
1.0
16
70
>>>
```

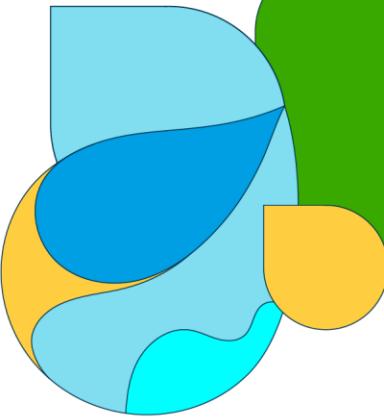


Operadores Aritméticos | Potencia

**

$z, a, b = 0, 2, 4$

$z = 2^{**4} \#z$ queda con 16



Operadores Aritméticos | Multiplicativos

*

$z,a,b=0,1,1$

$z= a * b$ #z queda con el valor de 1

/

$z,a,b=0,1,3$

$z= a / b$ #z queda con el valor de 0.3333

//

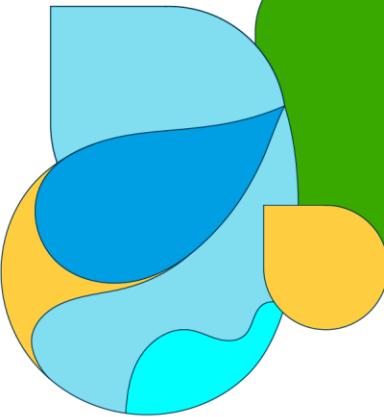
$z,a,b=0,1,3$

$z= a // b$ #z queda con el valor de 0

%

$z,a,b=0,5,3$

$z= a \% b$ #z queda con el valor de 2



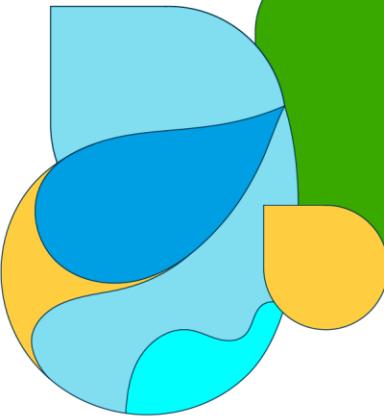
Operadores Aritméticos |

Suma & Resta

+

```
z,a,b=0,1,1 #Múltiple asignación
z= a + b #z queda con el valor de 2
z,a,b=0,1,1
z= a - b #z queda con el valor de 0
```

-



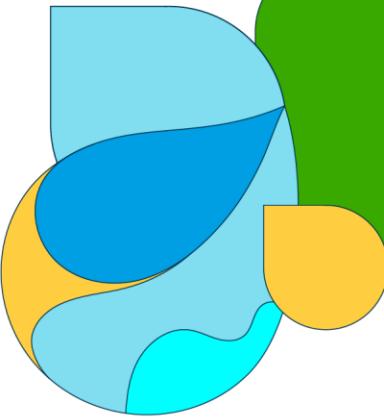
Operadores Extendidos | +

a,b= "Hola", "Python"
+ z= a + " " + c #z "Hola Python"
z= a + 3 #Error de compilación

*

a="Hola"

z= a**2 #z "HolaHola"



Operadores Asignación

lvalue = valor o expresión

lvalue **op=** valor o expresión # Equivalente a

lvalue = lvalue **op (valor o expresión)**

Resumen | Operadores

- Los operadores aritméticos realizan operaciones matemáticas.
- Los operadores binarios necesitan dos operandos.
- El operador * se utiliza con dos operandos numéricos.
- El operador * se extiende para usarse con una cadena y un entero.
- El operador ** se usa como exponente .
- El operador / realiza divisiones con decimales.
- El operador // realiza divisiones enteras.

Resumen | Operadores

- La prioridad de los operadores es: primero exponentiales, luego operadores de división, módulo o multiplicación, y finalmente sumas y restas.
- La asociatividad de los operadores aritméticos es de izquierda a derecha.
- Se usan los paréntesis () para alterar la jerarquía y asociatividad en las expresiones.
- Los operadores de asignación tienen la menor prioridad.

Resumen | Variables

- Las variables guardan los valores resultantes de las expresiones.
- Las variables hacen que el código sea más fácil de leer.
- Los nombres de las variables son constantes, pero no así el contenido en ellas.
- El tipo de la variable se decide en la ejecución, es decir, su tipado es dinámico.
- Una misma variable puede cambiar de tipo de dato almacenado según se necesite.

Resumen | Variables (Cont.)

- Los caracteres válidos para nombres de variables son letras, dígitos y guion bajo (_).
- No se pueden usar palabras clave (keywords) como nombres de variables.
- Los nombres de las funciones internas como *print* o *input*, no se consideran palabras clave, pero no es recomendable su uso.
- Case *sensitive*, es decir, hace diferencia entre mayúsculas y minúsculas.
- Las variables se inicializan con el operador de asignación (=), y deben de estar inicializadas antes de usarse.

Resumen | Literales & Comentarios

- Las literales en Python al igual que la mayoría de los lenguajes de programación sirven para representar constantes, p. ej. 0, 0.1, 2, 5e-10, True, False, None, "Python 3.0", etc.
- Para escribir comentarios en Python de una sola línea o parte de la línea se usa el carácter **#**.
- Para escribir comentarios en Python de múltiples líneas se usa la triple comilla (""""") o el triple apóstrofo ('').
- Los comentarios son ignorados en la ejecución del programa.

Unidad temática 2

Tipos, operadores, expresiones
y E/S y control de flujos

2.1 Tipos, operadores, expresiones y E/S

Objetivos:

- Listar los operadores bit a bit.
- Usar los operadores bit a bit con tipos de datos enteros.
- Usar los operadores de identidad.
- Usar los operadores relacionales e igualdad.
- Usar los operadores lógicos: and, or y not para construir expresiones booleanas.
- Usar las cadenas de caracteres.
- Usar el operador [] con las cadenas de caracteres.
- Usar las funciones input() y print().
- Usar las funciones de conversión float(), str() e int().
- Usar las funciones len(), count(), min() y max() con cadenas de caracteres.

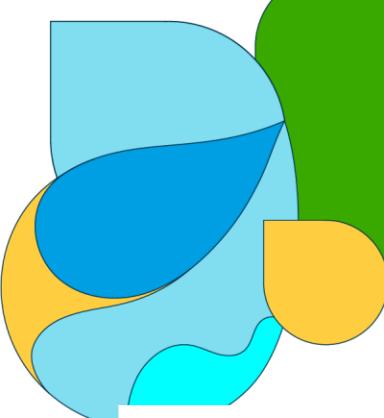
Introducción | Cambio de Base

- ¿Cómo cambiamos de decimal a binario?
- Un algoritmo simple es la división sucesiva entre 2 y obtener los residuos.

$$\begin{array}{r} 2 \longdiv{19} \\ \hline 2 \longdiv{9} \\ \hline 2 \longdiv{4} \\ \hline 2 \longdiv{2} \\ \hline 2 \longdiv{1} \\ \hline 0 \end{array} \quad \text{Residuos}$$

Above the division steps, the word "Residuos" is written in green, with an orange arrow pointing from the word to the first remainder (1).

$$19_{10} = 10011_2$$



Operadores Relacionales

==

a,b=10,10

a==b #True

!=

a,b=10,5

a!=b #True

>

a,b=20,10

a>b #True

>=

a,b=10,10

a>=b #True

<

a,b=10,20

a<b #True

<=

a,b=10,10

a<=b #True

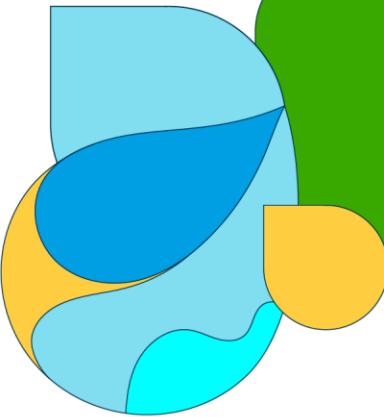
```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> a=3
>>> b=5
>>> print (a==b)
False
>>> print (a!=b)
True
>>> print (a<b)
True
>>> print (a<=b)
True
>>> print (a>=b)
False
>>> print (a>b)
False
>>> 
```

Operadores Lógicos | and

- Si ambos operandos son verdaderos, entonces la condición se vuelve verdadera.

and	True	False
True	True	False
False	False	False

- Cortocircuito, la segunda expresión no será evaluada si la primera expresión decide el resultado final.

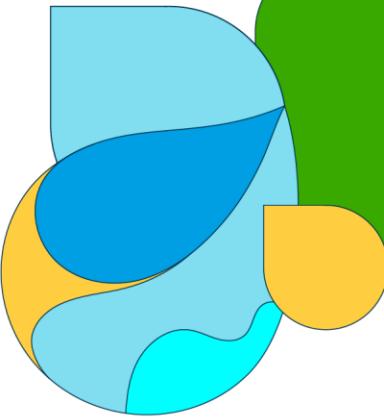


Operadores Lógicos | or

- Si alguno de los dos operandos es verdadero, entonces la condición se vuelve verdadera.

or	True	False
True	True	True
False	True	False

- Cortocircuito, la segunda expresión no será evaluada si la primera expresión decide el resultado final.



Operadores Lógicos | not

- Se usa para invertir el estado lógico de su operando.

not	
True	False
False	True

Operadores Bit a Bit | Binarios

- Los operadores bit a bit se utilizan para comparar números en su representación binaria, es decir, trabajan a nivel de bits.

& (and bit a bit)

&	0	1
0	0	0
1	0	1

| (or bit a bit)

	0	1
0	0	1
1	1	1

^ (xor bit a bit)

^	0	1
0	0	1
1	1	0

Operadores Bit a Bit

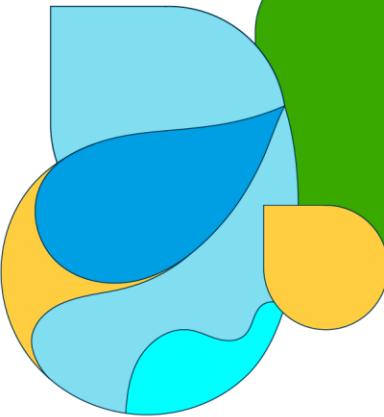
```
>>> a= 5 # a = 0b0101  
>>> b= 12 # b = 0b1100
```

&	0	1
0	0	0
1	0	1

	0	1
0	0	1
1	1	1

^	0	1
0	0	1
1	1	0

a	0	1	0	1
b	1	1	0	0
a & b	0	1	0	0
a b	1	1	0	1
a ^ b	1	0	0	1

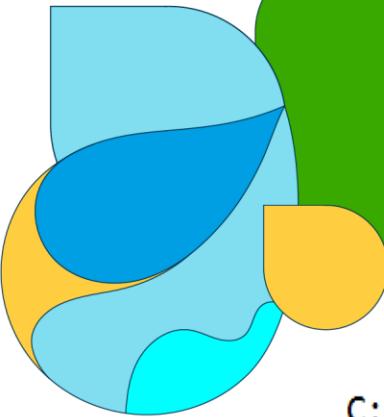


Operadores Bit a Bit | Unarios

- Los operadores bit a bit se utilizan para comparar números en su representación binaria, es decir, trabajan a nivel de bits.

\sim (negación bit a bit)

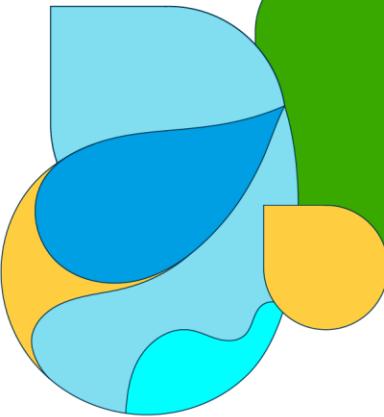
\sim	
0	1
1	0



Operadores Bit a Bit | bin()

```
C:\Users\Administrator\wp_essentials>python -q
>>> a=0b0101
>>> b=0b1100
>>> a
5
>>> b
12
>>> a & b
4
>>> a | b
13
>>> a ^ b
9
>>> ~a
-6
>>> ~-1
0
```

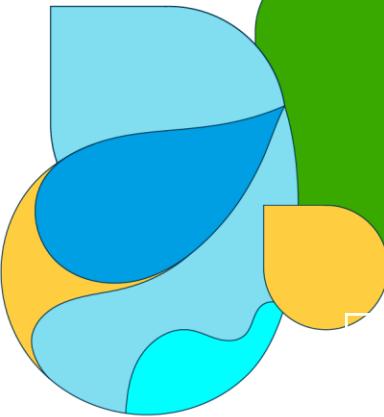
```
C:\Users\Administrator\wp_essentials>python -q
>>> a=5
>>> bin(a)
'0b101'
>>> b=12
>>> bin(b)
'0b1100'
>>> bin(a&b)
'0b100'
>>> bin(a|b)
'0b1101'
>>> bin(a^b)
'0b1001'
>>> bin(~a)
'-0b110'
>>> bin(-1)
'-0b1'
>>> bin(~-1)
'0b0'
```



Operadores Bit a Bit | Desplazamiento

```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> a=0b10
>>> a<<3
16
>>> a<<=3
>>> bin(a)
'0b10000'
>>>
>>> a>>=3
>>> bin(a)
'0b10'
>>> a
2
>>>
```

Desplazamiento binario a la izquierda (<<)
Desplazamiento binario a la derecha (>>)



Operadores de Identidad

is

a=10;b=20

a **is** b #False

a=10;b=10

a **is** b #True

is not

a=10;b=20

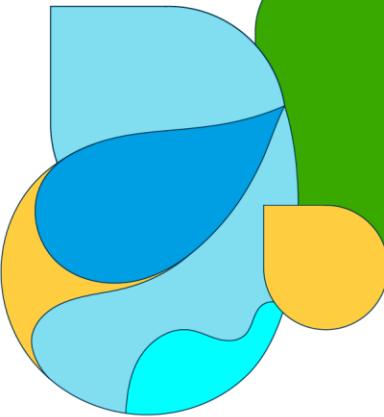
a **is not** b #True

a=10;b=10

a **is not** b #False

Operadores de Identidad | Ejemplo

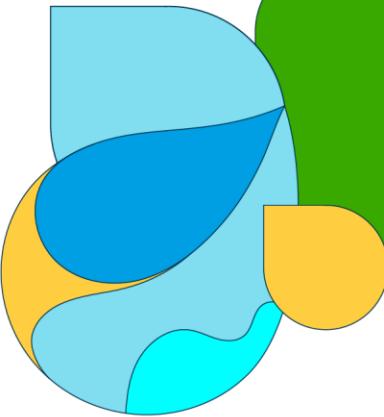
```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>> a= b = 100
>>> a is b
True
>>> b is a
True
>>>
>>> b=200
>>> a is b
False
>>> b is a
False
>>>
>>> b is not a
True
>>>
>>> b not is a
      File "<stdin>", line 1
        b not is a
          ^
SyntaxError: invalid syntax
>>>
```



Expresiones

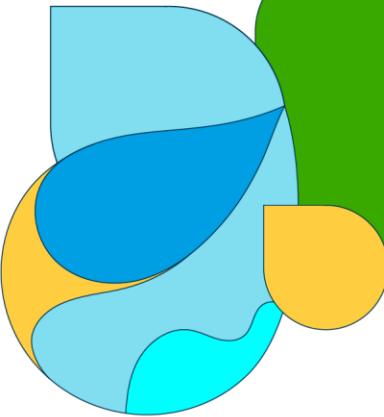
- Las expresiones están compuestas de uno o más operandos, con uno o más operadores.
- ¿Qué es el cortocircuito de una expresión lógica?
- Si la expresión ocurre en un contexto booleano, el resultado se evalúa como booleano.
- Python no necesariamente todos los operandos booleanos son literales True & False, observe las siguientes expresiones:

```
a= 1  
b= 3  
c= 0  
a and b #3 la expresión es True  
b and a #1 la expresión es True  
a and c #0 la expresión es False  
c and a #0 la expresión es False
```



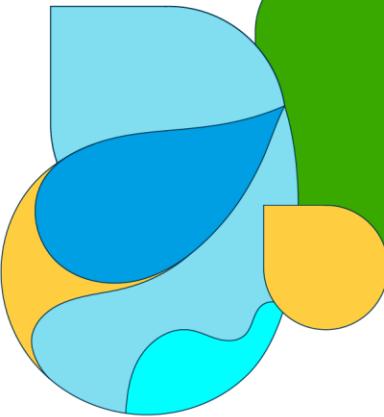
Expresiones | Ejemplos | and

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> a=1
>>> b=3
>>> c=0
>>>
>>> a and b
3
>>> b and c
0
>>> a and c
0
>>> c and a
0
>>> 5 and "A"
'A'
>>> "A" and 5
5
>>> "" and 0
''
>>> 0 and ""
0
>>> -
```



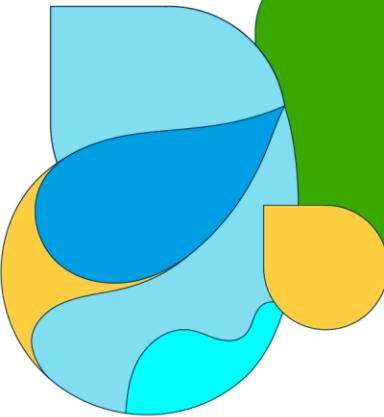
Expresiones | Ejemplos | or

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> x=0 ; y=1; z=2
>>> x or y
1
>>> y or x
1
>>> y or z
1
>>> x or 0
0
>>> 0 or "a"
'a'
>>> 'b' or 0
'b'
>>> '' or 0
0
```



Expresiones | Ejemplos | not

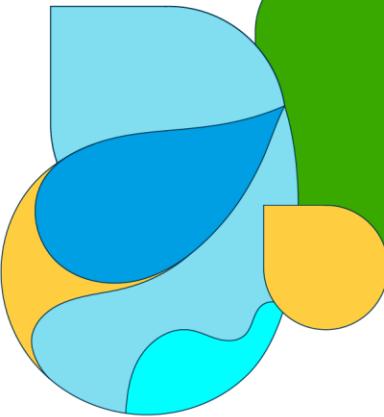
```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> a= b = c= 0
>>> not a
True
>>> not not a
False
>>>
>>> not a and not b
True
>>> not ""
True
>>> not " "
False
>>> not 'Python'
False
>>> _
```



Funciones de Entrada y Salida

- Función de salida: print()

```
C:\Users\Administrator>python -q
>>>
>>> print ("H","o","l","a","!")
H o l a !
>>> print("Hola")
Hola
>>> print ("Hola Mundo")
Hola Mundo
>>>
>>> print ("Hola","Curso de Python",1,2,3, 'a')
Hola Curso de Python 1 2 3 a
>>> -
```

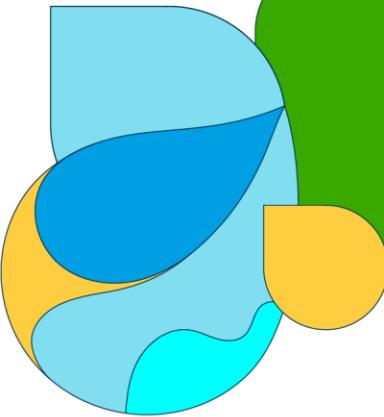


Funciones de Salida | print()

- Función de salida: print(), argumentos nombrados
 - sep & end
 - sep: separador de campos, por defecto es un espacio en blanco.
 - end: separador de líneas, por defecto es un cambio de línea.

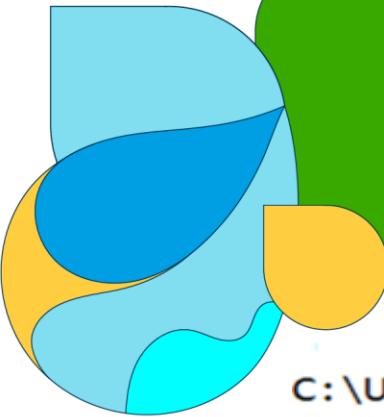
```
print("H","o","l","a","!")
```

```
print("H","o","l","a","!", sep=" ", end="\n")
```



print() | Ejemplos

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> print ("Este", "es", "un", "mensaje", "!")
Este es un mensaje !
>>>
>>> print ("Este", "es", "un", "mensaje", "!", sep="-")
Este-es-un-mensaje-
>>>
>>> print ("Este", "es", "un", "mensaje", "!", sep="")
Este es un mensaje !
>>>
>>> print ("Este", "es", "un", "mensaje", "!", sep=" ", end="FIN")
Este es un mensaje !FIN>>>
>>>
>>> print ("Este", "es", "un", "mensaje", "!", sep="\n", end="FIN\n")
Este
es
un
mensaje
!FIN
>>> _
```



Funciones de Entrada | input()

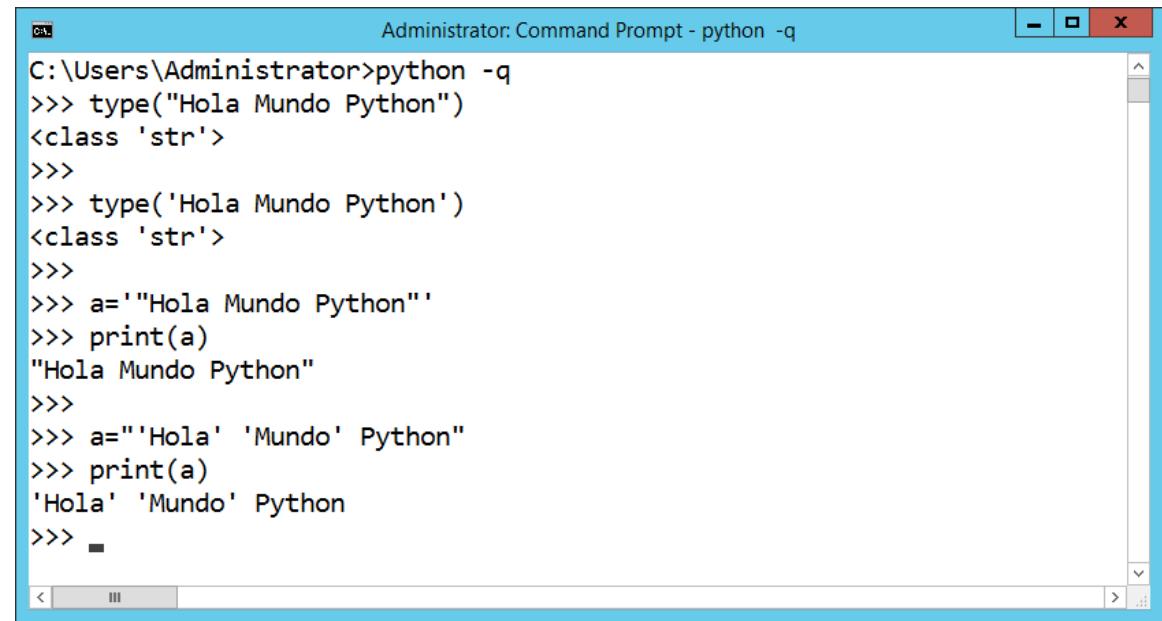
```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> nombre=input("¿Cuál es su nombre? ")
¿Cuál es su nombre? Everardo
>>>
>>> type(nombre)
<class 'str'>
>>>
>>> precio=input("¿Cuál es el precio del producto? ")
¿Cuál es el precio del producto? 99.99
>>>
>>> type(precio)
<class 'str'>
>>> float(precio)
99.99
>>> precio=float(precio)
>>> type(precio)
<class 'float'>
>>> _
```

- Función de entrada: `input()`
 - El argumento de la función sirve como un "prompt".
 - El valor de retorno siempre es una cadena de caracteres.
 - Existen funciones de conversión como `int()`, `float()`, etc.

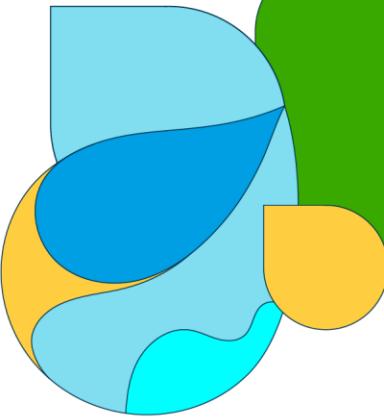
Cadenas

- Las cadenas de caracteres o simplemente cadenas son secuencias de caracteres delimitadas por comillas dobles o apóstrofos.

"Hola Mundo Python"
'Hola Mundo Python'
"Hola Mundo 'Python' "
'Hola Mundo "Python" '



```
C:\Users\Administrator>python -q
>>> type("Hola Mundo Python")
<class 'str'>
>>>
>>> type('Hola Mundo Python')
<class 'str'>
>>>
>>> a='''Hola Mundo Python'''
>>> print(a)
"Hello Mundo Python"
>>>
>>> a="''Hello' 'Mundo' Python"
>>> print(a)
'Hello' 'Mundo' Python
>>>
```



Cadenas | Secuencias de Escape

Carácter
de escape
\

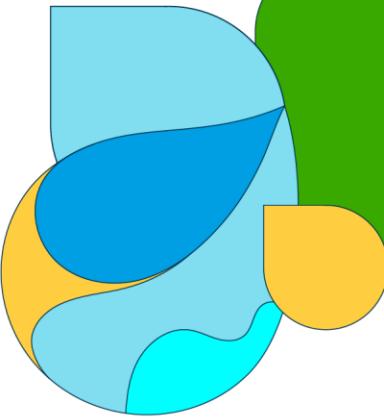
\'	Apostrofó simple
\\"	Diagonal invertida
\n	Cambio de línea
\r	Regreso de carro
\\"	Comilla doble
\f	Cambio de hoja
\a	Campana de la máquina
\b	Backspace
\t	Tabulador
\v	Tabulador vertical

Cadenas | Comillas Triples

```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>>
>>> """ Aquí se puede poner lo que sea, por lo general
... se usar para la documentación, pero lo que cuenta es:
... \n cambios de línea
... \t tabulaciones
...
... " ' @ # Simbolos se quedan hasta nuevamente poner tres
... comillas dobles
...
' Aquí se puede poner lo que sea, por lo general\nse usar para la documentación, pero lo que cuenta es:\n\n
>>>
```

Las comillas triples se utilizan para crear cadenas de varias líneas.

Se almacenan las secuencias de escape

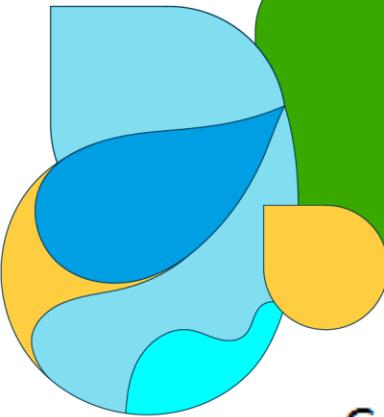


Cadenas | Indexación

- La indexación es el proceso de encontrar un elemento específico dentro de una secuencia de elementos a través de la posición del elemento.

H	o	l	a		M	u	n	d	o
0	1	2	3	4	5	6	7	8	9
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Índices



Cadenas | Indexación | Ejemplos

```
C:\Users\Administrator\wp_essentials>python string_indexing.py
Hola Mundo
0123456789
Primero H
Ultimo o
Tercero a
Tercero inverso n
```

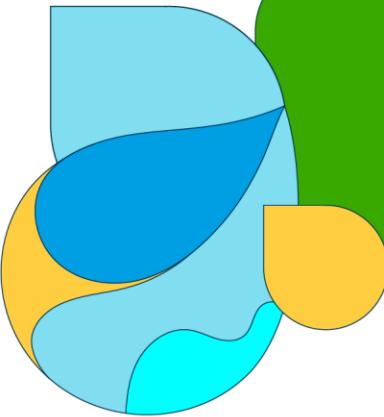
```
mensaje = "Hola Mundo"
print(mensaje)
print("0123456789")

primero = mensaje[0] # [H]ola Mundo
print("Primero", primero)

ultimo = mensaje[-1] #Hola Mund[o]
print("Ultimo", ultimo)

tercero = mensaje[3] #Hol[a] Mundo
print("Tercero", tercero)

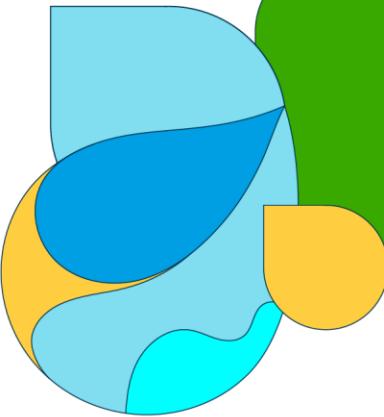
tercero_desde_final = mensaje[-3] #Hola Mu[n]do
print("Tercero inverso", tercero_desde_final)
```



Cadenas | Cortes | []

- A los corchetes cuadrados [] se les conoce como operador de slice o corte.
- Formas sintácticas:

```
secuencia[índice] #caracter específico
secuencia[inicio:] #de inicio al fin de cadena
secuencia[inicio:fin] #de inicio al fin - 1 de la cadena
secuencia[:fin] #desde 0 al fin - 1 de la cadena
secuencia[inicio:fin:paso] #de inicio al fin - 1 de paso en paso
secuencia[:] #todos los caracteres de la cadena
```



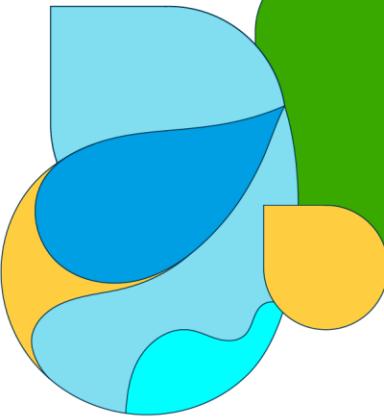
Cadenas | Cortes | Ejemplos

```
men = "Guillermo Tell"  
print(men)  
print("01234567890123")
```

```
primeros5 = men[0:5] #Los caracteres entre 0 y 5-1, es decir, 0 y 4, 'Guill'  
print("Guillermo Tell'[0:5]", primeros5)
```

```
del2al4 = men[1:4] #Los caracteres entre 1 y 4-1, es decir, 0 y 3, 'uil'  
print("Guillermo Tell'[1:4]", del2al4)
```

```
del5alfinal = men[4:] #Los caracteres desde la posición 4 al final  
print("Guillermo Tell'[4:]",del5alfinal)
```



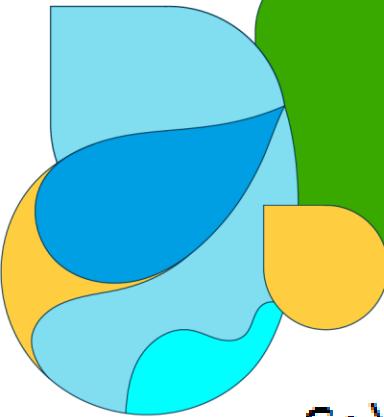
Cadenas | Cortes | Ejemplos (Cont.)

```
los3ultimos = men[-3:] #Los caracteres desde la posición -3 al final de la cadena
print("Guillermo Tell'[-3]",los3ultimos)
```

```
losprimeros3= men[:3] #Los caracteres entre 0 y 2, 'Gui'
print("Guillermo Tell'[:3]",losprimeros3)
```

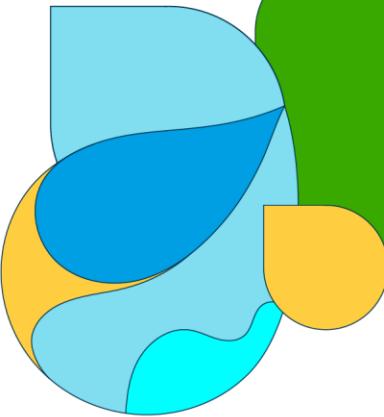
```
losultimos3antesdelfinal = men[-4:-1] #Los caracteres entre -4 y -2
print("Guillermo Tell'[-4:-1]",losultimos3antesdelfinal)
```

```
copia = men[:] #Los caracteres entre 0 y el fin de la cadena
print("Guillermo Tell'[:]",copia)
```



Cadenas | Cortes | Ejemplos (Cont.)

```
C:\Users\Administrator\wp_essentials>python string_slicing.py
Guillermo Tell
01234567890123
'Guillermo Tell'[0:5] Guill
'Guillermo Tell'[1:4] uil
'Guillermo Tell'[4:] lermo Tell
'Guillermo Tell'[-3] ell
'Guillermo Tell'[:3] Gui
'Guillermo Tell'[-4:-1] Tel
'Guillermo Tell'[:] Guillermo Tell
```



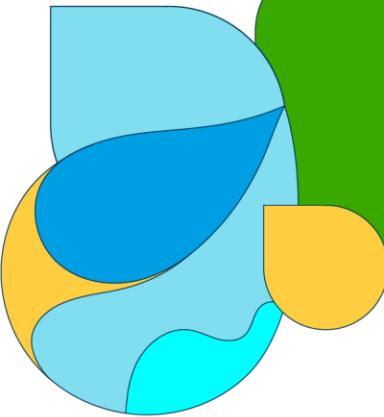
Concatenación | +

- La concatenación es una palabra elegante para unir cadenas de caracteres.
- Python usa el operador +

```
C:\Users\Administrator\wp_essentials>py cual_es_tu_nombre.txt
¿Cuál es tu nombre? Everardo
Hola, Everardo!
```

```
C:\Users\Administrator\wp_essentials>
```

```
nombre = input("¿Cuál es tu nombre? ")
mensaje = "Hola, " + nombre + "!"
print(mensaje)
```



Repetición | *

- La repetición es el proceso de concatenar una cadena varias veces.
- Python usa el operador *

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> palabra="Bla"
>>> dialogo= palabra * 10
>>> print(dialogo)
BlaBlaBlaBlaBlaBlaBlaBlaBlaBla
>>> ■
```

Concatenación & Repetición

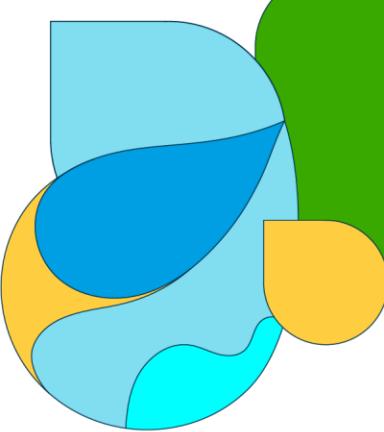
"a" + "b" * 3 + "c"

"abbbc"

("a" + "b") * 3 + "c"

"abababc"

Primero se realiza la expresión entre paréntesis



Cadenas | Métodos

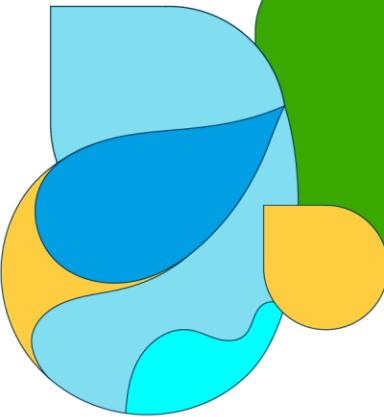
```
capitalize "hola".capitalize() # "Hola"
```

```
lower "HOLA".lower() # "hola"
```

```
swapcase "hoLA".swapcase() # "Hola"
```

```
title "hola mundo".title() # "Hola Mundo"
```

```
upper "holA".upper() # "HOLA"
```



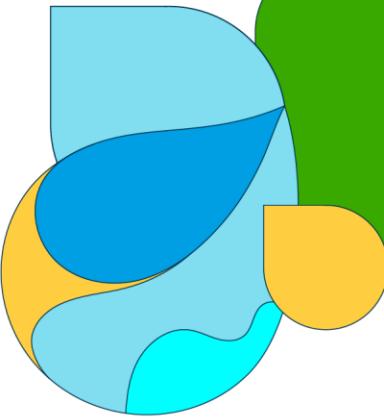
Cadenas | Métodos (Cont.)

replace(viejo,nuevo,[cuenta]) "Hola".replace("H","X") # "Xola"

strip([caracteres]) "xxxxxx".strip("x") # "xxx"

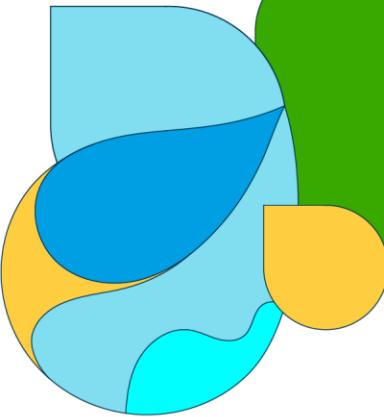
lstrip ([caracteres]) " hola ".lstrip() # "ola "

rstrip ([caracteres]) " hola ".rstrip() # " hola "



Cadenas | Métodos (Cont.)

isalnum	"¡Hola, mundo!".isalnum() #False
startswith(prefijo)	"hola".startswith("h") #True
endswith (sufijo)	"hola".endswith("a") #True
isnumeric	"5.5".isnumeric() #False
isdecimal	"5.5" .isdecimal() #False
isspace	" ".isspace() #True



Cadenas | isdigit & isdecimal

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> c='\u00B2'
>>> c.isdecimal()
False
>>> c.isdigit()
True
>>> print(c)
2
>>> c == 2
False
>>> ■
```

Diferencia entre
isdecimal() &
isdigit()

count (sub [,inicio [,fin]])

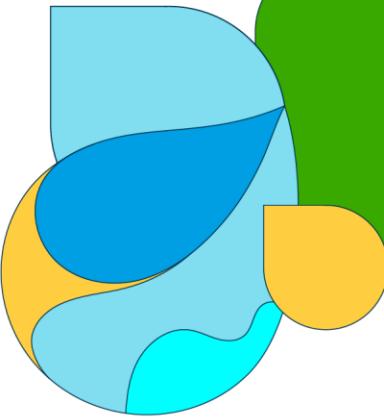
```
"Hola Mundo!".count("o") #2
```

```
"Hola Mundo!".count("o",2) #1
```

```
"Hola Mundo!".count("o",2,7) #0
```

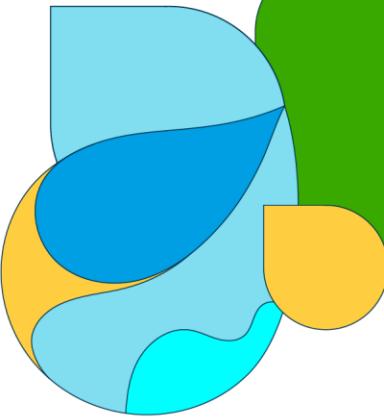
Cadenas | count() | Ejemplos

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> "oooxxooo".count("o")
6
>>> "oooxxooo".count("o",1)
5
>>> "oooxxooo".count("o",1,4)
2
>>> "oooxxooo".count("x",1,4)
1
>>> "oooxxooo".count("x")
2
>>> "oooxxooo".count("y")
0
>>> "oooxxooo".count("X")
0
>>>
```



Cadenas | Funciones de Conversión

```
str('México') #El objeto ya es una cadena. Devuelve 'México'  
str(666) #El objeto es entero. Devuelve '666'  
str(math.pi) #El objeto es flotante. Devuelve '3.141592653589793'
```



Cadenas | len(), min() & max()

```
len('México') #Devuelve 6
```

```
len("") #Devuelve 0, longitud de la cadena vacía
```

```
len(' ') #Devuelve 1, la cadena tiene un único espacio en blanco
```

- len()
- min()
- max()

```
min('w', 'e', 'b') #Devuelve 'b'
```

```
min('a', 'B', 'c') #Devuelve 'B'
```

```
max('w', 'e', 'b') #Devuelve 'w'
```

```
max('a', 'B', 'c') #Devuelve 'c'
```

Resumen

- Saber cómo manipular cadenas de caracteres, juega un papel fundamental en la mayoría de las tareas de procesamiento de texto.
- Las cadenas de caracteres son objetos de Python inmutables: p. ej.
`a="Hola"; a.replace("H","X") == "Xola"; a=="Hola" #True en ambos casos`
- Cuando los dos operandos del operador + son cadenas, el resultado es la concatenación de los operandos: "Curso" + "Python" = "CursoPython".
- El operador + cuando opera con cadenas de caracteres no es una operación commutativa.

Resumen (Cont.)

- El operador ***** con cadena y un entero, repite tantas veces la cadena como el operando entero de la derecha indique. "H"*3 = "HHH" y 3*"H" = "HHH".
- El operador ***** con cadena y un entero genera la cadena vacía si el entero es menor o igual a cero.
- El operador **[]** es un operador de indexación, que puede aceptar valores enteros positivos o negativos, p. ej. "aeiou"[0] == "a" and "aeiou"[-1]=="u"
- Los operadores booleanos se utilizan en una expresión booleana para devolver **True** o **False**.

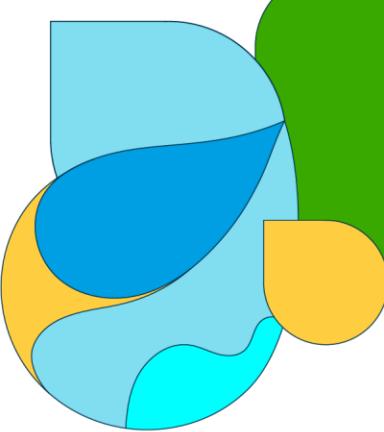
Resumen (Cont.)

- Los operadores booleanos de Python son: `and`, `or` y `not`.
- Python tiene operadores para operar bit a bit, los operadores son: `&`, `|`,
`^`, `~`, `>>` y `<<`
- Los operadores de comparación se utilizan para comparar dos valores de operandos.
- A los operadores de comparación también se les conoce como operadores relacionales.
- Los operadores relacionales de Python son: `<`, `>`, `<=`, `>=`, `==` y `!=`

2.2 Control de flujo

Objetivos:

- Usar las estructuras condicionales: if, if-else, elif.
- Usar las estructuras de ciclo: for y while.
- Identificar el uso de rangos.
- Usar la cláusula else en las estructuras de ciclo.



Introducción

Línea de código
Línea de código
Línea de código
Línea de código



switch & case

If, if-else & elif

for & while

Declaraciones Condicionales | if

Identación
o Sangría

```
if expresión_booleana:  
    cod1()  
    cod2()  
cod3()
```

Inicio de
bloque

Bloque de código,
si la expresión
booleana es True

Fin del bloque.
Instrucción fuera
del bloque del if

Declaraciones Condicionales | if - else

Identación o
Sangría

Inicio del
bloque else

Identación o
Sangría

if expresión_boleana:

[cod1()
cod2()]

else:

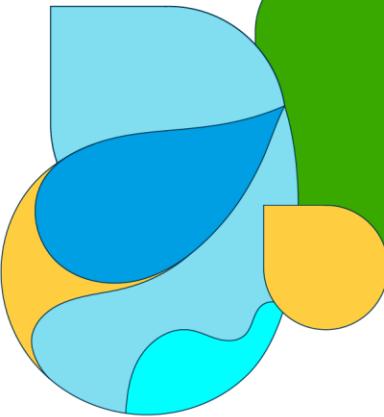
[cod3()
cod4()]

Inicio de bloque

Bloque de código, si la
expresión booleana es True

Bloque de código, si la
expresión booleana es False

Fin del bloque. Instrucción fuera del bloque del if-else



Declaraciones Condicionales |

if - elif- else

Identación o
Sangría

Fin del bloque.
Instrucción fuera
del bloque

if expresión_booleana:

cod1()
cod2()

Inicio de bloque

Sí la expresión booleana
es True

elif otra_expresión_booleana:

cod3()
cod4()

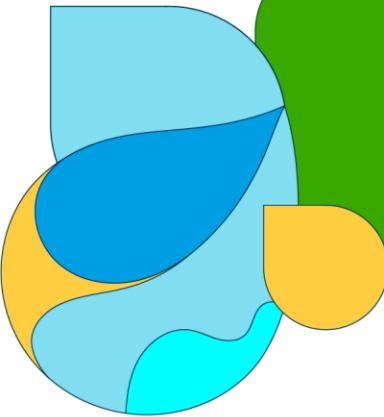
Sí la expresión booleana
es True

else:

cod5()
cod6()

cod7()

Sí la otra_expresión_booleana
es False



Declaraciones Condicionales | Ejemplo

```
edad = int(input('¿Cuántos años tiene? '))

if edad >= 21:
    print('Ahora puede votar y beber alcohol en USA.')
elif edad >= 18:
    print('En USA, ahora puede votar, pero no puede beber alcohol.')
else:
    print('En USA, no puede votar ni beber alcohol.')
```

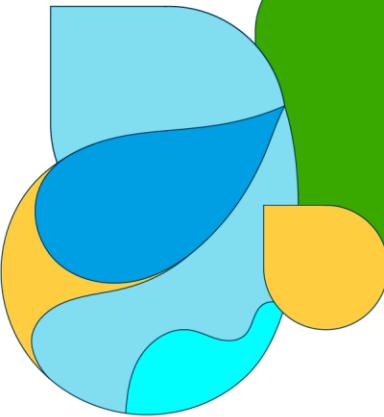
Declaraciones Condicionales | Salida

```
C:\> Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py if.py
¿Cuántos años tiene? 10
En USA, no puede votar ni beber alcohol.

C:\Users\Administrator\wp_essentials>py if.py
¿Cuántos años tiene? 20
En USA, ahora puede votar, pero no puede beber alcohol.

C:\Users\Administrator\wp_essentials>py if.py
¿Cuántos años tiene? 30
Ahora puede votar y beber alcohol en USA.

C:\Users\Administrator\wp_essentials>
```



Declaraciones Condicionales | Ejemplo

```
edad = int(input('¿Cuántos años tienes? '))

if input('¿Eres ciudadano americano? [S|N] ').lower() == 's':
    is_citizen = True
else:
    is_citizen = False

if edad >= 21 and is_citizen:
    print('Puedes votar y beber alcohol')
elif edad >= 21:
    print('Puede beber alcohol, pero no puedes votar.')
elif edad >= 18 and is_citizen:
    print('Puedes votar, pero no puedes beber alcohol.')
else:
    print('No puedes votar o beber alcohol')
```

Declaraciones Condicionales | Salida

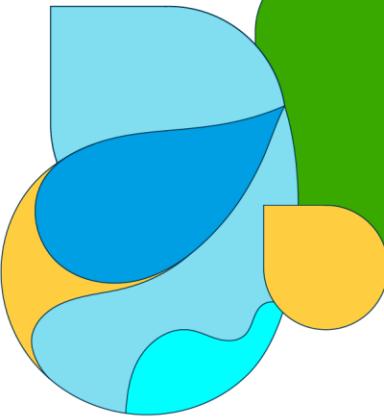
```
Administrator: Command Prompt

C:\Users\Administrator\wp_essentials>py if2.py
¿Cuántos años tienes? 20
¿Eres ciudadano americano?[S|N] S
Puedes votar, pero no puedes beber alcohol.

C:\Users\Administrator\wp_essentials>py if2.py
¿Cuántos años tienes? 10
¿Eres ciudadano americano?[S|N] N
No puedes votar o beber alcohol

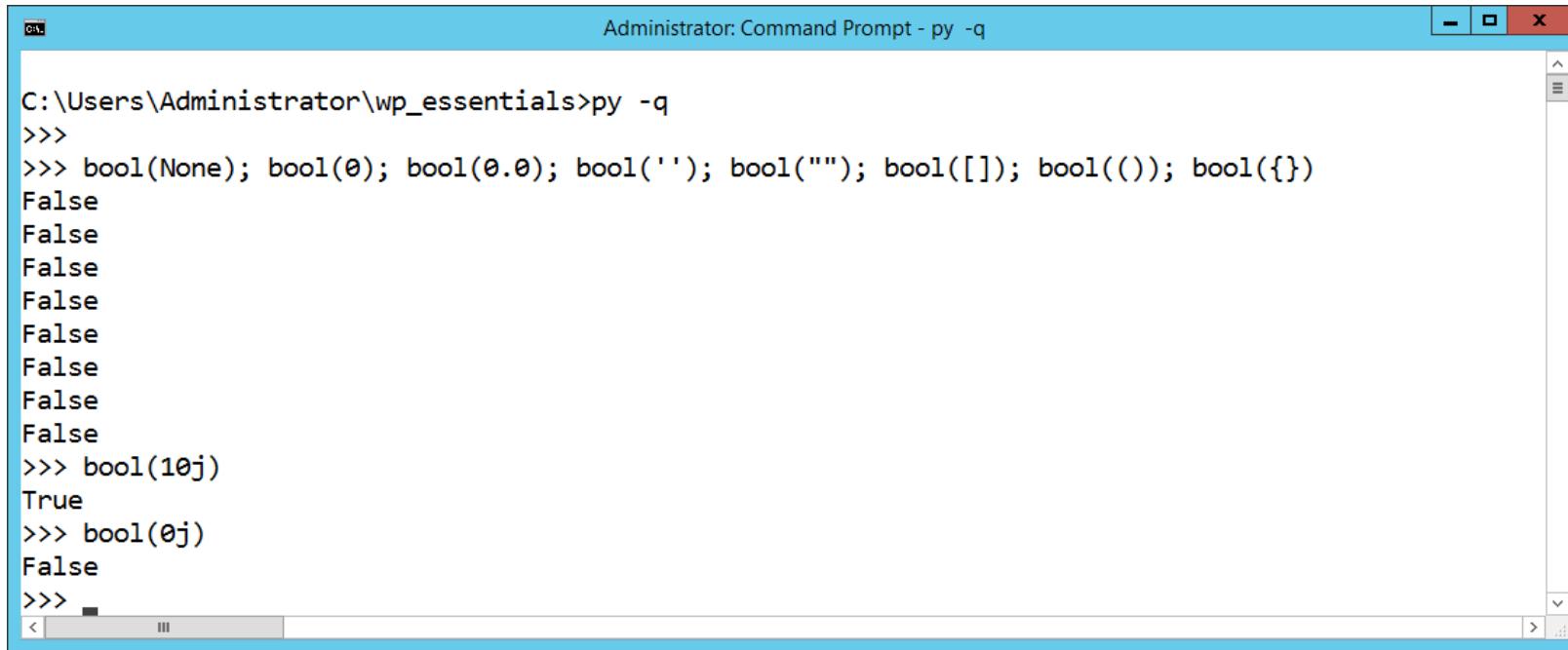
C:\Users\Administrator\wp_essentials>py if2.py
¿Cuántos años tienes? 30
¿Eres ciudadano americano?[S|N] S
Puedes votar y beber alcohol

C:\Users\Administrator\wp_essentials>
```

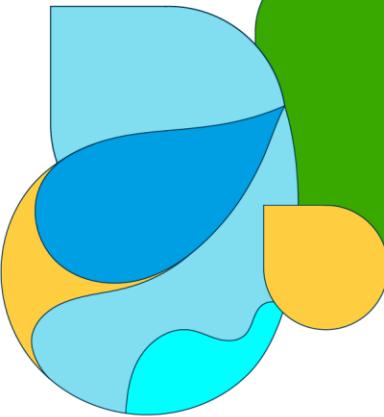


Expresiones Falsas

- None, 0, 0.0, "", "", [], (), () y 0j



```
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> bool(None); bool(0); bool(0.0); bool(''); bool(""); bool([]); bool(()); bool({})
False
False
False
False
False
False
False
False
>>> bool(10j)
True
>>> bool(0j)
False
>>>
```



Rangos

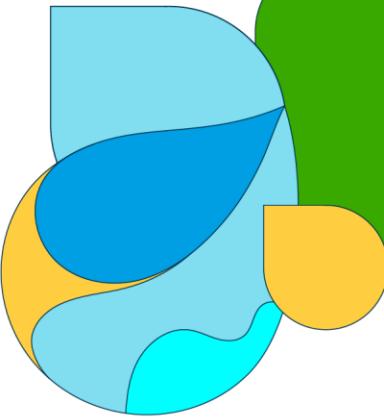
- Secuencia inmutable de números que a menudo se usan para declaraciones o estructuras de ciclo.
- Función preconstruida Python: range()
- Sintaxis:

`range(final) # [0,final]`

`range(inicio, final) # [inicio,final]`

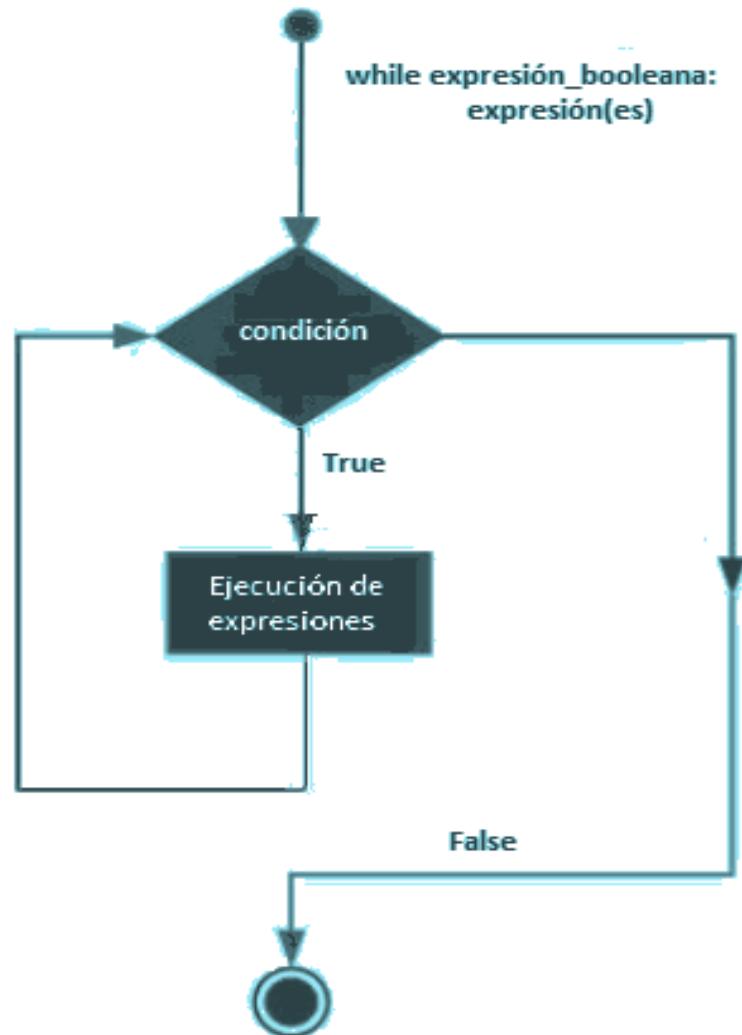
`range(inicio, final, paso) # [inicio,final] de paso en paso`

Rangos | Ejemplos



```
C:\> Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>py -q
>>> range(5) #Crea un rango iniciando en 0 y finalizando en 4
range(0, 5)
>>> range(5,10) #Crea un rango iniciando en 0 y finalizando en 9
range(5, 10)
>>> range(5,21,5) #Crea un rango iniciando en 0 y finalizando en 20, de cinco en cinco
range(5, 21, 5)
>>> range(5,-5) #Crea un rango vacío
range(5, -5)
>>> range(5,-5,-1) #Crea un rango iniciando en 5 y finalizado en en -4
range(5, -5, -1)
>>>
```

Declaraciones de Ciclo | while



while | Sintaxis

Identación
o Sangría

Inicio de bloque
del while

```
while expresión_booleana:  
    línea de código  
    línea de código  
    línea de código
```

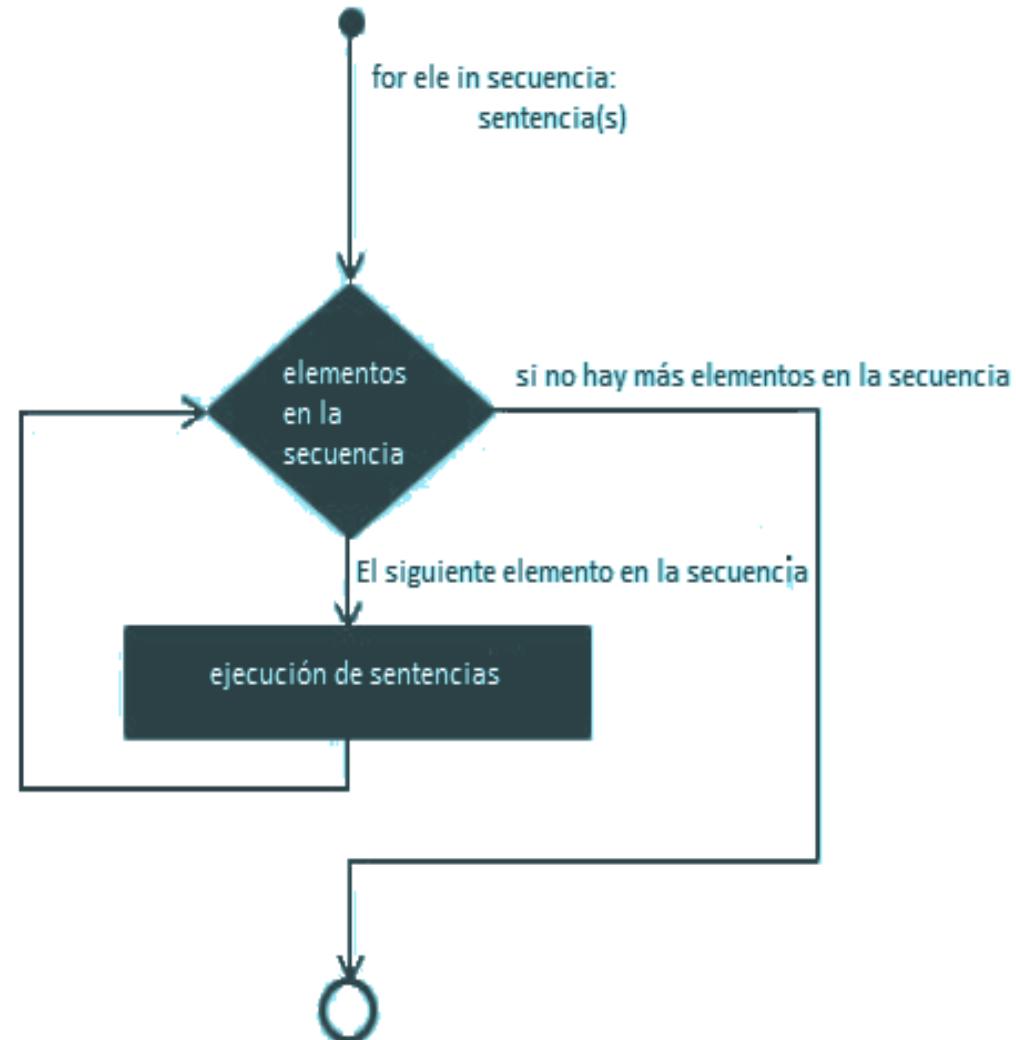
Fin del bloque
while. Instrucción
fuera del bloque

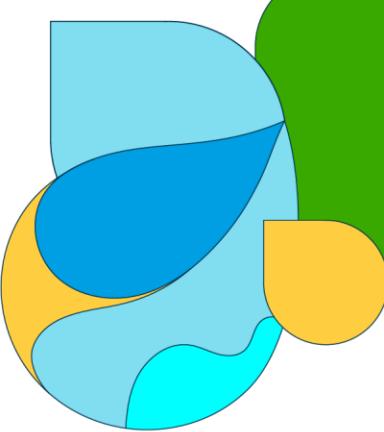
Sí la expresión
booleana es True

while | Ejemplo

```
C:\Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> num=0
>>> while num < 5:
...     print("\t",num)
...     num += 1
...
    0
    1
    2
    3
    4
>>> -
```

Declaraciones de Ciclo | for





for | Sintaxis

`for variable in <valores o rango o cadenas de caracteres>:`

`pass #keyword se sustituye por el bloque de código
 controlado por el for`

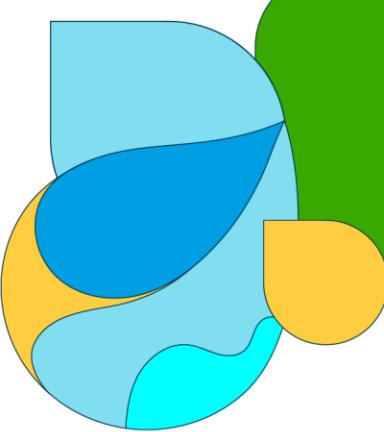


Identación
o Sangría



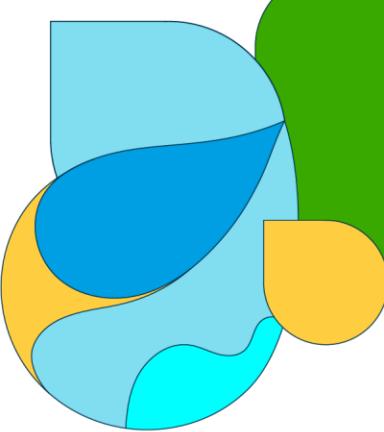
Inicio de bloque del while

for | Ejemplo 1



```
C:\> Administrator: Command Prompt - python -q
>>> for i in 1,2,3,4,5:
...     print(i)
...
1
2
3
4
5
>>> for i in "abcd":
...     print(i)
...
a
b
c
d
>>>
KeyboardInterrupt
>>>
```

for | Ejemplo 2



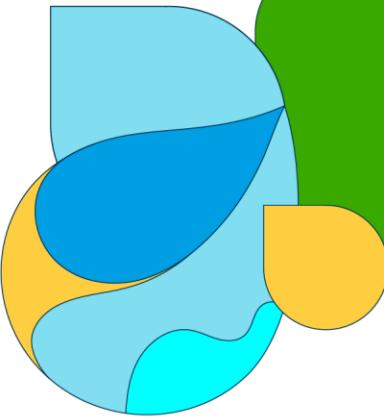
```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for num in range(5):
...     print (" ", num)
...
0
1
2
3
4
>>> for num in range(0,3):
...     print (" ", num)
...
0
1
2
>>>
```

for | Ejemplo 3

```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for v in 'a','e','i', 10, 20:
...     print (" ", v)
...
a
e
i
10
20
>>> for v in "PYTHON":
...     print (" ", v)
...
P
Y
T
H
O
N
>>>
```

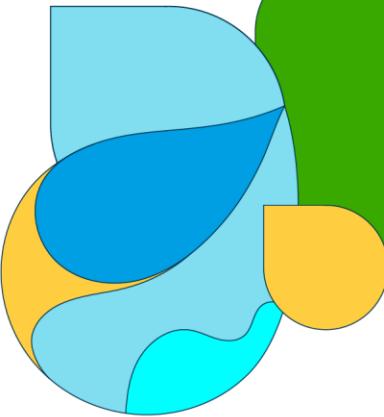
for | Ejemplo 4

```
Administrator: Command Prompt - python -q
>>> for num in range (1,11,2):
...     print(" ", num)
...
1
3
5
7
9
>>> for num in range (5,0,-1):
...     print(" ", num)
...
5
4
3
2
1
>>>
```



Ciclos Anidados

- Cuando dentro del bloque de una instrucción `for` o `while` nuevamente se tiene una instrucción `for` o `while` se conoce como ciclos anidados.
 - Sirven para procesar volúmenes grandes de información.
 - Son útiles en combinatoria.



Ciclos Anidados | Ejemplo 1

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>> for i in range (1,7):
...     for j in range (1,7):
...         print("(",i,",",",",j,")", end="")
...     print()
...
( 1 , 1 )( 1 , 2 )( 1 , 3 )( 1 , 4 )( 1 , 5 )( 1 , 6 )
( 2 , 1 )( 2 , 2 )( 2 , 3 )( 2 , 4 )( 2 , 5 )( 2 , 6 )
( 3 , 1 )( 3 , 2 )( 3 , 3 )( 3 , 4 )( 3 , 5 )( 3 , 6 )
( 4 , 1 )( 4 , 2 )( 4 , 3 )( 4 , 4 )( 4 , 5 )( 4 , 6 )
( 5 , 1 )( 5 , 2 )( 5 , 3 )( 5 , 4 )( 5 , 5 )( 5 , 6 )
( 6 , 1 )( 6 , 2 )( 6 , 3 )( 6 , 4 )( 6 , 5 )( 6 , 6 )
>>> -
```

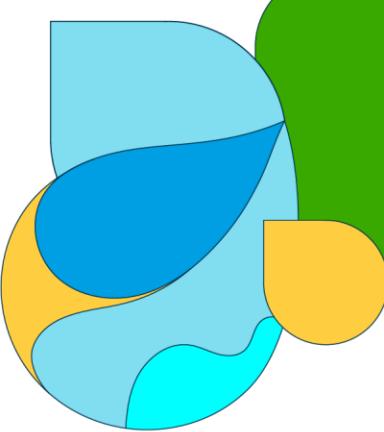
Ciclos Anidados | Ejemplo 2

```
Administrator: Command Prompt - python -q
>>>
>>> i= j = 1
>>> while i<7:
...     while j<7:
...         print("( ,i, , ,j, )", end="")
...         j+=1
...     j= 1
...     i+= 1
...     print()
...
( 1 , 1 )( 1 , 2 )( 1 , 3 )( 1 , 4 )( 1 , 5 )( 1 , 6 )
( 2 , 1 )( 2 , 2 )( 2 , 3 )( 2 , 4 )( 2 , 5 )( 2 , 6 )
( 3 , 1 )( 3 , 2 )( 3 , 3 )( 3 , 4 )( 3 , 5 )( 3 , 6 )
( 4 , 1 )( 4 , 2 )( 4 , 3 )( 4 , 4 )( 4 , 5 )( 4 , 6 )
( 5 , 1 )( 5 , 2 )( 5 , 3 )( 5 , 4 )( 5 , 5 )( 5 , 6 )
( 6 , 1 )( 6 , 2 )( 6 , 3 )( 6 , 4 )( 6 , 5 )( 6 , 6 )
>>>
```

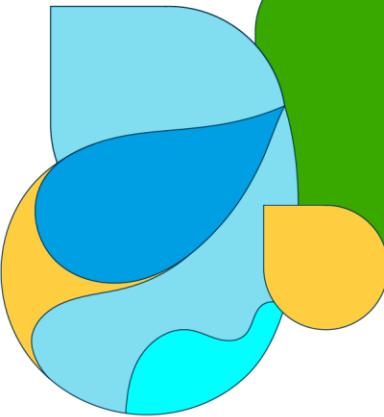
Sentencia break

- Para salir de un ciclo se puede salir de dos formas posibles:
 - Iterar un número controlado de veces.
 - Iterar y mediante una condición usar la opción **break**, esta última instrucción saldrá del ciclo.

Sentencia break | Ejemplo



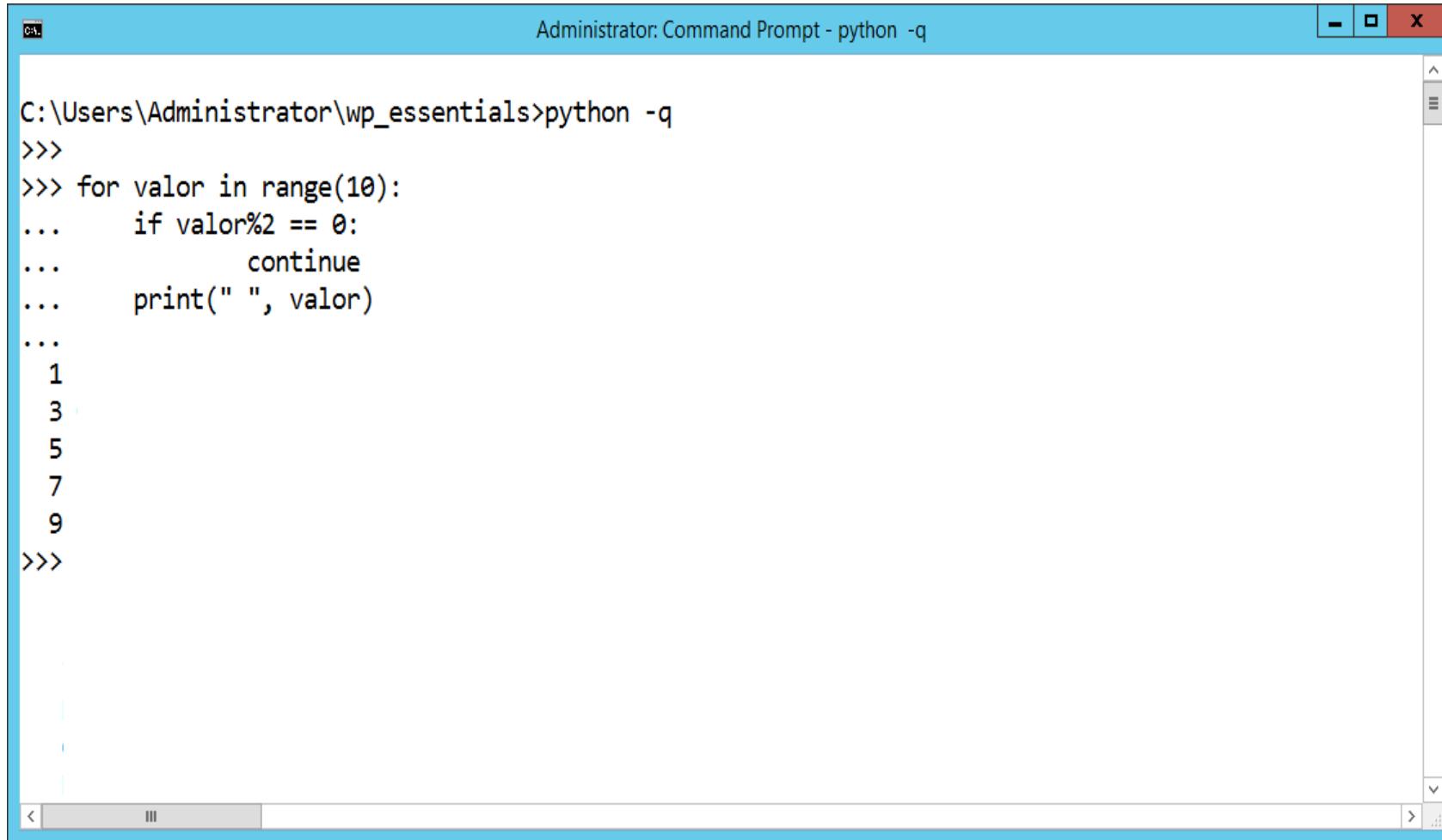
```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for valor in range(8):
...     print(" ", valor)
...     if (valor > 3):
...         break
...
0
1
2
3
4
>>> -
```



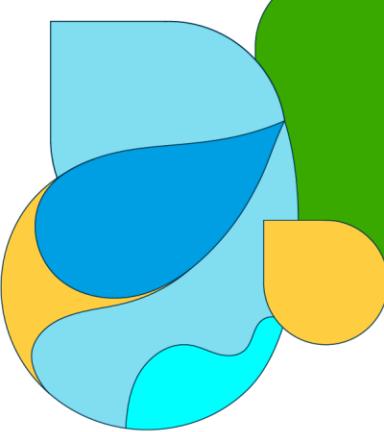
Sentencia continue

- Para "saltar" a la siguiente iteración de un ciclo sin ejecutar las declaraciones restantes en el bloque, se usa la declaración `continue`.
- Tanto la instrucción `break` como la instrucción `continue` solo pueden usarse en el contexto de ciclos.

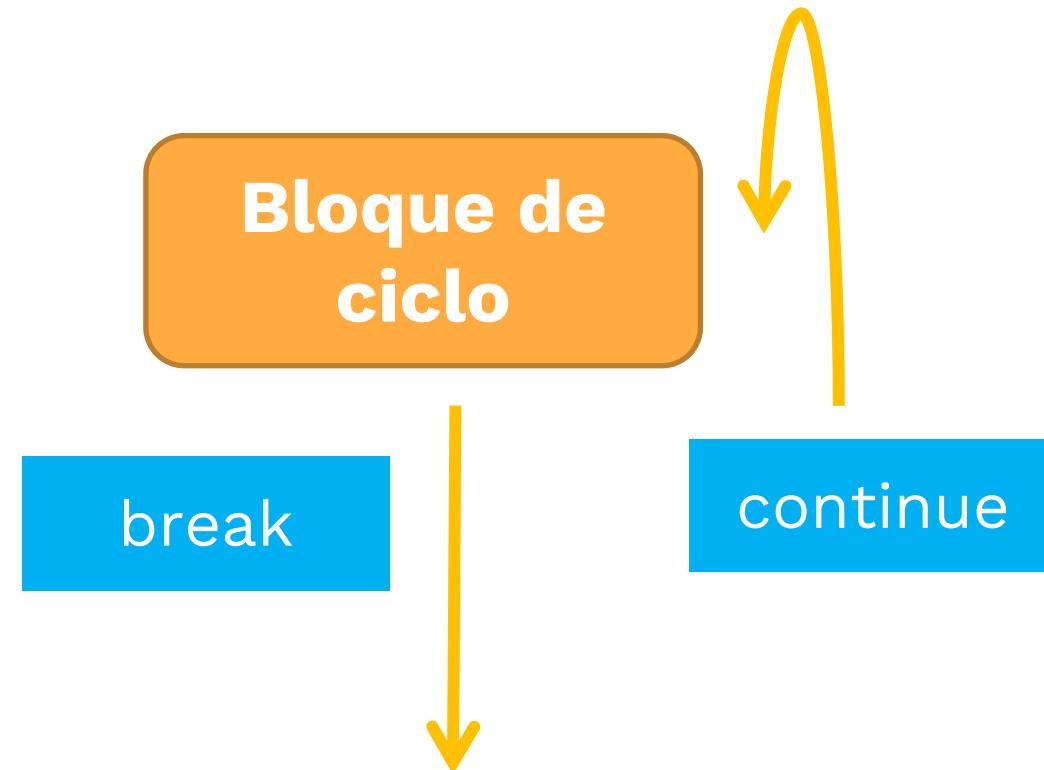
Sentencia continue | Ejemplo

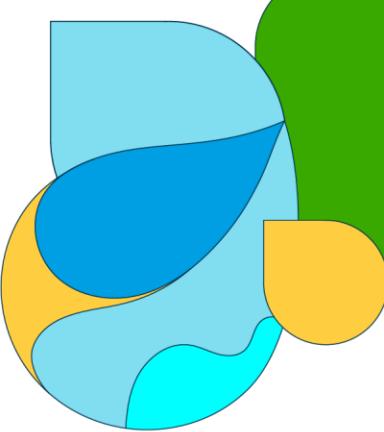


```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for valor in range(10):
...     if valor%2 == 0:
...         continue
...     print(" ", valor)
...
1
3
5
7
9
>>>
```



break vs. continue

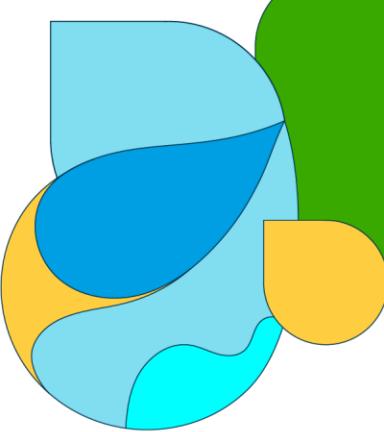




Cláusula else

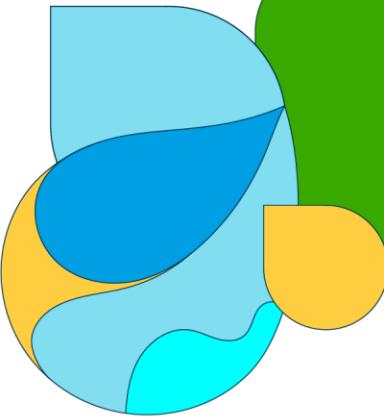
- La cláusula `else` puede usarse también con ciclos.
 - `if-else`, `if-elif-else`
- Se ejecuta después del que el ciclo (`while` o `for`) haya completado con éxito las iteraciones.
- Sin interrupción o haber salido con la sentencia `break`.

Cláusula else | Ejemplo 1



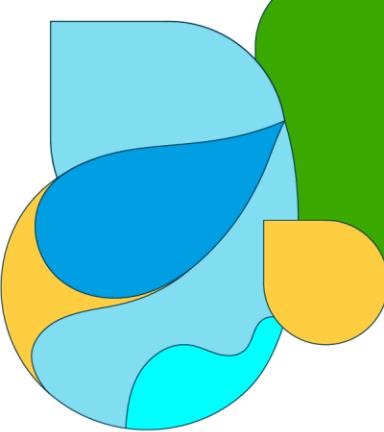
```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for i in range(4):
...     print(' ', i)
... else:
...     print(' Ciclo completado')
...
0
1
2
3
Ciclo completado
>>> -
```

Cláusula else | Ejemplo 2



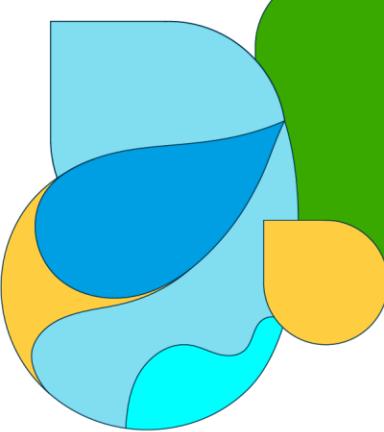
```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> for i in range(4):
...     if i == 3:
...         break
...     print(" ", i)
... else:
...     print("Ciclo completado")
...
0
1
2
>>>
```

Cláusula else | Ejemplo 3

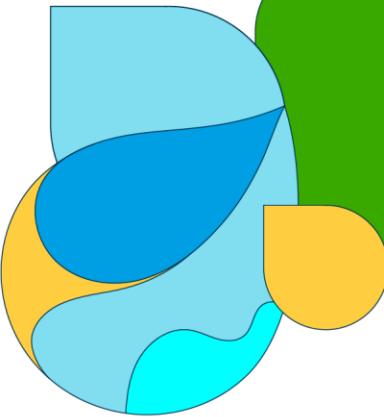


```
C:\Administrator>py -q
>>>
>>> num= 0
>>> while num<4:
...     print(" ", num)
...     num += 1
... else:
...     print(" Ciclo completado.")
...
0
1
2
3
Ciclo completado.
>>>
```

Cláusula else | Ejemplo 4



```
C:\Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>py -q
>>> 
>>> num= 0
>>> while num<4:
...     print (" ", num)
...     if num == 2:
...         break
...     num += 1
... else:
...     print (" Ciclo completado.")
...
 0
 1
 2
>>> 
```



Cláusula else | Ejemplo 5

```
for persona in personas:  
    if es_asombrosa(persona):  
        celebra()  
        break  
  
else:  
    print('No se encontro alguien asombroso en el grupo.')
```

Resumen

- Una declaración condicional en Python también se le llama declaración de control. Es una declaración que tiene expresiones condicionales y evalúa el resultado booleano (**True** o **False**).
- Python tiene las declaraciones condicionales: **if**, **if-else** & **if-elif-elif-....-else**
- En Python se usa la palabra reservada **pass** para especificar un bloque vacío.
- Los ciclos en Python auxilian en las tareas repetitivas, escribiendo código compacto dentro del bloque.

Resumen (Cont.)

- Es posible que dentro de un ciclo (`for` o `while`) haya otra declaración de ciclo (`for` o `while`), a esto se le conoce como **ciclos anidados**.
- Recuerde que la sentencia `else` puede ser parte de la estructura del `while`, y sólo se ejecutará cuando el `while` termina exitosamente.
- La instrucción `break` interrumpe las iteraciones del actual ciclo y sale del ciclo al ejecutarla.
- La instrucción `continue` omite la iteración actual del ciclo mientras continúa en el ciclo.

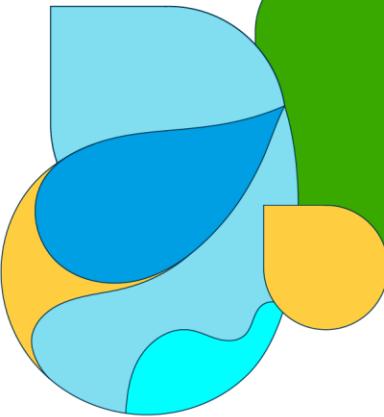
Unidad temática 3

Colecciones y funciones

3.1 Colecciones

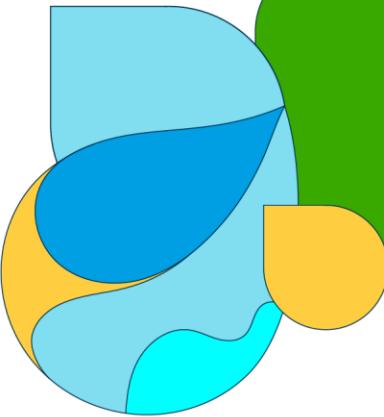
Objetivos:

- Crear, modificar y usar listas.
- Crear, modificar y usar tuplas.
- Crear, modificar y usar conjuntos.
- Crear, modificar y usar diccionarios.
- Identificar el uso de comprensión de listas.



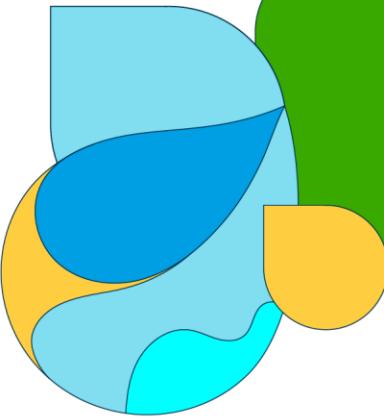
Introducción

- Listas (list): []
- Tuplas (tuple): ()
- Diccionarios (dict): { }
- Conjuntos (set): { }



Introducción

- Las secuencias son iterables que pueden devolver sus elementos según una posición dentro del iterable.
 - Ejemplos de secuencias son las cadenas, las listas, las tuplas y los rangos.
- Las **listas** son secuencias **mutables** similares a los arreglos en otros lenguajes de programación.
- Las **tuplas** son secuencias **inmutables**.
- Los **diccionarios** son asignaciones que usan claves arbitrarias para asignar valores. Los diccionarios son como arreglos asociativos en otros lenguajes de programación.
- Los **conjuntos** son colecciones **mutables** sin ordenar de distintos objetos **inmutables**.
 - El conjunto en sí puede modificarse, no puede llenarse con objetos que puedan modificarse.

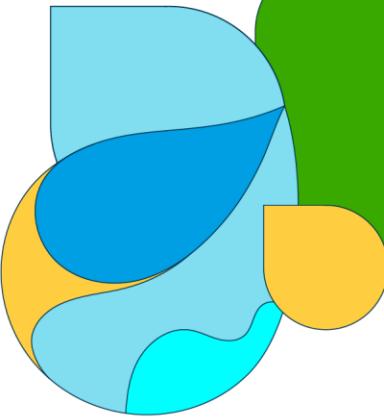


Listas | []

```
colores = ['rojo', 'verde', 'azul']
días = ['Lunes', 'Sabado', 'Domingo']
ventas= [ 1000, 2000, 3000, 5000]
switches=[ True, False, True, True]
```

Listas | [] | Ejemplo

```
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> colores = ['rojo', 'verde', 'azul']
>>> dias = ['Lunes', 'Sabado', 'Domingo']
>>> ventas= [ 1000, 2000, 3000, 5000]
>>> switches=[ True, False, True, True]
>>>
>>> type(colores); type(dias); type(ventas); type(switches)
<class 'list'>
<class 'list'>
<class 'list'>
<class 'list'>
>>>
```



Listas | Métodos

append(x)

sort()

count(i,x)

extend(olista)

remove(x)

reverse()

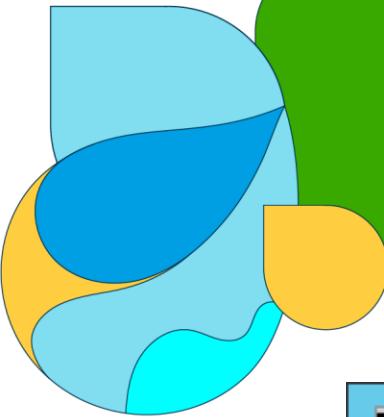
index(x)

list()

insert(i,x)

pop(n)

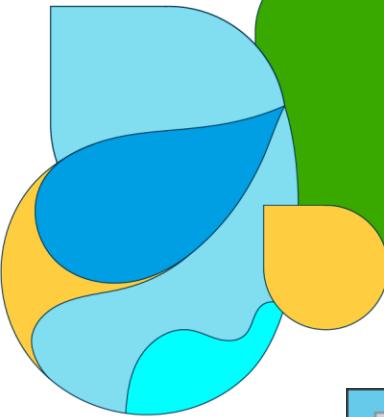
clear()



Listas | Métodos | Ejemplos

```
Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>>
>>> colores= ["amarillo", "rojo", "azul"]
>>> colores.append ("verde")
>>> colores
['amarillo', 'rojo', 'azul', 'verde']
>>>
>>> sabores=["dulce", "amargo", "salado"]
>>> colores.append(sabores)
>>> colores
['amarillo', 'rojo', 'azul', 'verde', ['dulce', 'amargo', 'salado']]
>>>
```

Listas | Métodos | Ejemplos (Cont.)



```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>> puntajes= [120, 100, 90, 70, 10, 200, 120, 100, 100]
>>> puntajes.count(100)
3
>>> puntajes.count(120)
2
>>> puntajes.count(10)
1
>>> puntajes.count(5)
0
>>>
```

Listas | Métodos | Ejemplos (Cont.)

```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>> marcas=["Mazda", "VMW", "Ford", "Acura"]
>>> marcas
['Mazda', 'VMW', 'Ford', 'Acura']
>>> marcas.reverse()
>>> marcas
['Acura', 'Ford', 'VMW', 'Mazda']
>>> marcas.sort()
>>> marcas
['Acura', 'Ford', 'Mazda', 'VMW']
>>> marcas.reverse()
>>> marcas
['VMW', 'Mazda', 'Ford', 'Acura']
>>>
```

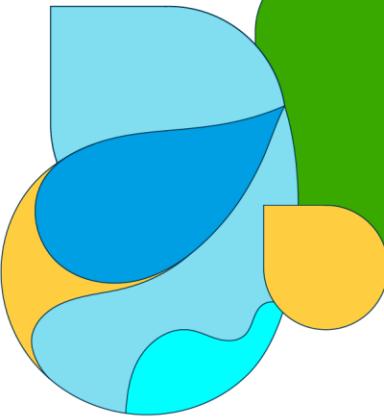
Listas | Métodos | Ejemplos (Cont.)

```
C:\Users\Administrator>python -q
>>> colores= ["amarillo", "rojo", "azul"]
>>> pos= colores.index("amarillo")
>>> pos
0
>>> pos= colores.index("Amarillo")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 'Amarillo' is not in list
>>> ■
```

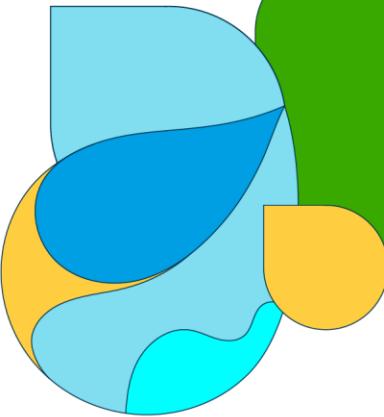
Listas | Métodos | Ejemplos (Cont.)

```
C:\Users\Administrator>python -q
>>> colores= ["amarillo", "rojo", "azul"]
>>> sabores= ["amargo", "salado", "dulce"]
>>> colores.extend(sabores)
>>> colores
['amarillo', 'rojo', 'azul', 'amargo', 'salado', 'dulce']
>>> sabores
['amargo', 'salado', 'dulce']
>>> len(sabores)
3
>>> min(colores)
'amargo'
>>> max(colores)
'salado'
>>>
```

Listas | Funciones Preconstruidas



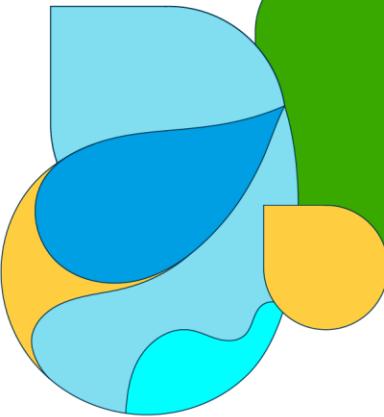
```
C:\> Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> colores=['rojo','verde','azul','negro','blanco']
>>> print(colores)
['rojo', 'verde', 'azul', 'negro', 'blanco']
>>> len(colores)
5
>>> del colores[0]
>>> print(colores)
['verde', 'azul', 'negro', 'blanco']
>>> len(colores)
4
>>> del colores[1:3]
>>> print(colores)
['verde', 'blanco']
>>> len(colores)
2
>>>
```



Operadores de Membresía

in
not in

1 in [1,2,3]	#True
5 in [1,2,3]	#False
1 not in [1,2,3]	#False
5 not in [1,2,3]	#True



Listas | Indexación & Slicing | Sintaxis

`lista[índice]` #El índice inicia en 0 y puede ser con valores negativos

`lista[inicio:]` #Del inicio al final de la lista

`lista[inicio:fin]` #Del inicio al fin-1, inclusive izquierda, exclusive derecha

`lista[:fin]` #Del cero al fin-1 de la lista

`lista[inicio:fin:paso]` #Del inicio al fin-1, de paso en paso

`lista[:]` #Todos los elementos de la lista

Listas | Indexación & Slicing

```
lista=['a','e','i','o','u']
```



Índices

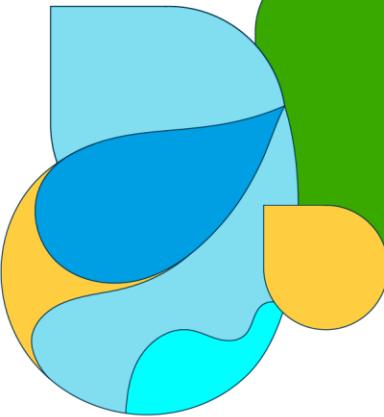
lista[5] IndexError: list index out of range

lista[0]='a'	lista[-1]='u'
lista[1]='e'	lista[-2]='o'
Lista[2]='i'	lista[-3]='i'
lista[3]='o'	lista[-4]='e'
lista[4]='u'	lista[-5]='a'

lista[0:] = ['a','e','i','o','u']	lista[:]= ['a','e','i','o','u']
lista[2:4]= ['i','o']	
lista[1:3]= ['e','i']	
lista[:4]= ['a','e','i','o']	
lista[-1:-5]= []	
lista[-5,-1]= ['a','e','i','o']	
lista[1:5:2]= ['e','o']	

Listas | Indexación & Slicing | Ejemplo

```
C:\>
Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> dias=['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo'] #Días
>>>
>>> #Días específicos
>>> dias[0]; dias[1]; dias[6]; dias[-2]
'Lunes'
'Martes'
'Domingo'
'Sabado'
>>> #Días del inicio al final de la lista
>>> dias[4:]
['Viernes', 'Sabado', 'Domingo']
>>>
>>> #Días en un rango
>>> dias[4:6]
['Viernes', 'Sabado']
>>> #Días del inicio 0 al final - 1
>>> dias[:7]
['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo']
>>> #Días del inicio al final de dos en dos
>>> dias[0:8:2]
['Lunes', 'Miércoles', 'Viernes', 'Domingo']
>>> #Todos los elementos de la lista
>>> dias[:]
['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo']
>>>
```

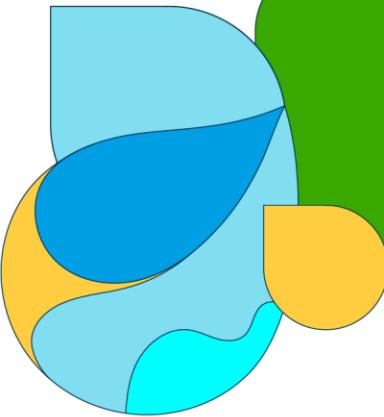


Tuplas | ()

- Similares a las listas.
- Se denotan usando paréntesis.
- Son objetos inmutables.
 - No pueden cambiar después de ser creados.
- Funciones preconstruidas:
 - `len()`

MAGENTA = (255, 0, 255)

Tuplas | Ejemplos



```
Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> MAGENTA=(255,0,255) #Notación RGB para los colores
>>>
>>> type(MAGENTA)
<class 'tuple'>
>>>
>>> MAGENTA=255,0,255
>>>
>>> type(MAGENTA)
<class 'tuple'>
>>>
```

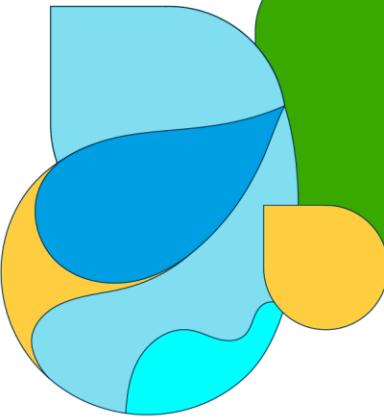
Tuplas | Ejemplos (Cont.)

```
C:\> Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> MAGENTA=255,0,255
>>> type(MAGENTA)
<class 'tuple'>
>>> len(MAGENTA)
3
>>> len(255,0,255)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: len() takes exactly one argument (3 given)
>>> len((255,0,255))
3
>>>
```

Tuplas | Ejemplos (Cont.)

```
C:\Administrator>
Administrator: Command Prompt - py -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> tupla=('2021',)
>>> tupla1='2021'
>>>
>>> type(tupla)
<class 'tuple'>
>>>
>>> type(tupla1)
<class 'str'>
>>>
```

- Tupla vacía: ()
- Tupla con un solo elemento: (elemento,)



Tuplas | Métodos

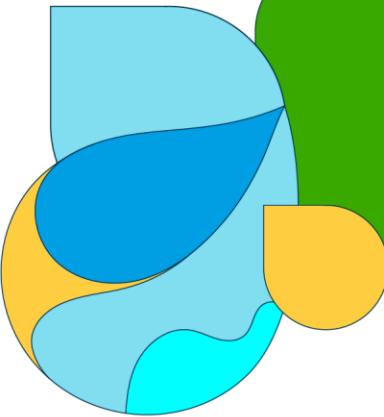
`count(x)`

`len()`

`index(x)`

`min()`

`max()`



Tuplas | Métodos | Ejemplos

```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>>
>>> tupla= (0,1,2,3,4,5,4,3,2,1,0)
>>> len(tupla)
11
>>> cuantos= tupla.count(5)
>>> cuantos
1
>>> cuantos= tupla.count(0)
>>> cuantos
2
>>> cuantos= tupla.count(10)
>>> cuantos
0
>>>
```

Tuplas | Métodos | Ejemplos (Cont.)

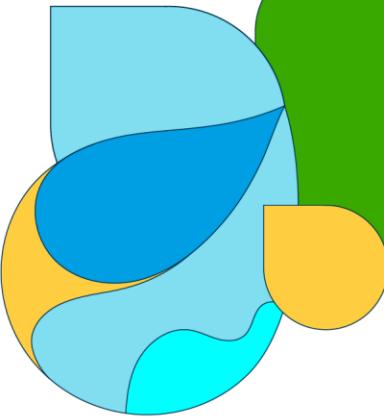
```
Administrator: Command Prompt - python -q
>>>
>>> tupla=('a','e','i','o','u')
>>> len(tupla)
5
>>> pos= tupla.index('a')
>>> pos
0
>>> pos= tupla.index('i')
>>> pos
2
>>> pos= tupla.index('z')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: tuple.index(x): x not in tuple
>>>
```

Tuplas | Métodos | Ejemplos (Cont.)

```
Administrator: Command Prompt - python -q
C:\Users\Administrator>
C:\Users\Administrator>python -q
>>>
>>> tupla=('a','e','i','o','u')
>>> len(tupla)
5
>>> dir(tuple)
['__add__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__',
>>>
>>> min(tupla)
'a'
>>> max(tupla)
'u'
>>>
```

Tuplas | Ejemplos | Inmutabilidad

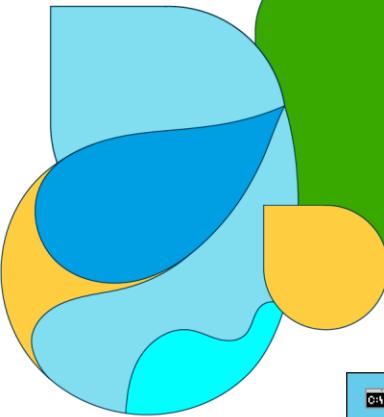
```
C:\Users\Administrator\wp_essentials>py -q
>>>
>>> frutas=['fresas','manzanas','uvas']
>>> carne=['res','pollo','cerdo']
>>> lacteos=['leche', 'yogurt','helado']
>>> vegetales=['brocoli','chicharos','zanahorias']
>>>
>>> comestibles=(frutas,carne,lacteos) # Tupla
>>> comestibles[1]= vegetales
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> print(comestibles)
(['fresas', 'manzanas', 'uvas'], ['res', 'pollo', 'cerdo'], ['leche', 'yogurt', 'helado'])
>>> carne[1]='cordero'
>>> print(comestibles)
(['fresas', 'manzanas', 'uvas'], ['res', 'cordero', 'cerdo'], ['leche', 'yogurt', 'helado'])
>>>
```



Diccionarios | { clave: valor }

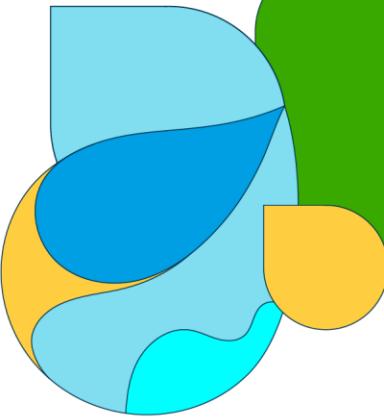
- Colección desordenada, modificable e indexada.
- { clave1 : valor1, clave2 : valor2,... }

```
dict={clave:valor, clave2:valor, clave3:valor,...}
dict[clave2] = nuevo_valor #Asignando un nuevo valor a la clave2
dict[clave4] = valor #Asignando un nuevo valor a la clave4
print(dict[clave1]) #Desplegar el valor asociado a la clave1
```



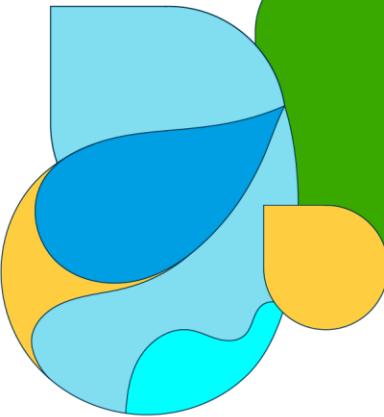
Diccionarios | Ejemplos

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>> print(calificaciones, type(calificaciones))
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 70, 'Arte': 74, 'Danza': 90} <class 'dict'>
>>> len(calificaciones)
5
>>> calificaciones['Matemáticas']
10
>>> calificaciones['Educación Física']=87
>>> print(calificaciones, len(calificaciones))
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 70, 'Arte': 74, 'Danza': 90, 'Educación Física': 87} 6
>>>
>>> calificaciones['Danza']=10
>>> print(calificaciones, len(calificaciones))
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 70, 'Arte': 74, 'Danza': 10, 'Educación Física': 87} 6
>>>
>>> del(calificaciones['Arte'])
>>> print(calificaciones, len(calificaciones))
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 70, 'Danza': 10, 'Educación Física': 87} 5
>>>
```



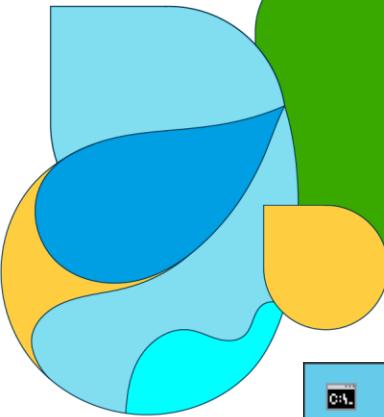
Diccionarios | Métodos

Métodos	Descripción
get(clave [, predeterminado])	Devuelve el valor de la clave si está en el diccionario, de lo contrario, devuelve el valor predeterminado si se pasó en los argumentos.
pop(clave [, predeterminado])	Elimina y devuelve la clave si está en el diccionario de lo contrario, devuelve el valor predeterminado si se pasó como argumento o un KeyError si no lo es.
popitem()	Elimina y devuelve una clave/valor.
copy()	Devuelve una copia del diccionario.
clear()	Elimina todos los elementos del diccionario.



Método update() | Diccionario

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>> calificaciones.update({'Música':90, 'Historia':89}) # Cambia y agrega
>>>
>>> print(calificaciones)
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 89, 'Arte': 74, 'Danza': 90, 'Música': 90}
>>>
>>> # Actualización del diccionario con otro diccionario
>>>
```

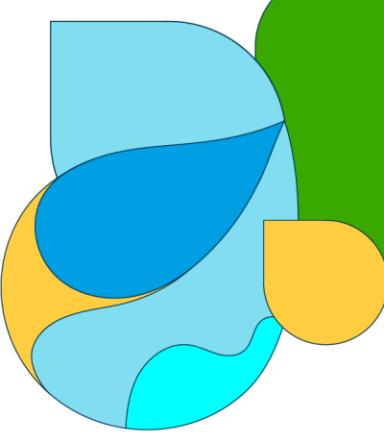


Método update() | Parámetros Nombrados

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>> calificaciones.update(Música=90, Historia=89) # Cambia y agrega
>>>
>>> print(calificaciones)
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 89, 'Arte': 74, 'Danza': 90, 'Música': 90}
>>>
>>> #Actualiza el diccionario con parámetros nombrados clave=valor
>>> -
```

Método update() | Lista de Tuplas

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>> calificaciones.update([('Música',90), ('Historia',89)]) # Cambia y agrega
>>> print(calificaciones)
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 89, 'Arte': 74, 'Danza': 90, 'Música': 90}
>>>
>>> #Actualiza con un lista de tuplas clave:valor
>>> calificaciones['Matemáticas']=100
>>> print(calificaciones)
{'Inglés': 85, 'Matemáticas': 100, 'Historia': 89, 'Arte': 74, 'Danza': 90, 'Música': 90}
>>>
```



Diccionarios | Vistas

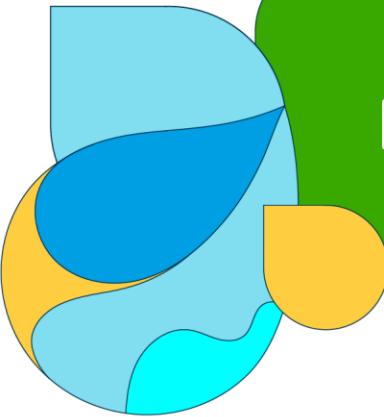
- Los siguientes tres métodos devuelven objetos de tipos vistas (View) del diccionario.

```
mi_dict.keys() # Devuelve las claves o llaves.  
mi_dict.values() # Devuelve los valores.  
mi_dict.items() # Devuelve los pares de llave/valor.
```

Diccionarios | Vistas | Ejemplo

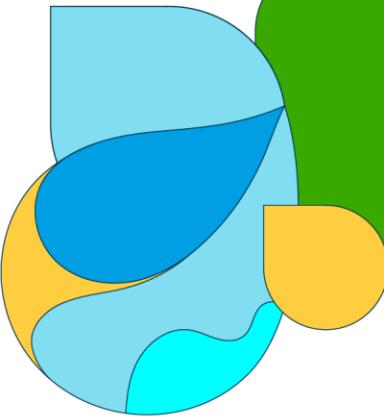
```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>>
>>> print(calificaciones, type(calificaciones), len(calificaciones))
{'Inglés': 85, 'Matemáticas': 10, 'Historia': 70, 'Arte': 74, 'Danza': 90} <class 'dict'> 5
>>>
>>> print(calificaciones.keys(), type(calificaciones.keys()))
dict_keys(['Inglés', 'Matemáticas', 'Historia', 'Arte', 'Danza']) <class 'dict_keys'>
>>>
>>> print(calificaciones.values(), type(calificaciones.values()))
dict_values([85, 10, 70, 74, 90]) <class 'dict_values'>
>>>
>>> print(calificaciones.items(), type(calificaciones.items()))
dict_items([('Inglés', 85), ('Matemáticas', 10), ('Historia', 70), ('Arte', 74), ('Danza', 90)]) <class 'dict_items'>
>>>
>>>
```

Tipos



Diccionarios | Vistas | Observaciones

- `dict_keys` & `dict_values` parecen listas.
- `dict_items` parece una lista de tuplas.
- Las vistas del diccionario no admiten indexación ni segmentación
 - Por no tener un orden establecido.
- Las vistas no se pueden modificar.
- Proporcionan listas dinámicas de un diccionario
 - Cuando el diccionario cambia, la vista también cambiará.



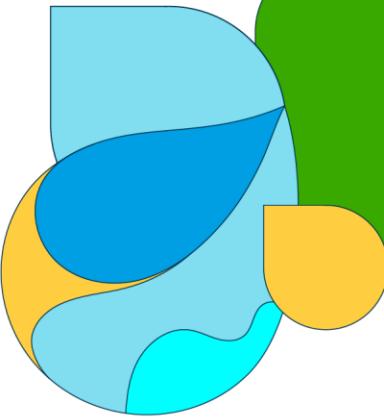
Diccionarios | Vistas | Ejemplos

```
C:\Administrator>
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10, 'Historia':70, 'Arte':74, 'Danza':90 }
>>> numeros= calificaciones.values()
>>> print (numeros)
dict_values([85, 10, 70, 74, 90])
>>> calificaciones['Matemáticas']=100
>>> print (numeros)
dict_values([85, 100, 70, 74, 90])
>>> -
```

Diccionarios | Eliminar Claves

```
C:\>Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> calificaciones= {'Inglés':85, 'Matemáticas':10,'Historia':70, 'Arte':74, 'Danza':90 }
>>>
>>> print(calificaciones['Matemáticas'])
10
>>> del calificaciones['Matemáticas']
>>> print(calificaciones['Matemáticas'])
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'Matemáticas'
>>>
>>> print (calificaciones)
{'Inglés': 85, 'Historia': 70, 'Arte': 74, 'Danza': 90}
>>>
```

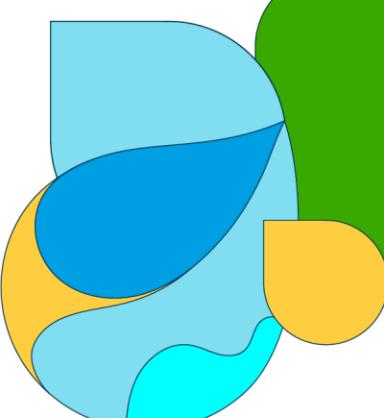
Función preconstruida del()



Conjuntos (set)

- Colecciones mutables sin ordenar de distintos objetos inmutables.
- Puede pensar en conjuntos como diccionarios sin claves.
- Se usan con menos frecuencia.
- Se usan para eliminar los elementos duplicados en una lista.

```
medidas = {'chico', 'mediano', 'grande'}
vocales = {'a', 'e ', 'i', 'o', 'u'}
materias = {'Inglés', 'Matemáticas', 'Historia', 'Arte', 'Deportes'}
dígitos = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
```



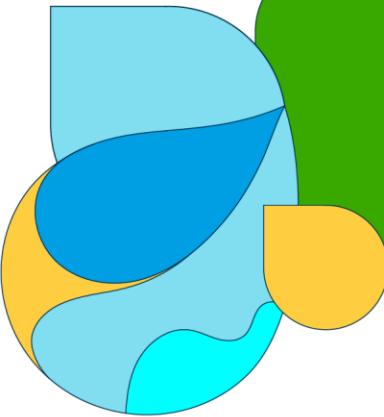
Conjuntos | Ejemplos

```
vegetales = ['tomates', 'espinacas', 'pimientos', 'brocoli', 'tomates', 'brocoli']
print("Lista: ", vegetales)
```

```
v_set = set(vegetales) # de lista a conjuntos
print("Conjunto: ", v_set)
```

```
vegetales = list(v_set) # de conjunto a lista
print("Lista: ", vegetales) # lista sin duplicados
```

```
C:\Material_Python\códigos\PythonFundamentos\demos>python remove_dups.py
Lista: ['tomates', 'espinacas', 'pimientos', 'brocoli', 'tomates', 'brocoli']
Conjunto: {'brocoli', 'tomates', 'espinacas', 'pimientos'}
Lista: ['brocoli', 'tomates', 'espinacas', 'pimientos']
```



Conjuntos | Métodos

add

clear

copy

difference

discard

intersection

isdisjoint

issubset

issuperset

pop

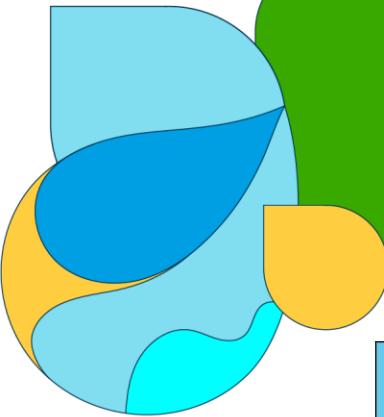
remove

union

update

set

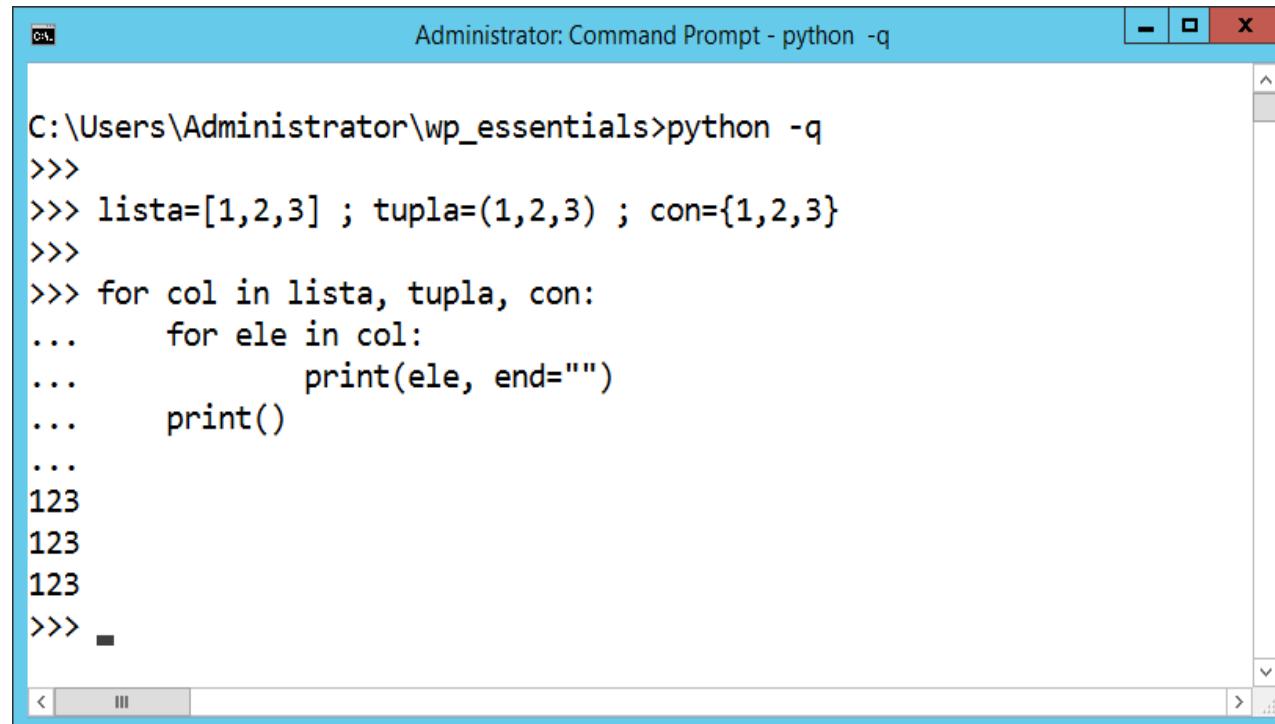
Conjuntos | Métodos | Ejemplos



```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>> cereales= { "arroz", "maiz", "cebada" }
>>> cereales.add("lentejas")
>>> cereales
{'maiz', 'arroz', 'lentejas', 'cebada'}
>>> cereales.clear()
>>> cereales
set()
>>> cereales= { "arroz", "maiz", "cebada" }
>>> copia= cereales.copy()
>>> copia
{'maiz', 'arroz', 'cebada'}
>>> cereales.difference(copia)
set()
>>> cereales.intersection(copia)
{'maiz', 'cebada', 'arroz'}
>>> cereales.union(copia)
{'maiz', 'arroz', 'cebada'}
>>> cereales.issubset(copia)
True
```

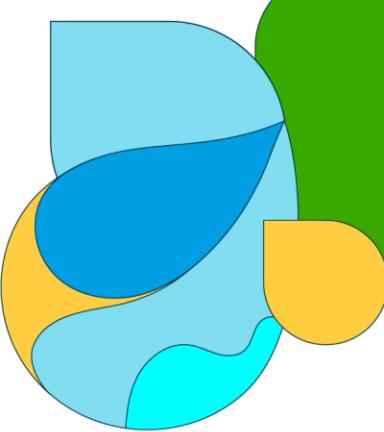
Iterables

- Las colecciones son objetos iterables
 - Esto es, que se puede iterar sobre los elementos de la colección.
 - Lista, tupla, diccionario o conjunto.

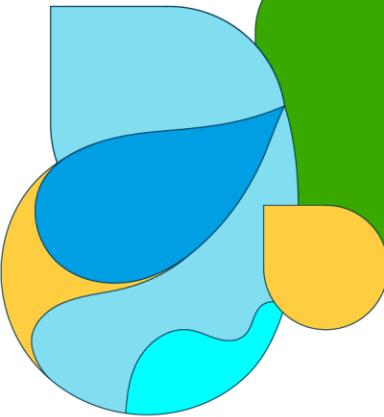


```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> lista=[1,2,3] ; tupla=(1,2,3) ; con={1,2,3}
>>>
>>> for col in lista, tupla, con:
...     for ele in col:
...         print(ele, end="")
...     print()
...
123
123
123
>>> -
```

len()



```
C:\>Administrator: Command Prompt - python -q
C:\Users\Administrator>python -q
>>>
>>> len('Python')
6
>>> len(['a','e','i','o','u'])
5
>>> len((255,255,255))
3
>>> len({'Matematicas':100, 'Español':100, 'Arte':100})
3
>>> -
```



Comprensión de Listas

- Comprensión de listas (list comprehension) se utiliza para la creación de nuevas listas a partir de secuencias existentes al tomar un subconjunto de esa secuencia y/o modificar sus miembros.

```
lista_nueva = [ expression(ele) for ele in lista_original if filter(ele) ]
```

Comprensión de Listas (Cont.)

```
lista_nueva = []
for ele in lista_original:
    if filter(ele):
        lista_nueva.append(expressions(ele))
```

Aplicar una función boolena o filtro a los elementos de la lista original

Iniciar una nueva lista

Iterar sobre los elementos de la lista original

Acumulación de los **elementos** en la nueva lista

Transformar al elemento de la lista original antes de agregarlo a la **nueva** lista

Comprensión de Listas (Cont.)

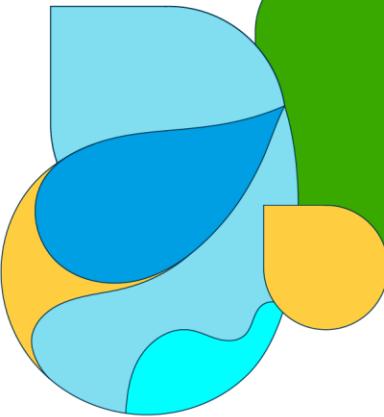
Transformar al elemento de la lista original
antes de agregarlo a la nueva lista

Iterar sobre los elementos de la lista original

```
lista_nueva = [ expression(ele) for ele in lista_original if filter(ele) ]
```

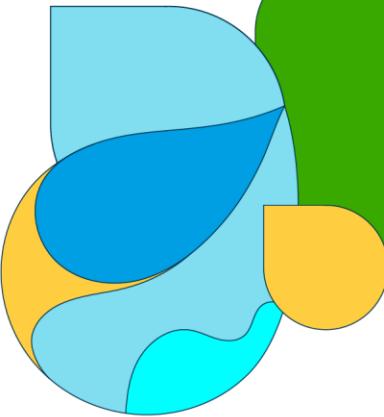
Acumulación de los elementos en la nueva lista

Aplicar una función boolena o filtro a los
elementos de la lista original

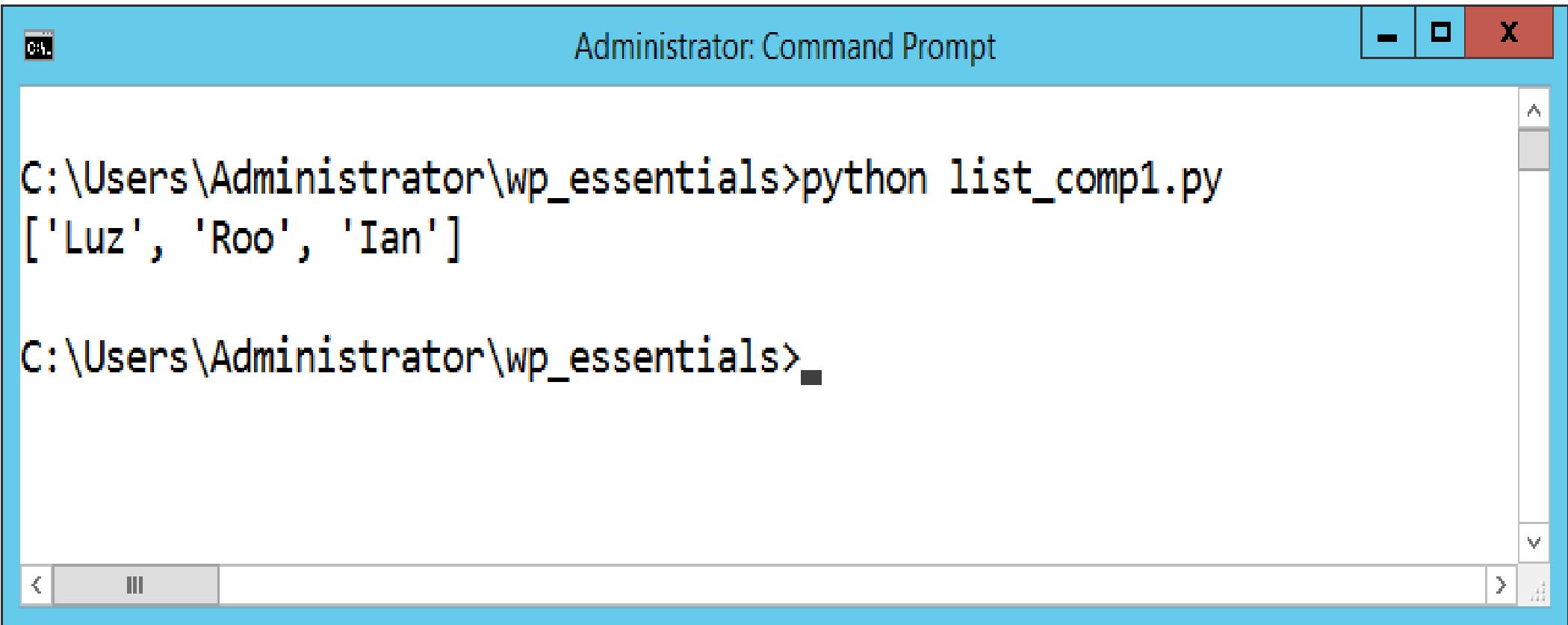


Comprensión de Listas | Ejemplo

```
words = ['Marisol', 'Saúl', 'Alejandra', 'Patricia', 'Abraham', 'Yuri',
         'Luz', 'Carmen', 'Violeta', 'Greta', 'Adriana', 'Alicia', "Roo", "Yum"
         'Adriana', 'Ximena', 'David', 'Ricardo', 'Eduardo', 'Enrique',
         'Almudena', 'Gabriela', 'Leticia', 'Lucia', 'Eduardo', 'Ian']
tres_letras = [w for w in words if len(w) == 3]
print(tres_letras)
```



Comprensión de Listas | Ejemplo (Cont.)



```
Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>python list_compl.py
['Luz', 'Roo', 'Ian']

C:\Users\Administrator\wp_essentials>
```

Resumen

- Las listas son mutables mientras que las tuplas son inmutables, lo que significa que se puede cambiar el contenido de una lista dinámicamente.
- Las tuplas son inmutables, es decir, no cambian una vez creadas. No así sus elementos.
- Cuando las iteraciones se aplican a los objetos de la lista, el tiempo de procesamiento es mayor. Por el contrario, los objetos de tupla iteran de manera rápida.
- El tipo de datos Tuple es apropiado para acceder a los elementos. Por el contrario, la lista es mejor para realizar operaciones, como la inserción y eliminación.
- Si el usuario desea utilizar la secuencia como clave de diccionario, una tupla es una mejor opción porque las claves de diccionario son inmutables.

Resumen (Cont.)

- Las listas consumen más memoria en comparación con la tupla.
- En las listas, es más probable que ocurran cambios y errores inesperados, pero en el caso de una tupla, es difícil que ocurra.
- Las listas tienen varios métodos incorporados, mientras que las tuplas no tienen métodos incorporados obligatorios.
- La comprensión de listas es simplemente una reorganización de un ciclo `for` para fines prácticos.
- Internamente Python sí hace mejoras de optimización en la comprensión de listas.

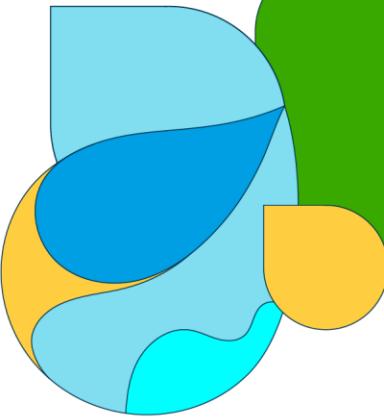
Resumen (Cont.)

- Las colecciones son objetos iterables.
- La función **len()** devuelve la cardinalidad de la colección.
- La función **sum()** devuelve la suma de los elementos de la colección.
- Los elementos de las listas, tuplas, y conjuntos pueden ser a su vez listas, tuplas y conjuntos.

3.2 Funciones

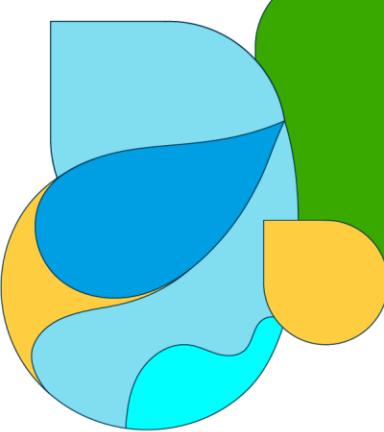
Objetivos:

- Definir e invocar funciones.
- Definir funciones con lista de parámetros variables.
- Invocar funciones con listas de argumentos.
- Establecer valores predeterminados a los parámetros de una función.
- Identificar el alcance, ámbito o scope de una variable local y global.
- Conocer y usar los enumeradores.
- Conocer y usar los generadores



Introducción

- Evitan la repetición de fragmentos de código.
- Puede considerarse como una variable que "encierra" código.
- Apoya el desarrollo modular.
- En Python un módulo puede verse como una parte de un programa en cualquiera de sus formas y variados contextos.
 - Un módulo no necesariamente es una función o un procedimiento.
 - Los módulos deben de ser cohesivos y de bajo acoplamiento.



Funciones | Definición

Inicio de bloque

Sangría

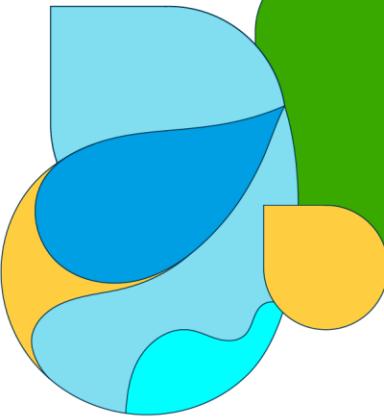
def nombre_función([lista de argumentos opcionales]):

■ # El contenido de la función debe ir identado,

■ # indicando el bloque de la función

■ # Realizar algo

■ algo()



Funciones | Invocación

- Para invocar se utiliza el nombre de la función seguida de los paréntesis, y opcionalmente puede tener parámetros.
- Si la función tiene parámetros, entonces deben de agregarse en la invocación

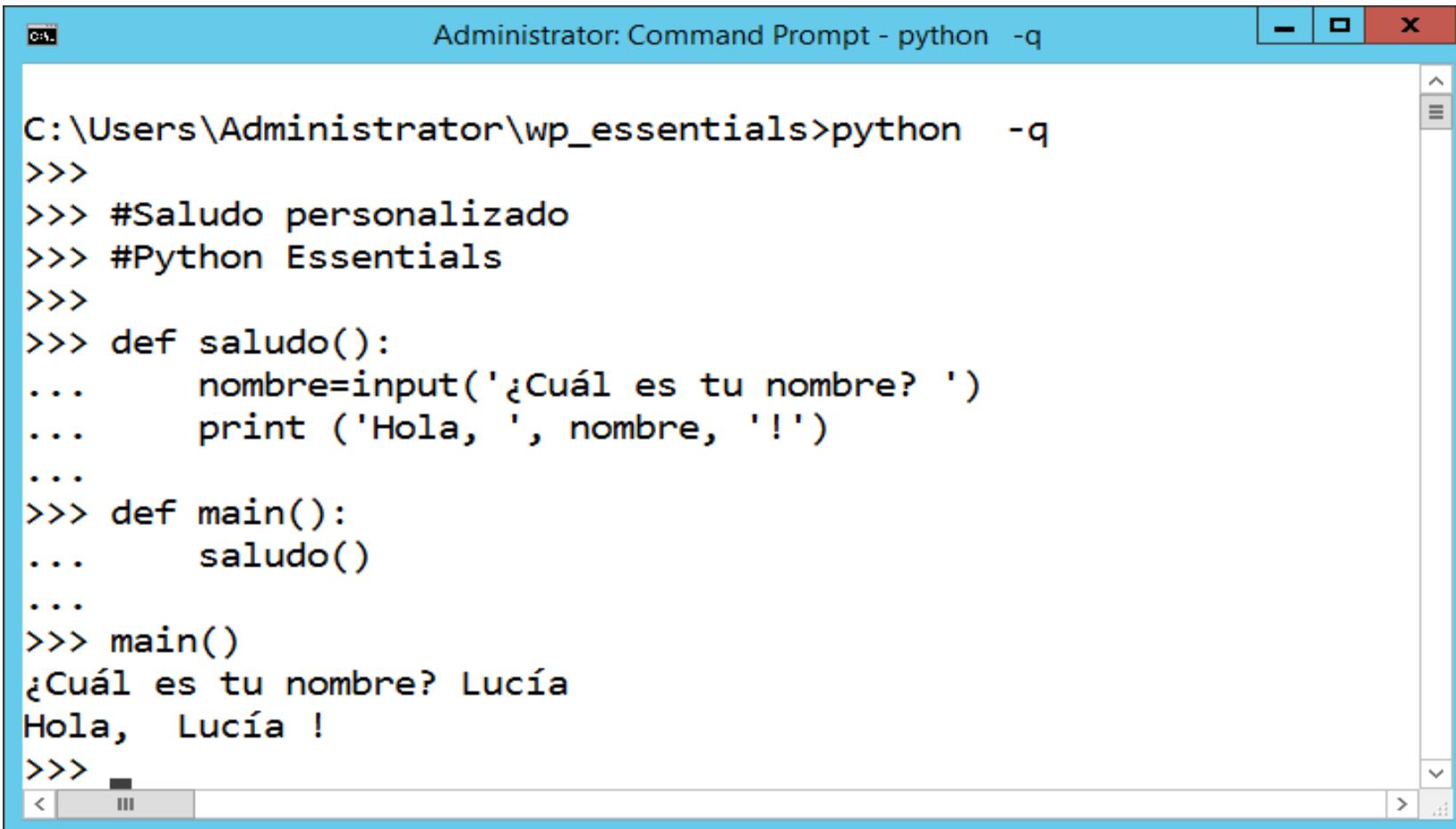
```
def nombre_función([lista de argumentosopcionales]):  
    pass
```

Definición

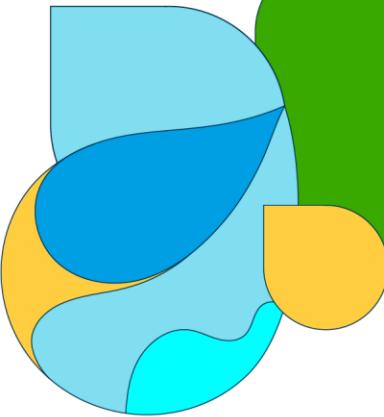
```
nombre_función()
```

Invocación

Funciones | Ejemplo



```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> #Saludo personalizado
>>> #Python Essentials
>>>
>>> def saludo():
...     nombre=input('¿Cuál es tu nombre? ')
...     print ('Hola, ', nombre, '!')
...
>>> def main():
...     saludo()
...
>>> main()
¿Cuál es tu nombre? Lucía
Hola, Lucía !
>>>
```



Funciones | Invocación

```
print() #Invocación sin argumentos
```

```
print("Lista de agumentos", "a", "imprimir") #Invocación con tres argumentos
```

```
print("Esto solo es un ejemplo con print") #Invocación con un argumento
```

```
print("Una línea","mas","de ejemplo",sep="-", end="***")
```

```
#Invocación con parámetros nombrados
```

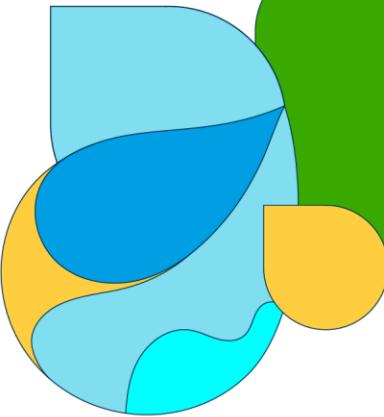
Funciones | help()

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> print("Una línea", "mas", "de ejemplo", sep="-", end="***")
Una línea-mas-de ejemplo***>>>
>>>
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
        file: a file-like object (stream); defaults to the current sys.stdout.
        sep: string inserted between values, default a space.
        end: string appended after the last value, default a newline.
        flush: whether to forcibly flush the stream.

>>> -
```

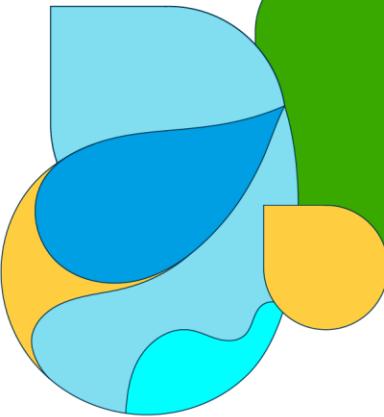


Funciones | Ámbito

- El ámbito, scope en inglés, de una variable es la parte del programa donde la variable es accesible.
- Los parámetros de una función son variables locales.
- Las variables declaradas dentro de una función son locales a esa función donde se encuentra.
- No existe una palabra reservada para indicar que la variable es local.
- Las variables que se encuentran fuera de todas las funciones se les llama variables globales.
- Para tener acceso a una variable de ámbito global se puede usar la palabra reservada **global**.

Funciones | Ámbito | Ejemplo 1

```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> def set_x():
...     x=10 #Variable local
...
>>> def get_x():
...     print(x) #Variable global
...
>>> def main():
...     set_x()
...     get_x()
...
>>> x=100
>>> main()
100
>>>
```



Funciones | Ámbito

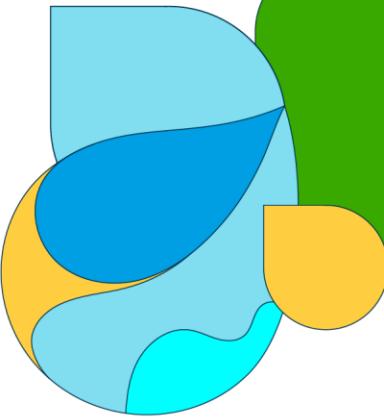
- Si la variable no se encuentra definida previo a su uso, Python genera un error.

NameError: name '<nombre>' is not defined

Funciones | Ámbito | Ejemplo 2

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> def set_x():
...     x=10 #Variable local
...
>>> def get_x():
...     print(x) #Variable global
...
>>> def main():
...     set_x()
...     get_x()
...
>>> main()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 3, in main
    File "<stdin>", line 2, in get_x
NameError: name 'x' is not defined
>>>
```

Ámbito Global | Ejemplo 3



```
C:\> Administrator: Command Prompt
# Variables globales

x=0

def set_x():
    x=1
    print("En set_x():", x) # Usa la variable local x

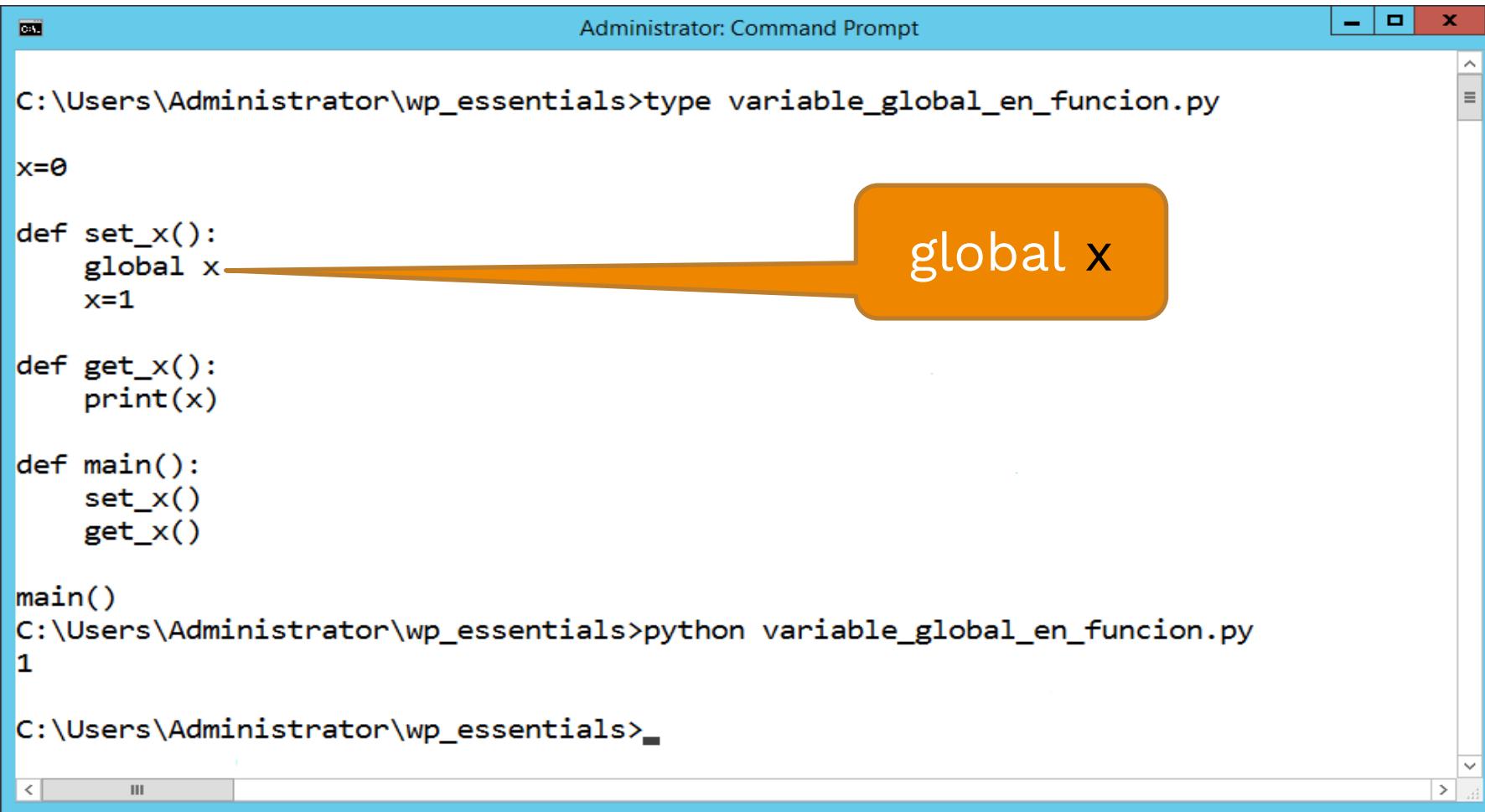
def get_x():
    print("En get_x():", x) # Usa la variable global x

def main():
    set_x()
    get_x()

main()
C:\Users\Administrator\wp_essentials>py variables_globales.py
En set_x(): 1
En get_x(): 0

C:\Users\Administrator\wp_essentials>
```

Ámbito Global | Ejemplo 4



```
C:\Users\Administrator\wp_essentials>type variable_global_en_funcion.py

x=0

def set_x():
    global x
    x=1

def get_x():
    print(x)

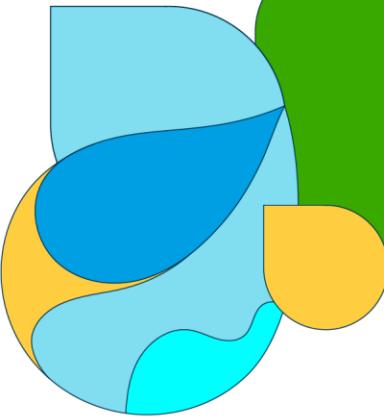
def main():
    set_x()
    get_x()

main()
C:\Users\Administrator\wp_essentials>python variable_global_en_funcion.py
1

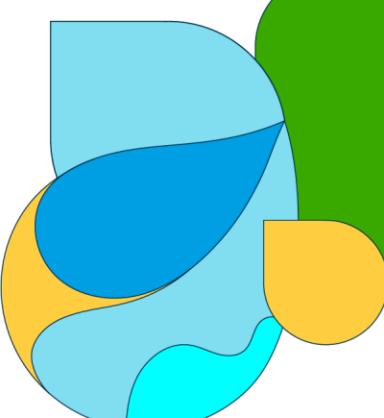
C:\Users\Administrator\wp_essentials>
```

global x

Ámbito Global | Ejemplo 5



```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> a=100
>>> def fun():
...     print(a)
...     a=500
...     print(a)
...
>>> def gen():
...     print(a)
...
>>> gen()
100
>>>
>>> fun()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
    File "<stdin>", line 2, in fun
UnboundLocalError: local variable 'a' referenced before assignment
>>>
```



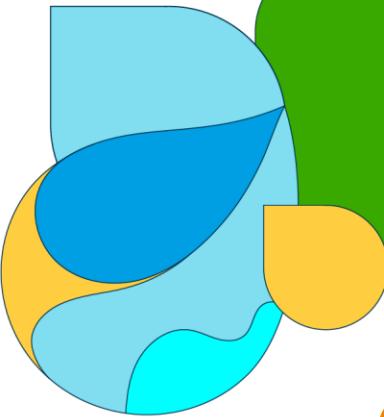
Recursión

- Cuando en la definición de una función, esta se invoca a sí misma, se le conoce como función recursiva.

```
def nombre_función([lista de argumentosopcionales]):  
    # ... Código omitido  
    nombre_función()
```

Recursión | Ejemplo

```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> def factorial(num):
...     if num == 0:
...         return 1
...     return num * factorial(num-1)
...
>>>
>>> factorial(0)
1
>>> factorial(1)
1
>>> factorial(2)
2
>>> factorial(3)
6
>>> factorial(5)
120
>>>
```



Parámetros Posicionales & Defecto

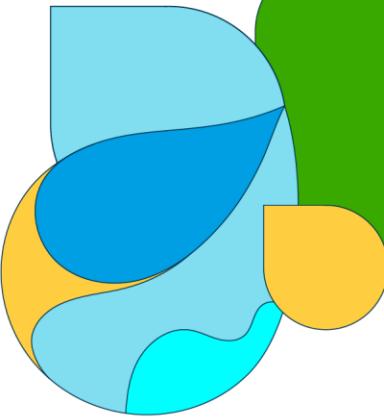
Los valores por defecto se indican en la definición de la función usando el operador de asignación =

```
def
nombre_función(parámetro=valor_defecto):
    hacer_algo(parámetro)
```

Parámetros Posicionales & Defecto (Cont.)

```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> def sep(s==""):
...     print(s,s,s,s,sep="")
...
>>> sep()
=====
>>> sep('x')
xxxx
>>> sep('-')
---
>>> sep('----')
-----
>>>
```

Cuando la función se invoca sin ningún parámetro se usa el valor por defecto



Devolución de Valores | return

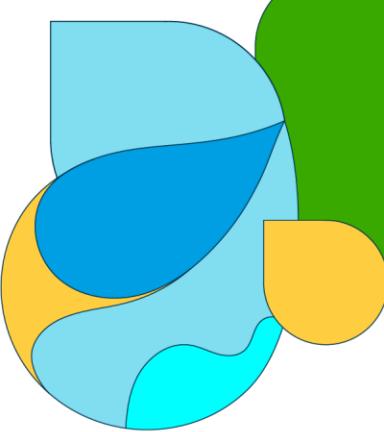
- Las funciones utilizan la palabra reservada `return` para devolver un valor.
- La instrucción `return` hace que la función termine inmediatamente.
- Las funciones no indican el tipo de dato que devuelven, por lo tanto, puede regresar valores de diferente tipo según se necesite.

Devolución de Valores | Ejemplo

```
def suma(num1, num2, num3=0, num4=0, num5=0):
    res = num1 + num2 + num3 + num4 + num5
    return res

def main():
    s = suma(1, 2) # s queda con el valor de 3
    print(s)
    s = suma(s, 3) # s queda con el valor de 6
    print(s)
    s = suma(s, 4) # s queda con el valor de 10
    print(s)
    s = suma(s, 5) # s queda con el valor de 15
    print(s)
    s = suma(s, 6) # s queda con el valor de 21
    print(s)

main()
```



Funciones Preconstruidas

`int()`

`abs()`

`pow()`

`len()`

`bin()`

`min()`

`round()`

`enumerate()`

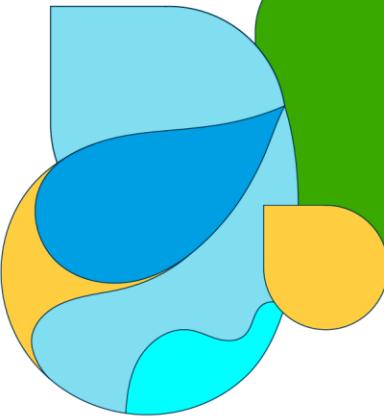
`float()`

`max()`

`sum()`

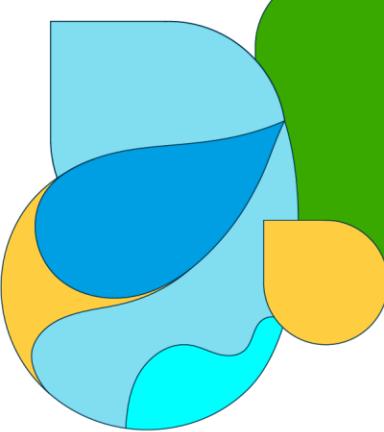
`hex()`

int() | Ejemplo



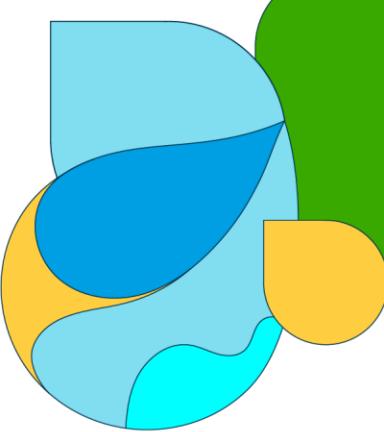
```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> int(1)
1
>>> int('1')
1
>>> int(1.5)
1
>>> int(1.999)
1
>>> int (-1.9999)
-1
>>> int ('1.5')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1.5'
>>>
```

bin() | Ejemplo



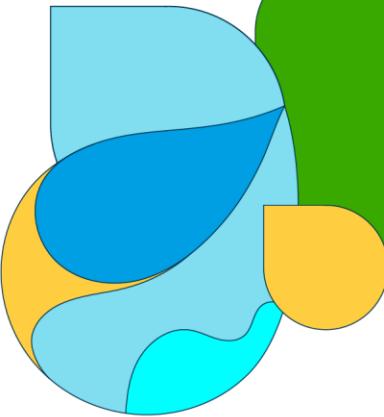
```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> bin(2)
'0b10'
>>> bin(8)
'0b1000'
>>> bin(-1)
'-0b1'
>>> bin(5.5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'float' object cannot be interpreted as an integer
>>> bin('8')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object cannot be interpreted as an integer
>>>
```

float() | Ejemplo

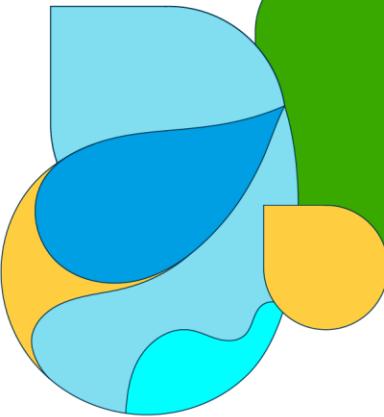


```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> float(5)
5.0
>>> float('5')
5.0
>>> float("5")
5.0
>>> float('5.3')
5.3
>>> float('-5.3')
-5.3
>>> float('-5.9999')
-5.9999
>>> -
```

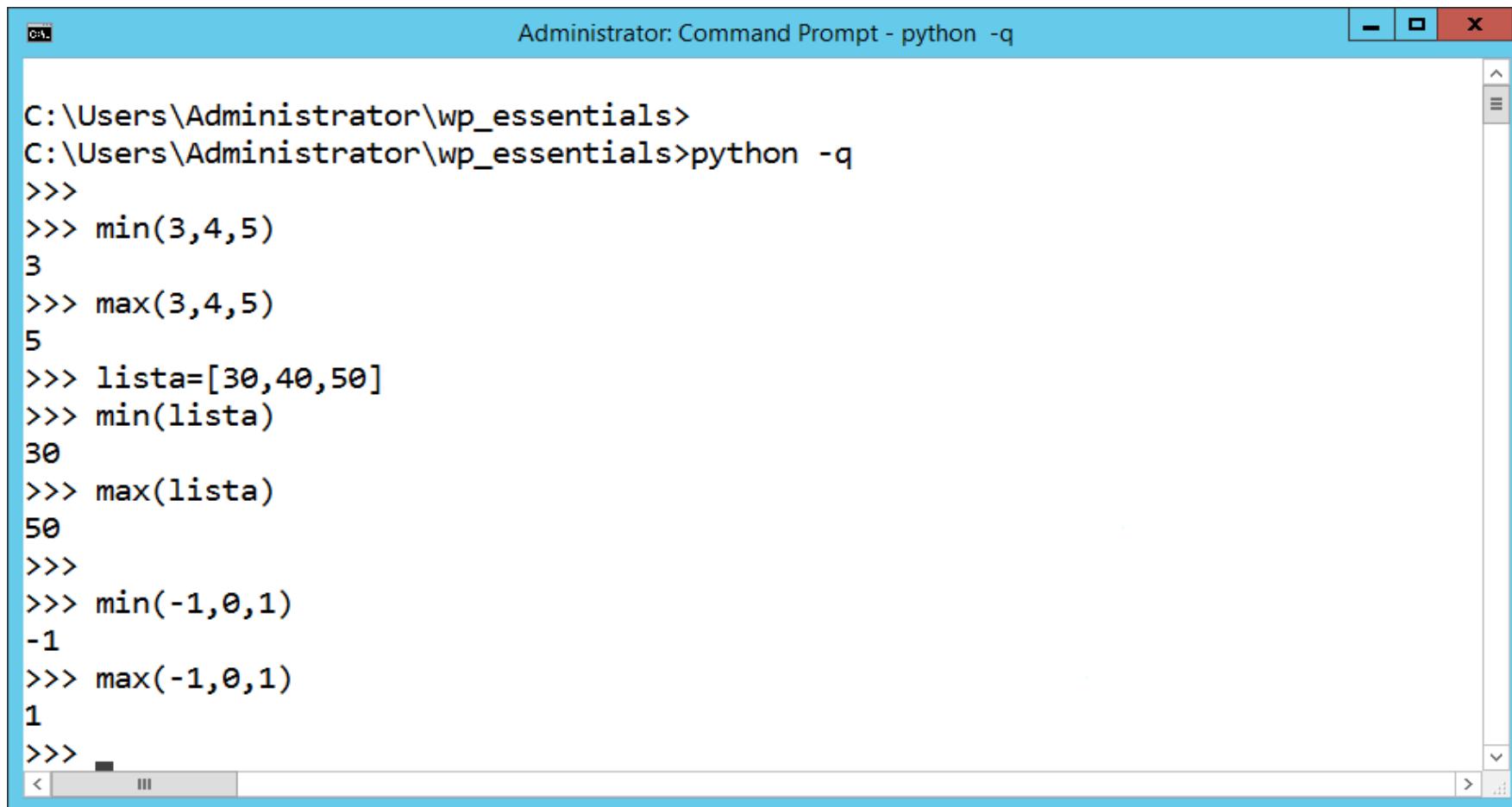
abs() | Ejemplo



```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> abs(-2)
2
>>> abs(2)
2
>>> abs(2.5)
2.5
>>> abs(-2.5)
2.5
>>>
>>> abs('2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: bad operand type for abs(): 'str'
>>> -
```

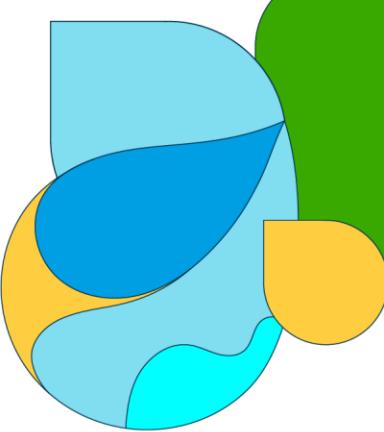


min() & max() | Ejemplo



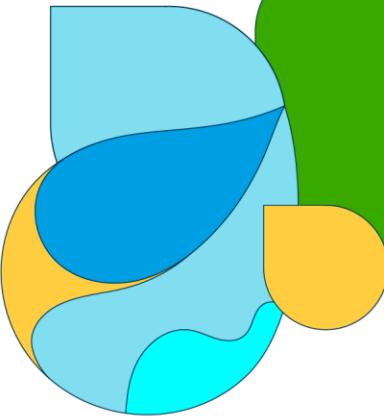
```
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> min(3,4,5)
3
>>> max(3,4,5)
5
>>> lista=[30,40,50]
>>> min(lista)
30
>>> max(lista)
50
>>>
>>> min(-1,0,1)
-1
>>> max(-1,0,1)
1
>>>
```

pow() | Ejemplo



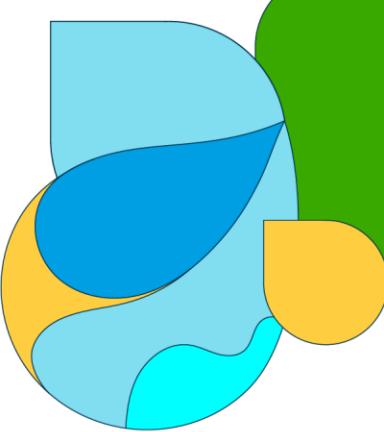
```
C:\Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> pow(2,3)
8
>>> 2**3
8
>>>
>>> pow(3,3)
27
>>> 3**3
27
>>>
>>> pow(2,1/2)
1.4142135623730951
>>> -
```

pow() | Ejemplo (Cont.)



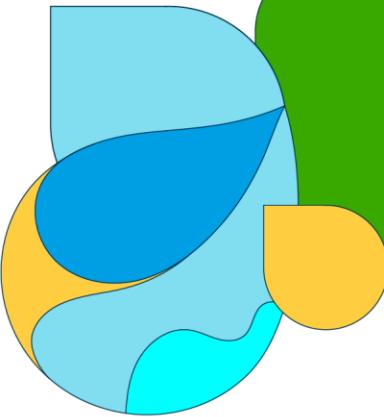
```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> pow(2,3,3)
2
>>> 2**3%3
2
>>> pow(5,2,10)
5
>>> 5**2%10
5
>>> -
```

round() | Ejemplo



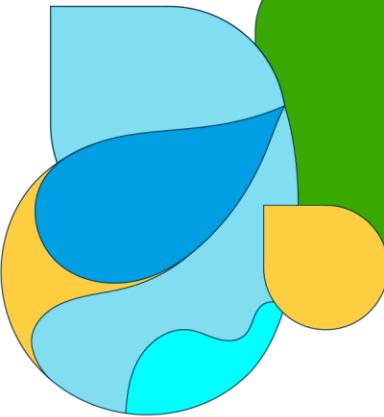
```
Administrator: Command Prompt - python -q
>>>
>>> round(3.1416)
3
>>> round(-3.1416)
-3
>>> round(3.1415,1)
3.1
>>> round(3.5000001,1)
3.5
>>> round(3.950001,1)
4.0
>>>
>>> round (1111,0)
1111
>>> round (1111,-1)
1110
>>> round (1111,-2)
1100
>>> round (1111,-3)
1000
>>> round (1111,-4)
0
>>>
```

sum() | Ejemplo



```
Administrator: Command Prompt - python -q
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> sum(4,3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>>
>>> sum((4,3))
7
>>> a=[1,2,3,4,5]
>>> sum(a)
15
>>> sum(a,10)
25
>>> sum(range(1,101))
5050
>>>
```

len() | Ejemplo



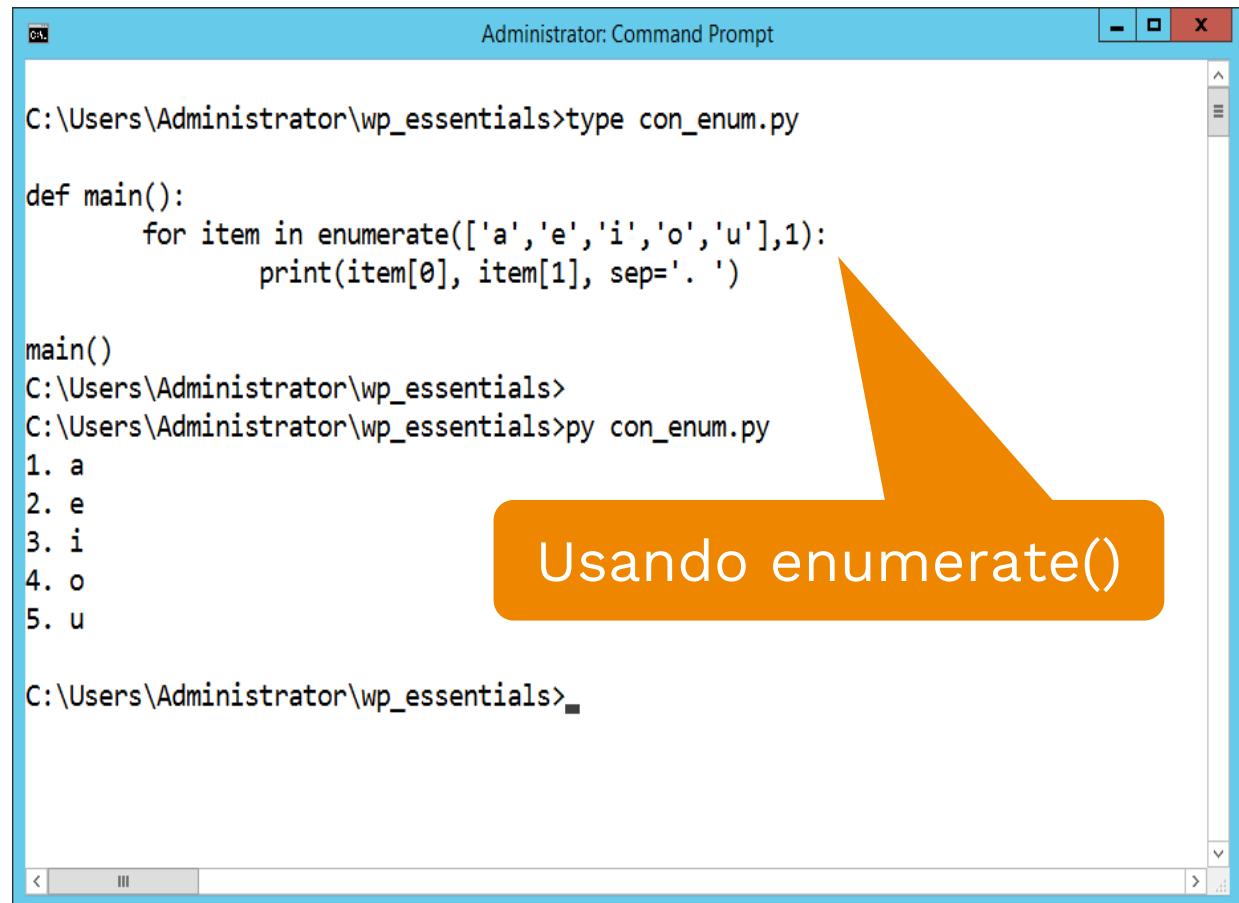
```
C:\Users\Administrator\wp_essentials>python -q
>>>
>>> lista=[1,2,3]
>>> tupla=(0,1,2)
>>> set={'a','e','i','o','u'}
>>> dict={'a':10,'b':20,'c':30}
>>> len (lista)
3
>>> len(tupla)
3
>>> len(set)
5
>>> len(dict)
3
>>> len("0123456789")
10
>>> len(range(1,100))
99
>>>
```

enumerate() | Ejemplo

Sin usar enumerate()

```
def main():
    i = 1
    for item in ['a','e','i','o','u']:
        print(i, item, sep=' . ')
        i += 1

main()
```



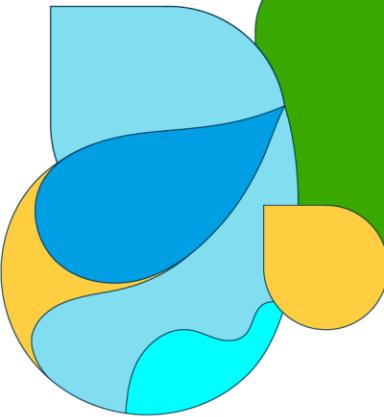
```
Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>type con_enum.py

def main():
    for item in enumerate(['a','e','i','o','u'],1):
        print(item[0], item[1], sep=' . ')

main()
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>py con_enum.py
1. a
2. e
3. i
4. o
5. u

C:\Users\Administrator\wp_essentials>
```

Usando enumerate()



Generadores

- Iteración es la repetición de "algo" una y otra vez.
 - En Python generalmente está asociada con el ciclo for.
 - Es una forma de iterar sobre una lista de valores.
- Un iterable es un objeto que admite la iteración.
 - En Python son iterables: list, tuples, dict, str & set.
- Los generadores son iteradores especiales que solo pueden usarse una vez.
- Son creados a través de funciones.
 - Usan la palabra `yield` en lugar de `return`.
- Producen los datos sobre la marcha y permiten acceder a ellos siempre que los necesite.
- Optimizan los recursos, en especial la memoria, y permiten trabajar con flujos infinitos de datos.

Función Regular o Normal

```
import random

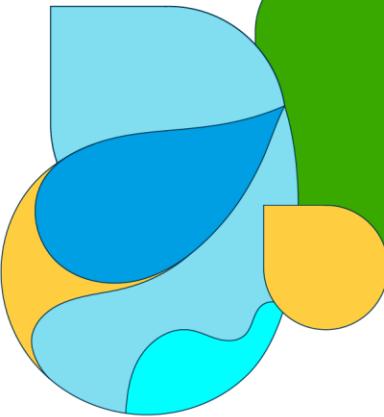
def genera(inicio,fin,num):
    numeros = []
    for numero in range(num):
        numeros.append(random.randint(inicio,fin))
    return numeros

numeros = genera(1,100,5)

print('Iterando a través de la lista:')
for num in numeros:
    print(num)

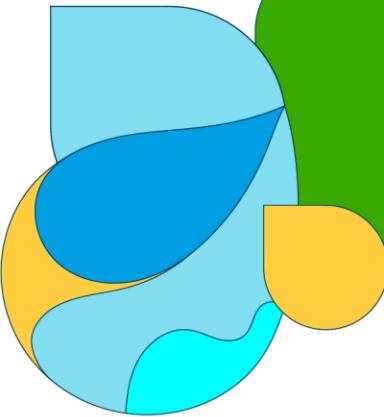
print('Nuevament iterando en la lista: ')
for num in numeros:
    print(num)
```

Función Regular | Ejecución & Salida



```
C:\> Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>py list_loop.py
Iterando a través de la lista:
92
14
92
19
75
Nuevamente iterando en la lista:
92
14
92
19
75

C:\Users\Administrator\wp_essentials>
```



Generadores vs. Funciones

```
# Función Regular
def función_a():
    return "a"
```

```
# Función Generadora
def generador_a():
    yield "a"
```

Las dos funciones realizan exactamente la misma acción, devolver o producir la misma cadena

Generadores | Ejemplo 1

```
import random

def genera(inicio,fin,num):
    for numero in range(num):
        yield random.randint(inicio,fin)    # yield

numeros = genera(1,100,5)

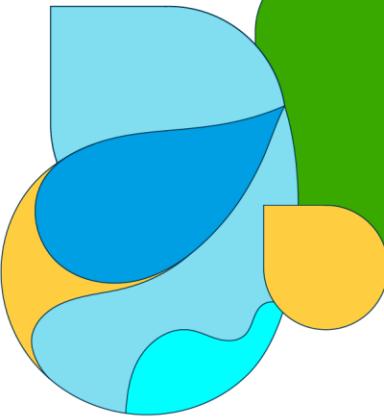
print('Iteramos en el generador:')
for num in numeros:
    print(num)

print('Nuevamente iteramos en el generador:')
for num in numeros:
    print(num)
```

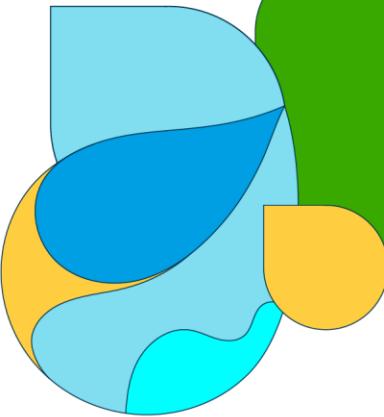
La función genera() produce cada resultado uno por uno a medida que se avanza por el ciclo for

yield en lugar de return

Generador | Ejecución & Salida



```
Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>python generator_loop.py
Iteramos en el generador:
63
20
27
41
76
Nuevamente iteramos en el generador:
C:\Users\Administrator\wp_essentials>
```



Generadores | Ejemplo 2

```
import random

def genera(inicio,fin,num):
    for numero in range(num):
        yield random.randint(inicio,fin)    # yield

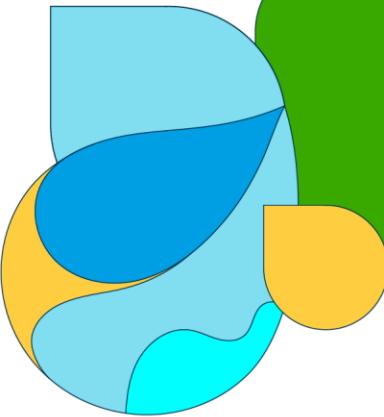
numeros = genera(1,100,5)

print('Iteramos en el generador:')
for num in numeros:
    print(num)

numeros = genera(1,100,5)

print('Nuevamente iteramos en el generador:')
for num in numeros:
    print(num)
```

Generador | Ejecución & Salida



```
Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>
C:\Users\Administrator\wp_essentials>python generator_loop.py
Iteramos en el generador:
54
43
73
16
52
Nuevamente iteramos en el generador:
57
19
34
89
48

C:\Users\Administrator\wp_essentials>
```

Resumen

- La conversión de un tipo de datos en otros tipos de datos se le conoce como typecasting o simplemente casting.
- Python cuenta con varias funciones de conversión de tipos como: int(), bin(), oct(), hex(), float(), list(), str(), bool(), tuple(), set(), dict(), complex(), etc.
- Para verificar el tipo de datos Python tiene la función [type\(\)](#).
- Python tiene las funciones round(), sum(), pow() para redondear cifras, sumar todos sus elementos y obtener la potencia de un número.

Resumen (Cont.)

- El módulo de Python para usar funciones logarítmicas y trigonométricas se llama `math`.
- El módulo de Python para matemáticas con números complejos se llama `cmath`.
- La función para devolver un enumerado, se llama `enumerate(iterable, inicio=0, paso)`.
- Las funciones típicas de entrada y salida son: `input()` y `print()`
- Cuando se define una función que se llama a sí misma se le conoce como función recursiva.

Resumen (Cont.)

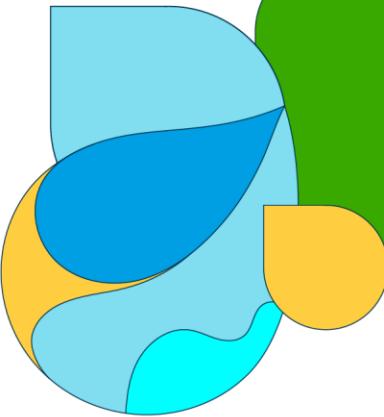
- Cuando la función devuelve un valor en la definición de dicha función se usa la palabra `return`
- Cuando un generador devuelve un valor en la definición del generador se usa la palabra `yield`
- Los valores por defecto en las funciones se colocan en las definiciones de estas: `def función(parámetro=valor):`

Unidad temática 4

Introducción al uso aplicado de Python
para el desarrollo

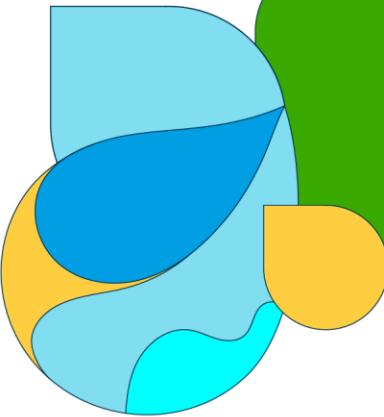
Objetivos:

- Listar las características más sobresalientes acerca de Python.
- Configurar ambiente y herramientas de desarrollo en Python.



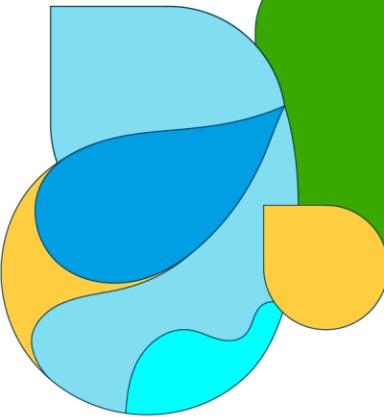
Introducción | Características

- Lenguaje de propósito general de alto nivel, apareció a principios de los 90s, creado por Guido Van Rossum.
- Lenguaje Orientado a Objetos.
 - Encapsulamiento, Abstracción, Polimorfismo & Herencia.
- Interpretado, tipado dinámico & fuertemente tipificado.
- Libre & gratuito.
- Multiplataforma.
- Datos de alto nivel: [Listas, conjuntos, diccionarios, tuplas, etc.](#)
- Implementaciones: [Jython, CPython, IronPython, etc.](#)



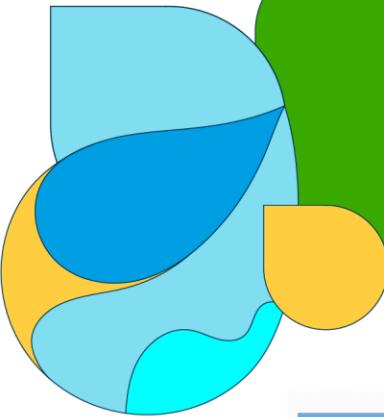
Introducción | Usos

- Desarrollo web
 - Frameworks: Django, CherryPy, TurboGears, Web2Py, Falcom & Flask.
- Ciencias de datos
 - [Aprendizaje automático](#), análisis de datos y visualización de datos.
 - [Scikit-Learn & Tensor-Flow](#).
 - Matplotlib, SeaBorn, etc.
- Scripting
 - Secuencia de comandos, automatizar la ejecución de tareas.
- Otros
 - Video juegos, aplicaciones de escritorio, aplicaciones integradas, seguridad, etc.



Introducción | Herramientas & IDEs

- **IDE:** Integrated Development Environment.
- Programa dedicado al desarrollo de software con varias herramientas auxiliares en la construcción de software.
 - Un editor diseñado resaltando de sintaxis.
 - Atajos para la inserción de bloques de código y autocompletado.
 - Opciones para la compilación, la ejecución y depuración del código.
 - Integración con herramientas de control de versiones.



Introducción | Python SDK

Looking for a specific release?

Python releases by version number:

Release version	Release date	Click for more
Python 3.10.4	March 24, 2022	 Download Release Notes
Python 3.9.12	March 23, 2022	 Download Release Notes
Python 3.10.3	March 16, 2022	 Download Release Notes
Python 3.9.11	March 16, 2022	 Download Release Notes
Python 3.8.13	March 16, 2022	 Download Release Notes
Python 3.7.13	March 16, 2022	 Download Release Notes
Python 3.9.10	Jan. 14, 2022	 Download Release Notes
Python 3.10.2	Jan. 14, 2022	 Download Release Notes

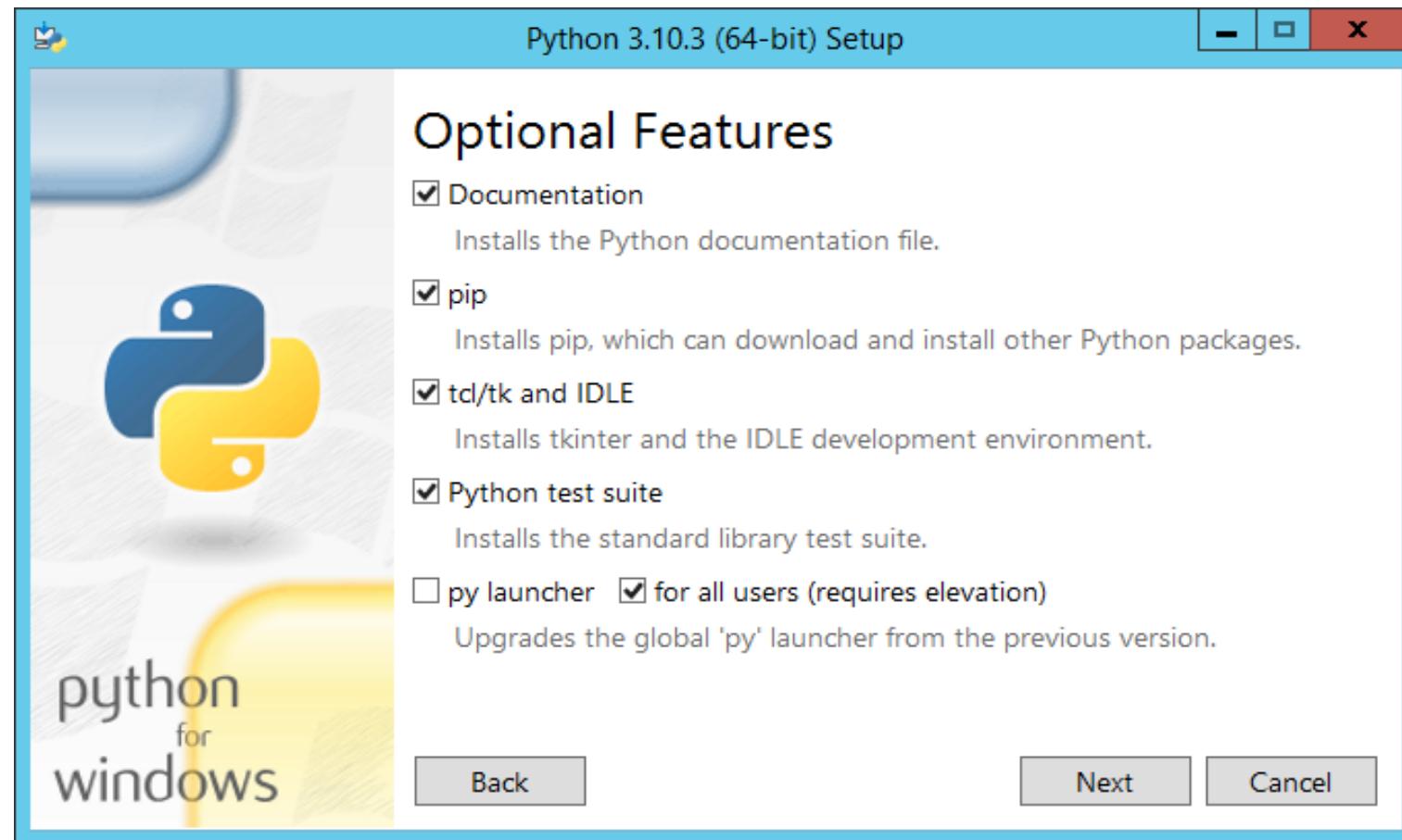
[View older releases](#)

Introducción | Python SDK

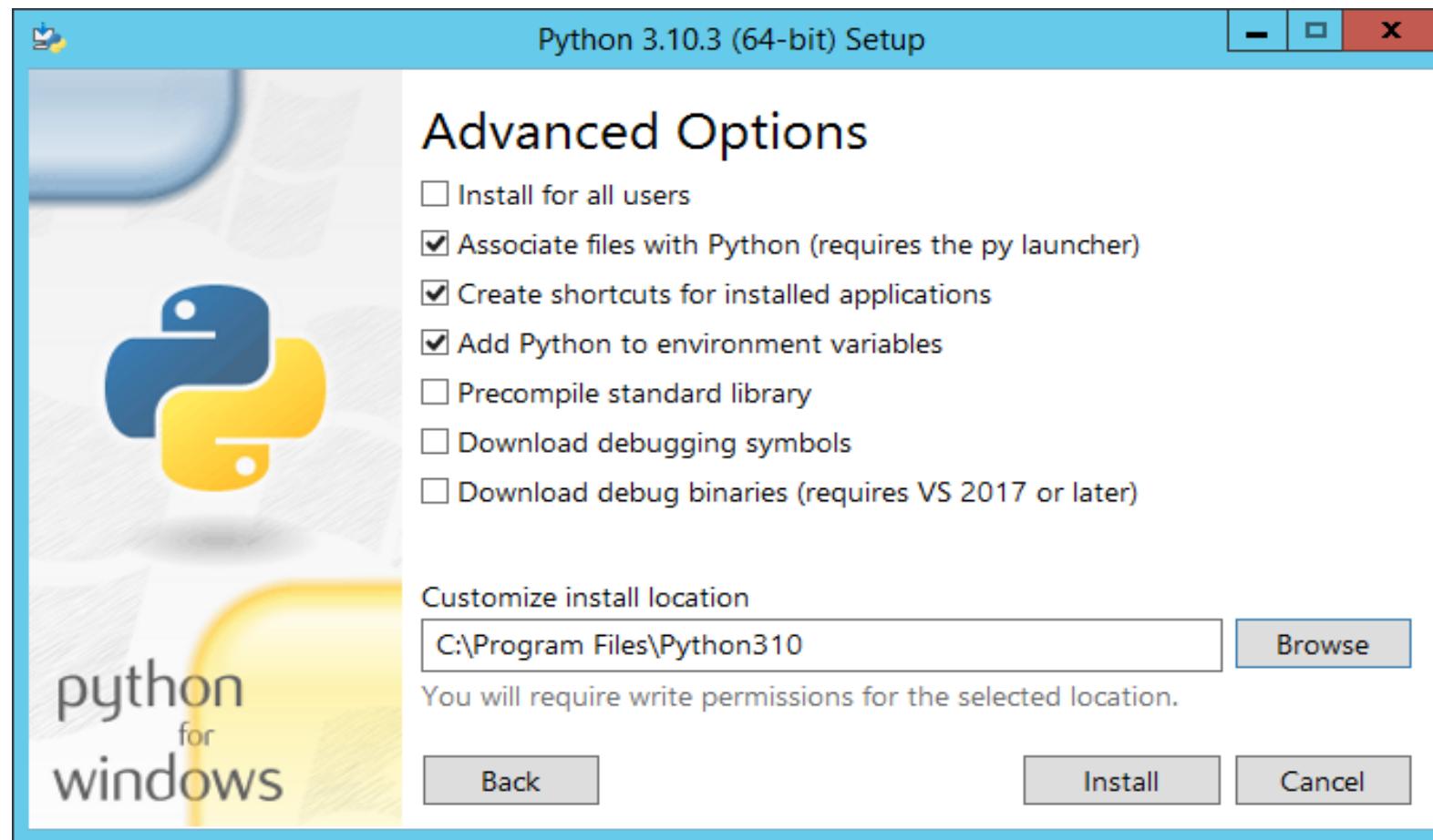
- La instalación de Python SDK, una vez seleccionado para el sistema operativo es muy sencillo de seguir.

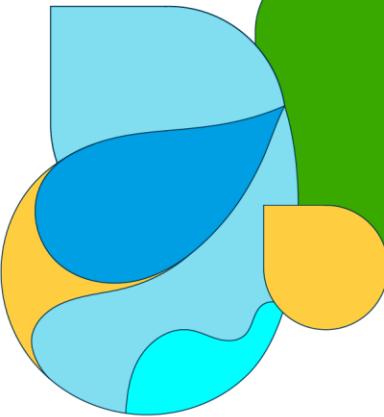


Introducción | Python SDK



Introducción | Python SDK





Introducción | Python SDK

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>python --version
Python 3.10.3

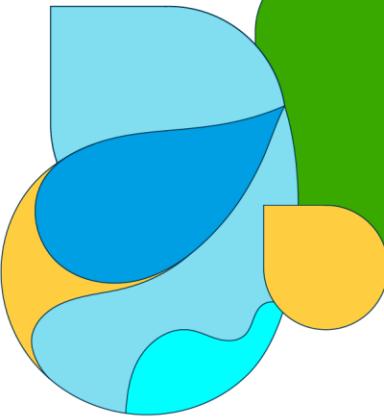
C:\Users\Administrator>set PATH=C:\Program Files\Python310;%PATH%

C:\Users\Administrator>python --version
Python 3.10.3

C:\Users\Administrator>
```

Introducción | Python Interactivo

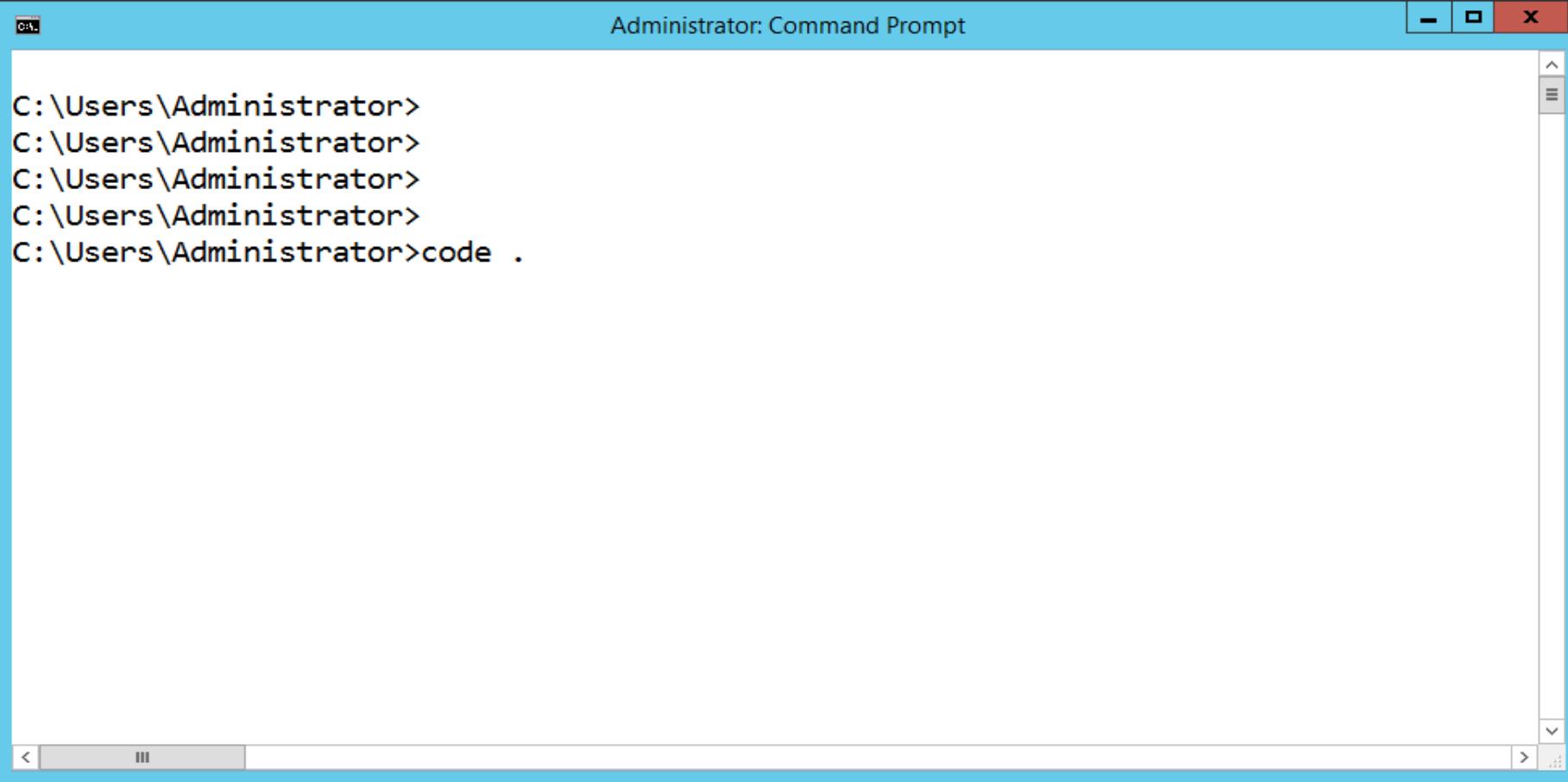
```
C:\Python>
C:\Python>python
Administrator: Command Prompt - python
Python 3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)] on
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> print ("Hola Mundo")
Hola Mundo
>>>
>>> 100 + 200
300
>>>
>>> num=100+200
>>> print (num)
300
>>> num
300
>>>
```



Introducción | Visual Studio Code

- VSC Visual Studio Code.
- Editor multiplataforma.
- Código abierto.
- Altamente extendible y configurable a través de complementos (plugins).
 - Cuenta con una gran cantidad de complementos para Python, los cuales permiten desde resaltar las palabras reservadas del lenguaje hasta depuración.

Introducción | Visual Studio Code



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The window is blue with a white background. It displays the following text:

```
C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>
C:\Users\Administrator>code .
```

The window has standard Windows controls at the top right: minimize, maximize, and close. On the left side, there are vertical scroll bars and a small icon. At the bottom, there is a navigation bar with icons for back, forward, and search.

Introducción | VSC

The diagram illustrates the Visual Studio Code (VSC) interface with numbered callouts:

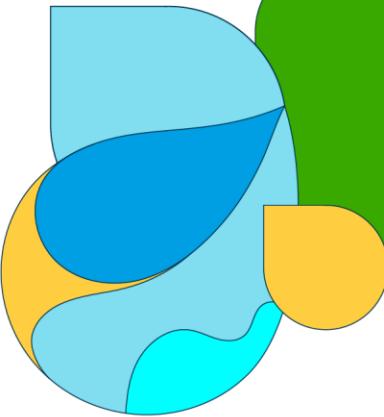
- 1**: Explorer icon in the sidebar.
- 2**: Python code in the editor:

```
1 import sys
2
3 def saludo(mensaje):
4     print(mensaje)
5     print(sys.version)
6
7 saludo("Hola Mundo Python")
```

- 3**: Terminal output:

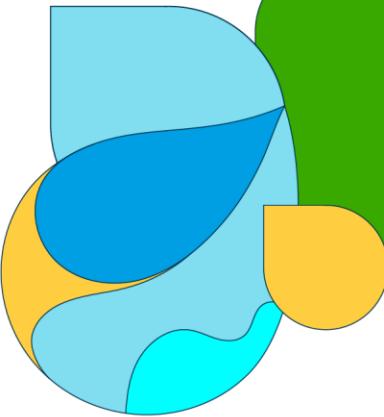
```
C:\Users\Administrator\mis_proyectos>python Saludo.py
Hola Mundo Python
3.5.4 (v3.5.4:3f56838, Aug  8 2017, 02:17:05) [MSC v.1900 64 bit (AMD64)]
```

- 4**: Terminal icon in the sidebar.
- 5**: Outline icon in the sidebar.
- 6**: Projects icon in the sidebar.



Introducción | Documentación

Documentación	Link
Python 3.10.4 Documentation.	https://docs.python.org/3/
Python 2.7.18 Documentation.	https://docs.python.org/2.7/
Beginner's Guide to Python.	https://wiki.python.org/moin/BeginnersGuide
Download Python 3.10.4 Documentation.	https://docs.python.org/3/download.html
Ecosistema de software de código abierto basado en Python para matemáticas, ciencias e ingeniería.	https://scipy.org/
Conferencias y reuniones.	https://pyvideo.org/
Comunidad Python.	https://www.python.org/community/lists/



Introducción | Versiones

- Python 2.x vs Python 3.x
- Python 2.7 llegó a su fin en el año 2020.
- No habrá versión 2.8
- Existen herramientas para la ayuda en la migración de Python 2 a Python 3.
- El siguiente link te puede ser útil para ver el registro de cambios del lenguaje Python.
 - <https://docs.python.org/3/whatsnew/changelog.html>

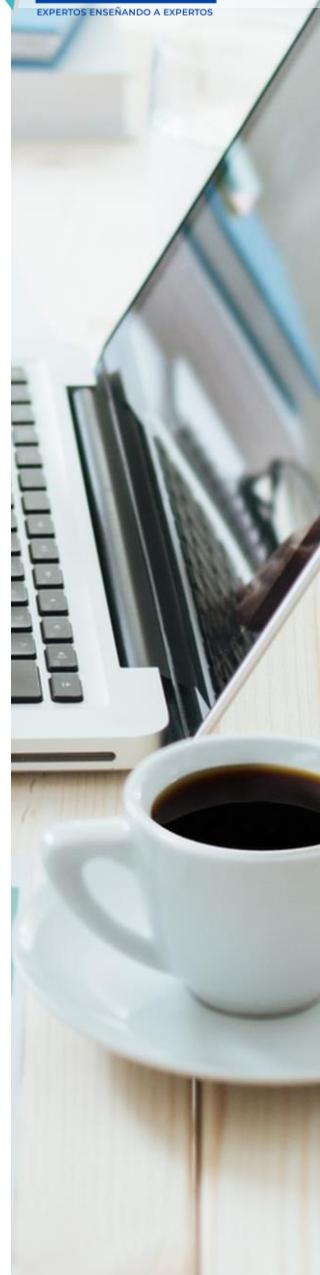
Resumen del capítulo

- Existen varias formas de trabajar con Python dependerá del tipo de desarrollo, las herramientas a descargar, instalar y configurar para sentirse a gusto como desarrollador Python.

IDEs	Editores	Frameworks
<u>Eclipse + PyDev</u>	<u>Visual Studio Code</u>	<u>Django</u>
<u>Visual Studio</u>	<u>Sublime Text</u>	<u>Flask</u>
<u>PyCharm</u>	<u>Atom</u>	<u>Kivy</u>
<u>Thonny</u>	<u>GNU Emacs</u>	
	<u>Vi/Vim</u>	
	<u>Notepad++</u>	

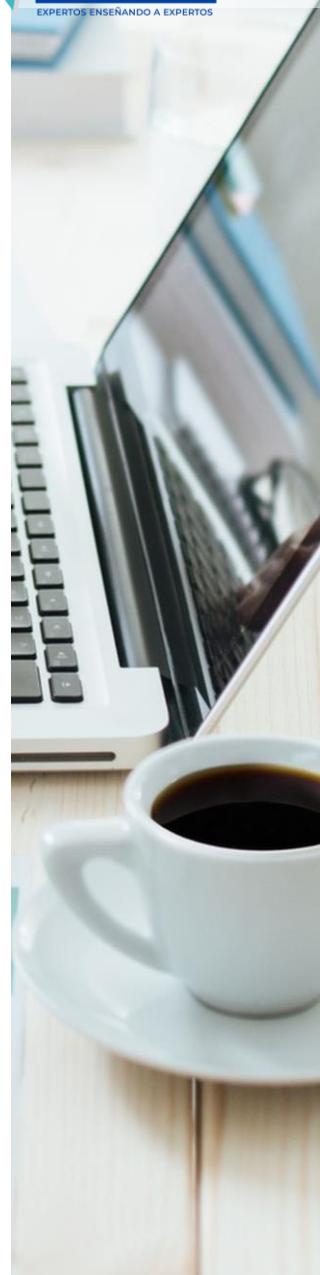
Práctica: Tendencias

- Utiliza el siguiente link: <https://trends.google.com.mx/trends/?geo=MX>
- Realiza un comparativo entre los lenguajes de programación Python, C#, Java, C y C++.
- Utiliza el siguiente link: <https://www.amazon.com.mx/>
- Realiza un comparativo entre los libros referentes a Python, C#, Java, C y C++.
- Comenta tus observaciones respecto a las tendencias y recursos sobre Python.



Práctica: Ambiente de Desarrollo

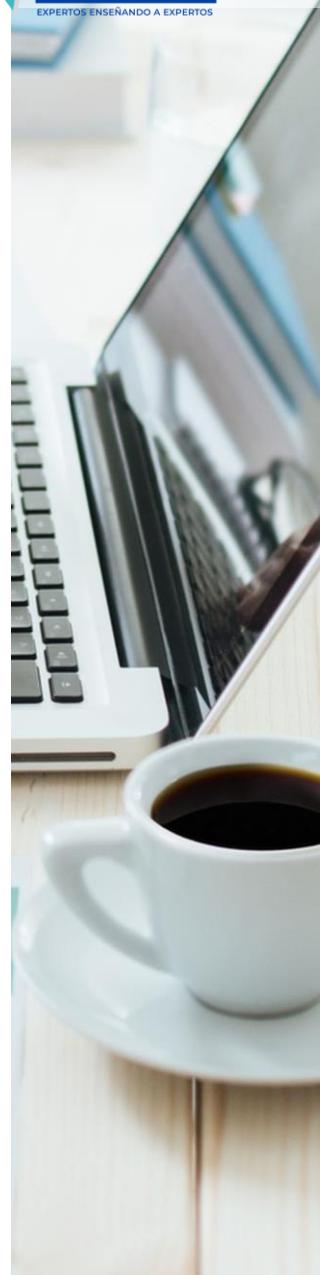
- Verifica la versión de Python que tienes instalada en la máquina asignada al curso: `python -V` o `python --v`
- En caso de no tener el producto instalado, puedes descargarlo del siguiente link: <https://www.python.org/downloads/>



Práctica: Hola Mundo

- Crea una carpeta/directorio de trabajo con el nombre de ws_curso: mkdir ws_curso
- Inicia el REPL/Consola Python en cualquier terminal de comandos: python -q
- Escribe el código en la consola Python:

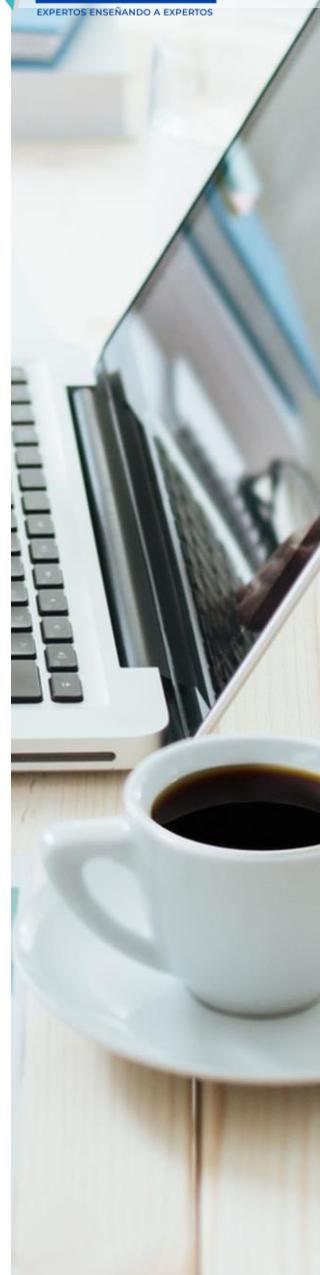
```
print("Hola Mundo Python")
import sys
print (sys.version)
exit()
```



Práctica: Hola Mundo

```
C:\> Administrator: Command Prompt
D:\MaterialPythonProgrammingA\codigos>
D:\MaterialPythonProgrammingA\codigos>
D:\MaterialPythonProgrammingA\codigos>
D:\MaterialPythonProgrammingA\codigos>python -q
>>>
>>> print ("Hola Mundo Python")
Hola Mundo Python
>>>
>>> import sys
>>> print (sys.version)
3.10.3 (tags/v3.10.3:a342a49, Mar 16 2022, 13:07:40) [MSC v.1929 64 bit (AMD64)]
>>>
>>> exit()

D:\MaterialPythonProgrammingA\codigos>
```



Referencias Bibliográficas

- Tiobe: <https://www.tiobe.com/tiobe-index/>
- The Top Programming Languages 2019:
<https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019>
- PYPL PopularY of Programming Language:
<http://pypl.github.io/PYPL.html>
- Python supera a R y SAS en la encuesta de herramientas de análisis:
<https://www.informationweek.com/strategic-cio/team-building-and-staffing/python-beats-r-and-sas-in-analytics-tool-survey/d/d-id/1335694? mc=rss x iwr edt aud iw x x-rss-simple>

Referencias Bibliográficas

- Documento para migrar Python de la versión 2 a 3:
http://ptgmedia.pearsoncmg.com/imprint_downloads/informat/promotions/python/python2python3.pdf
- PythonEditors: <https://wiki.python.org/moin/PythonEditors>
- Comparison of integrated development environments:
https://en.wikipedia.org/wiki/Comparison_of_integrated_development_environments#Python
- 10 Best Python Frameworks For Web Development, by Ambika Choudhury: <https://analyticsindiamag.com/10-best-python-frameworks-for-web-development/>

Unidad temática 5

Python Essentials para el
desarrollo de software

Objetivos:

- Usar la consola de Python para codificar y ejecutar código.
- Listar los tipos de comentarios de Python.
- Usar variables con tipado dinámico.
- Conocer los tipos de asignación que tiene Python.
- Enumerar los tipos de datos base con los que cuenta Python.
- Comprender el tipo de sangría para definir bloques de código.
- Usar los operadores Python en una expresión.
- Utilizar las funciones print() & input().

Comentario de una sola línea

```
....
```

Comentarios multilínea
línea 1
línea 2

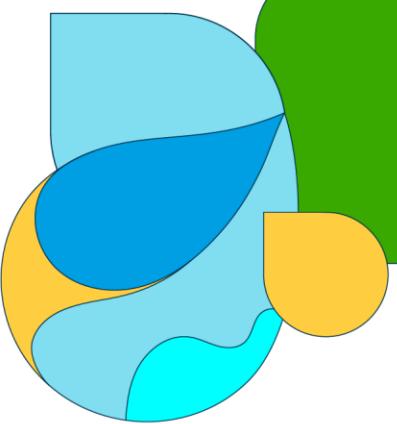
```
....
```

```
'''
```

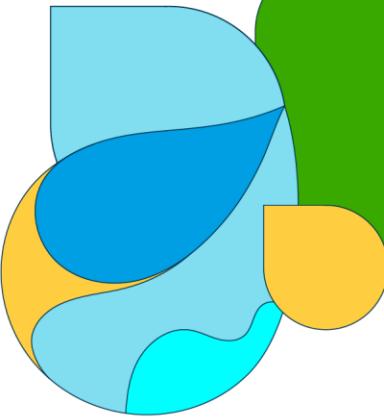
Comentarios multilínea
línea 1
línea 2

```
'''
```

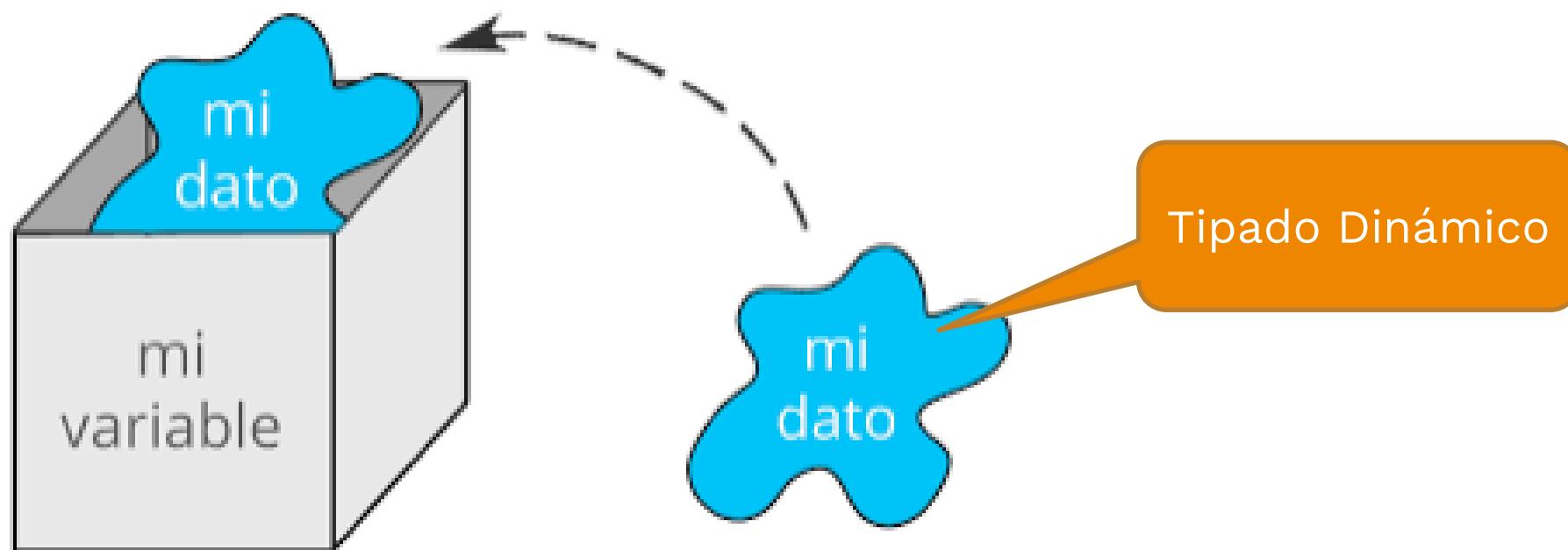
Comentarios |

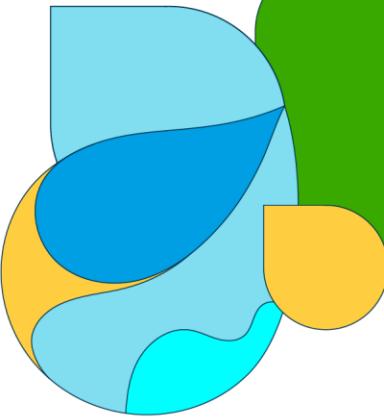


```
Administrator: Command Prompt - python -q
D:\w_python>python -q
>>> # En el ejemplo se calcula el área y perímetro de un círculo de radio 10
>>> import math
>>> print (math.pi * 10 **2) # Área
314.1592653589793
>>> print (math.pi * 10 * 2) # Perímetro
62.83185307179586
>>>
```



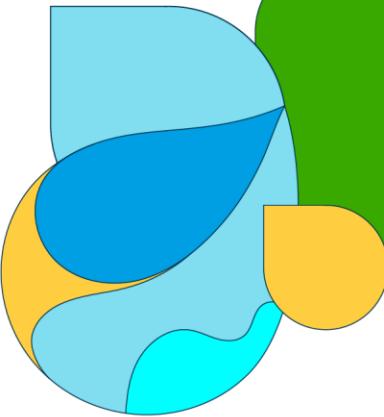
Variables





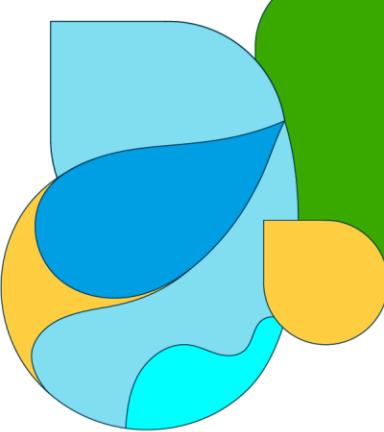
Palabras Reservadas | Keywords

False	class	finally	is	return
None	continue	for	lambda	try
True	def	from	nonlocal	while
and	del	global	not	with
as	elif	if	or	yield
assert	else	import	pass	
break	except	in	raise	



Asignación

- | | |
|--------|---|
| lvalue | = valor o expresión |
| lvalue | op= valor o expresión # Equivalente a: |
| lvalue | = lvalue op (valor o expresión) |



Asignación Simultánea

`var1, var2, ..., varn = valor1, valor2, ..., valorn`

`inteligente, lindo, divertido, sabio = "John", "Paul", "Ringo", "George"`

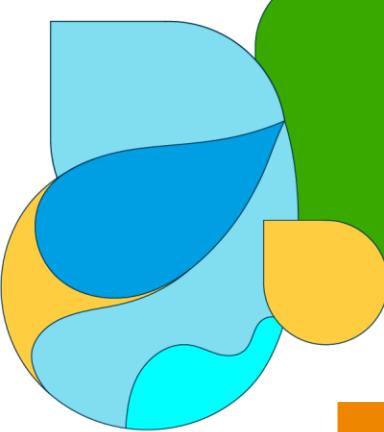
Equivalente a:

`inteligente= "John"`

`lindo= "Paul"`

`divertido = "Ringo"`

`sabio = "George"`



Tipos de Datos

str

int

float

complex

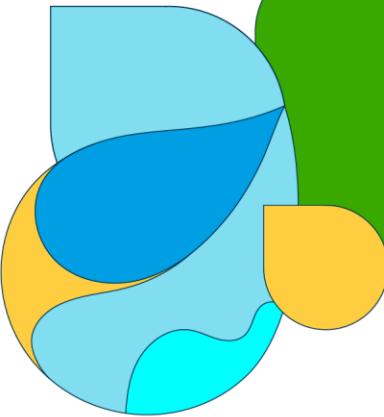
list

tuple

dict

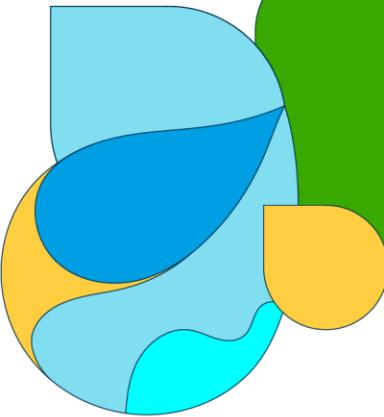
set

bool



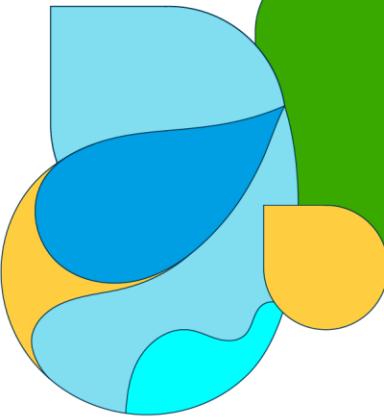
Tipos de Datos

boolean	bool	True o False.
integer	int	-1, 0, 1, 5, 0xffee, 0o647, 0b0101
float	float	3.1416, 0.10, 10e-1, 100E-2
string	str	'Hola', "Hola", 'Python "es un lenguaje" '
complex	complex	10+5j, 5j



Literales

int	-3, -2, -1, 0, 1, 2, 3
float	3.5, math.pi, 0.16
str	"", "Hola", "Netec México", “Hola\n\rMundo”
None	None podría ser considerada una literal, y no una palabra reservada.
bool	True y False, son consideradas literales booleanas y/o palabras reservadas.



Constantes

```
>>> import math
>>> print (math.pi)
3.141592653589793
>>> print (math.e)
2.718281828459045
>>>
>>> math.pi=100
>>>
>>> print (math.e)
2.718281828459045
>>> print (math.pi)
100
>>> print (math.pi)
100
>>> math.pi=5
>>> print (math.pi)
5
```

Instrucción del

```
>>> a="Hola"
>>> type(a)
<class 'str'>
>>> print(a)
Hola
>>> del a
>>> print(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>> type(a)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>>
```

Bloques de Código

Sangría

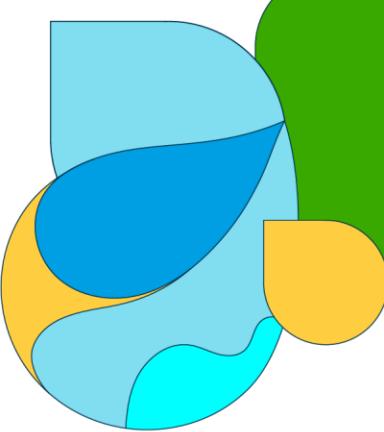
Fin del
bloque
condicional
if

```
C:\> Administrator: Command Prompt - python -q
C:\Users\Administrator\ws_python>python -q
>>>
>>> def suma(a,b):
...     print("La suma de",a,"+",b,"=",a+b)
...     return a+b
...
>>> suma(5,3)
La suma de 5 + 3 = 8
8
>>>
>>> # Otro ejemplo
>>> a=5
>>> if (a>0):
...     print("Número positivo")
... else:
...     print("Número negativo")
...
Número positivo
>>> if (true)
File "<stdin>", line 1
    if (true)
        ^
SyntaxError: expected ':'
```

Inicio de bloque, :

No forma parte
del bloque

Inicio del
bloque



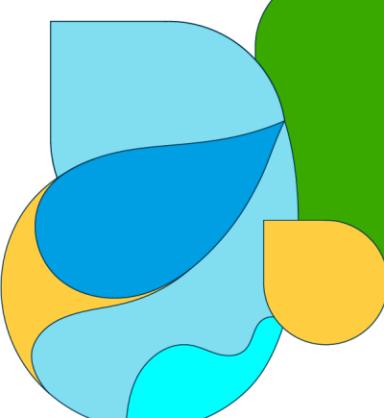
Expresiones & Operadores

¿A qué se refiere cuando se habla de expresiones?

$$2 + 3$$

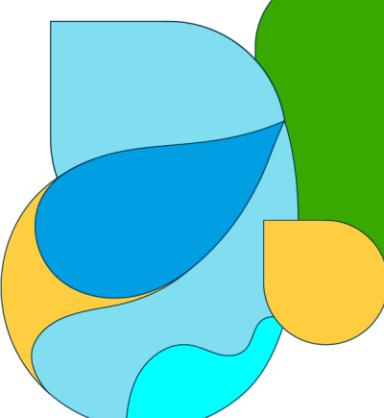
$$a = 5$$

$$b = a + 2 + 3$$



Expresiones

```
C:\Users\Administrator>python
Python 3.10.0 (tags/v3.10.0:b494f59, Oct  4 2021, 19:00:18) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
>>> 1 + 1
2
>>> 10 + 4 + 3*2
20
>>>
>>> "Hola" + " " + "Mundo" + 'Python'
'Hola MundoPython'
>>>
>>> exit
Use exit() or Ctrl-Z plus Return to exit
>>> exit()
```

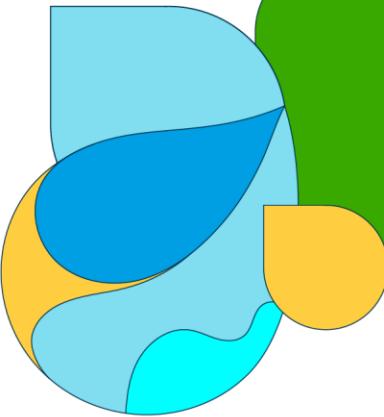


Operadores

```
>>>help('symbols')
```

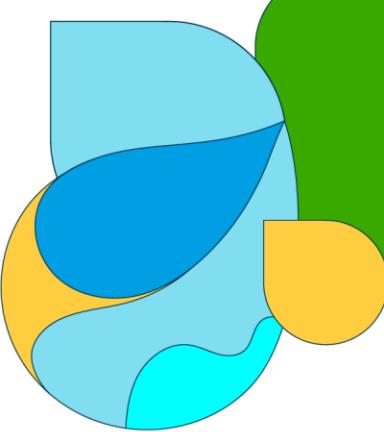
**
*/%
+ -
=

```
10 + 30  
minutos - 1  
hora*60 + minutos  
minutos / 60  
2 ** 4  
(1 + 9) * (10 - 3)
```



Operadores | Ejemplos

```
>>> hora=1
>>> minutos=60
>>> 10 + 30; minutos-1; hora*60 + minutos; minutos/60; 2**4; (1+9)*(10-3)
40
59
120
1.0
16
70
```



Operadores Aritméticos | Potencia

**

$z, a, b = 0, 2, 4$

$z = 2^{**4} #z$ queda con 16

*

$z,a,b=0,1,1$

$z= a * b$ #z queda con el valor de 1

/

$z,a,b=0,1,3$

$z= a / b$ #z queda con el valor de 0.3333

//

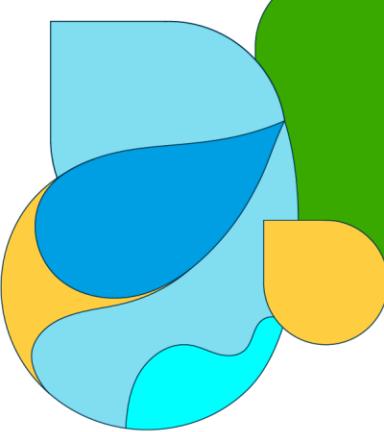
$z,a,b=0,1,3$

$z= a // b$ #z queda con el valor de 0

%

$z,a,b=0,5,3$

$z= a \% b$ #z queda con el valor de 2



Operadores Aritméticos |

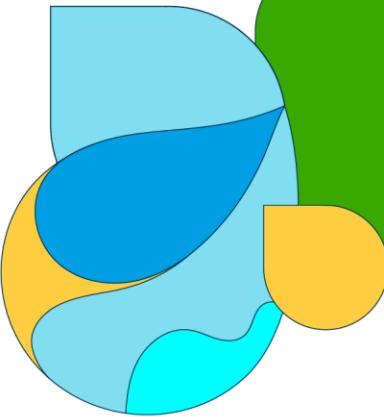
Suma & Resta

+

`z,a,b=0,1,1 #Múltiple asignación
z= a + b #z queda con el valor de 2`

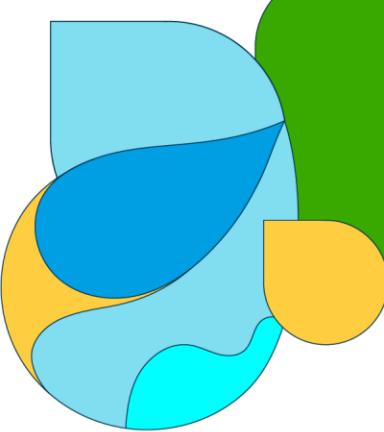
-

`z,a,b=0,1,1
z= a - b #z queda con el valor de 0`



Operadores Extendidos | +

```
a,b= "Hola", "Python"  
+      z= a + " " + c #z queda "Hola Python"  
      z= a + 3 # Error de compilación
```

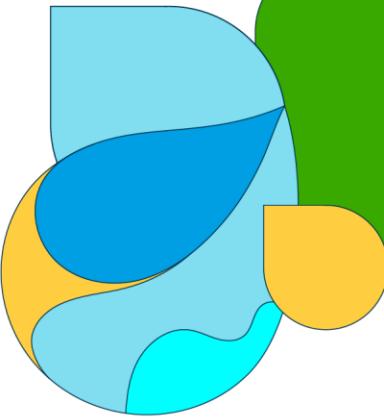


Operadores Extendidos | *

**

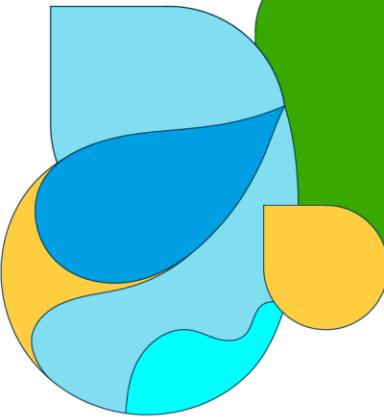
a="Hola"

z= a**2 #z "HolaHola"



Operadores Relacionales

==	a,b = 10, 10 a == b # True
!=	a,b= 10,5 a != b # True
>	a,b = 20, 10 a > b # True
<	a, b = 10,20 a < b # True



Función print()

- La función preconstruida print() despliega el valor de la expresión a la salida estándar.
- Puede tomar múltiples argumentos.
 - Se usa una coma como separador.
 - De forma predeterminada despliega los argumentos con un espacio en blanco.
 - Puede cambiar el separador por defecto, usando `sep=<expresión>`
 - También puede modificar el cambio de línea usando `end=<expresión>`

Función print()

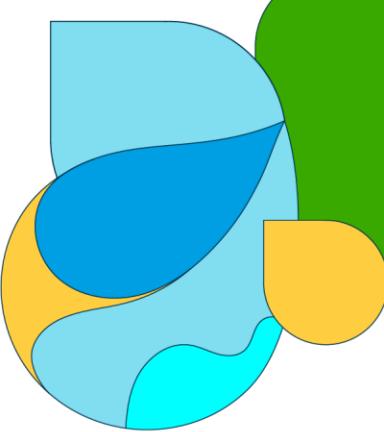
```
C:\Users\Administrator\ws_python>python -q
>>>
>>> print ("Este", "es", "un", "mensaje", sep="-", end="\n\n")
Este-es-un-mensaje

>>> print ("Estes", "es", "un", "mensaje")
Estes es un mensaje
>>>
>>> print ("Estes", "es", "un", "mensaje", ".")
Estes es un mensaje .
>>>
```

Función input()

```
C:\Users\Administrator\ws_python>cat input.py
def main():
    today = input("¿Qué día es hoy? ")
    print("Guauu!! hoy es", today, end=" ")
    print("asombroso.")

main()
C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>python input.py
¿Qué día es hoy? Miércoles
Guauu!! hoy es Miércoles asombroso.
```



Condicionales

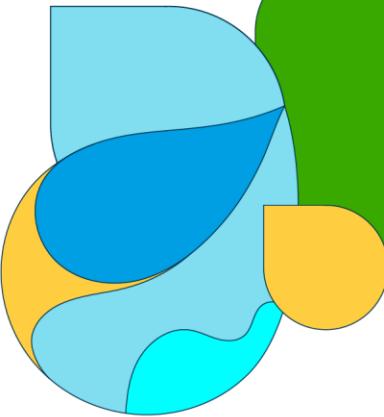
- if
- if-else
- if-elif-else

```
if expresión_booleana:  
    cod1()  
    cod2()  
    cod3()
```

```
if expresión_booleana:  
    cod1()  
    cod2()  
else:  
    cod3()  
cod4()
```

```
if expresión_booleana:  
    cod1()  
    cod2()  
elif otra_expresión_booleana:  
    cod3()
```

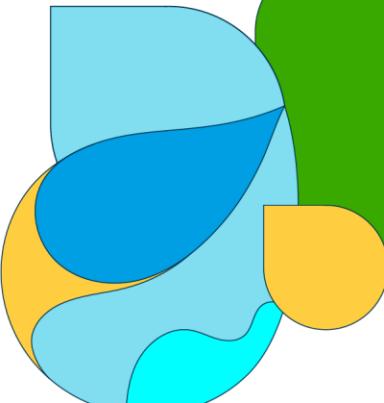
```
    cod4()  
else:  
    cod5()  
    cod6()  
cod7()
```



Condicionales | Ejemplo 1

```
edad = int(input('¿Cuántos años tiene? '))

if edad >= 21:
    print('Ahora puede votar y beber alcohol en USA.')
elif edad >= 18:
    print('En USA, ahora puede votar, pero no puede beber alcohol.')
else:
    print('En USA, no puede votar ni beber alcohol.')
```

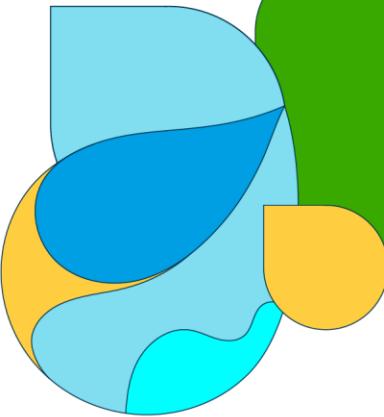


Condicionales | Ejemplo 2

```
edad = int(input('¿Cuántos años tienes? '))

if input('¿Eres ciudadano americano? [S|N] ').lower() == 's':
    is_citizen = True
else:
    is_citizen = False

if edad >= 21 and is_citizen:
    print('Puedes votar y beber alcohol')
elif edad >= 21:
    print('Puede beber alcohol, pero no puedes votar.')
elif edad >= 18 and is_citizen:
    print('Puedes votar, pero no puedes beber alcohol.')
else:
    print('No puedes votar o beber alcohol')
```

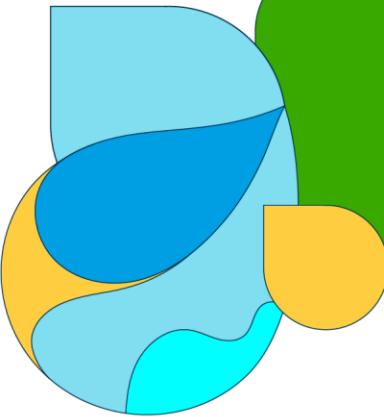


Ciclos

- while
- for

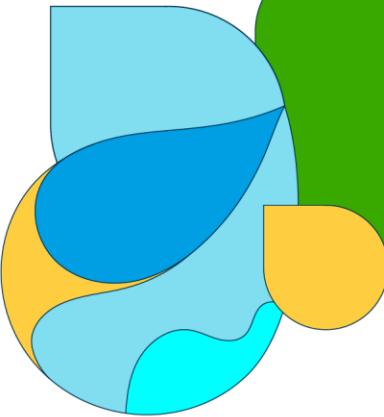
```
while expresión_booleana:  
    # Bloque de código  
    exp1  
    exp2
```

```
for variable in <lista de valores o rango o cadenas de caracteres>:  
    # Bloque de código  
    exp1  
    exp2
```



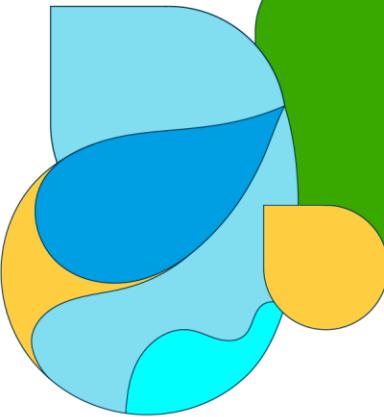
Ciclos | while | Ejemplo

```
>>>  
>>> var=0  
>>> while var < 5:  
...     print ("\t",var)  
...     var += 1  
  
...  
0  
1  
2  
3  
4  
>>>
```



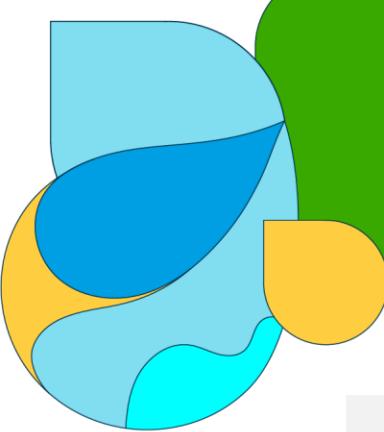
Ciclos | for | Ejemplo

```
>>>
>>> for x in 1, 2, 3:
...     print(x)
...
1
2
3
>>> for var in "Python":
...     print (var)
...
P
y
t
h
o
n
>>>
```



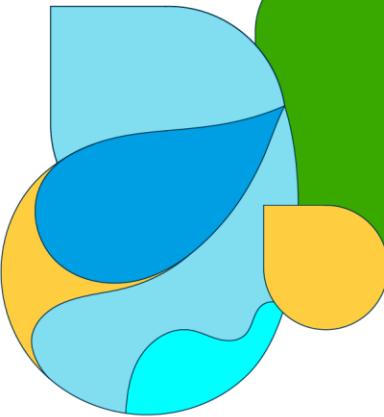
Ciclos | for | Ejemplo

```
>>>
>>> for x in range(5): # elementos en el rango 0, 1, 2, 3 y 4
...     print (" ", x)
...
0
1
2
3
4
>>> for x in range(0,3): # elementos en el rango 0, 1 y 2
...     print(" ",x)
...
0
1
2
>>>
```



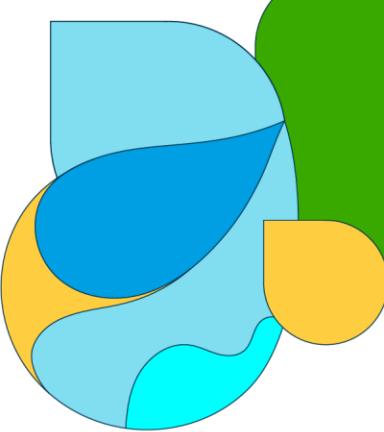
Funciones

```
def nombre_función([lista de argumentos
opcionales]):  
    # El contenido de la función debe ir identado,  
    # indicando el bloque de la función  
    # Realizar algo  
    algo()  
  
    # Ya no es parte del cuerpo de la función.  
    hacer_algo()
```



Funciones | Ejemplo

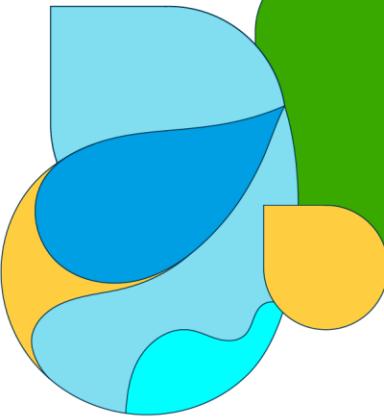
```
D:\>python -q
>>>
>>> # Saludo Personalizado
>>> # Python Programming
>>> def saludo():
...     nombre=input('¿Cuál es tu nombre?')
...     print ("Hola, ",nombre, "!")
...
>>> def main():
...     saludo()
...
>>> main()
¿Cuál es tu nombre?Leticia
Hola, Leticia !
```



Ámbito

- El **ámbito** de una variable es la parte del programa donde la variable es accesible.
- Por su **ámbito** las variables se clasifican: **locales & globales**.
- Si las variables no se encuentran previamente definidas antes de usarlas Python genera un error:

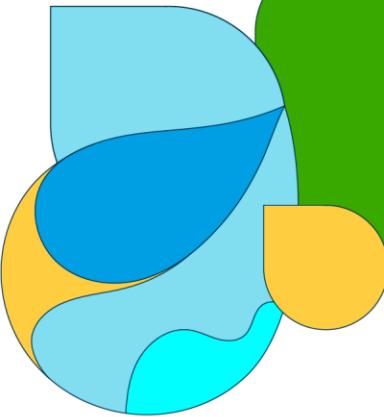
NameError: name <variable> is not defined



Scope | Ejemplo

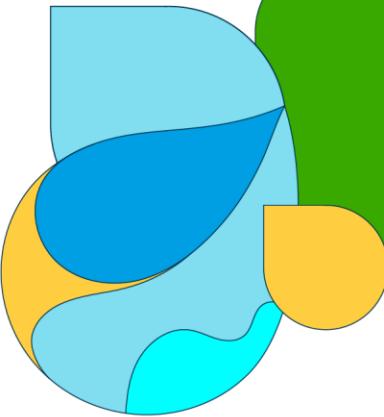
```
a= 100
def fun():
    print (a)
    a= 500
    print (a)
def gen():
    print (a)

gen() # la salida es 100
fun() # UnboundLocalError: local variable 'a' referenced before
      assignment
```



Variables Globales

- Las variables globales en Python se encuentran fuera de los bloques.
 - clases, métodos o funciones.
- Python busca primero **localmente** en el bloque si no encuentra la variable, entonces busca **globalmente**.
- Las variables globales pueden ser **referenciadas**, pero no **modificadas**.
- Para modificar variables globales dentro de un bloque se debe indicar explícitamente que se trata de una variable **global**.
 - Palabra clave: **global**.
 - No existe la palabra local, pero si se tiene la palabra **nonlocal**.

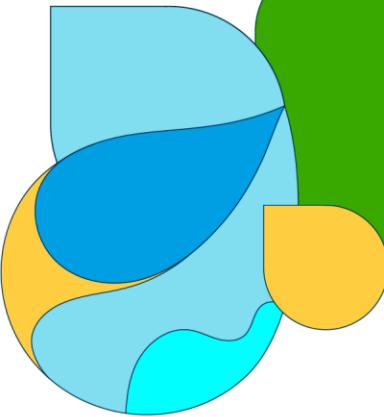


Variables Globales | Ejemplo

```
# Define una función
def fun():
    global a
    a = "En la función"

# Invoca a la función
fun()

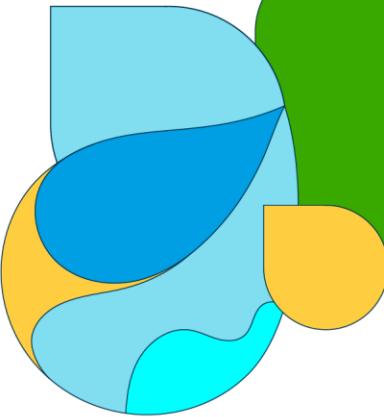
# a es una variable global tiene alcance fuera de la función
print(a)
```



Parámetros Posicionales & Defecto

- Python permite asignar por defecto los valores de los parámetros utilizando: def, nombre de la función y la lista de parámetros inicializados.
- Sintaxis:

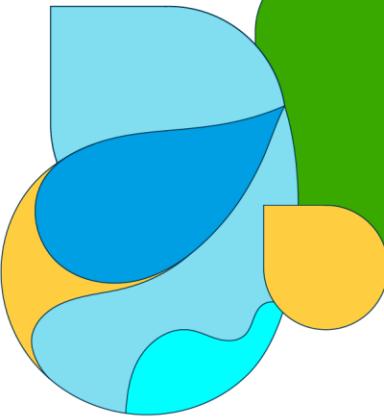
```
def nombre_función(parámetro = valor_por_defecto):  
    hacer_algo(parámetro)
```



Devolución

- Para que una función devuelva un valor, se usa la palabra reservada return.

```
def suma(num1, num2, num3=0, num4=0, num5=0):  
    res = num1 + num2 + num3 + num4 + num5  
    return res  
  
def main():  
    s = suma(1, 2)    # s queda con el valor de 3  
    print(s)  
    s = suma(s, 3)    # s queda con el valor de 6  
    print(s)  
    s = suma(s, 4)    # s queda con el valor de 10  
    print(s)  
  
main()
```



Funciones Preconstruidas

`int()`

`bin()`

`float()`

`abs()`

`min()`

`max()`

`round()`

`len()`

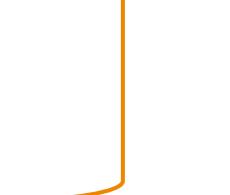
Función Enumerate

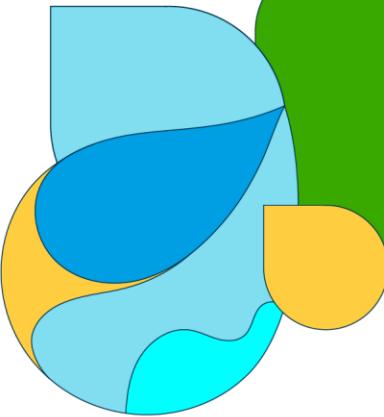
```
def main():
    i = 1
    for item in ['a','e','i','o','u']:
        print(i, item, sep='. ')
        i += 1

main()
```

```
def main():
    for item in enumerate(['a','e','i','o','u'], 1):
        print (item[0], item[1],sep='. ')

main()
```

- 
- 
1. a
 2. e
 3. i
 4. o
 5. u



Generadores | Sin

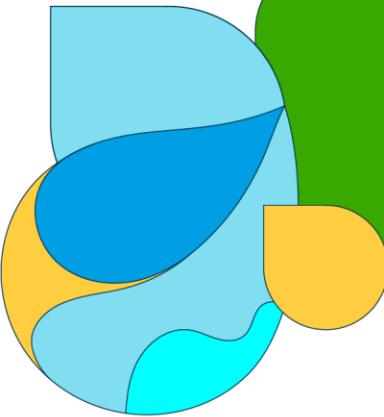
```
import random

def genera(inicio,fin,num):
    numeros = []
    for numero in range(num):
        numeros.append(random.randint(inicio,fin))
    return numeros

numeros = genera(1,100,5)

print('Iterando a través de la lista:')
for num in numeros:
    print(num)

print('Nuevamente iterando en la lista: ')
for num in numeros:
    print(num)
```



Generadores | Con

yield

```
import random

def genera(inicio,fin,num):
    for numero in range(num):
        yield random.randint(inicio,fin)    # yield

numeros = genera(1,100,5)

print('Iteramos en el generador:')
for num in numeros:
    print(num)

print('Nuevamente iteramos en el generador:')
for num in numeros:
    print(num)
```

Resumen del capítulo

- Se ha repasado: el uso de variables, obtención de datos del usuario, desplegar el resultado de expresiones y el uso de los diferentes operadores, su **jerarquía** y su **asociatividad**.
- Python cuenta con varios tipos para representar datos, números enteros, números reales, números complejos, caracteres, cadenas de caracteres, booleanos, conjuntos, tuplas, listas y diccionarios.
- Python tiene con varias funciones de conversión de tipos como: **int()**, **bin()**, **oct()**, **hex()**, **float()**, **list()**, **str()**, **bool()**, **tuple()**, **set()**, **dict()**, **complex()**, etc.

Resumen del capítulo (Cont.)

- Para verificar el tipo de datos Python tiene a la función `type()`.
- Python tiene las funciones preconstruidas: `round()`, `sum()` & `pow()`.
- El módulo de Python para usar funciones logarítmicas y trigonométricas se llama `math`.
- El módulo de Python para matemáticas con números complejos se llama `cmath`.
- La función para devolver un enumerado, se llama `enumerate(iterable, inicio=0)`.
- Las funciones típicas de entrada y salida son: `input()` & `print()`

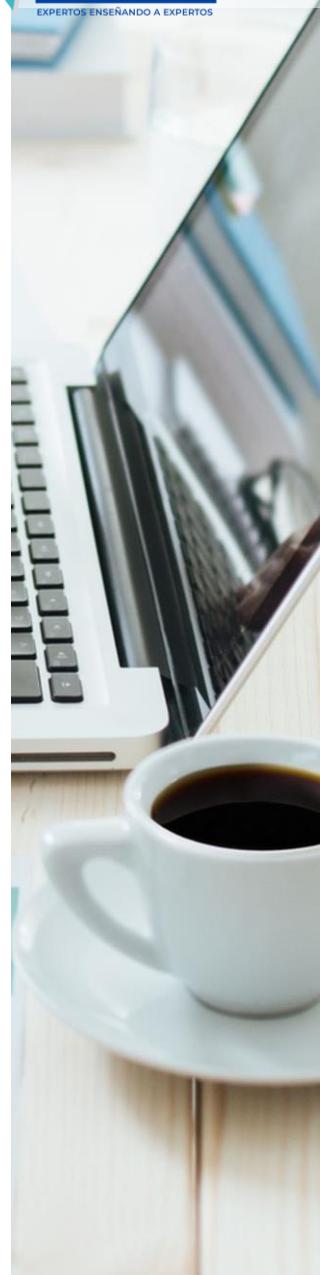
Resumen del capítulo (Cont.)

- Cuando se define una función que se llama así misma se le conoce como función recursiva.
- Cuando la función devuelve un valor en la definición de dicha función se usa la palabra `return`.
- Cuando un generador devuelve un valor en la definición del generador se usa la palabra `yield`.
- Los valores por defecto en las funciones se colocan en las definiciones de estas; `def fun(variable=valor):`

Práctica: Explorando Tipos de Datos

- El objetivo de esta práctica es utilizar la función preconstruida `type()` para explorar los diferentes tipos de datos con los que cuenta el lenguaje Python, no te preocupes si aún no estás familiarizado con todos los tipos.

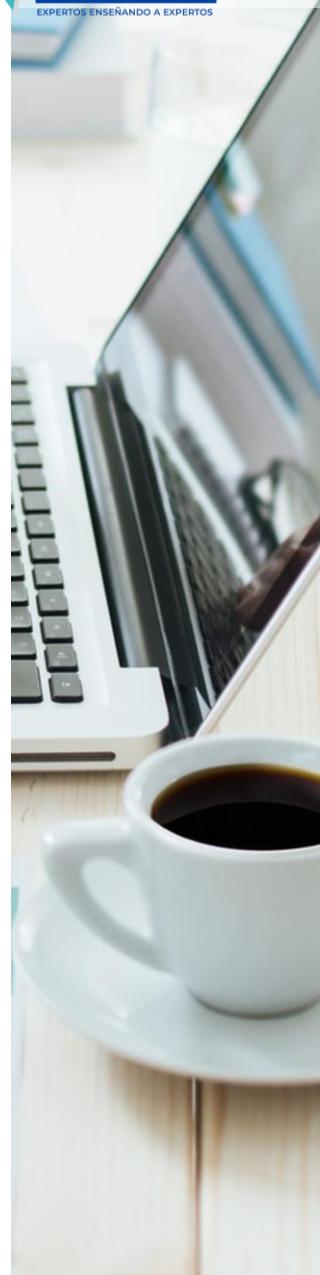
`type(<expresión>)`



Práctica: Explorando Tipos de Datos (Cont.)

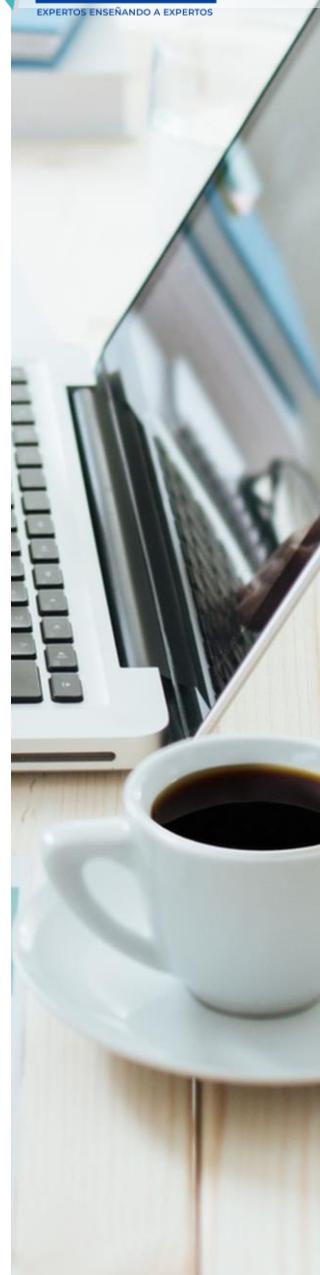
Desde una terminal o consola de Python (Python REPL).

- a. Inserta `type(3)` y presiona [Enter].
- b. Inserta `type(3.1)` y presiona [Enter].
- c. Inserta `type("3")` y presiona [Enter].
- d. Inserta `type('3')` y presiona [Enter].
- e. Inserta `type("pizza")` y presiona [Enter].
- f. Inserta `type(1==1)` y presiona [Enter].



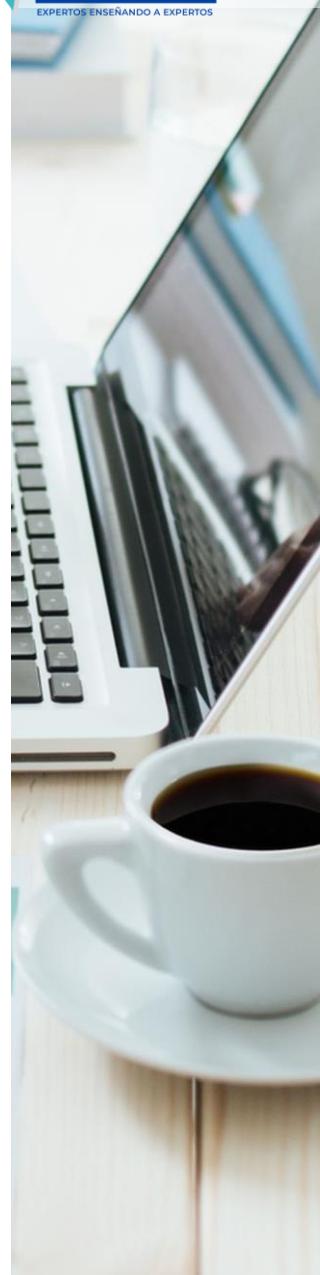
Práctica: Explorando Tipos de Datos (Cont.)

- g. Inserta `type(['1','2','3'])` y presiona [Enter].
 - h. Inserta `type([1,2,3])` y presiona [Enter].
 - i. Inserta `type({"1","2","3"})` y presiona [Enter].
 - j. Inserta `type({"1": "Hoal", "2":"Adios", "3":"Buen día"})` y presiona [Enter],
2. ¿Qué otro tipo de dato Python conoce que no aparece en la lista anterior? Comenta en la clase.



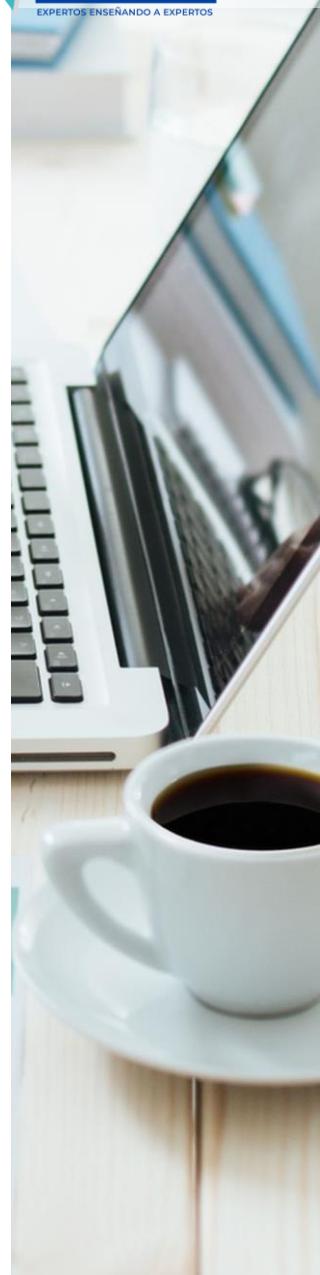
Práctica: Un Script Básico

1. Inicia el VSC, crea un nuevo documento de nombre `p2_2.py` en una carpeta dedicada a tus prácticas `ws_curso`.
2. Crea una variable de nombre `hoy` que contenga el día actual de la semana como literal de tipo texto, es decir, entre comillas dobles "Lunes", "Martes", etc.
3. Utiliza la función `print()` para desplegar a la salida estándar el valor de la variable.
4. Ejecuta tu script `p2_2.py`
5. ¿Conoces alguna otra forma de ejecutar el script de la práctica?
Comenta en la clase.



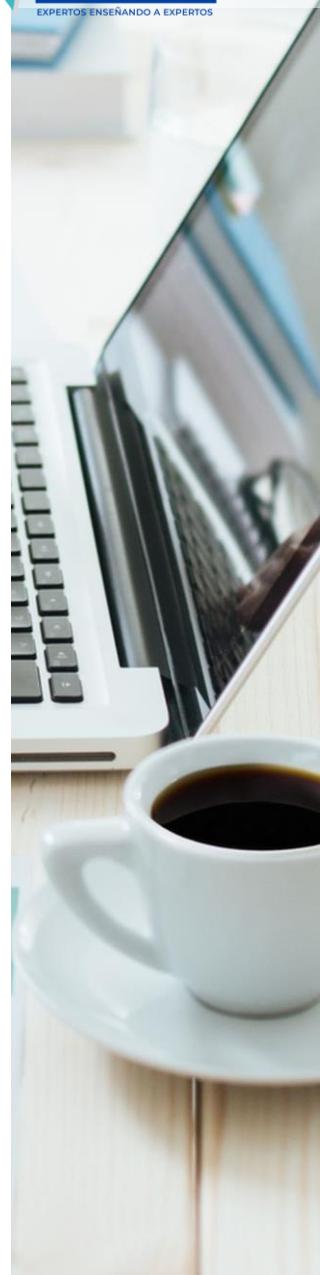
Práctica: Interactuando con el usuario

1. Inicia VSC, cree un nuevo archivo de nombre `p2_3.py` en la carpeta `ws_curso`.
2. Solicita al usuario ingresar el día de la semana y almacena el resultado en una variable de nombre **hoy**.
3. Utiliza la función `print()` para desplegar a la salida estándar el valor de la variable hoy.
4. Ejecuta tu código varias veces, dándole valores diferentes a tu script.
5. ¿Recibes algún error si no se ingresa ningún valor al script o si se ingresa un número?



Práctica: Interando

- Inicia el VSC, cree un nuevo archivo de nombre `p2_4.py` en la carpeta `ws_curso`.
- Crea una función llamada `iter()`, que reciba tres parámetros, el inicio, el final y el paso. Los tres parámetros y el valor del paso por defecto es uno.
- La función debe desplegar la iteración del valor de inicio, al valor final incluyendo el valor final de uno en uno.
- Toda la salida de la función del punto anterior debe de ser en una sola línea.
- Invoca la función con los siguientes valores: `iter(1,10)`, `iter(1,10,2)`, `iter(1,10,5)`.
- Invoca la función de tal manera que la línea del resultado sea: `5 4 3 2 1 0`.
- Invoca la función de tal manera que la línea del resultado sea: `-3, -2, -1, 0, 1, 2, 3`.



Referencias Bibliográficas

- Las principales diferencias entre Python 2 y 3 con ejemplos: <https://www.pythonmania.net/es/2016/02/29/las-principales-diferencias-entre-python-2-y-3-con-ejemplos/>
- Built-in Functions: <https://docs.python.org/3/library/functions.html>
- print(): <https://docs.python.org/3/library/functions.html#print>
- input(): <https://docs.python.org/3/library/functions.html#input>
- Built-in Constants: <https://docs.python.org/3/library/constants.html>

Referencias Bibliográficas

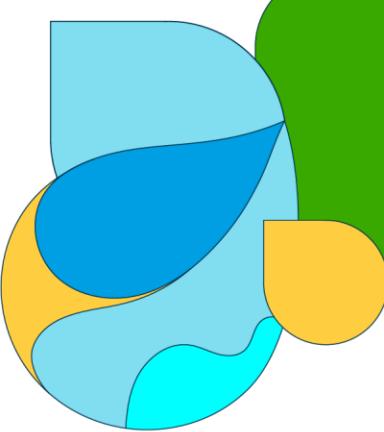
- Built-in Types: <https://docs.python.org/3/library/stdtypes.html>
- Python DataTypes:
https://www.w3schools.com/python/python_datatypes.asp
- Built-in Functions: <https://docs.python.org/3/library/functions.html>
- Defining Functions:
<https://docs.python.org/3/tutorial/controlflow.html#defineing-functions>

Unidad temática 6

Módulos y Paquetes de phyton
para el desarrollo de software

Objetivos:

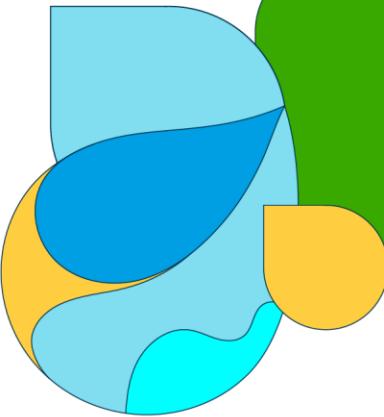
- Comprender el uso de módulos Python.
- Importar módulos de Python.
- Usar el comando pip para la gestión de paquetes Python.
- Crear programas organizados en paquetes.
- Usar los módulos de paquetes propios y de terceros.
- Crear de forma práctica un paquete distribuible.



Módulos



- Los módulos proporcionan la flexibilidad de organizar el código con cierta lógica.
- Un módulo puede definirse como un archivo de código Python.
- Archivos con extensión `.py`
- `import` & `from-import`



Sentencia import

- Para incluir un módulo en el código fuente se utiliza la palabra reservada import, o las palabras from-import.
- Se importa una sola vez, aun cuando se indique más de una vez.
- Sintaxis:
 - `import <modulo1>, <modulo2>,..., <modulon>`
 - `from <nombre_del_módulo> import <nombre1>, <nombre2>,...<nombre3>`
 - `from <nombre_del_módulo> import *`

import | Ejemplo

mensajes.py

```
# -*- coding: utf-8 -*-
# Función que despliega un mensaje a la salida estándar

def display(nombre):
    print("Bienvenido al curso: ", nombre)
```

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 26 12:50:36 2019

@author: Administrator
"""

import mensajes
nombre= input("Ingresa tu nombre: ")
mensajes.display(nombre)
```

from-import | Ejemplo

operaciones.py

```
def sumar(a,b):
    return a + b

def restar(a,b):
    return a - b

def multiplicar(a,b):
    return a*b

def dividir(a,b):
    return a/b
```

```
from operaciones import sumar, restar

a=int(input("Ingresa un numero: "))

b=int(input("Ingresa otro numero:"))

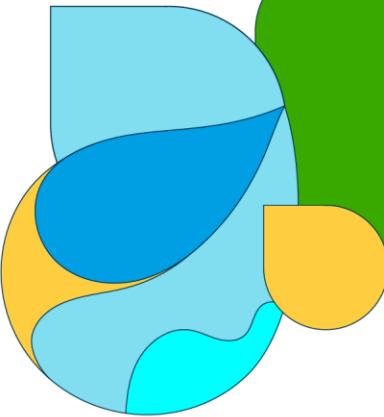
print("")

print ("sumar","",a,"","",b,"")= ", sumar(a,b) , sep="")

print ("restar","",a,"","",b,"")= ", restar(a,b), sep="")
```

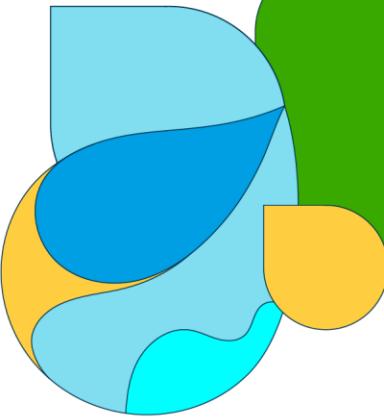
NameError: name 'multiplicar' is not defined

```
from <nombre_del_módulo> import *
```



Import .. as | Renombrar Módulos

```
import operaciones as oo
a=int(input("Ingresa un numero: "))
b=int(input("Ingresa otro numero: "))
print("")
print ("sumar(",a,",",b,")= ", oo.sumar(a,b) , sep="")
print ("restar(",a,",",b,")= ", oo.restar(a,b) , sep="")
print ("multiplicar(",a,",",b,")= ", oo.multiplicar(a,b) , sep="")
```



Función dir()

- La función preconstruida dir(<módulo>).
 - Devuelve una lista con nombres definidos en el módulo.
 - La lista contiene todos los submódulos, variables, funciones o clases definidas en el módulo especificado.

```
import operaciones  
  
lista= dir(operaciones)  
  
print (lista)  
print ()  
  
  
import json  
  
lista= dir(json)  
  
print (lista)
```

Función reload()

- `reload(<nombre del módulo>)`
- Para usar la función es necesario importar el módulo: `importlib`
- El código del módulo recargado se vuelve a compilar y a ejecutar.

```
import operaciones
lista= dir(operaciones)
print (lista)
import importlib
importlib.reload(operaciones)
```

Ámbito

Ámbito Local

Local

Se encuentran definidas dentro de una función

1

Global

No se encuentran definidas dentro de una función

2

- Ámbito: Zona o fragmento del programa donde la variable es visible.

Ámbito | Scope

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 26 18:39:53 2019

@author: Administrator
"""

nombre= "Gabriel" # Variable global
print(nombre)

def display(nombre):
    nombre="Lucía" # Variable local
    print ("¡Hola ", nombre, "!", sep="")

nombre= input("Ingresa tu nombre: ") # Variable global
print(nombre)

display(nombre)

print(nombre)
```

```
In [50]: runfile('C:/Users/Administrator/Documents/ProyectosPython/test_scope.py',
wdir='C:/Users/Administrator/Documents/ProyectosPython')
```

```
Gabriel
```

```
Ingresa tu nombre: Leticia
```

```
Leticia
```

```
¡Hola Lucía!
```

```
Leticia
```

```
In [51]:
```

¿Por qué no es
¡Hola Leticia!?

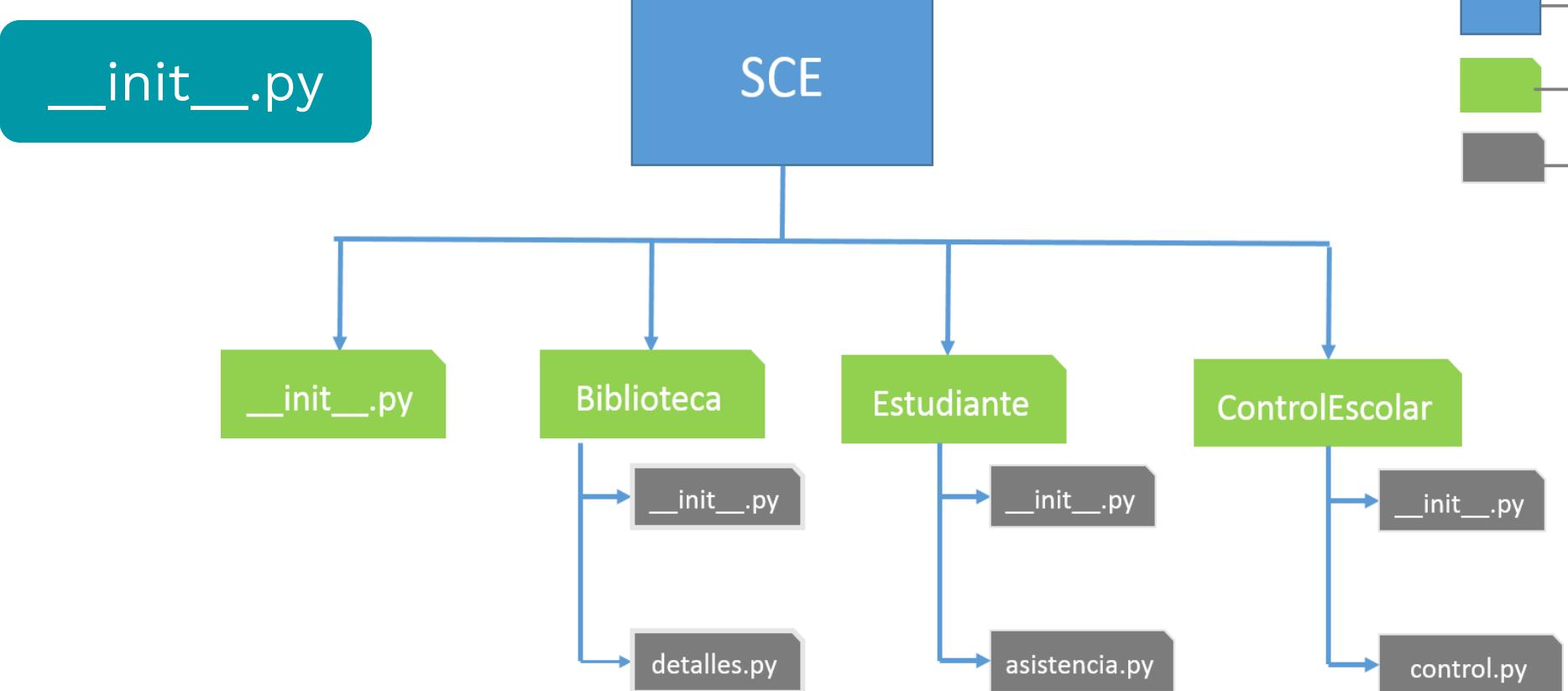
Función Principal

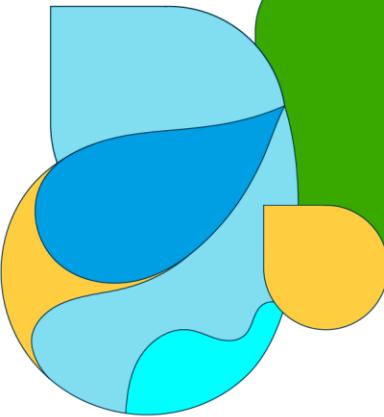
```
# Operaciones básicas aritméticas
def sumar(a,b):
    return a + b
def restar(a,b):
    return a - b
def multiplicar(a,b):
    return a*b
def dividir(a,b):
    return a/b

# Mini Test
def main():
    if __name__ == '__main__':
        print("Desde el módulo operaciones")
        print ("sumar(2,3)= ", sumar(2,3))
        print ("restar(2,3)= ", restar(2,3))
        print ("multiplicar(2,3)= ", multiplicar(2,3))
        print ("dividir(2,3)= ", dividir(2,3))
    else:
        print ("__name__ : ", __name__)
main()
```

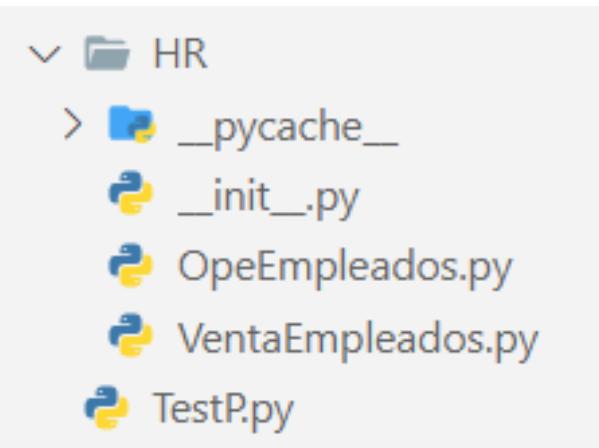
__name__

Paquetes





Paquetes | Ejemplo

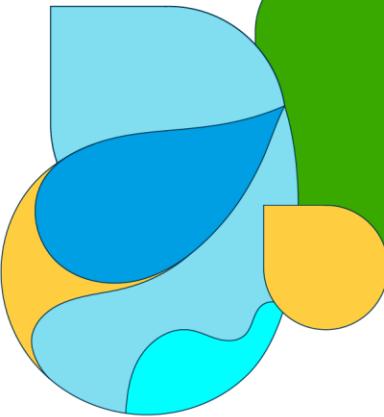


OpeEmpleados.py

```
def getOpeNombres():
    lista=["Everardo", "Jesús", "Roberto", "Marco"]
    return lista;
```

VentaEmpleados.py

```
def getVentasNombres():
    lista= ["Leticia", "Judith", "Andrea", "Verónica", "Liliana"]
    return lista
```



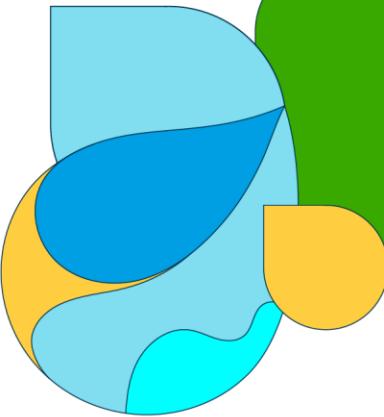
Paquetes | Ejemplo

```
from HR import OpeEmpleados # 1

import HR.VentaEmpleados # 2
from HR.VentaEmpleados import getVentasNombres as gVN # 3

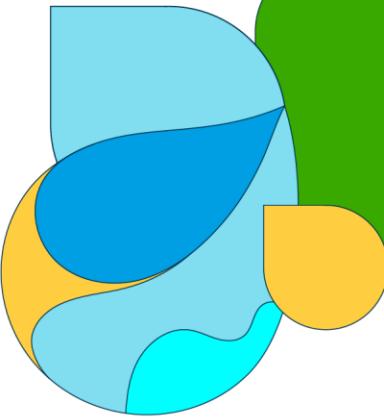
print (OpeEmpleados.getOpeNombres()) # 4
print (HR.VentaEmpleados.getVentasNombres()) # 5
print (gVN()) # 6
```

```
['Everardo', 'Jesús', 'Roberto', 'Marco']
['Leticia', 'Judith', 'Andrea', 'Verónica', 'Liliana']
['Leticia', 'Judith', 'Andrea', 'Verónica', 'Liliana']
```



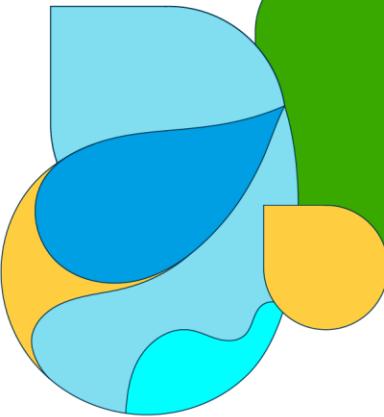
Paquetes Distribuibles

- pip: Python Intall Packages
 - Por lo general se instalan paquetes adicionales a la biblioteca estándar de Python.
 - Incluido desde la version 3.4
 - pip -V para desplegar la versión.
 - pip -h o pip --help para desplegar el uso de la instrucción.
 - pip permite instalar, actualizar, listar, y desinstalar paquetes Python.
- PyPI: Python Package Index
 - Extensa colección de paquetes que incluyen marcos de trabajo, herramientas y bibliotecas.



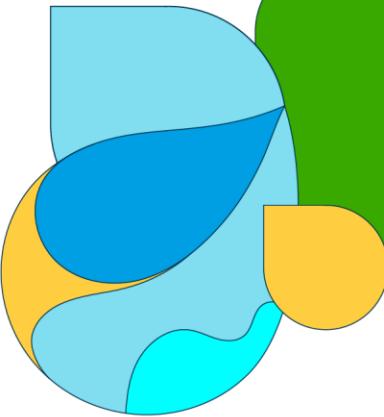
PIP | pip --help

- install: Instala paquetes.
- download: Descarga paquetes.
- uninstall: Desinstala paquetes.
- list: Lista los paquetes instalados.
- check: Verifica si los paquetes instalados tienen dependencias compatibles.
- config: Gestiona la configuración local y global.



PIP | Upgrade

- `python -m pip install upgrade pip`
 - Actualiza la versión de la instrucción pip.
 - La opción `-m` le dice que ejecute el módulo.
- `pip install <nombre_del_paquete>`
 - Instalará el paquete y las dependencias compatibles, se instala la última versión del paquete.
- `pip list`
 - Despliega la lista de nombres paquetes y versiones instalados.
- `pip show <nombre_del_paquete>`
 - Muestra los detalles o metadatos del paquete instalado.



Alternativas al Comando pip

CONDA



Módulo math

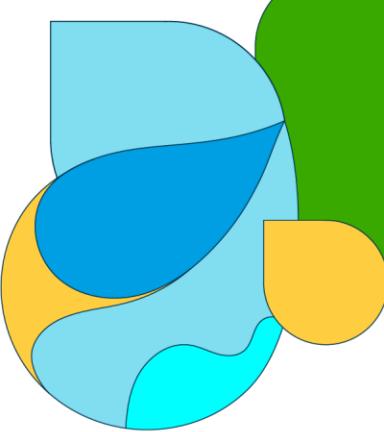
- El módulo math tiene las constantes: `math.pi` & `math.e`
- Para usarse se debe importar.
 - `import math`
- `dir(math)` o `help(math)` para documentación en línea de comandos.

Logarítmicas

Trigonométricas

Angulares

Hiperbólicas



Módulo math

ceil

floor

trunc

fabs

factorial

fmod

pow

sqrt

pi

e

Módulo random

random

randint

randrange

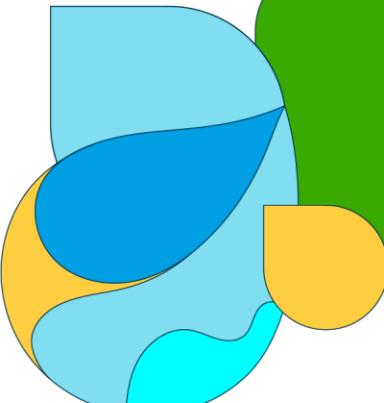
uniform

randrange

choice

shuffle

seed

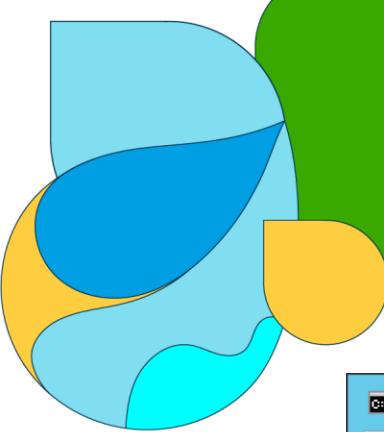


Módulo random

```
>>>  
>>> a=[1,3,5,7,9]  
>>>  
>>> import random  
>>>  
>>> random.shuffle(a)  
>>> a  
[3, 1, 5, 9, 7]  
>>> random.shuffle(a)  
>>> a  
[3, 7, 5, 1, 9]  
>>> random.shuffle(a)  
>>> a  
[7, 1, 5, 3, 9]
```

```
>>> import random  
>>>  
>>> random.seed(10)  
>>> random.randint(1,100)  
74  
>>> random.randint(1,100)  
5  
>>> random.seed(10)  
>>> random.randint(1,100)  
74  
>>> random.randint(1,100)
```

sys.path



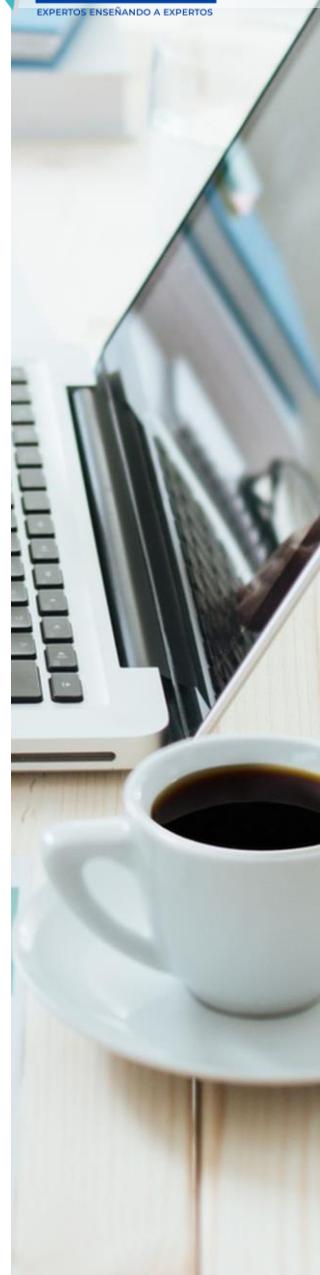
```
Administrator: Command Prompt - python -q
C:\Users\Administrator\ws_python>python -q
>>>
>>> import sys
>>>
>>> for i in sys.path:
...     print (i)
...
C:\Program Files\Python310\python310.zip
C:\Program Files\Python310\DLLs
C:\Program Files\Python310\lib
C:\Program Files\Python310
C:\Program Files\Python310\lib\site-packages
>>>
```

Resumen del capítulo

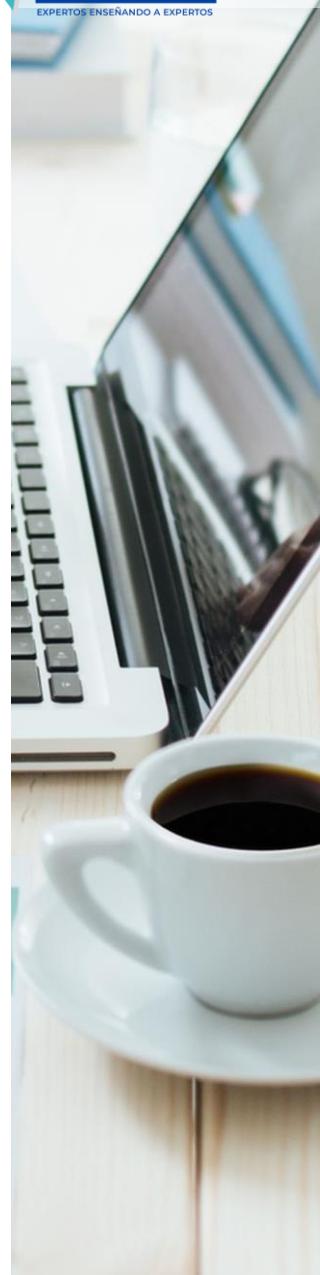
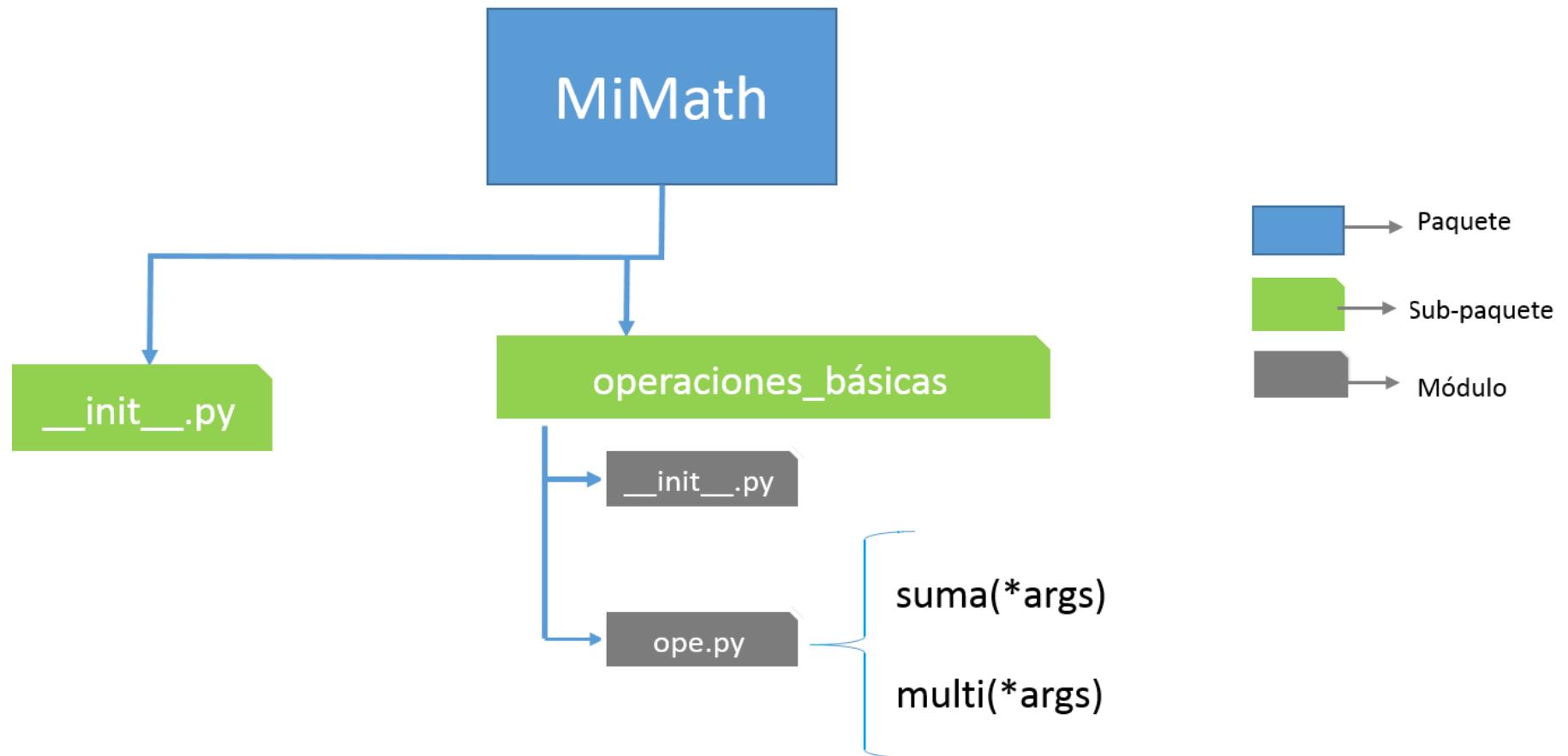
- pip es una herramienta para gestionar la dependencia en Python.
- pip significa Python Installer Package.
- pip permite instalar, desinstalar, mostrar, las dependencias de los paquetes Python.
- PyPI significa Python Package Index, acumulador de paquetes y módulos desarrollados por la comunidad Python.
- Es buena práctica analizar las dependencias de un programa Python antes de desinstalar un paquete.

Práctica: Paquetes

1. Crea un paquete de nombre MiMath.
2. Crea un subpaquete de nombre operaciones_básicas.
3. Crea un módulo de nombre ope.
4. En el módulo del punto anterior crea dos funciones, suma(*args) & multi(*args), que reciban un número variable de argumentos enteros, las funciones entregan la suma y la multiplicación de los valores pasados como argumentos.
5. Prueba las funciones suma(1,2,3) y multi(1,2,3).

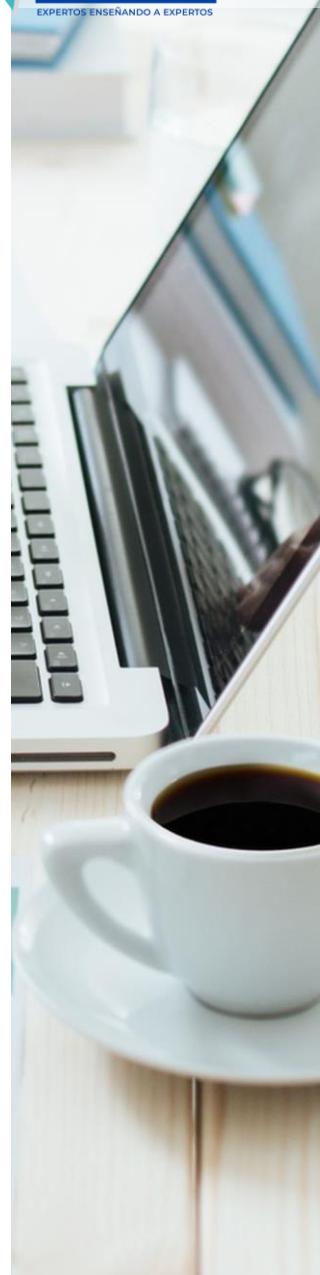


Práctica: Paquetes (Cont.)



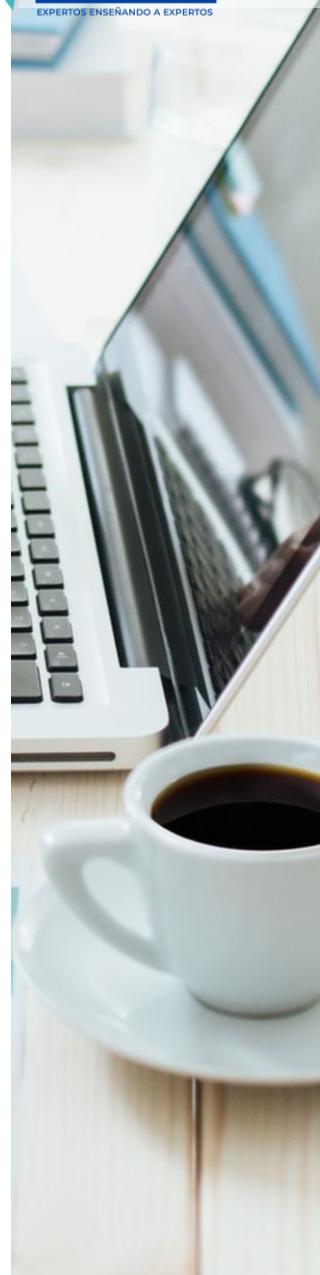
Práctica: Módulos y Programa Principal

1. Crea un programa Python de nombre Test.py que importe las funciones suma() y multi() del paquete MiMath, subpaquete operaciones_básicas y módulo ope, realizadas en la práctica 3.1.
2. Despliega la suma de: 2, 3, 5, 7, 11 y 13.
3. Despliega la multiplicación de: 2, 3, 5, 7, 11 y 13.
4. Asegúrate que los únicos mensajes que se desplieguen sean los del punto 2 y 3.
5. Verifica la creación del archivo: __pycache__
6. ¿Cuál es el uso del archivo del punto 5?



Práctica: Paquetes Distribuibles

1. Crea una carpeta llamada Test.
2. Copia el archivo Test.py de la práctica anterior en la carpeta del punto uno.
3. Cambia al directorio y ejecuta el archivo Test.py.
4. ¿Qué fue lo que sucedió? ¿Por qué Python no encontró el paquete? ¿Cómo corregirías este problema?.
5. Crea un archivo de nombre setup.py en directorio padre del directorio o carpeta MiMath.
6. Este archivo debe tener la descripción del paquete a distribuir con la siguiente información:



Práctica: Paquetes Distribuibles (Cont.)

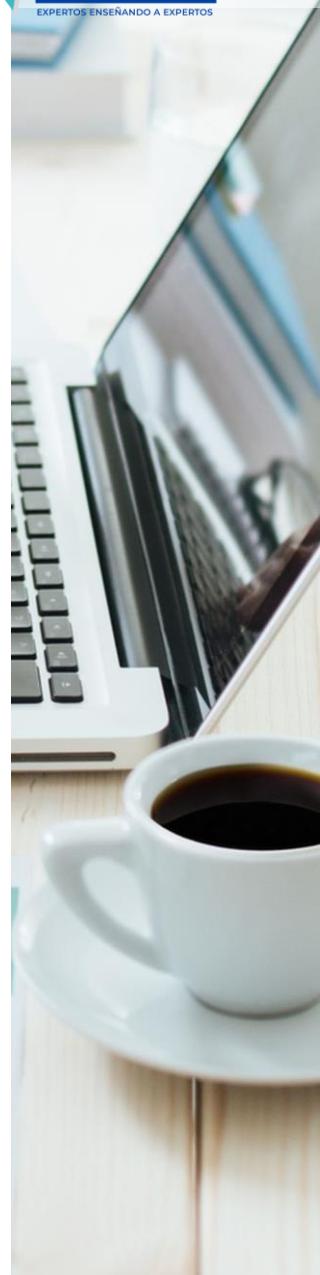
```
# Archivo del paquete distribuible

from setuptools import setup

setup (

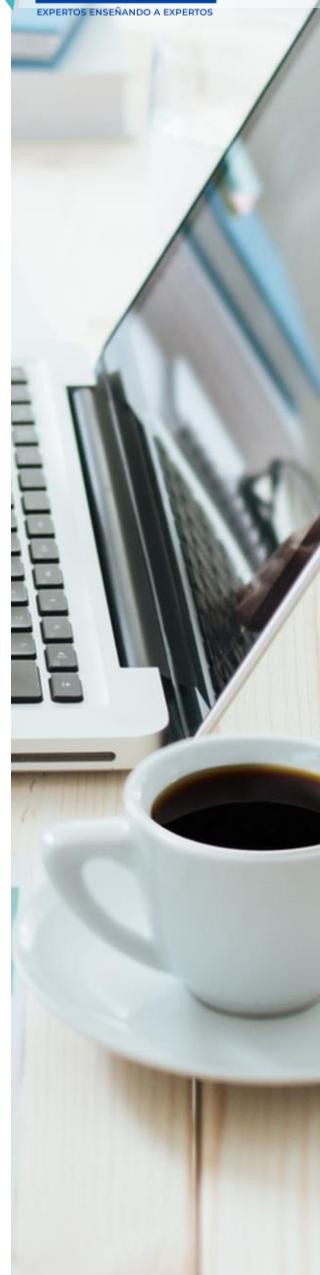
    name="MiPaqueteBB",      # Nombre del paquete
    version="1.0",           # Versión del paquete
    description=" Paquete creado para la práctica 2.3 y ver el uso de paquetes distribuibles", #Descripción
    author="Netec",          # Autor
    author_email="operaciones@netec.com", # Correo del autor
    url="https://www.netec.com", # Sitio del autor
    packages=["MiMath", "MiMath.ope"], # Paquetes & subpaquetes

    # Para más variables del setup ir a la documentación oficial
    # docs.python.org
)
```



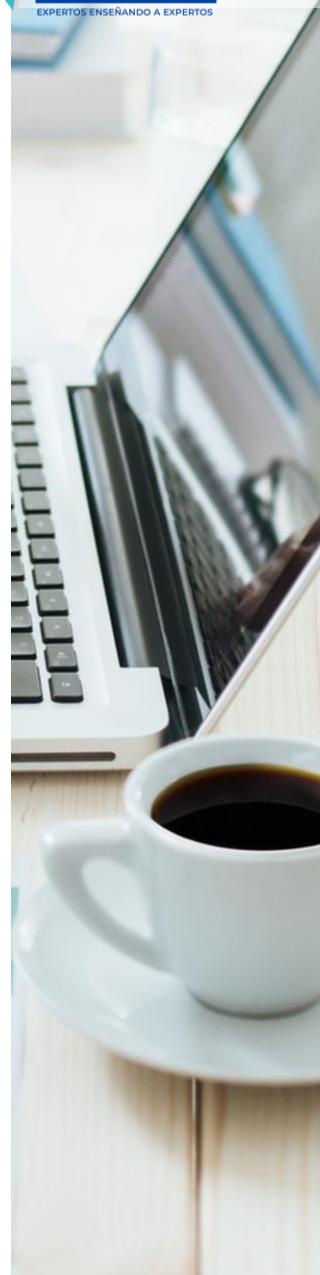
Práctica: Paquetes Distribuibles (Cont.)

7. En una terminal posiciónate en el directorio donde está el archivo `setup.py`.
8. Crea el paquete de Python distribuible con la siguiente instrucción: **`python setup.py sdist`**
9. Verifica las creaciones de las carpetas **`dist`** y **`MiPaqueteBB.egg-info`**.
10. Verifica el archivo `dist\MiPaqueteBB-1.0-tar`.
11. ¿Conoces los archivos tar? Pregunta a tu instructor asignado
12. Cámbiate al directorio de trabajo `dist`.
13. Usa la instrucción **`pip list`** y verifica los paquetes hasta este momento instalados.
14. Instala con pip el paquete distribuible recientemente creado en el punto 8 de la práctica.
 - Sugerencia: **`pip install MiPaqueteBB-1.0.tar.gz`**



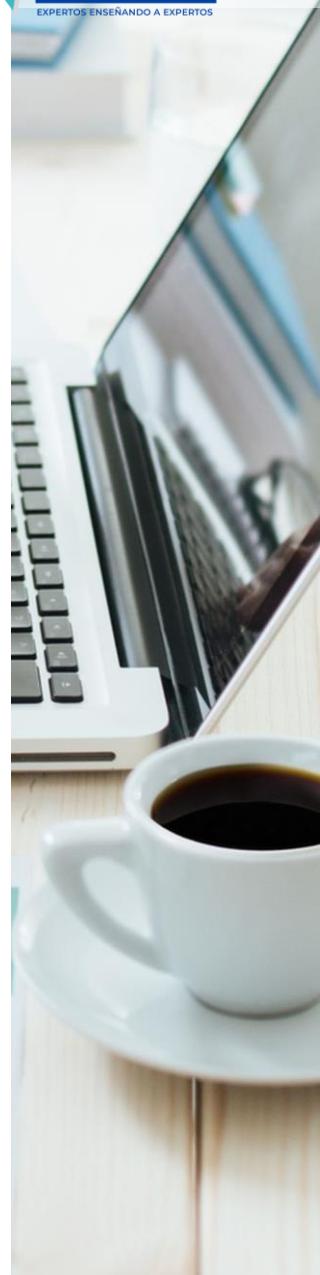
Práctica: Paquetes Distribuibles (Cont.)

15. Verifica con **pip list**, los paquetes instalados, la salida de la instrucción es en orden alfabético.
16. Cámbiate al directorio del punto 1 y verifica que el programa se ejecute correctamente.
17. Verifica que estando el archivo **Test.py** en cualquier carpeta, siempre encuentra el paquete distribuible.
18. Borra el paquete del instalado en el punto 14, y verifica la desinstalación. Sugerencia: **pip uninstall MiPaqueteBB**.
19. Ejecuta nuevamente el programa **Test.py**.



Práctica: Pizzas

1. El programa debe solicitar al usuario que ingrese la información siguiente:
 - a) El número de personas que comerán pizza.
 - b) El número de rebanadas de pizza que comerá cada persona.
 - c) El número de rebanas por pizza.
2. Usando la información suministrada, el programa debe calcular cuántas pizzas son necesarias para compartir con todos.
3. La siguiente captura de pantalla muestra cómo debería funcionar el programa.



Práctica: Pizzas (Cont.)

```
Administrator: Command Prompt

C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>python pizza_slices.py
¿Cuántas personas en la reunión? 11
¿Cuántas rebanadas por persona? 2.5
¿Cuántas rebanadas por pizza? 8
Se necesitaran 4 pizzas para alimentar a 11 personas.
Sobrarán 4.5 rebanas.

C:\Users\Administrator\ws_python>
```



Referencias Bibliográficas

- Format Specification Mini-Language:
<https://docs.python.org/3/library/string.html#format-spec>
- Maximum Line LengthMaximum Line Length:
<https://www.python.org/dev/peps/pep-0008/#maximum-line-length>
- Introduction to Unicode:
<https://docs.python.org/3/howto/unicode.html>
- Characters, Symbols and the Unicode Miracle – Computerphile:
<https://www.youtube.com/watch?v=MijmeoH9LT4>

Referencias Bibliográficas

- Module Unicode Character Database (UCD):
<https://docs.python.org/3/library/unicodedata.html>
- Regular Expression HOWTO:
<https://docs.python.org/3/howto/regex.html>
- Mastering Regular Expressions:
<http://www2.ii.uj.edu.pl/~tabor/prl09-10/perl/master.pdf>



Referencias Bibliográficas

- Pythex: <https://pythex.org/>
- Regular expression 101: <https://regex101.com/>
- Python RegEx:
https://www.w3schools.com/python/python_regex.asp
- Python Regular Expressions:
<https://developers.google.com/edu/python/regular-expressions>

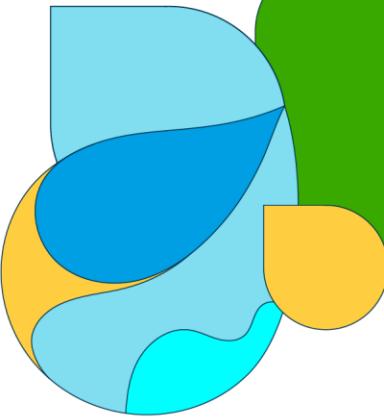
Unidad temática 7

Cadenas y colecciones
para el desarrollo de software

7.1 Cadenas

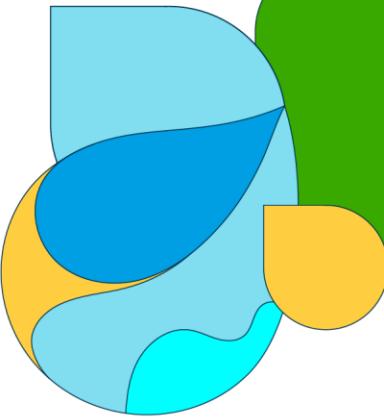
Objetivos:

- Trabajar con cadenas de caracteres.
- Listar las secuencias de escape o caracteres especiales en una cadena.
- Codificar cadenas de caracteres de varias líneas.
- Indexar y cortar cadenas de caracteres.
- Utilizar métodos y operadores comunes para la manipulación de cadenas de caracteres.
- Formatear cadenas de caracteres.
- Usar funciones intrínsecas de Python para manipulación de cadenas de caracteres.
- Crear expresiones regulares.
- Usar expresiones regulares en Python.



Marcación

```
>>>
>>> type("Hola Python")
<class 'str'>
>>> type('Hola Python')
<class 'str'>
>>>
>>> a='''Hola Python'''
>>> print (a)
"Hola Python"
>>>
>>> a="''Hola Python''"
>>> print (a)
'Hola Python'
```

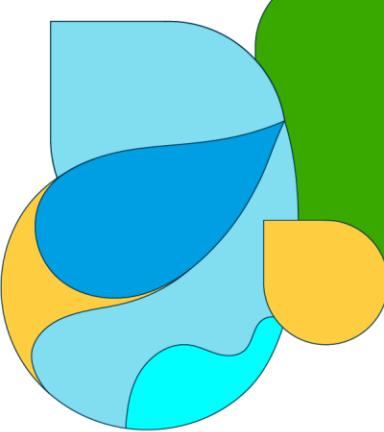


Secuencia de Escape

Winston Churchill dijo "Mantén la clama y continúa".

"Winston Churchill dijo \"Mantén la clama y continúa\"."

'Winston Churchill dijo "Mantén la clama y continúa".'



Secuencia de Escape

\'

\"

\\"

\n

\r

\t

\f

\b

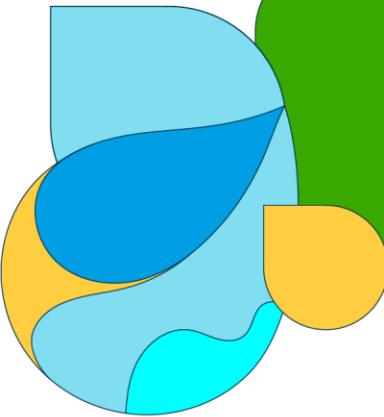
\a

Comillas Triples

```
>>> """ Aquí se puede poner lo que sea
... \n cambios de línea
... \t tabuladores
...
... " ' @ hasta poner nuevamente tres comillas dobles
...
... """
' Aquí se puede poner lo que sea\n\n cambios de línea\n\t tabuladores\n\n" \' @ hasta poner nuevamente
les\n'
>>>
```

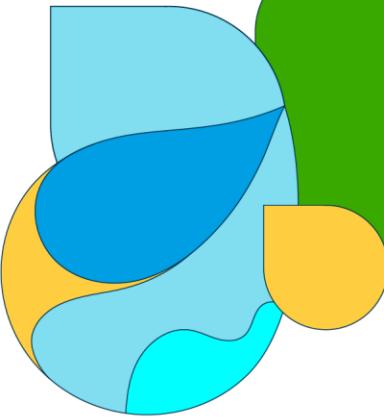
Indexación

H	o	l	a		M	u	n	d	o
0	1	2	3	4	5	6	7	8	9
-10	-9	-8	-7	-6	-5	-4	-3	-2	-1



Indexación

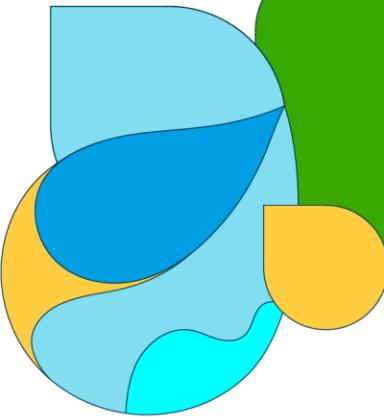
- mensaje = "Hola Mundo"
- print(mensaje)
- print("0123456789")
-
- primero = mensaje[0]
- # "[H]ola Mundo"
- print("Primero",primero)
-
- ultimo = mensaje[-1]
- #Hola Mund[o]
- print("Ultimo", ultimo)
-
- tercero = mensaje[3]
- #Hol[a] Mundo
- print("Tercero", tercero)



Separación, Concatenación y Repetición

```
subcadena = cadena_original[a:b]
```

- Donde:
a y b son número enteros.



Separación, Concatenación y Repetición

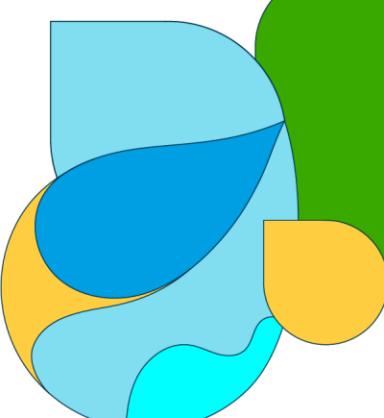
```
C:\> Administrator: Command Prompt
C:\Users\Administrator\ws_python\demos>python string_slicing.py
Guillermo Tell
01234567890123
'Guillermo Tell'[0:5] Guill
'Guillermo Tell'[1:4] uil
'Guillermo Tell'[4:] lermo Tell
'Guillermo Tell'[-3] ell
'Guillermo Tell'[:3] Gui
'Guillermo Tell'[-4:-1] Tel
'Guillermo Tell'[:] Guillermo Tell

C:\Users\Administrator\ws_python\demos>
```

Concatenación

```
nombre= input("¿Cuál es tu nombre? ")  
mensaje= "Hola, " + nombre + "!"  
print(mensaje)
```





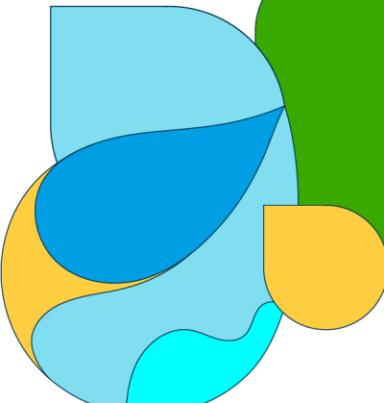
Concatenación

```
personaje_dice="bla"  
que_dice=personaje_dice * 10  
print(que_dice) # blablablablablablablablabla
```

*

```
mensaje="a" + "b" * 3 + "c" # abbbc  
mensaje=("a" + "b") * 3 + "c" # abababc
```

Prioridad/Asociatividad



Métodos

capitalize

lower

upper

swapcase

title

replace

strip

lstrip

rstrip

isalnum

startswith

endswith

isnumeric

isdecimal

isdigit

isspace

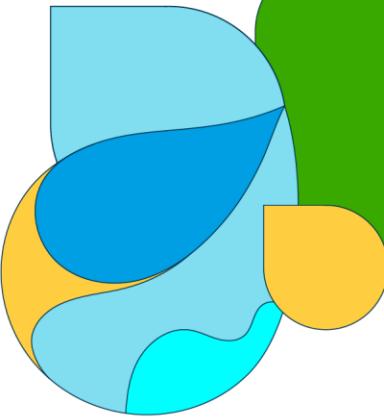
istitle

isupper

islower

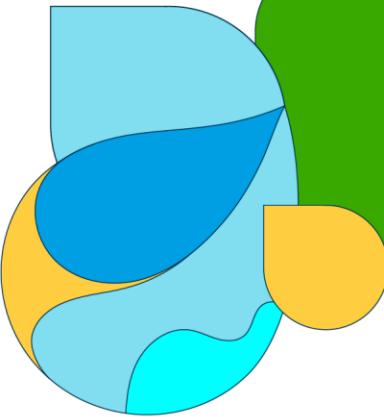
isalpha

join



Métodos

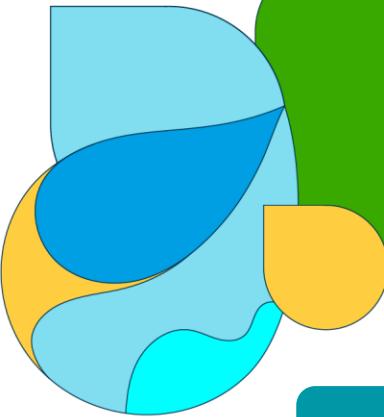
Método	Ejemplo
find(sub [,inicio [,fin]])	"Hola Mundo!".find("o") # 1 "Hola Mundo!".find("A") # -1
rfind(sub [,inicio [,fin]])	"Hola Mundo!".rfind("o") # 9
index(sub [,inicio [,fin]])	"Hola Mundo!".index("o") # 1 "Hola Mundo!".index("A") # TypeError
rindex(sub [,inicio [,fin]])	"Hola Mundo!".rindex("o") # 9



Método count()

- `count(sub[,inicio [,fin]])`
- La función cuenta el número de veces que se encuentra `sub` en la cadena.
- `inicio` es inclusivo.
- `fin` es exclusivo.

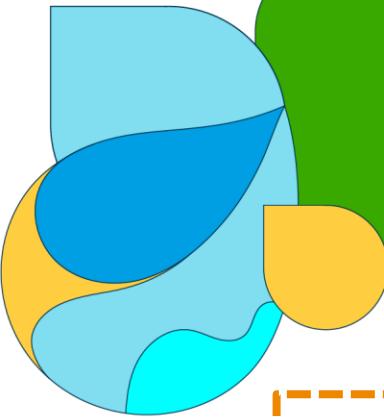
```
>>> "oooxxooo".count("o")
6
>>> "oooxxooo".count("o",1)
5
>>> "oooxxooo".count("o",1,5)
2
>>> "oooxxooo".count("x",1,5)
2
>>> "oooxxooo".count("x")
2
>>> "oooxxooo".count("y")
0
```



Formato

```
[[fill]align][sign][#][0][width][group][.precision][type]
```

- **type:** tipo de dato a formatear.
 - **b** binario, **c** caracter, **d** decimal, **o** octal, **x** o **X** hexadecimal, **f** float.
- **precision:** cuantos caracteres son tomados a la derecha del punto.
- **group:** al indicar con un caracter coma (,) agrupa los miles con comas.
- **width:** número de caracteres de la cadena resultante.
- **sign:** signo, negativo **-**, positivo **+**
- **align:** alineación.
 - **<** a la izquierda, **>** a la derecha, **^** centrado
- **fill:** relleno, por defecto espacios en blanco.

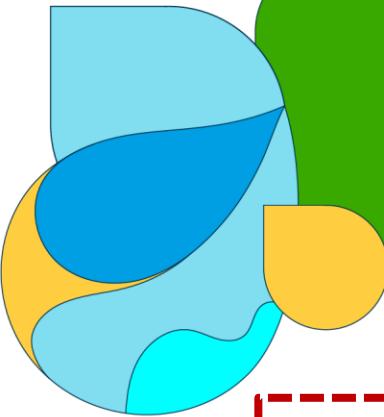


Formato | Ejemplos

```
>>> import math
>>>
>>> 'PI es igual a {:.f}'.format(math.pi)
'PI es igual a 3.141593'
>>>
>>> 'PI es igual a {:.}.'.format(math.pi)
'PI es igual a 3.141592653589793'
>>>
```

```
>>> import math
>>>
>>> 'PI es igual a {:.f}'.format(math.pi)
'PI es igual a 3.141593'
>>>
>>> 'PI es igual a {:.}'.format(math.pi)
'PI es igual a 3.141592653589793'
>>>
>>> 'PI es igual a {:.2f}'.format(math.pi)
'PI es igual a 3.14'
>>>
```

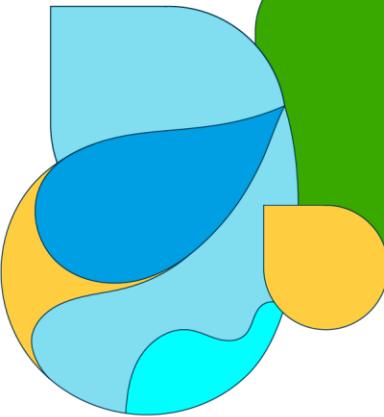
```
>>> 'El cliente compró {:.,.2f}.'.format(1000000000)
'El cliente compró 1,000,000,000.00.'
>>>
>>> 'El cliente compró ${:,.2f}.'.format(1000000000)
'El cliente compró $1,000,000,000.00.'
```



Formato | Ejemplos

```
>>> '{:5}'.format('123')
'123 '
>>> '{:5}'.format(123)
' 123'
>>> '{:5.2f}'.format(123)
'123.00'
```

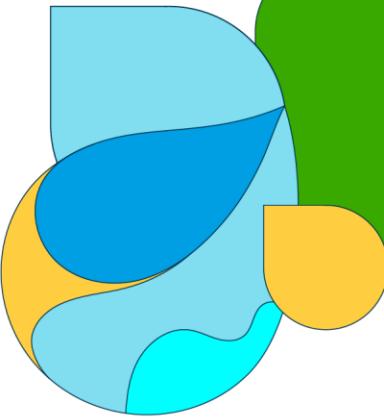
```
>>> import math
>>>
>>> 'Pi es aproximadamente {:.2f}'.format(math.pi)
'Pi es aproximadamente +3.14'
```



Formato | Ejemplos

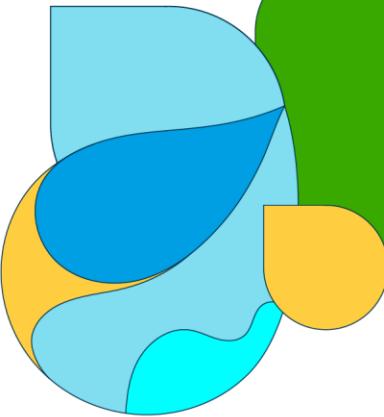
```
>>> '{:>5}'.format('xyz')
' xyz'
>>> '{:<5}'.format('xyz')
'xyz '
>>> '{:^5}'.format('xyz')
' xyz '
>>>
>>> '{:+=5}'.format(123)
'+ 123'
>>> '{:+=5}'.format(-123)
'- 123'
```

```
>>> import math
>>>
>>> '{.:^12.2f}'.format(math.pi)
'....3.14....'
>>> '{:#^12.2f}'.format(math.pi)
'####3.14#####'
```



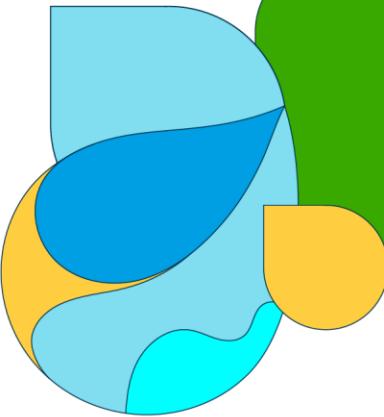
Funciones Precostruidas

- **str(arg):** Convierte el argumento a una cadena de caracteres.
 - str('México') -> 'México', str(123) -> '123', str(math.pi) -> '3.141592653589793'
- **len(arg):** Devuelve la longitud de la cadena de caracteres.
 - len('México') -> 6, len("") -> 0, len("12345") -> 5
- **min(arg):** Devuelve el valor más pequeño de los argumentos.
 - min("Hola") -> "H", min("Zorro") -> 'Z', min ("abc123" -> '1'
- **max(arg):** Devuelve el valor más grande de los argumentos.
 - max("Hola") -> "o", max("Zorro") -> 'r', max("zorro") -> 'z'



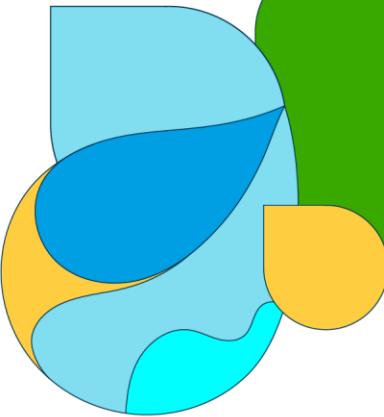
Expresiones Regulares

- Una expresión regular es una secuencia de caracteres que especifica un patrón de búsqueda en el texto.
- Especificación **POSIX** usada por **Perl**.
- Las expresiones regulares se utilizan en motores de búsqueda, en remplazo de procesadores de texto, en editores de texto, en análisis léxico, etc.
- La mayoría de los lenguajes de programación admiten capacidades de expresión regular de forma nativa o a través de bibliotecas.



Expresiones Regulares | Metacaracteres

^	\$	\b	\B	.	?	+
*	{}	\	\d	\D	\w	\W
\s	\S	[]	[-]	[^]	()	

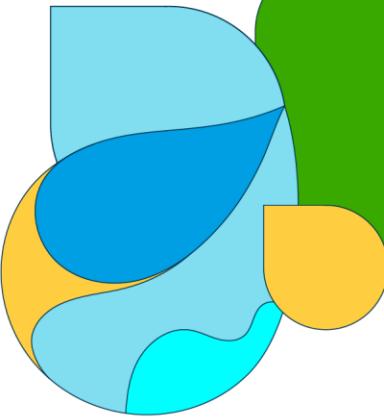


Expresiones Regulares | Módulo re

```
import re
p = re.compile('fl[aeiou]r')
result = p.search ("La rosa es una flor hermosa")
print (result)
```

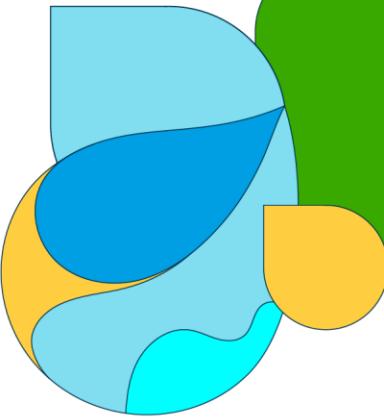
La rosa es una flor hermosa
012345678901234567890123456

<re.Match object; span=(15, 19), match='flor'>



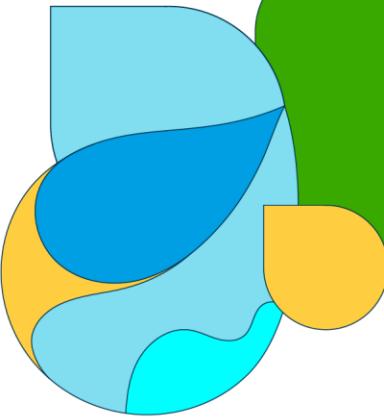
Módulo re | Métodos

Método	Descripción
p.search(str)	Encuentra la primera subcadena que coincide con la expreg.
p.match(str)	La coincidencia se encuentra al comienzo de la cadena.
p.fullmatch(str)	Toda la cadena debe coincidir. Python 3.4
p.findall(str)	Encuentra todas las coincidencias, devuelve lista de cadenas.
p.finditer(str)	Encuentra todas las coincidencias, devuelve un iterador.
p.split(str,maxsplit=0)	Divide la cadena en coincidencias de patrones. Limita las divisiones a maxsplit.
p.sub(repl,str,count=0)	Reemplaza todas las coincidencias cpm repl. Limita el remplazo a count.



Módulo re | Banderas

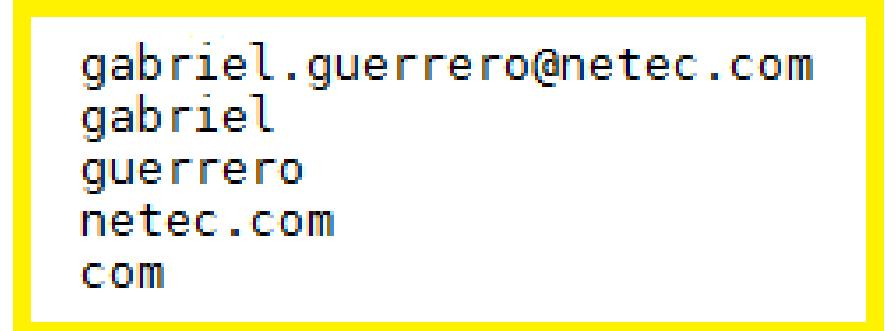
Bandera	Abreviación	Descripción
re.IGNORECASE	re.I	Ignora diferencia entre mayúsculas y minúsculas.
re.,MULTILINE	re.M	Hace que ^ y \$ consideren cada línea como una cadena independiente.
re.DOTALL	re.S	Hace que punto (.) coincida con cualquier carácter, incluido el cambio de línea.
re.ASCII	re.A	Hace que los caracteres especiales coincidan solo con caracteres ASCII.
re.VERBOSE	re.X	Se ignora el espacio en blanco.
re.DEBUG	No aplica	Muestra información de depuración.
re.LOCAL	re.L	Toma de forma local los caracteres especiales.



Módulo re | Grupos

```
p = re.compile(r'(\w+)\.(\w+)@(\w+\.\w+)')
match = p.match('gabriel.guerrero@netec.com')
email = match.group(0)
handle = match.group(1)
handle2 = match.group(2)
domain = match.group(3)
domain_type = match.group(4)

print(email, handle, handle2, domain, domain_type, sep='\n')
```



gabriel.guerrero@netec.com
gabriel
guerrero
netec.com
com

Resumen del capítulo

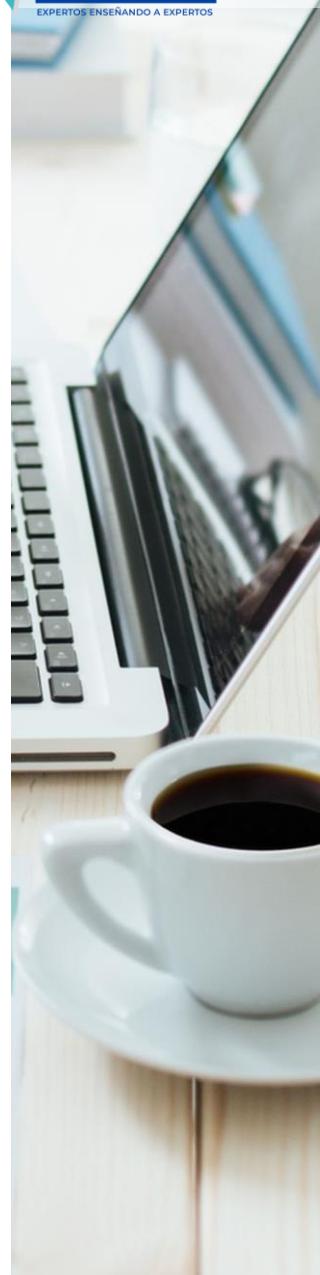
- En este capítulo se dio una introducción a técnicas de manipulación de cadenas de caracteres en Python.
- Las cadenas en Python son instancias de la clase str.
- Saber cómo manipular cadenas de caracteres juega un papel fundamental en la mayoría de las tareas de procesamiento de texto. Si quieras experimentar como debes escribir y ejecutar pequeños programas tal como se hizo en capítulos previos, o puedes abrir el intérprete de comandos de Python para probarlos ahí.
- Las funciones preconstruidas min(), max() y len() funcionan con cadenas y otros tipos de datos como colección o iterables.

Resumen del capítulo

- El libro más completo sobre expresiones regulares con certeza es el *Mastering Regular Expressions*, de Jeffrey Friedl, editorial O'Reilly.
- Una herramienta en línea puede ayudarte a comprender más claramente cada uno de los metacaracteres de las expresiones regulares antes de usar el módulo `re` de Python.

Práctica: Cadenas de Indexación

1. Crear la función `main()` de tal modo que realice lo siguiente:
 - a) Solicitar al usuario que ingrese una frase y almacenarla en una variable de nombre `frase`.
 - b) Solicitar al usuario un número y almacenarla en una variable de nombre `posición`.
 - c) Indicar al usuario qué carácter se encuentra en esa posición en la frase del usuario.
 - d) Utilizar cadenas formateadas.



Práctica: Cadenas de Indexación

```
C:\> Administrator: Command Prompt
C:\Users\Administrator\ws_python\soluciones>
C:\Users\Administrator\ws_python\soluciones>python indexacion.py
Ingesa un frase y pulsa [Enter]: Hola mundo cruel
Tu frase es: 'Hola mundo cruel'
¿Qué caracter quieres desplegar? [Número & Enter] 5
El carácter en la posición 5 es m

C:\Users\Administrator\ws_python\soluciones>python indexacion.py
Ingesa un frase y pulsa [Enter]: Sigue al conejo blanco
Tu frase es: 'Sigue al conejo blanco'
¿Qué carácter quieres desplegar? [Número & Enter] 11
El carácter en la posición 11 es n

C:\Users\Administrator\ws_python\soluciones>
```

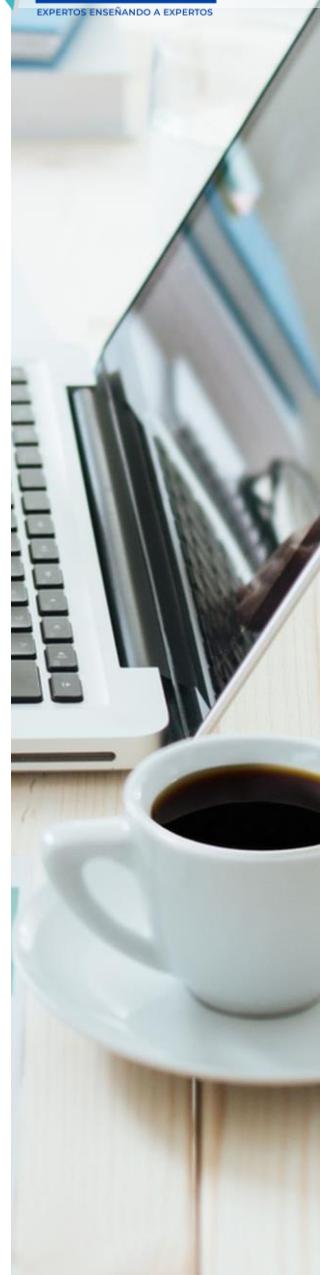


Práctica: Cadenas de Indexación

```
Administrator: Command Prompt

C:\Users\Administrator\ws_python\soluciones>python indexacion_reto.py
Ingesa un frase y pulsa [Enter]: Hola Mundo cruel
Tu frase es: 'Hola Mundo cruel'
¿Qué carácter quieres desplegar? [Número & Enter] 6
El carácter en la posición 6 es M

C:\Users\Administrator\ws_python\soluciones>
```



Práctica: Split de Cadena de Caracteres

En esta práctica escribe un programa que reciba una subcadena (o corte) de una frase introducida por el usuario.

1. Captura el siguiente fragmento de código.

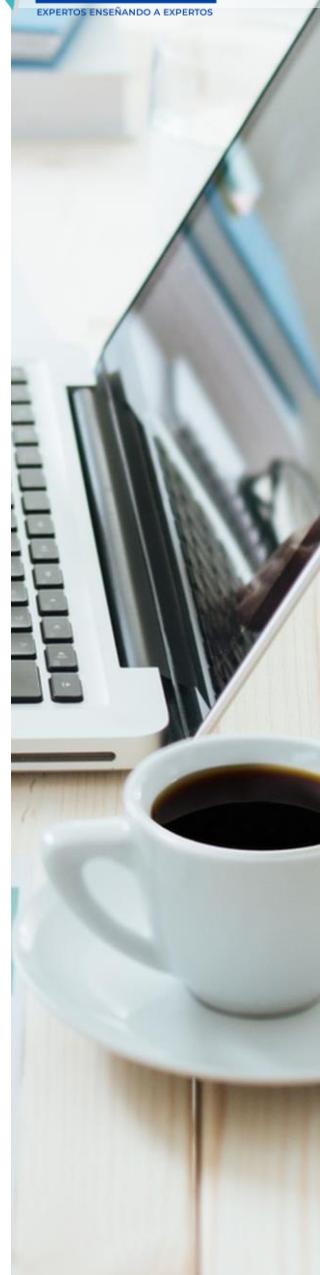
```
def main():
    phrase = input("Ingrese una frase: ")
    # Escriba su código aquí

main()
```



Práctica: Split de Cadena de Caracteres

2. Modifica la función `main()` de modo que:
- Solicite al usuario que ingrese una frase.
 - Despligue la frase ingresada por el usuario.
 - Solicite al usuario un número para el inicio de la frase.
 - Solicite al usuario un número para el final de la frase.
 - Indique al usuario la subcadena, entre comillas simples, que comienza con el número inicial y termina con el número final.
3. La salida del programa puede ser similar a la siguiente:



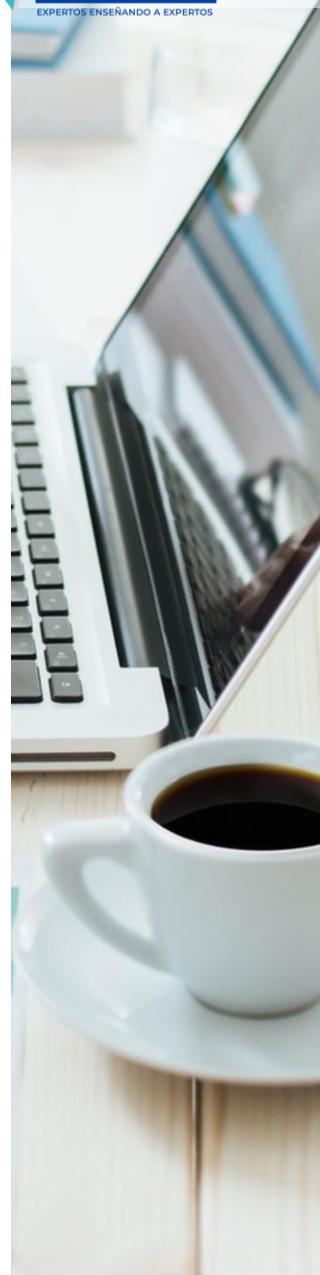
Práctica: Split de Cadena de Caracteres

```
Administrator: Command Prompt

C:\Users\Administrator\ws_python\soluciones>
C:\Users\Administrator\ws_python\soluciones>python slicing.py
Ingrese una frase: 1234567890
Su frase es: '1234567890'
¿Ingrese la posición del carácter inicial? 3
¿Ingrese la posición del carácter final? 3
La subcadena es: '4'

C:\Users\Administrator\ws_python\soluciones>python slicing.py
Ingrese una frase: abcdefghijk
Su frase es: 'abcdefghijklm'
¿Ingrese la posición del carácter inicial? 0
¿Ingrese la posición del carácter final? 4
La subcadena es: 'abcde'

C:\Users\Administrator\ws_python\soluciones>
```



Práctica: Split de Cadena de Caracteres

1. Cambia el programa para un usuario final, donde las cadenas de caracteres no inicien en 0.

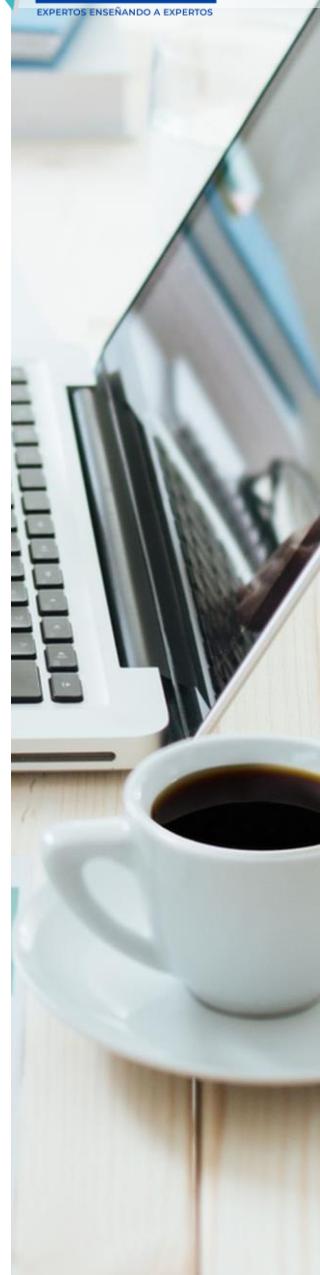
```
Administrator: Command Prompt

C:\Users\Administrator\ws_python\soluciones>python slicing_reto.py
Ingrese una frase: 1234567890
Su frase es: '1234567890'
¿Ingrese la posición del carácter inicial? 3
¿Ingrese la posición del carácter final? 3
La subcadena es: '3'

C:\Users\Administrator\ws_python\soluciones>python slicing_reto.py
Ingrese una frase: abcdefghijk
Su frase es: 'abcdefghijklm'
¿Ingrese la posición del carácter inicial? 0
¿Ingrese la posición del carácter final? 4
La subcadena es: 'abcd'

C:\Users\Administrator\ws_python\soluciones>python slicing_reto.py
Ingrese una frase: abcdefghijk
Su frase es: 'abcdefghijklm'
¿Ingrese la posición del carácter inicial? 1
¿Ingrese la posición del carácter final? 4
La subcadena es: 'abcd'

C:\Users\Administrator\ws_python\soluciones>
```

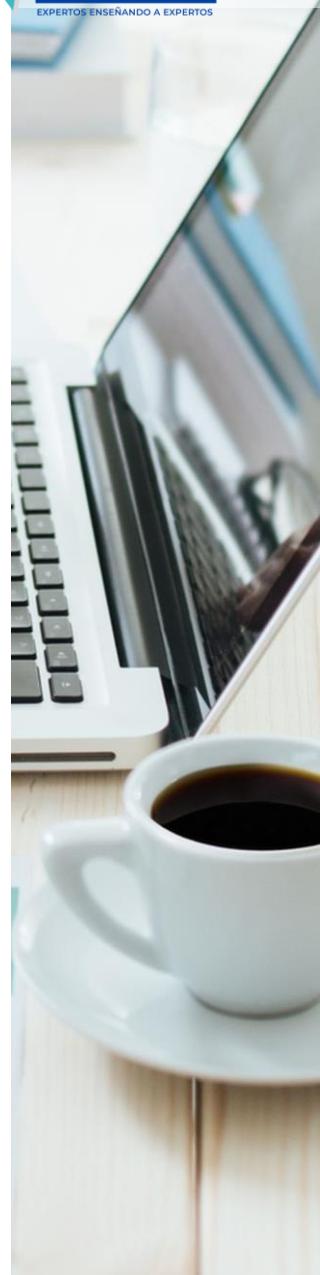


Práctica: Expresiones Regulares

1. Crea un script Python con el siguiente código:

```
import re
codigos = ['se1', 'se9', 'ma2', 'se:','se.', 'se2', 'hu2', 'se3', 'sea', 'sec']
sep = "-"*40

for ele in codigos:
    patron = None # el 3er carácter puede ser nº de 0-5
    if re.match(patron, ele):
        print(ele)
    print(sep)
```

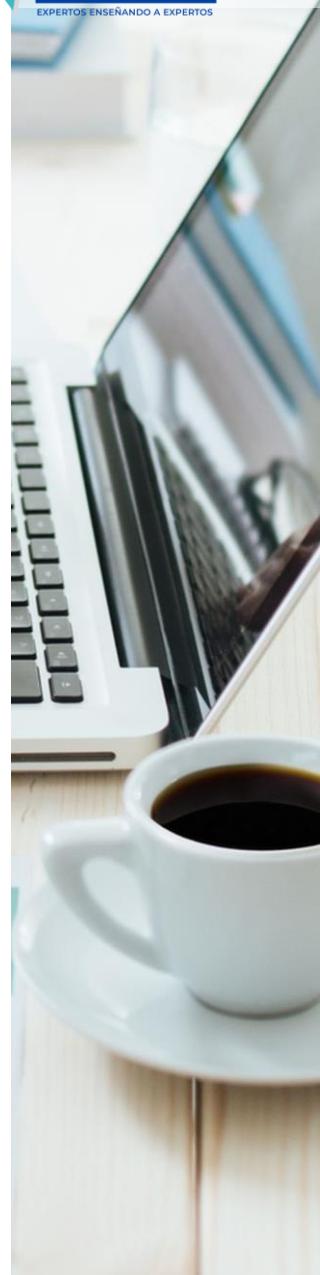


Práctica: Expresiones Regulares

```
for ele in codigos:  
    patron= None # nº de 0 a 5 y letra de a a z  
    if re.match(patron, ele):  
        print(ele)  
    print(sep)
```

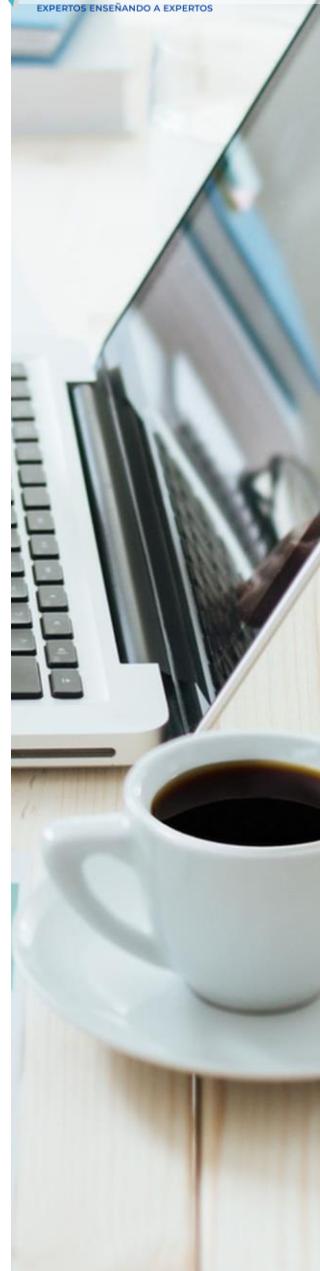
```
for ele in codigos:  
    patron = None # el tercer carácter puede ser . ó :  
    if re.match(patron, ele):  
        print(ele)  
    print(sep)
```

```
for ele in codigos:  
    patron = None # debe comenzar por nº de 0 a 2  
    if re.match(patron, ele):  
        print(ele)  
    print(sep)
```



Práctica: Expresiones Regulares

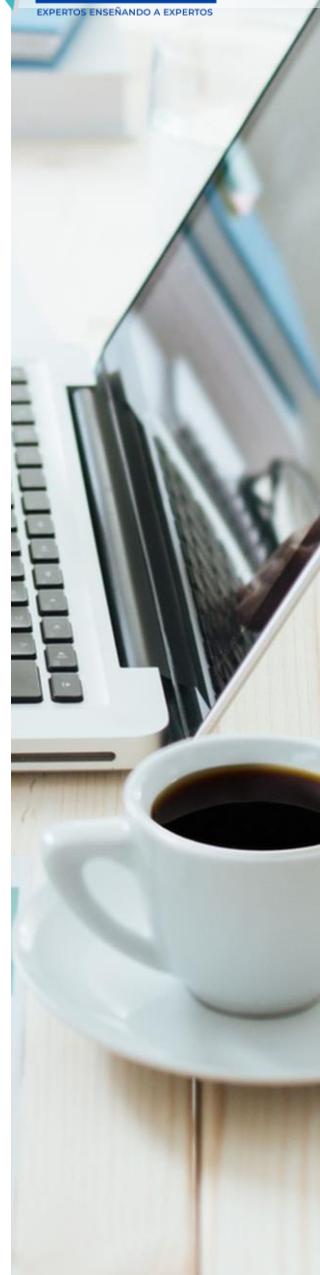
2. En el código debes sustituir el valor de None por el valor de la expresión regular necesaria para:
- a) Encontrar el tercer carácter, este puede ser un número del cero al cinco.
 - b) Encontrar el tercer carácter, este puede ser del cero al cinco o una letra minúscula.
 - c) Encontrar el tercer carácter, este puede ser . ó :
 - d) Encontrar el tercer carácter, este no puede ser 0, 1 ó 2.



Práctica: Expresiones Regulares

3. La salida del script puede ser similar a lo siguiente:

```
se1  
se2  
se3  
-----  
se1|  
se2  
se3  
sea  
sec  
-----  
se:  
se.  
-----  
se9  
se:  
se.  
se3  
sea  
sec  
-----
```



Referencias Bibliográficas

- Python List:
https://www.w3schools.com/python/python_lists.asp
- Python Tuples:
https://www.w3schools.com/python/python_tuples.asp
- Python Sets:
https://www.w3schools.com/python/python_sets.asp



Referencias Bibliográficas

- Python Dictionaries:
https://www.w3schools.com/python/python_dictionaries.asp
- Argumentos en funciones: <https://recursospython.com/guias-y-manuales/argumentos-args-kwargs>
- Python Tuples are Not Just Constant Lists:
http://jtauber.com/blog/2006/04/15/python_tuples_are_not_just_constant_lists/



7.2 Colecciones

Objetivos:

- Crear, modificar y trabajar con listas.
- Crear y trabajar con tuplas.
- Usar rangos.
- Crear, modificar y trabajar con diccionarios.
- Crear y usar conjuntos.
- Usar conjuntos para eliminar elementos duplicados en listas.
- Conocer las diferencias entre los parámetros *args & **kwargs.

Listas []:

- Similares a los arreglos en lenguajes como Java, C, C++, etc.

```
colores = ['rojo','verde','azul ]
paises = ['México', 'Chile', 'Perú','Colombia', 'Argentina']
numeros [ 1, 2, 3, 4, 5 ]
arreglos = [colores, paises, numeros]
switches = [ True, False, False, True]
empty = []
uno = [1]
```

Tuplas ():

MAGENTA = (255,0,255)

BLANCO = (255,255,255)

NEGRO = (0,0,0)

TAMAÑOS = ('CH','M','G','XG', 'XCH', 'L')

DIAS = ("LUNES", "MARTES", "MIERCOLES", "JUEVES", "VIERNES")

empty= () # tupla vacía

astro = ('Luna',)

satélite= ('Luna') # Esta asignación no es una tupla.

Tuplas ():

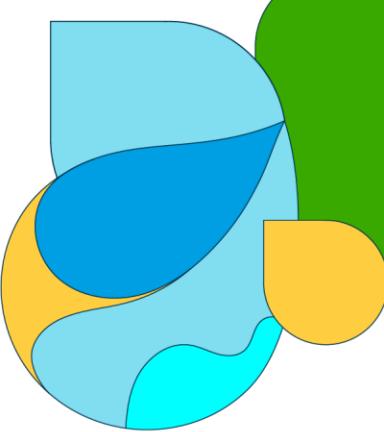
```
>>> tupla=(1,2,3)
>>> print(tupla, tupla[0])
(1, 2, 3) 1
>>> tupla[0]=10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

```
>>> MAGENTA=(255,0,255)
>>>
>>> type(MAGENTA)
<class 'tuple'>
>>>
>>> MAGENTA=255,0,255
>>>
>>> type(MAGENTA)
<class 'tuple'>
```

Tuplas ():

```
>>> def show_type(obj):
...     print (type(obj))
...
>>>
>>> MAGENTA=255, 0, 255 # R G B
>>>
>>> show_type(MAGENTA)
<class 'tuple'>
>>>
```

```
>>> show_type(255,0,255)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: show_type() takes 1 positional argument but 3 were given
>>>
>>> show_type((255,0,255))
<class 'tuple'>
>>> show_type([255,0,255])
<class 'list'>
```

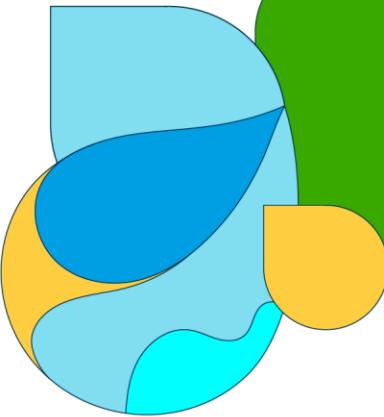


Tuplas | Métodos:

count(x)

copy()

index()



Tuplas | Ejemplos

```
>>> frutas=['platanos','manzanas']
>>> carne=['res','pollo','cerdo']
>>> lacteos=['leche','yogurt','helado']
>>> vegetales=['brocoli','chicharos','zanahorias']
```

```
>>> comestibles=(frutas,carne,lacteos) # Tupla
>>> comestibles[1]=vegetales
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> print(comestibles)
(['platanos', 'manzanas'], ['res', 'pollo', 'cerdo'], ['leche', 'yogurt', 'helad
>>> carne[1]='cordero' # pollo -> cordero
>>>
>>> print (comestibles)
(['platanos', 'manzanas'], ['res', 'cordero', 'cerdo'], ['leche', 'yogurt', 'hel
```

Indexación & Slicing | Listas

```
lista=['a','e','i','o','u']
```

'a'	'e'	'i'	'o'	'u'
0	1	2	3	4
-5	-4	-3	-2	-1

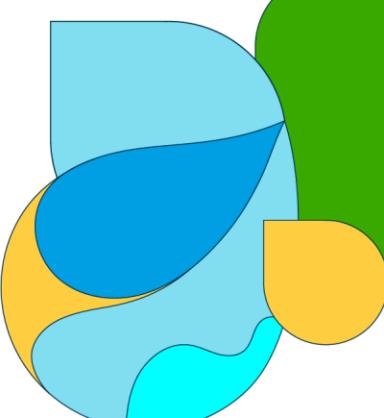
Índices

```
lista[5] IndexError: list index out of range
```

```
lista[0]='a'  
lista[1]='e'  
Lista[2]='i'  
lista[3]='o'  
lista[4]='u'
```

```
lista[-1]='u'  
lista[-2]= 'o'  
lista[-3]='i'  
lista[-4]='e'  
lista[-5]='a'
```

```
lista[0:]= ['a','e','i','o','u']  
lista[:]= ['a','e','i','o','u']  
lista[2:4]= ['i','o']  
lista[1:3]= ['e','i']  
lista[:4]= ['a','e','i','o']  
lista[-1:-5]= []  
lista[-5,-1]= ['a','e','i','o']  
lista[1:5:2]= ['e','o']
```



Indexación & Slicing

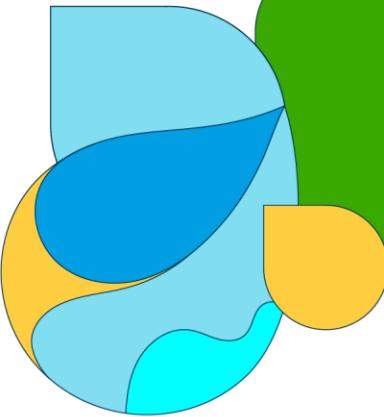
```
frutas = ['manzana', 'naranja', 'fresa', 'pera', 'lima', 'sandía']
print(frutas)

primeras_5_frutas = frutas[0:5]
print(primeras_5_frutas) #['manzana', 'naranja', 'fresa', 'pera', 'lima']

frutas_2_a_4 = frutas[1:4]
print(frutas_2_a_4) #['naranja', 'fresa', 'pera']

frutas_5_a_fin = frutas[4:]
print(frutas_5_a_fin) #['lima', 'sandía']
```

sub_sequence = sequence [pos_inicial : pos_final]



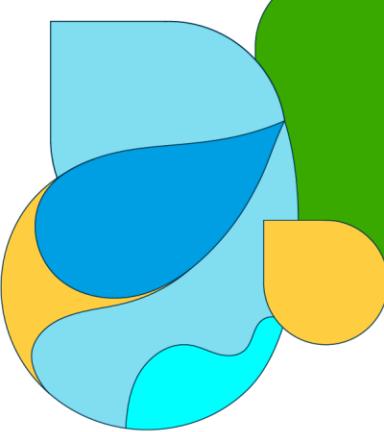
Indexación & Slicing

```
las_ultimas_3_frutas = frutas[-3:]
print(las_ultimas_3_frutas) #['pera', 'lima', 'sandía']

las_primeras_3_frutas = frutas[:3]
print(las_primeras_3_frutas) #['manzana', 'naranja', 'fresa']

tres_frutas_antes_ultima = frutas[-4:-1]
print(tres_frutas_antes_ultima) #['fresa', 'pera', 'lima']

copia = frutas[:]
print(copia) #['manzana', 'naranja', 'fresa', 'pera', 'lima', 'sandia']
```



Funciones Preconstruidas

`sum(iterable)`

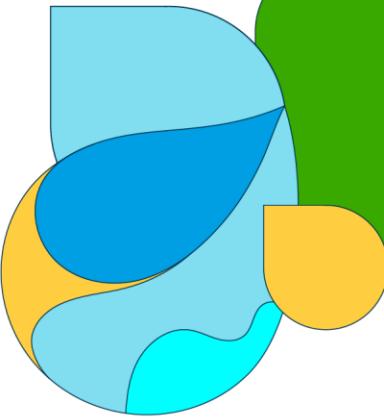
`join()`

`split()`

`min()`

`max()`

`splitlines()`

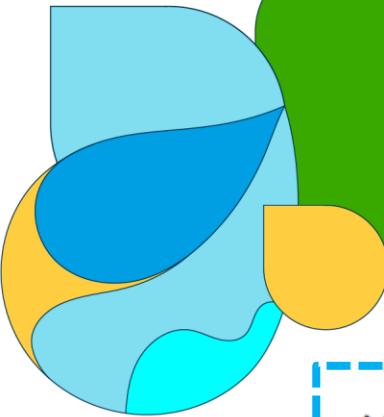


Funciones Preconstruidas min & max

```
>>> def main():
...     personas= ['Lucy','Blanca','Miguel','Greta','Leticia']
...     edades= [ 52, 72, 28, 22, 50]
...     print("min(personas):",min(personas))
...     print("max(personas):",max(personas))
...     print("min(edades):",min(edades))
...     print("maz(edades):",max(edades))
...
>>> main()
min(personas): Blanca
max(personas): Miguel
min(edades): 22
maz(edades): 72
```

Funciones | join()

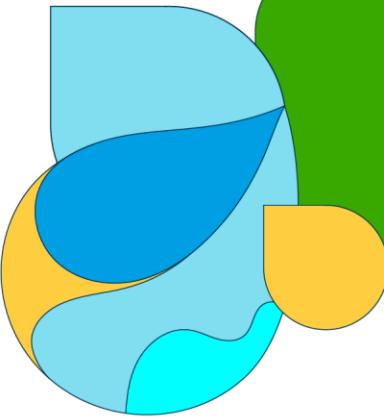
```
>>> colores=['rojo','azul','verde','naranja']
>>>
>>> ','.join(colores)
'rojo,azul,verde,naranja'
>>> ':'.join(colores)
'rojo:azul:verde:naranja'
>>> ' '.join(colores)
'rojo azul verde naranja'
>>> ''.join(colores)
'rojoazulverdenaranja'
>>> type(' '.join(colores))
<class 'str'>
```



Funciones | split()

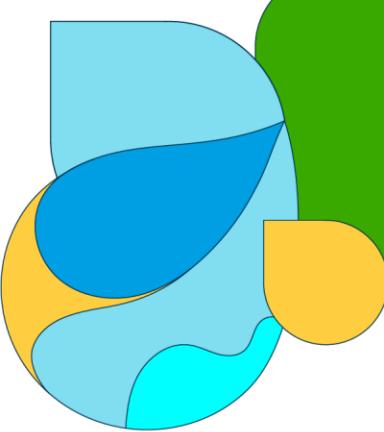
```
>>> nombre = "Adraina Estefanía Melchora Policarpa 'María del Pilar' Guadalupe Mosqueda Islas"  
>>> nombres= nombre.split()  
>>>  
>>> print (nombres[0],nombres[5],nombres[-1])  
Adraina del Islas
```

```
>>> frutas = 'cerezas, limones, guayabas, melones, peras'  
>>> lista = frutas.split(',')  
>>> print(lista)  
['cerezas', ' limones', ' guayabas', ' melones', ' peras']  
>>> lista[-1]  
' peras'  
>>> lista[-2]  
' melones'  
>>> lista[-2].lstrip()  
'melones'
```



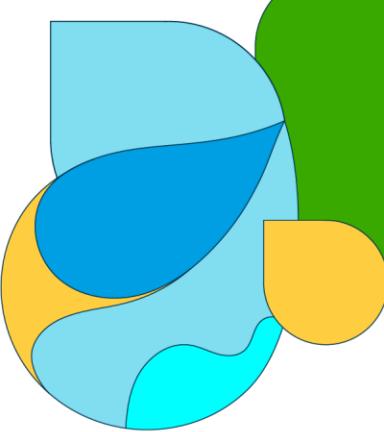
Funciones | split() con tope

```
>>> vegetales = 'zanahorias, chicharons, tomates, espinacas'  
>>> lista= vegetales.split(', ',2); print (lista, len(lista))  
['zanahorias', 'chicharons', 'tomates, espinacas'] 3  
>>>  
>>> lista= vegetales.split(', ',3); print (lista, len(lista))  
['zanahorias', 'chicharons', 'tomates', 'espinacas'] 4  
>>>  
>>> lista= vegetales.split(', ',1); print (lista, len(lista))  
['zanahorias', 'chicharons, tomates, espinacas'] 2
```



Funciones | splitlines()

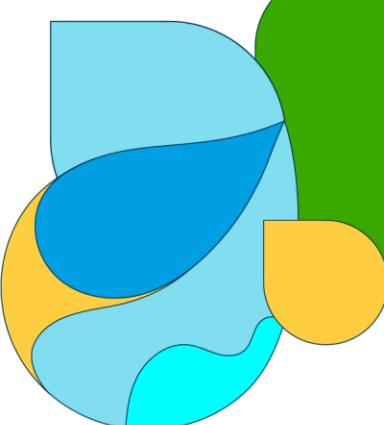
```
>>> frutas= '''manzana,  
... pera,  
... melón,  
... uvas,  
... guayabas'''  
>>> lista= frutas.splitlines()  
>>> print (len(lista), lista)  
5 ['manzana,', 'pera,', 'melón,', 'uvas,', 'guayabas']
```



Desempaquetamiento

j, p, g, r = "John", "Paul", "George", "Ringo" a,

```
>>> beatles=["John", "Paul", "George", "Ringo"]
>>> inteligente, tierno, serio, divertido = beatles
>>> print (inteligente, divertido)
John Ringo
```



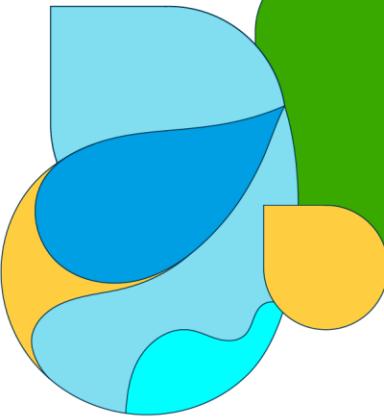
Diccionarios

```
dict={clave:valor, clave2:valor, clave3:valor,...}
```

```
dict[clave2] = nuevo_valor #Asignando un nuevo valor a la clave2
```

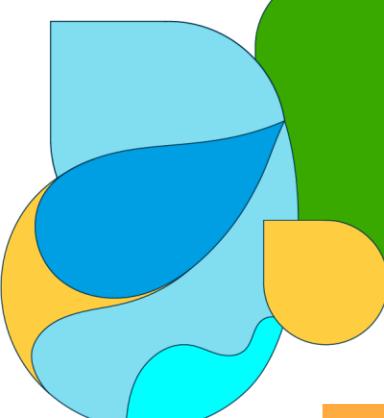
```
dict[clave4] = nuevo_valor #Asignando un nuevo valor a la clave4
```

```
print(dict[clave1]) #Desplegar el valor asociado a la clave1
```



Diccionarios | Ejemplos

```
>>> calificaciones = {'Inglés':85, 'Mate':10, 'Historia':70, 'Arte':74, 'Danza':90}
>>> print (len(calificaciones), calificaciones)
5 {'Inglés': 85, 'Mate': 10, 'Historia': 70, 'Arte': 74, 'Danza': 90}
>>> print (calificaciones['Mate'])
10
>>> calificaciones['Educación Física']=87
>>> calificaciones['Historia']=90
>>> print (len(calificaciones), calificaciones)
6 {'Inglés': 85, 'Mate': 10, 'Historia': 90, 'Arte': 74, 'Danza': 90, 'Educación Física': 87}
```



Diccionarios | Métodos

`get(clave, valor)`

`pop(clave,valor)`

`popitem()`

`copy()`

`clean()`

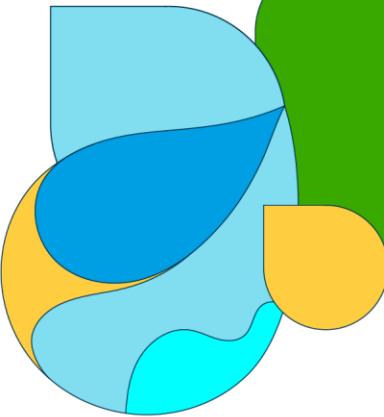
`update()`

`setdefault()`

`keys()`

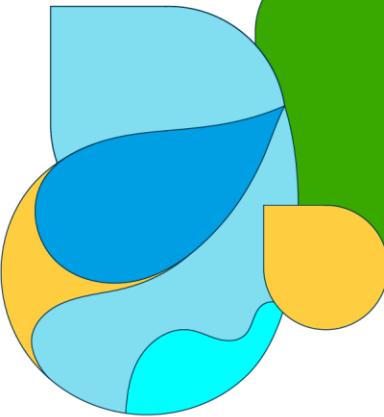
`values()`

`items()`



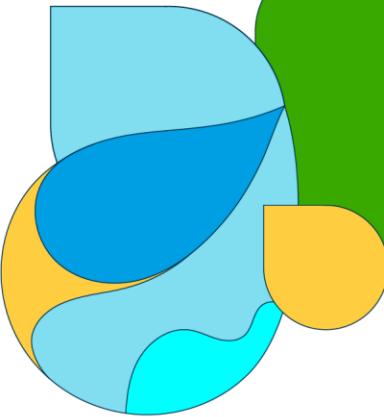
Diccionarios | update | Ejemplos

```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}
>>>
>>> calificaciones.update({'Música':80, 'Historia':89}) # Cambia & agrega
>>> print (calificaciones)
{'Inglés': 80, 'Música': 80, 'Matemáticas': 100, 'Español': 88, 'Historia': 89}
>>>
>>> # Observe que se actualiza el diccionario calificaciones con otro diccionario
```



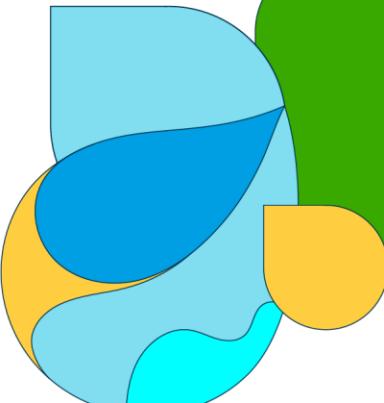
Diccionarios | update | Ejemplos

```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}  
>>>  
>>> calificaciones.update(Matematicas=95, Deportes=100) # Cambia & agrega  
>>> print(calificaciones)  
{'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88, 'Matematicas': 95, 'Deportes': 100}  
>>>  
>>> # Actualiza el diccionario con clave=valor, clave=valor, ...
```



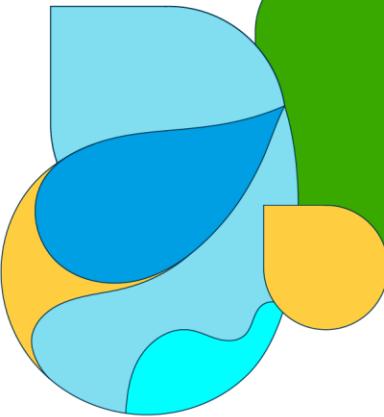
Diccionarios | update | Ejemplos

```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}
>>>
>>> calificaciones.update([ ('Matematicas',80),('Historia',80) ]) # Actualiza & agrega
>>> print(calificaciones)
{'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88, 'Matematicas': 80, 'Historia': 80}
>>>
>>> # Actualiza con un iterable (list) de pares clave/valor
>>>
>>> calificaciones['Matemáticas']=100
>>> print(calificaciones)
{'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88, 'Matematicas': 80, 'Historia': 80}
>>>
>>> # ¿Porqué aparentemente Matemáticas aparece dos veces?
```



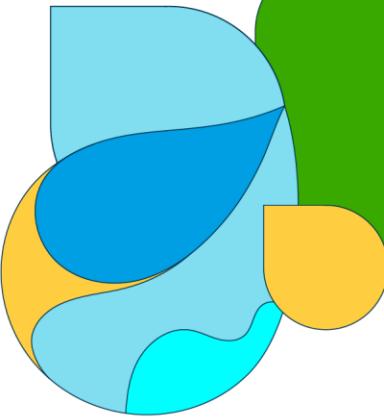
Diccionarios | setdefault | Ejemplos

```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}
>>>
>>> calificaciones.setdefault('Español',80) # Español existe, no cambia
88
>>> print (calificaciones)
{'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88}
>>> print (calificaciones['Español'])
88
>>> calificaciones.setdefault('Deportes',100) # Deportes es nuevo, se agrega
100
>>> print (calificaciones['Deportes'])
100
>>> print (len(calificaciones),calificaciones)
5 {'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88, 'Deportes': 100}
```



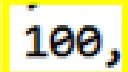
Diccionarios | Vistas | Ejemplos

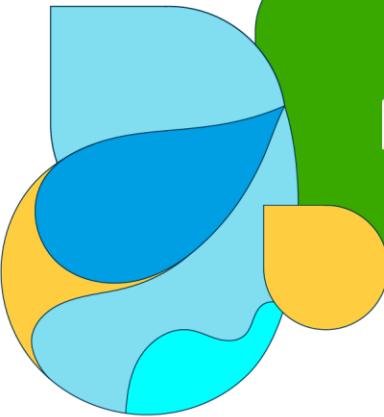
```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}
>>> print (len(calificaciones), calificaciones)
4 {'Inglés': 80, 'Música': 60, 'Matemáticas': 100, 'Español': 88}
>>>
>>> type(calificaciones.keys())
<class 'dict_keys'>
>>> calificaciones.keys()
dict_keys(['Inglés', 'Música', 'Matemáticas', 'Español'])
>>>
>>> type(calificaciones.values())
<class 'dict_values'>
>>> calificaciones.values()
dict_values([80, 60, 100, 88])
>>>
>>> type(calificaciones.items())
<class 'dict_items'>
>>> calificaciones.items()
dict_items([('Inglés', 80), ('Música', 60), ('Matemáticas', 100), ('Español', 88)])
```



Diccionarios | Vistas | Ejemplos

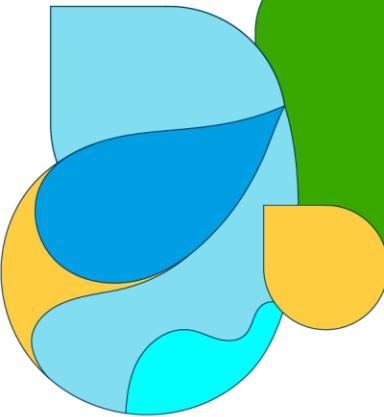
```
>>> calificaciones = {'Inglés':80, 'Música':60, 'Matemáticas':100, 'Español':88}  
>>>  
>>> numeros = calificaciones.values()  
>>>  
>>> print (len(numeros), numeros)  
4 dict_values([80, 60, 100, 88])  
>>> calificaciones['Matemáticas']=80  
>>> print (len(numeros), numeros)  
4 dict_values([80, 60, 80, 88])
```





Función Preconstruida len | Ejemplos

```
>>> len('Netec')
5
>>> len('Python' 'Netec')
11
>>> len(['a','e','i','o','u'])
5
>>> len((255,255,255))
3
>>> len({"Mate":100,"Español":100,"Arte":100})
3
>>>
```



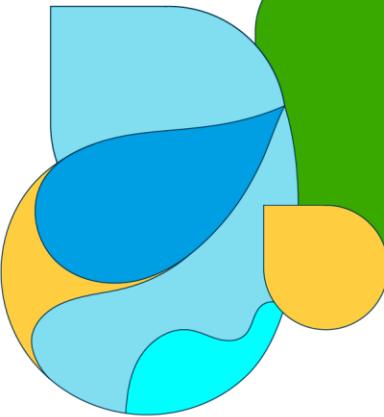
Conjuntos

medidas = { 'chico', 'mediano', 'grande'}

vocales = {'a', 'e ', 'i', 'o', 'u'}

materias = {'English', 'Matemáticas', 'Historia', 'Arte',
'Deportes'}

dígitos = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 0 }



Conjuntos | Ejemplos

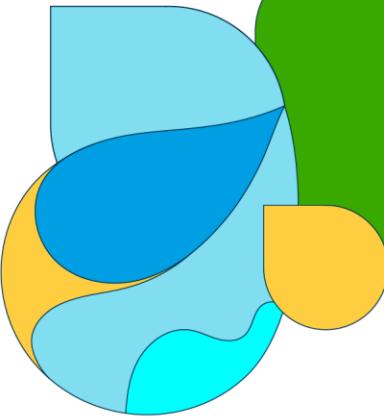
```
vegetales = ['tomates','espinacas','pimientos','brocoli','tomates','brocoli']

print("Lista: ", vegetales)

v_set = set(vegetales) # de lista a conjuntos
print("Conjunto: ", v_set)

vegetales = list(v_set) # de conjunto a lista
print("Lista: ", vegetales) # lista sin duplicados
```

list() & set()

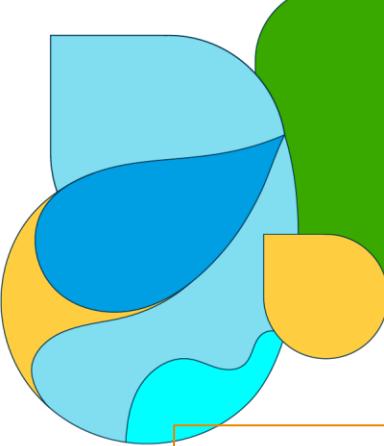


Conjuntos

```
Administrator: Command Prompt

C:\Users\Administrator\ws_python\demos>python remove_dups.py
Lista: ['tomates', 'espinacas', 'pimientos', 'brocolí', 'tomates', 'brocolí']
Conjunto: {'tomates', 'brocolí', 'espinacas', 'pimientos'}
Lista: ['tomates', 'brocolí', 'espinacas', 'pimientos']

C:\Users\Administrator\ws_python\demos>
```



Conjuntos | Métodos

`add()`

`clear()`

`copy()`

`difference()`

`intersection()`

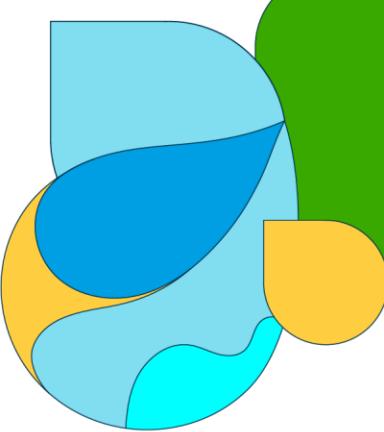
`union()`

`issubset()`

`issuperset()`

`pop()`

`remove()`



Compresión de Listas | Sin

```
lista_nueva = []
for ele in lista_original:
    if filter(ele):
        lista_nueva.append(expressions(ele))
```

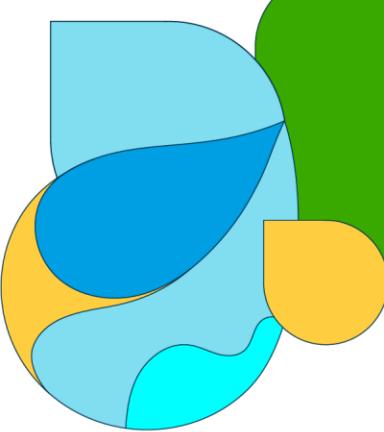
3

1

2

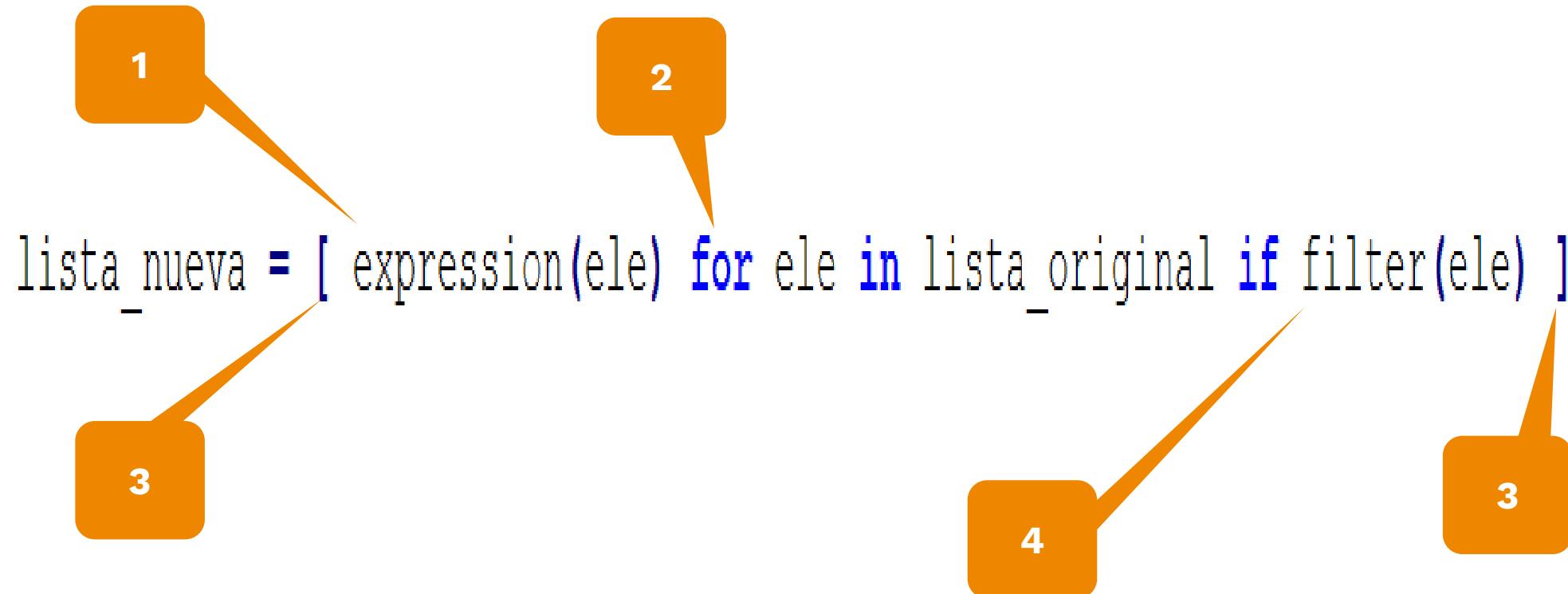
4

5

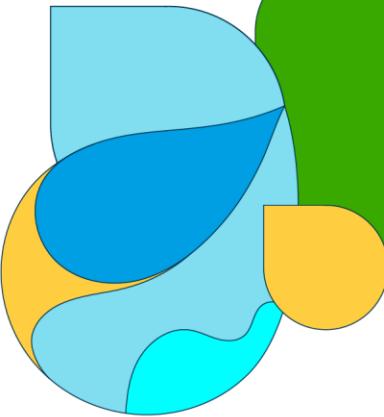


Compresión de Listas | Con

```
lista_nueva = [ expression(ele) for ele in lista_original if filter(ele) ]
```

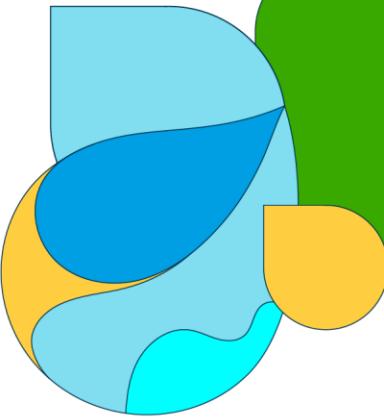


- 1: expression(ele)
- 2: for ele in lista_original
- 3: if filter(ele)
- 4: lista_nueva = [

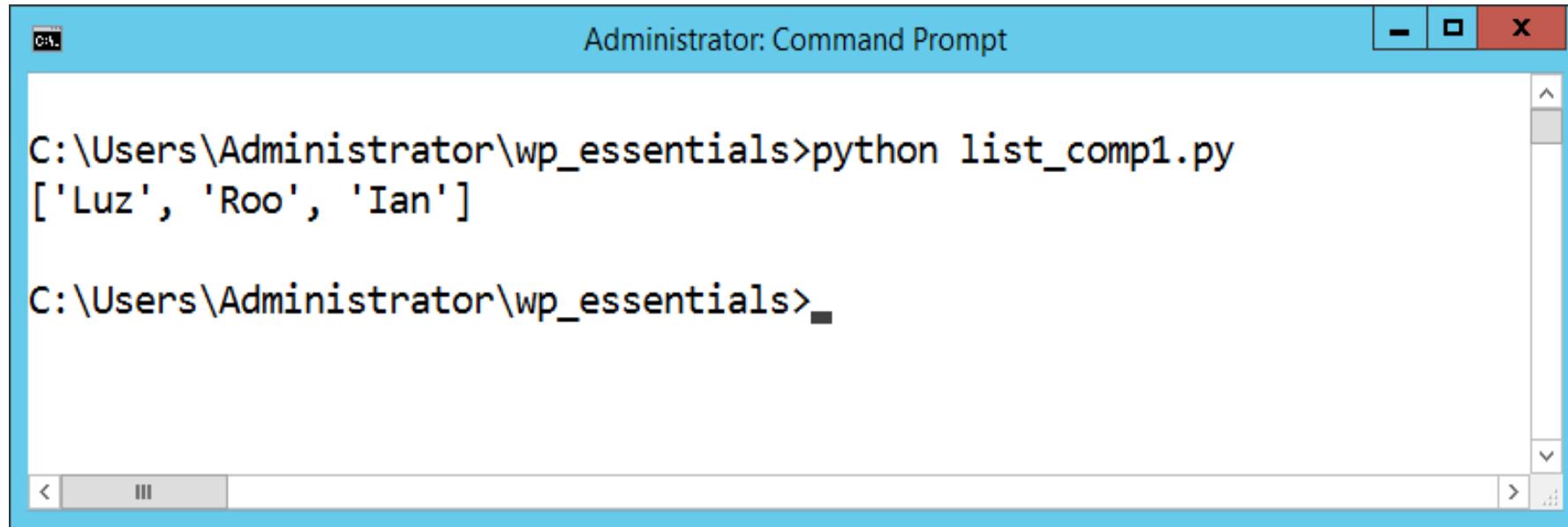


Compresión de Listas | Ejemplo

```
words = ['Marisol', 'Saúl', 'Alejandra', 'Patricia', 'Abraham', 'Yuri',
         'Luz', 'Carmen', 'Violeta', 'Greta', 'Adriana', 'Alicia', "Roo", "Yum"
         'Adriana', 'Ximena', 'David', 'Ricardo', 'Eduardo', 'Enrique',
         'Almudena', 'Gabriela', 'Leticia', 'Lucia', 'Eduardo', 'Ian']
tres_letras = [w for w in words if len(w) == 3]
print(tres_letras)
```



Compresión de Listas | Ejemplo



```
Administrator: Command Prompt
C:\Users\Administrator\wp_essentials>python list_comp1.py
['Luz', 'Roo', 'Ian']

C:\Users\Administrator\wp_essentials>
```

*args & **kwargs

- Ambos se utilizan en funciones para aceptar un número arbitrario de argumentos.
- *args indica un número variable de argumentos.
- **kwargs un número variable de argumentos nombrados, clave=valor.

*args | Ejemplo

```
def suma(num1, num2, num3 = 0, num4 = 0, num5 = 0):
    total = num1 + num2 + num3 + num4 + num5
    print(num1, ' + ', num2, ' + ', num3, ' + ', num4, ' + ', num5, ' = ', total)

def main():
    suma(1,2,3,4,5)
    suma(1,2,3,4)
    suma(1,2,3)
    suma(1,2)
    # suma(1) genera un error

main()
```

```
C:\Users\Administrator\ws_python\demos>python demo_funcion.py
1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 + 0 = 10
1 + 2 + 3 + 0 + 0 = 6
1 + 2 + 0 + 0 + 0 = 3
```

*args | Ejemplo

```
def suma2(num1, *nums):
    total = sum(nums, num1)
    str_nums= [str(i) for i in nums]
    print(num1, ' + ', ' + '.join(str_nums), ' = ', total)

def main():
    suma2(1,2,3,4,5)
    suma2(1,2,3,4)
    suma2(1,2,3)
    suma2(1,2)
    suma2(1) # Ya no tiene error

main()
```

1 + 2 + 3 + 4 + 5 = 15
1 + 2 + 3 + 4 = 10
1 + 2 + 3 = 6
1 + 2 = 3
1 + = 1

Resumen del capítulo

- En este capítulo, se ha aprendido a usar las listas, las tuplas, los rangos, los diccionarios y los conjuntos.
- Las listas son mutables mientras que las tuplas no.
- Las tuplas tienen mejor rendimiento al iterar que las listas.
- Las colecciones: listas, tuplas, diccionarios y conjuntos tienen un número considerable de métodos que los mostrados en esta presentación.
- *args y **kwargs son usados cuando se necesita definir una función que no se sabe con cuantos parámetros debería ser invocada.

Práctica: Piedra, Papel o Tijeras

1. Copia las siguientes líneas de código en un archivo de nombre `p5_1.py`:

```
import random
def main():
    pass # Ponga aquí su código de la práctica
main()
```



Práctica: Piedra, Papel o Tijeras

1. Escribe la función principal `main()` que realice lo siguiente:
 - a) Crea una secuencia con tres elementos: "Piedra", "Papel" y "Tijeras".
 - b) Realiza una elección al azar.
 - c) Solicita al usuario lo siguiente: 1 para Roca, 2 para Papel, 3 para Tijeras.
2. Despliega la elección del usuario y la elección de la máquina.
3. La salida de la práctica puede ser similar a la siguiente captura de ventana:



Práctica: Piedra, Papel o Tijeras

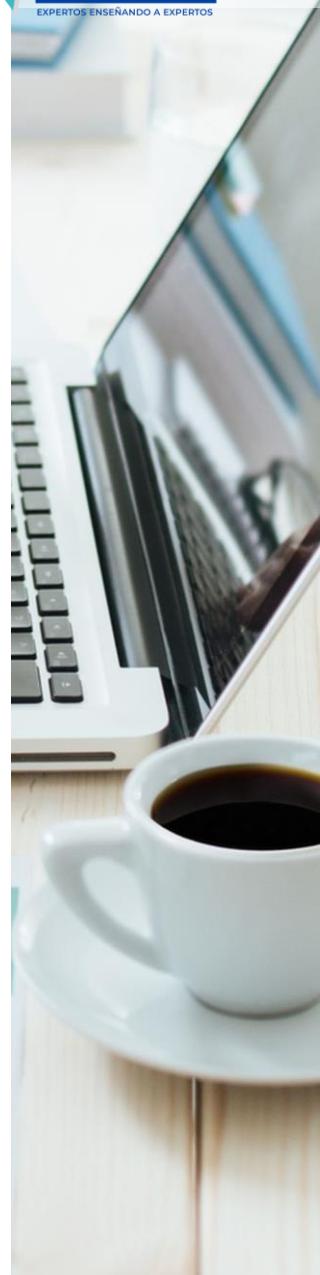
```
C:\> Administrator: Command Prompt
C:\Users\Administrator\ws_python\soluciones>python p5_1.py
1 para Roca, 2 para Papel, 3 para Tijeras: 1
Computadora: Tijeras
Userio: Roca

C:\Users\Administrator\ws_python\soluciones>python p5_1.py
1 para Roca, 2 para Papel, 3 para Tijeras: 1
Computadora: Papel
Userio: Roca

C:\Users\Administrator\ws_python\soluciones>python p5_1.py
1 para Roca, 2 para Papel, 3 para Tijeras: 1
Computadora: Roca
Userio: Roca

C:\Users\Administrator\ws_python\soluciones>python p5_1.py
1 para Roca, 2 para Papel, 3 para Tijeras: 3
Computadora: Papel
Userio: Tijeras

C:\Users\Administrator\ws_python\soluciones>
```



Práctica: Split

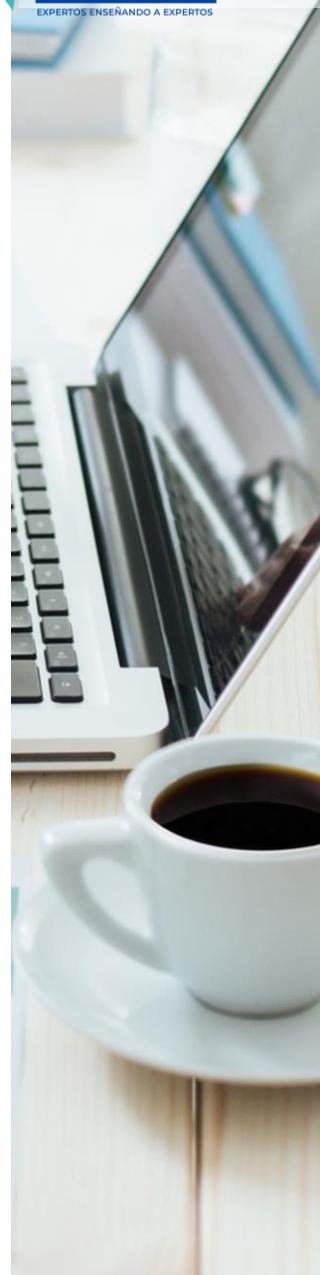
1. Copia el siguiente código en un archivo de nombre p5_2.py:

```
import math

def split_list(orig_list):
    pass # Ponga aquí su código.

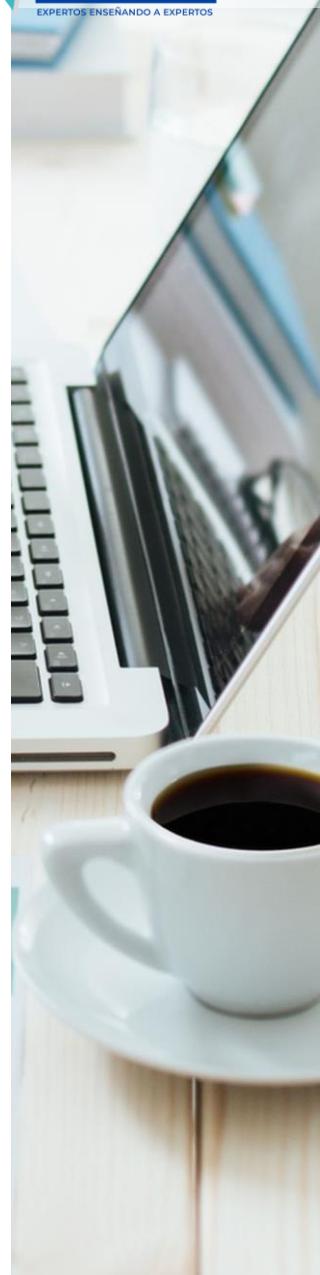
def main():
    colores = ['rojo', 'azul', 'verde', 'naranja', 'morado']
    colores_split = split_list(colores)
    print(colores_split[0])
    print(colores_split[1])

main()
```



Práctica: Split

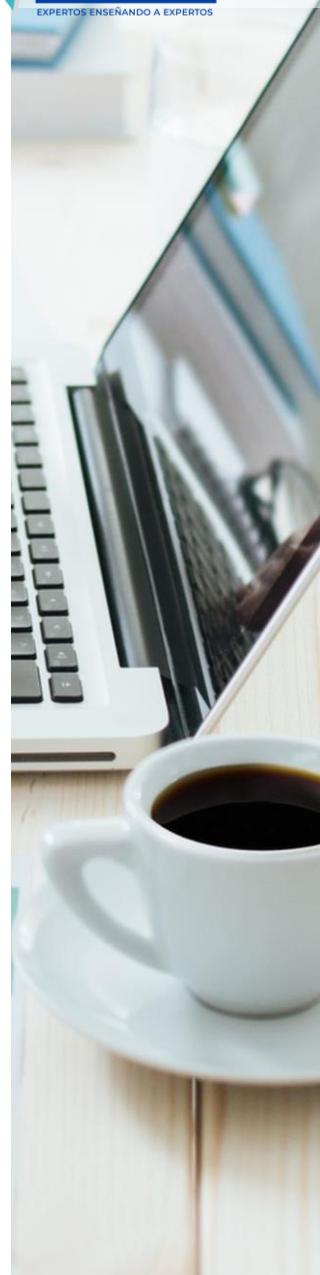
1. Escribe la función **split_list()** de modo que devuelva una lista con dos sublistas: la primera y la segunda mitad de la lista original. Por ejemplo, si se pasa como argumento a la función `[1, 2, 3, 4]`, **split_list()** debe devolver: `[[1, 2], [3, 4]]`.
2. Si la lista original tiene un número impar de elementos, la función debe colocar el elemento adicional en la primera lista. Por ejemplo, cuando se pasa como argumento a la función `[1, 2, 3, 4, 5]`, **split_list()** se devolverá: `[[1, 2, 3], [4,5]]`.
3. Cuando se ejecute el programa, la salida deberá ser similar a la siguiente captura de ventana:



Práctica: Split

```
C:\>Administrator: Command Prompt
C:\Users\Administrator\ws_python\soluciones>
C:\Users\Administrator\ws_python\soluciones>python p5_2.py
['rojo', 'azul', 'verde']
['naranja', 'morado']

C:\Users\Administrator\ws_python\soluciones>
```



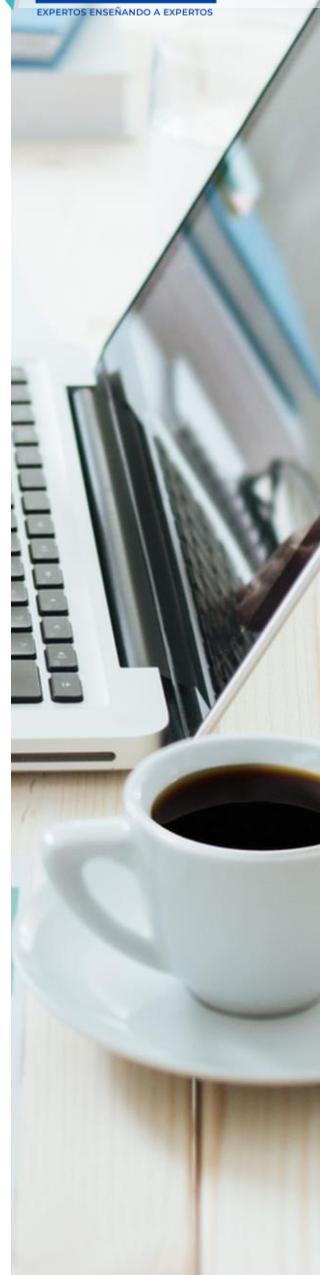
Práctica: Diccionarios

1. Captura el siguiente fragmento de código a un archivo de nombre `p5_3.py`:

```
def main():
    pass # Reemplace esto por su código de la práctica.
main()
```

2. Escribe la función `main()` de modo que:

- a) Cree un diccionario de nombre `calificaciones` y agregue al diccionario las calificaciones introducidas por el usuario para las materias siguientes: Inglés, Matemáticas, Historia, Arte y Música.
- b) Determine el promedio e imprímelo a la salida. Ten en cuenta que al hacer esto se necesitará convertir los números ingresados a enteros.



Práctica: Diccionarios

- c) La captura de pantalla siguiente muestra cómo debe ejecutarse el programa:



```
Administrator: Command Prompt
C:\Users\Administrator\ws_python\soluciones>
C:\Users\Administrator\ws_python\soluciones>python p5_3.py
Ingrese las calificaciones para las siguientes materias
Inglés: 10
Matemáticas: 10
Historia: 9
Arte: 8
Música: 7
El promedio es: 8.8

C:\Users\Administrator\ws_python\soluciones>python p5_3.py
Ingrese las calificaciones para las siguientes materias
Inglés: 6
Matemáticas: 6
Historia: 10
Arte: 10
Música: 10
El promedio es: 8.4

C:\Users\Administrator\ws_python\soluciones>
```

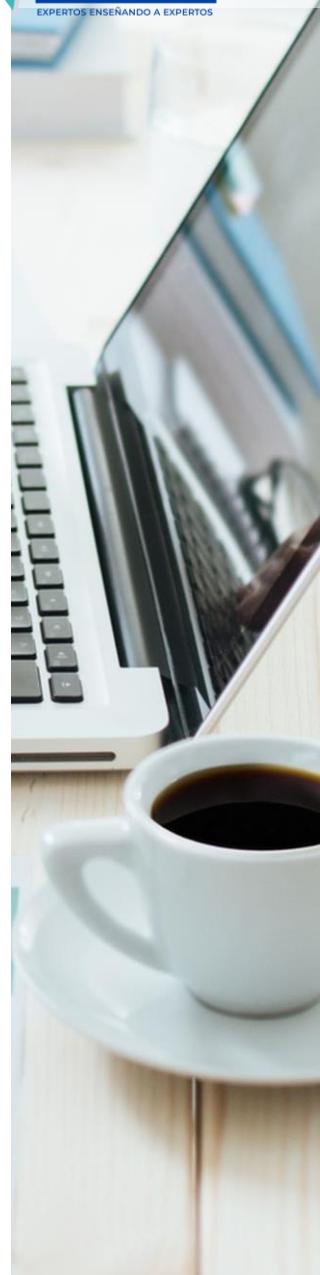
Práctica: Diccionarios

3. Después de la impresión del promedio, pide al usuario actualizar la calificación de alguna materia y luego obtén el nuevo promedio e imprima el nuevo resultado.

```
C:\> Administrator: Command Prompt
C:\Users\Administrator\ws_python\soluciones>python p5_3r.py
Inglés: 10
Matemáticas: 10
Historia: 10
Arte: 10
Música: 10
El promedio es: 10.0
Seleccione la materia a cambiar: Música
¿Cuál es la nueva calificación para Música? 6
El nuevo promedio es: 9.33

C:\Users\Administrator\ws_python\soluciones>python p5_3r.py
Inglés: 8
Matemáticas: 7
Historia: 5
Arte: 6
Música: 8
El promedio es: 6.8
Seleccione la materia a cambiar: Historia
¿Cuál es la nueva calificación para Historia? 8
El nuevo promedio es: 7.00

C:\Users\Administrator\ws_python\soluciones>
```





Referencias Bibliográficas

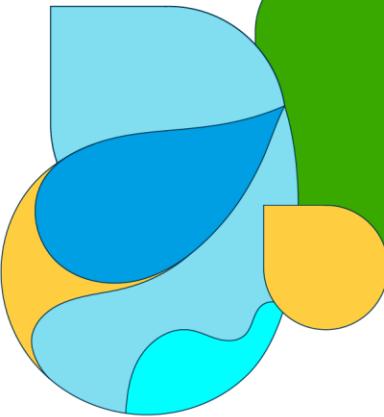
- Python List: https://www.w3schools.com/python/python_lists.asp
- Python Tuples:
https://www.w3schools.com/python/python_tuples.asp
- Python Sets: https://www.w3schools.com/python/python_sets.asp
- Python Dictionaries:
https://www.w3schools.com/python/python_dictionaries.asp
- Argumentos en funciones: <https://recursospython.com/guias-y-manuales/argumentos-args-kwargs/>
- Python Tuples are Not Just Constant Lists:
http://jtauber.com/blog/2006/04/15/python_tuples_are_not_just_constant_lists/

Unidad temática 8

Excepciones

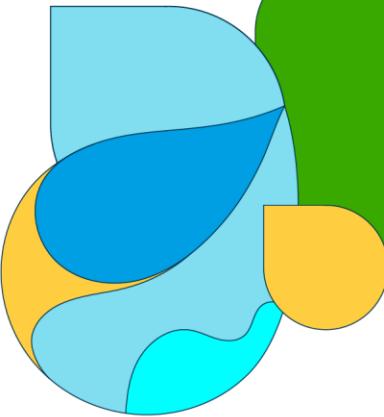
Objetivos:

- Trabajar con excepciones en Python.
- Listar las palabras reservadas para el control de excepciones.
- Usar try y except para administrar excepciones.
- Enumerar las excepciones preconstruidas en Python.



Introducción

```
>>> 100 + x
      File "<stdin>", line 1
          100 + x
IndentationError: unexpected indent
>>>
>>> 100 * (1/0)
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>>
>>> "100" + 100
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```



Gestión de Excepciones

try

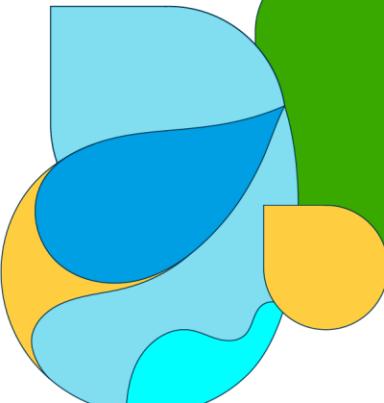
except

as

else

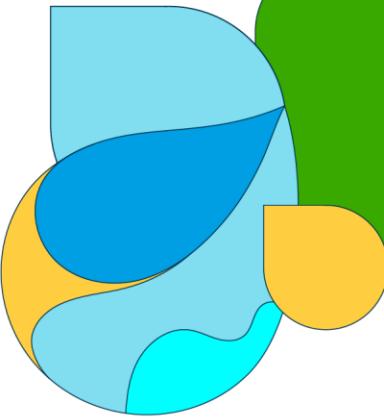
finally

raise



Gestión de Excepciones

```
>>>
>>> def div(x):
...     try:
...         return 100 + (x/0)
...     except:
...         print ("No se permite dividir entre cero")
...     print("El programa continua")
...
>>> div(10)
No se permite dividir entre cero
El programa continua
```



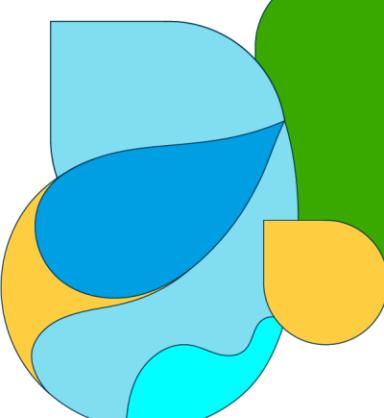
Gestión de Excepciones

```
def divide():
    try:
        numerator = int(input('Ingrese el numerador: '))
        denominator = int(input('Ingrese el denominador: '))
        result = numerator / denominator
        print(numerator, '/', denominator, '=', result)
    except ValueError:
        print('Ingrese solo enteros. Intente de nuevo.')
        divide()
    except ZeroDivisionError:
        print('No puede dividir entre cero. Intente de nuevo.')
        divide()
    except KeyboardInterrupt:
        print('Fin')
    except:
        print('No se tiene idea de lo que sucede')
```

Gestión de Excepciones

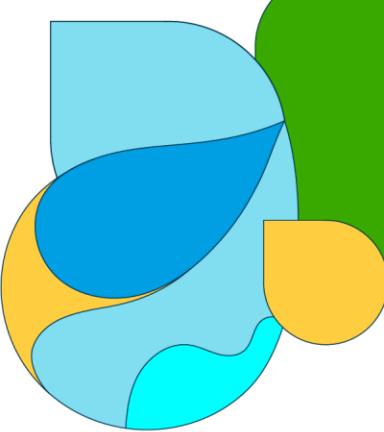
```
C:\> Administrator: Command Prompt
C:\Users\Administrator\ws_python\demos>python specific.py
Ingrese el numerador: 100
Ingrese el denominador: x
Ingrese solo enteros. Intente de nuevo.
Ingrese el numerador:
Ingrese solo enteros. Intente de nuevo.
Ingrese el numerador: 100
Ingrese el denominador: 0
No puede dividir entre cero. Intente de nuevo.
Ingrese el numerador: ^Z
No se tiene idea de lo que sucede

C:\Users\Administrator\ws_python\demos>python specific.py
Ingrese el numerador: 100
Ingrese el denominador: 10
100 / 10 = 10.0
```



Gestión de Excepciones

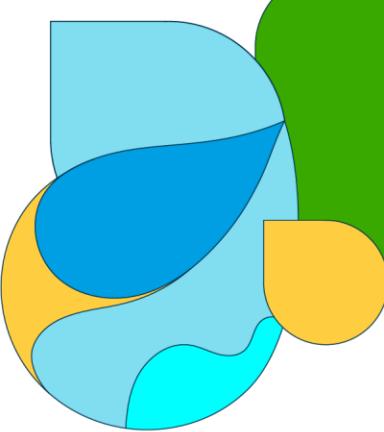
```
try:  
    100/0  
except Exception as exc:  
    print(type(exc)) # Despliega el tipo de excepción  
    print(exc) # Despliega el valor de str(exc)
```



Cláusula else

```
def divide():
    try:
        numerador = int(input('Ingrese el numerador: '))
        denominador = int(input('Ingrese el denominador: '))
        resultado = numerador / denominador
    except ValueError:
        print('Ingrese solo numeros. Intente de nuevo.')
        divide()
    except ZeroDivisionError:
        print('No puede dividir entre cero. Intente de nuevo.')
        divide()
    except KeyboardInterrupt:
        print('Fin!')
        raise
    except:
        print('No se tiene idea de lo que pasa, comodín')
    else:
        print(numerador, '/', denominador, '=', resultado)
```

Cláusula else



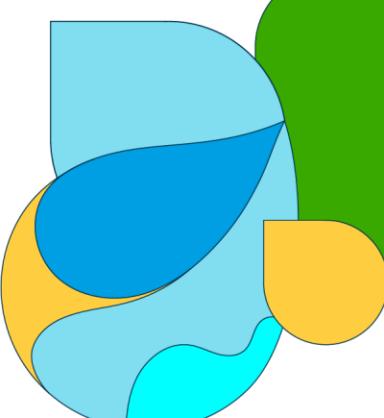
```
Administrator: Command Prompt
C:\Users\Administrator\ws_python\demos>python else.py
Ingrese el numerador:
Ingrese solo numeros. Intente de nuevo.
Ingrese el numerador: letras
Ingrese solo numeros. Intente de nuevo.
Ingrese el numerador: 100
Ingrese el denominador: 0
No puede dividir entre cero. Intente de nuevo.
Ingrese el numerador: 100
Ingrese el denominador: 100
100 / 100 = 1.0

C:\Users\Administrator\ws_python\demos>
```

Cláusula finally

```
def divide():
    try:
        numerador = int(input('Ingrese el numerador: '))
        denominador = int(input('Ingrese el denominador: '))
        resultado = numerador / denominador
    except ValueError:
        print('Ingrese solo numeros. Intente de nuevo.')
        divide()
    except ZeroDivisionError:
        print('No puede dividir entre cero. Intente de nuevo.')
        divide()
    except KeyboardInterrupt:
        print('Fin!')
        raise
    except:
        print('No se tiene idea de lo que pasa, comodín')
    else:
        print(numerador, '/', denominador, '=', resultado)
    finally:
        print("Siempre se ejecuta....")
```

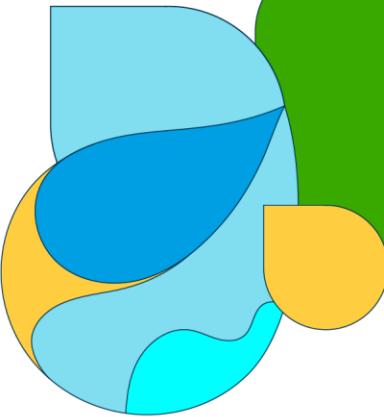
```
Ingrese el numerador:
Ingrese solo numeros. Intente de nuevo.
Ingrese el numerador: a
Ingrese solo numeros. Intente de nuevo.
Ingrese el numerador: 100
Ingrese el denominador: 3
100 / 3 = 33.33333333333336
Siempre se ejecuta....
Siempre se ejecuta....
Siempre se ejecuta....
```



Control de flujo

```
def pote2():
    try:
        num = int(input('Ingrese un entero: '))
    except ValueError:
        print('Esto no es un entero')
    else:
        print(num, '**2 = ', num**2)

def pote3():
    num = input('Ingrese un entero: ')
    if num.isdigit():
        print(num, '**3', int(num)**3)
    else:
        print('No es un numero entero.')
```



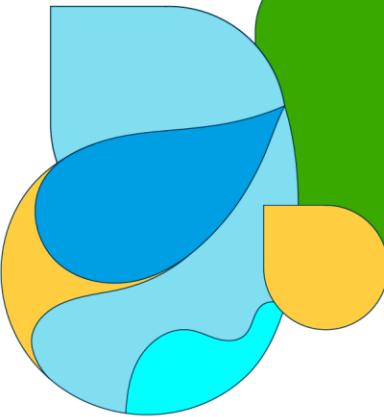
Excepciones Personalizadas

```
def bidict(d):
    d2 = d.copy()
    for k,v in d.items():
        d2[v] = k
    return d2
```

{'inicio': '¡Hola!', 'fin': '¡Adiós!'}



{'inicio':'¡Hola!', 'fin':'¡Adiós!', '¡Hola!':'inicio', '¡Adiós!':'fin'}

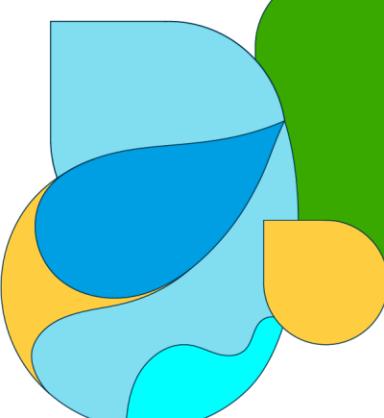


Excepciones Personalizadas

```
def bidict(d):
    d2 = d.copy()
    for k,v in d.items():
        if v in d2.keys():
            raise KeyError('No se puede crear un diccionario invertido ' + ' con claves duplicadas.')
        d2[v] = k
    return d2
```



raise



Excepciones Predefinidas

BaseException

Exception

SystemExit

ZeroDivisionError

EOFError

ImportError

NameError

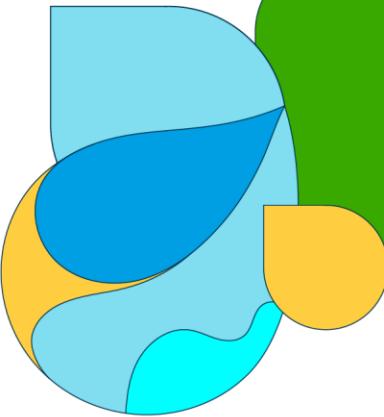
FileExistError

PermissionError

AritmeticError

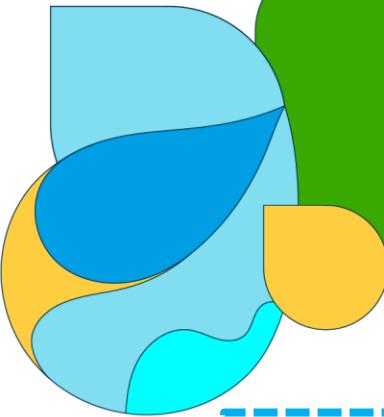
InterruptedError

FileNotFoundException



Aserciones

- Una aserción o afirmación es un predicado, es decir, una sentencia verdadera o falsa.
- Indica que dicho predicado siempre se cumple.
- Suelen ser útiles para programas específicos, así como para la corrección de los mismos.
- Su uso frecuente es para pruebas unitarias y para depurar código.
- Si el predicado es falso se genera un **AssertionError**.



Aserciones

```
valor="Netec"
```

```
#Si la condición siguiente es False, AssertionError es lanzado.
```

```
assert valor == "Gobal", "valor debería ser 'Netec'"
```

```
Traceback (most recent call last):
```

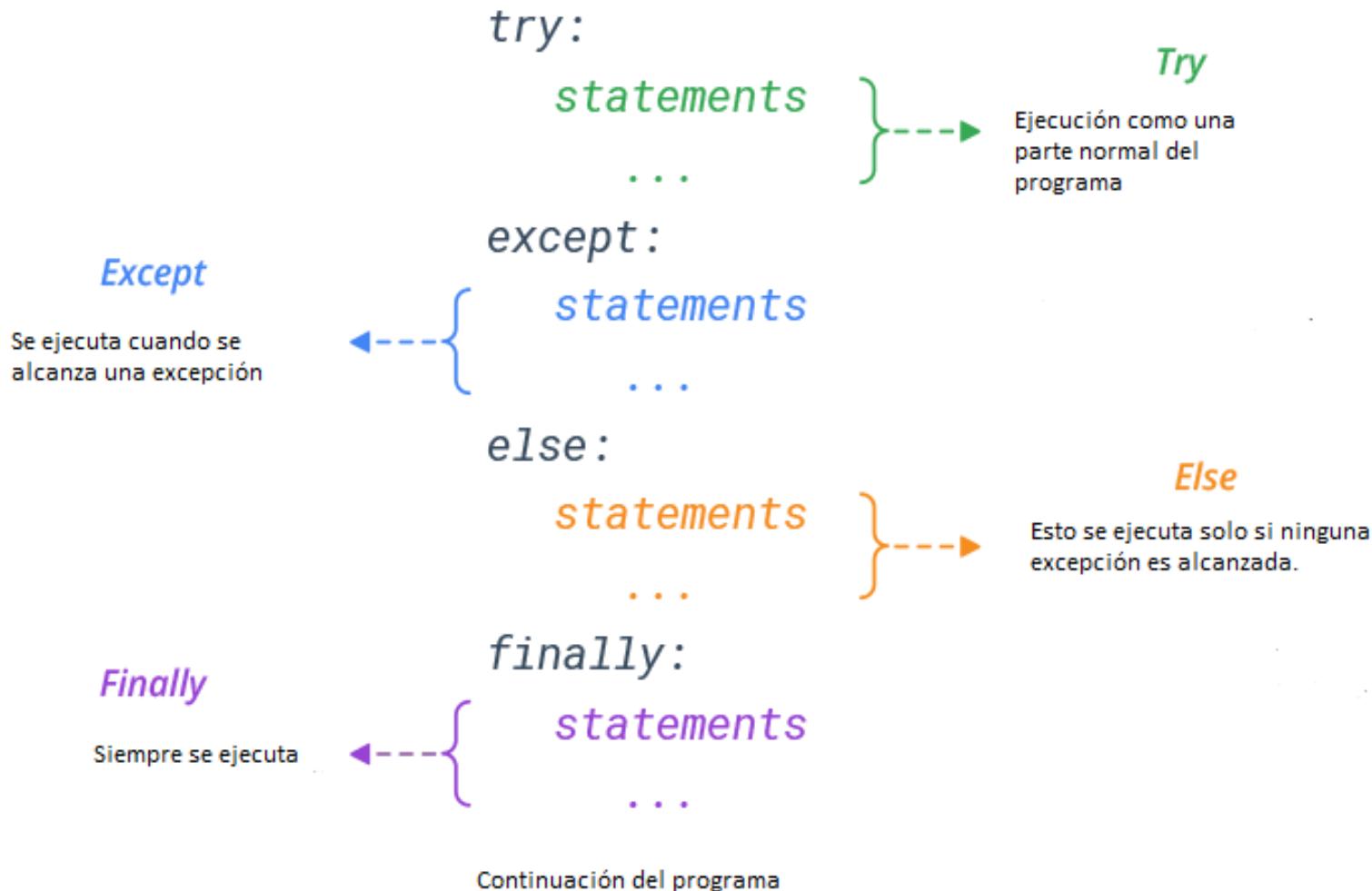
```
  File "C:\Users\Administrator\ws_python\demos\assert.py", line 6, in <module>
    assert valor == "Gobal", "valor debería ser 'Netec'"
```

```
AssertionError: valor debería ser 'Netec'
```

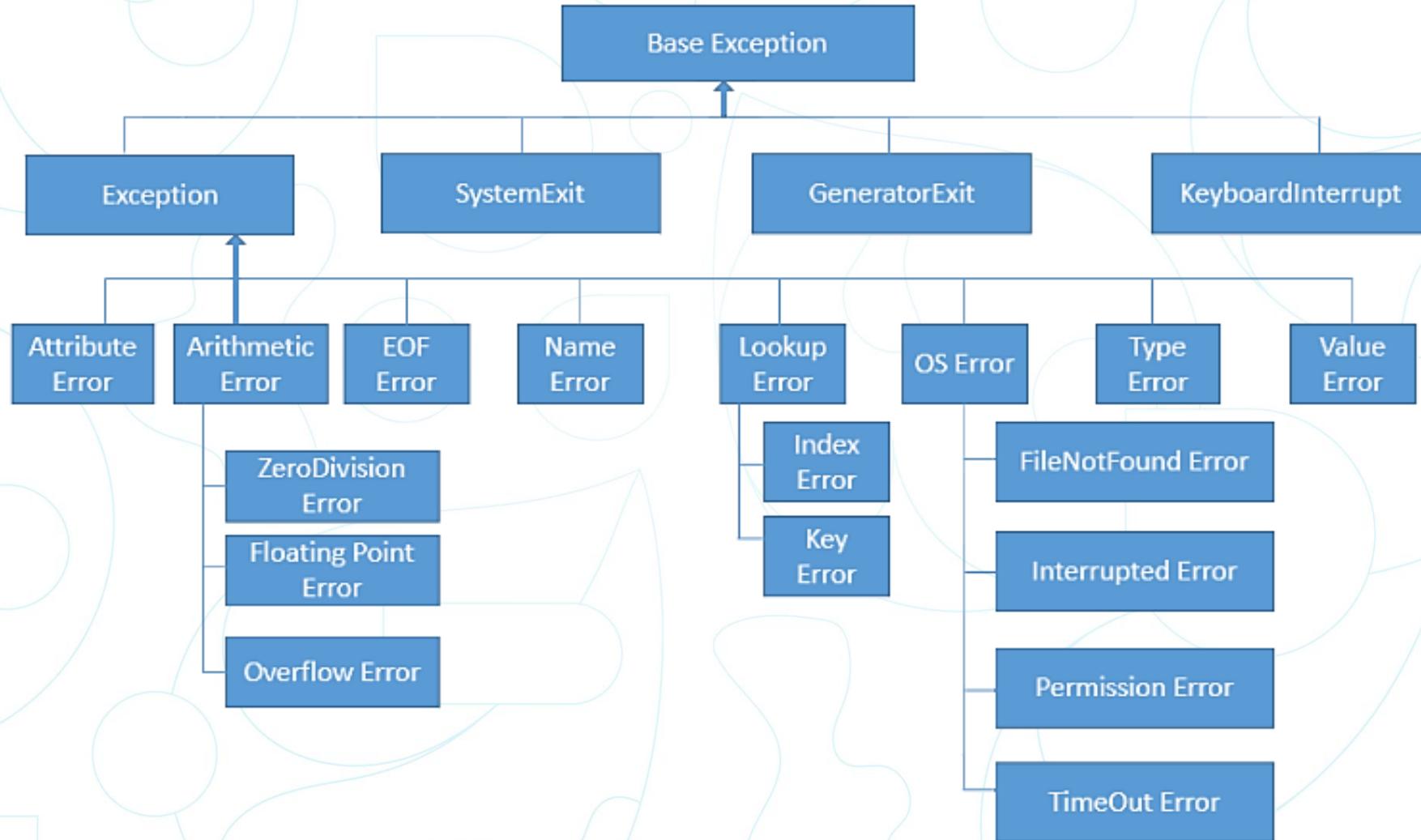
Resumen del capítulo

Keyword	Descripción
try	Protege un bloque de código.
except	Ejecuta un bloque de código en caso de que suceda una excepción.
as	Pone un alias a la excepción generada, se usa en combinación con except.
else	Sirve para especificar un bloque de código que solamente se ejecutará si el flujo del programa es exitoso.
finally	Indica un bloque de código que siempre se ejecuta. Este bloque de código es adecuado para cerrar recursos.
raise	Sirve para lanzar excepciones.

Resumen del capítulo



Resumen del capítulo



Práctica: Manejo de Excepciones

1. En cada bloque try/except inserta el código para generar los siguientes tipos de errores:
 - a) ZeroDivisionError.
 - b) ValueError.
 - c) NameError.
 - d) FileNotFoundError.
 - e) ImportError.
 - f) Error de tecleado.
 - g) AttributeError.
 - h) StopIteration.
 - i) KeyError.



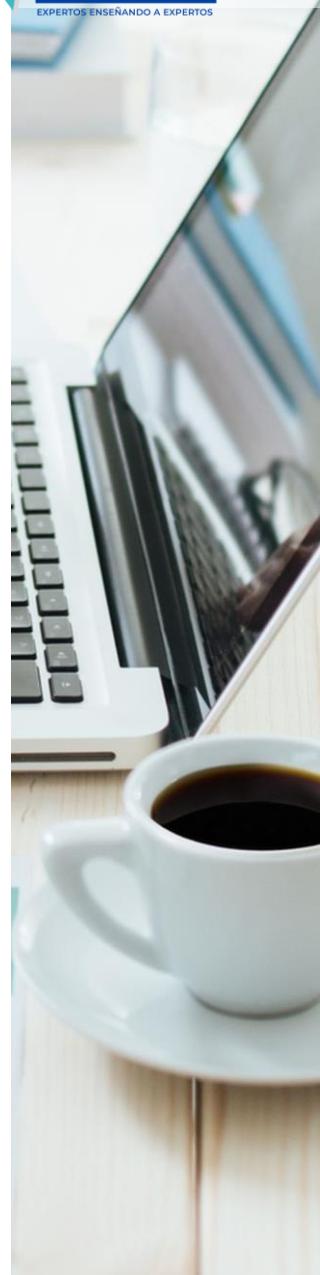
Práctica: Manejo de Excepciones

2. Repite el código en el recuadro naranja para cada excepción de la lista:

```
try:  
    1/0  
except Exception as e:  
    print(type(e))  
    print(e, '\n')
```

```
try:  
    pass #Ingrese el código para generar la excepción <Exception>  
except Exception as e:  
    print(type(e))  
    print(e, '\n')
```

3. Usa la documentación en <https://docs.python.org/3/library/exceptions.html>



Práctica: Sumar Indefinidamente

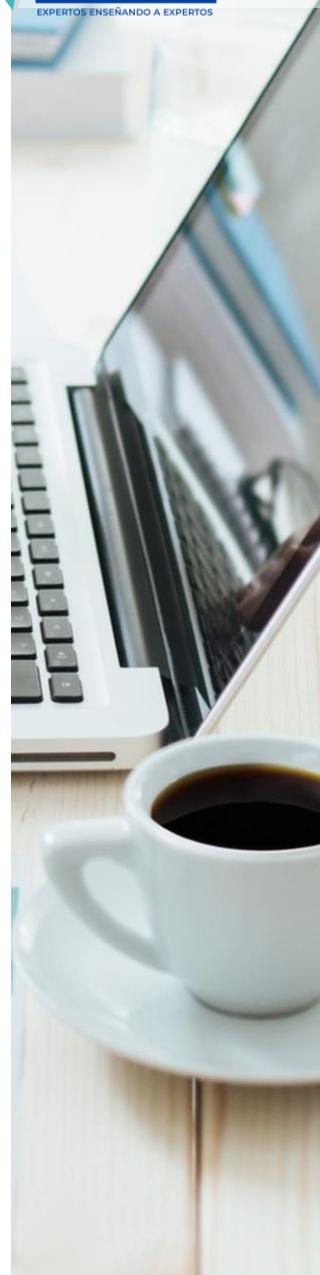
```
# Función suma recursiva, salir con [Ctrl]+C
def sumar(total=0):
    num = input('Ingrese un número: ')

    if not num.isdigit():
        print('Ingrese solamente números, intente nuevamente.')
    else:
        total += int(num)
        print('El total actual es:', total)

    sumar(total)

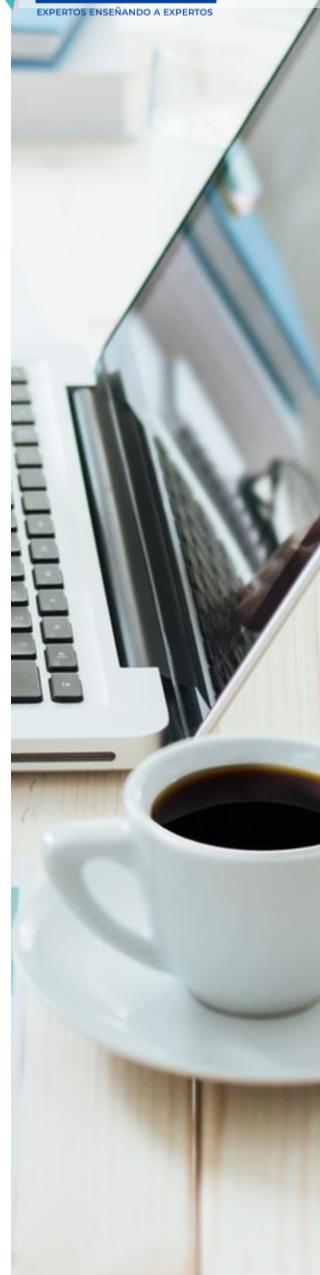
def main():
    sumar()

main()
```



Práctica: Sumar Indefinidamente

4. Prueba el programa del punto anterior.
5. Reemplaza el bloque if-else en la función `sumar()` con un bloque try/except. El resultado deberá ser el mismo que con el uso de if-else.





Referencias Bibliográficas

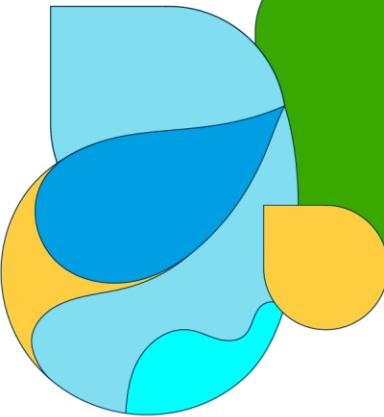
- Errors and Exceptions:
<https://docs.python.org/3/tutorial/errors.html>
- Built-in Exceptions:
<https://docs.python.org/3/library/exceptions.html#ConnectionResetError>
- Exception Hierarchy:
<https://o7planning.org/en/11421/python-exception-handling-tutorial#a1356423>

Unidad temática 9

La programación orientada a
objetos usando python

Objetivos:

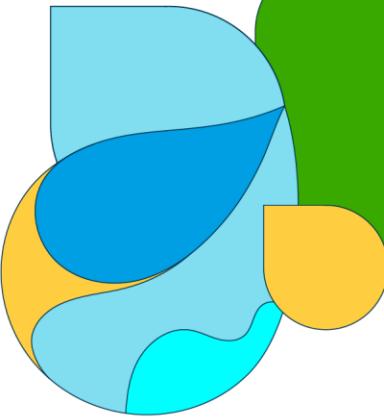
- Reafirmar conceptos de la OOP usando Python.
- Hacer abstracciones del mundo real.
- Crear clases con propiedades y métodos.
- Generar instancias de las clases definidas.
- Codificar métodos instancia, métodos de la clase y métodos estáticos.
- Crear subclases usando herencia.
- Crear clases abstractas.



Atributos

- Los atributos son generalmente sustantivos:
 - ramas, árboles, rocas, partidos, juegos, ligas, personas, mesas, sillas, dados, laptops, monitores, teclados, etc.
- Los atributos también pueden ser adjetivos:
 - pesado, robusto, débil, duro, largo, corto, alto, rápido, lento, claro, obscuro, cifrado, ofuscado, etc.

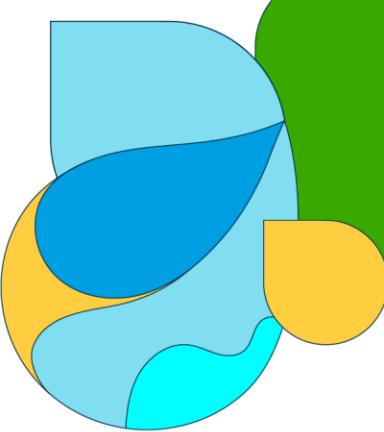
```
roca.peso = 10;  
patio.trasero.arbol.ramas= [rama1, rama2]  
partido.tenis.sets= [set1, set2, set3]
```



Comportamiento

- Los comportamientos por lo general son verbos/acciones.
- Los comportamientos se denominan en OOP, métodos.
 - Funciones definidas dentro de un bloque de clase.

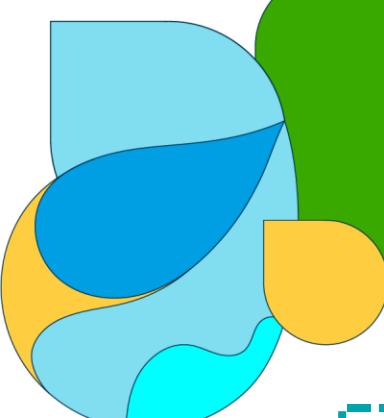
```
roca.caída('rápido')
patio.trasero.arbol.dio(datetime.date(2022, 05, 10))
partido.tenis.finalizo(datetime.time(10, 0))
```



Clases & Objetos

- Una clase es una plantilla para un objeto.
 - Tenista: `reves_a_dos_manos` atributo booleano.
- Un objeto es una instancia de la clase.
 - Serena, Roger, Venus, Nadal.

```
serena.reves_a_dos_manos = True
venus.reves_a_dos_manos = True
rafael.reves_a_dos_manos = True
roger.reves_a_dos_manos = False
```



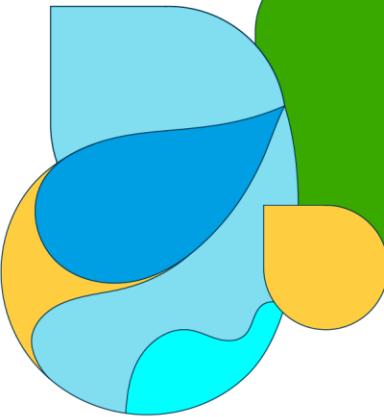
Clases & Objetos

```
class Tenista:  
    pass # Definimos atributos & comportamiento
```

Clase

Objetos

```
serena = Tenista()  
venus = Tenista()  
  
rafael = Tenista()  
roger = Tenista()
```



Clases & Objetos | type

```
print (type('Hola'), isinstance('Hola',str))  
  
print (type(1), isinstance(1, int))  
  
print (type(['a','e','i','o','u']), isinstance ([ 'a','e','i','o','u'], list))  
  
print (type((1,2,3)), isinstance ((1,2,3), tuple))  
  
print (type({'a':1, 'b':2 , 'c':3}), isinstance ({'a':1, 'b':2 , 'c':3}, dict))  
  
print (type(print))
```

```
<class 'str'> True  
<class 'int'> True  
<class 'list'> True  
<class 'tuple'> True  
<class 'dict'> True  
<class 'builtin_function_or_method'>
```

Clases & Objetos | type

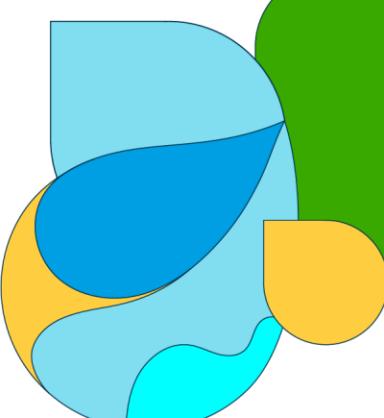
```
mayor = range(18,100)
print(type(mayor), type(range))
```

```
<class 'range'> <class 'type'>
```

```
class Tenista:
    pass

serena = Tenista()
print("serena:", type(serena))
print("Tenista:", type(Tenista))
print("Serena es una tenista:", isinstance(serena, Tenista))
```

```
serena: <class '__main__.Tenista'>
Tenista: <class 'type'>
Serena es una tenista: True
```



Clases & Objetos

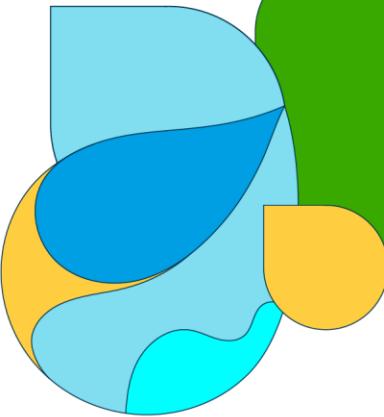


```
class Dado:  
    pass
```

```
import Die  
  
dd = Die.Dado()
```

```
from Die import Dado  
  
dd = Dado()
```

```
>>> import Die  
>>> dd=Die.Dado()  
>>> dd  
<Die.Dado object at 0x00000028AF3BA590>  
>>>  
>>> from Die import Dado  
>>> dd= Dado()  
>>> dd  
<Die.Dado object at 0x00000028AF3BA4D0>  
>>>
```



Constructores

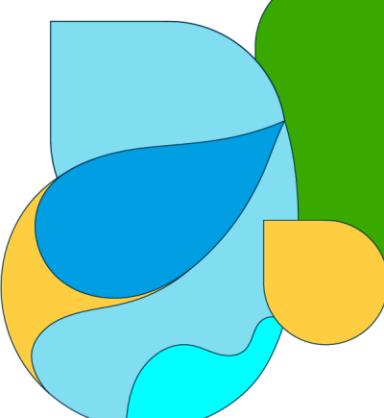
Atributos
`self.*`

```
import math

class Circulo:
    def __init__(self, val, prop='r'):
        if prop == 'r':
            self.radio = val
        elif prop == 'd':
            self.radio = val / 2
        elif prop == 'p':
            self.radio = val / (2 * math.pi)
        elif prop == 'a':
            self.radio = (val / math.pi) ** .5
        else:
            # r: radio, d: diametro, p:perímetro y a:area
            raise Exception('La propiedad debe ser r, d, p, a a')

        self.diametro = self.radio * 2
        self.perimetro = self.radio * 2 * math.pi
        self.area = self.radio ** 2 * math.pi
```

Importante en Python colocar los atributos de la clase fuera del constructor, podría tener otras implicaciones

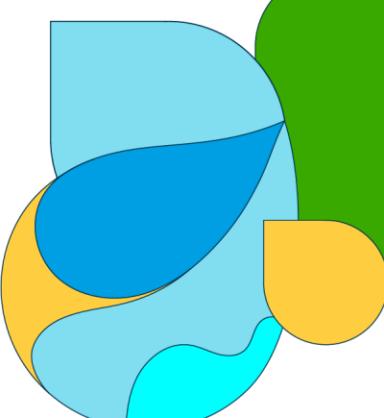


Constructores

```
c = Circulo(8,'d')
print ("Radio: " , c.radio)
print ("Diametro: " , c.diametro)
print ("Perímetro: " , c.perimetro)
print ("Área: " , c.area)
print ('-'*40)

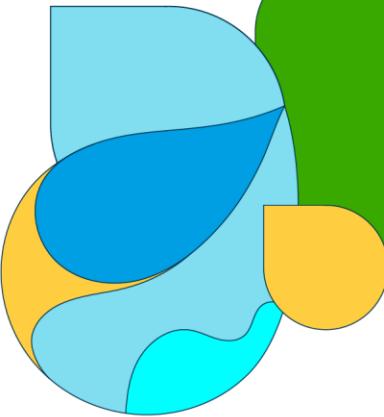
c = Circulo(10,'d')
print ("Radio: " , c.radio)
print ("Diametro: " , c.diametro)
print ("Perímetro: " , c.perimetro)
print ("Área: " , c.area)
print ('-'*40)
```

```
Radio: 4.0
Diametro: 8.0
Perímetro: 25.132741228718345
Área: 50.26548245743669
-----
Radio: 5.0
Diametro: 10.0
Perímetro: 31.41592653589793
Área: 78.53981633974483
```



Métodos

```
def redimensionar(self, amt):
    self.radio *= amt
    self.diametro = self.radio * 2
    self.perimetro = self.radio * 2 * math.pi
    self.area = self.radio ** 2 * math.pi
```



Métodos

```
c = Circulo(8,'d')
print ("Radio: " , c.radio)
print ("Diametro: " , c.diametro)
print ("Perímetro: " , c.perimetro)
print ("Área: " , c.area)
print ('-*40)

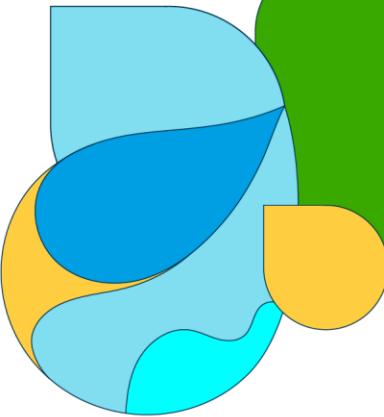
c.redimensionar(1); # Mismas dimensiones

print ("Radio: " , c.radio)
print ("Diametro: " , c.diametro)
print ("Perímetro: " , c.perimetro)
print ("Área: " , c.area)
print ('-*40)

c.redimensionar(2); # El doble del radio.

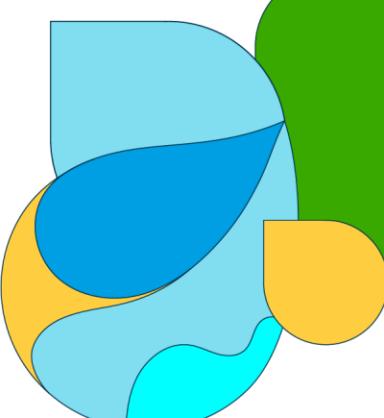
print ("Radio: " , c.radio)
print ("Diametro: " , c.diametro)
print ("Perímetro: " , c.perimetro)
print ("Área: " , c.area)
print ('-*40)
```

```
Radio: 4.0
Diametro: 8.0
Perímetro: 25.132741228718345
Área: 50.26548245743669
-----
Radio: 4.0
Diametro: 8.0
Perímetro: 25.132741228718345
Área: 50.26548245743669
-----
Radio: 8.0
Diametro: 16.0
Perímetro: 50.26548245743669
Área: 201.06192982974676
```



Encapsulamiento

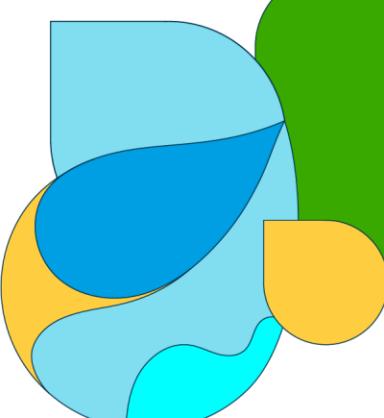
- Concepto fundamental de la programación orientada a objetos.
- Agrupar datos y métodos que funcionan con los datos dentro de una unidad en Python, se logra con una clase.
- El concepto también se usa a menudo para ocultar la representación interna o el estado de un objeto desde el exterior.
- Es común usar en otros lenguajes OOP las palabras reservadas para el acceso, `public`, `protected` y `private`.
- Los atributos “`private`” solo pueden modificarse dentro de la definición de la clase.
- El encapsulamiento favorece que el objeto siempre tenga un estado válido.



Encapsulamiento

```
class Circulo:  
    def __init__(self, val, prop='r'):  
        if prop == 'r':  
            self.set_radio(val)  
        elif prop == 'd':  
            self.set_diametro(val)  
        elif prop == 'p':  
            self.set_perimetro(val)  
        elif prop == 'a':  
            self.set_area(val)  
        else:  
            raise Exception('La propiedad solo puede ser; r, d, p o a')  
  
    def set_radio(self, r):  
        self._radio = r  
        self._diametro = r * 2  
        self._perimetro = r * 2 * math.pi  
        self._area = r ** 2 * math.pi
```

```
def get_radio(self):  
    return self._radio  
  
def set_diametro(self, d):  
    self.set_radio(d / 2)  
  
def get_diametro(self):  
    return self._diametro  
  
def set_perimetro(self, c):  
    self.set_radio(c / (2 * math.pi))  
  
def get_perimetro(self):  
    return self._perimetro  
  
def set_area(self, a):  
    self.set_radio((a / math.pi) ** .5)  
  
def get_area(self):  
    return self._area  
  
def redimensionar(self, amt):  
    r = self._radio * amt  
    self.set_radio(r)
```



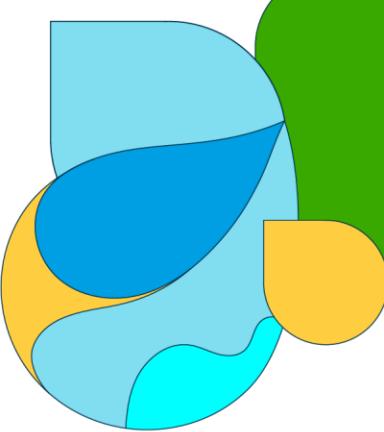
Encapsulamiento

```
from Circle3 import Circulo  
  
c= Circulo(10,'d')  
  
print ("El área de un círculo de radio: ", c.get_radio(), "es", c.get_area())  
c.set_radio(8)  
  
print ("El área de un círculo de radio: ", c.get_radio(), "es", c.get_area())
```

El área de un círculo de radio: 5.0 es 78.53981633974483
El área de un círculo de radio: 8 es 201.06192982974676

```
c._radio = 5  
print (c.get_area())
```

201.06192982974676

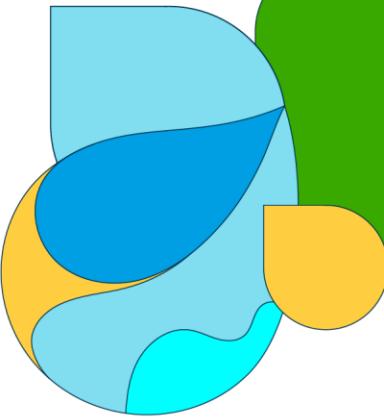


Propiedades

```
c.set_area(25)      # mutador  
a = get_area()      # accesor
```



```
c.area= 25          # modificar  
a= c.area           # obtener
```

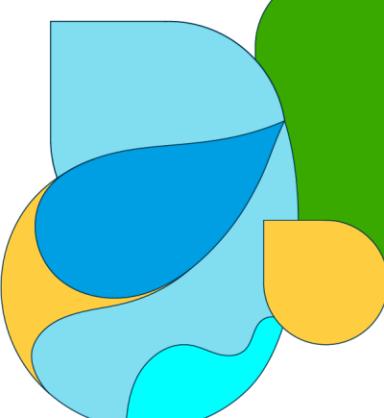


Propiedades | Función property()

```
class Circulo:
    def __init__(self):
        self._radio= None
    def get_radio():
        return self._radio

    def set_radio(self,r):
        self._radio = r

    radio = property(get_radio,set_radio) # Propiedad de la clase
```

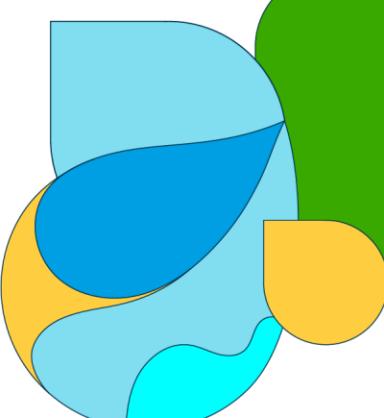


Propiedades

```
import math

class Circulo:
    def __init__(self, val, prop='r'):
        if prop == 'r':
            self.set_radio(val)
        elif prop == 'd':
            self.set_diametro(val)
        elif prop == 'c':
            self.set_perimetro(val)
        elif prop == 'a':
            self.set_area(val)
        else:
            raise Exception('La propiedad debe ser r, d, p, o a')

    def redimensionar(self, amount):
        r = self._radio * amount
        self.set_radio(r)
```



Propiedades

```
@property
def radio(self):
    return self._radio

@radius.setter
def radius(self, r):
    self._radio = r
    self._diametro = r * 2
    self.perimetro = r * 2 * math.pi
    self._area = r ** 2 * math.pi

@property
def diametro(self):
    return self._diametro

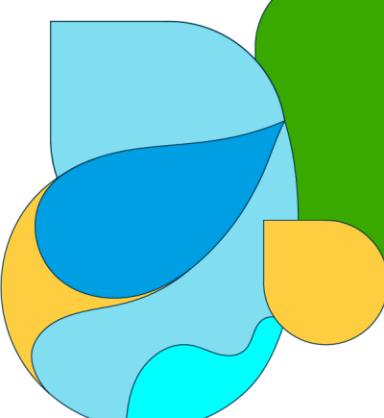
@diametro.setter
def diameter(self, d):
    self.radio = d / 2
```

```
@property
def perimetro(self):
    return self._perimetro

@perimetro.setter
def perimetro(self, c):
    self.radio = c / (2 * math.pi)

@property
def area(self):
    return self._area

@area.setter
def area(self, a):
    self.radio = (a / math.pi) ** .5
```



Propiedades

```
class Perro:  
  
    @property  
    def nombre(self):  
        return self.nombre  
  
a = Perro()  
print(a.nombre)
```

Ciclo
Infinito

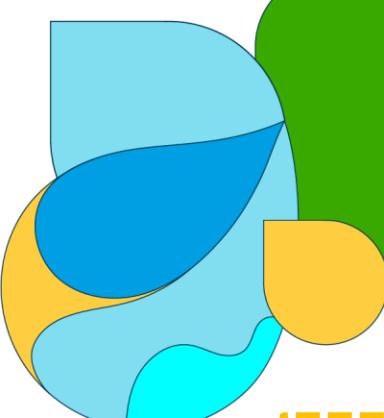
Herencia

```
class A:  
    pass
```

Superclase
Clase Base
Clase Padre

```
clase B(A):  
    pass
```

Subclase
Clase Derivada
Clase Hija

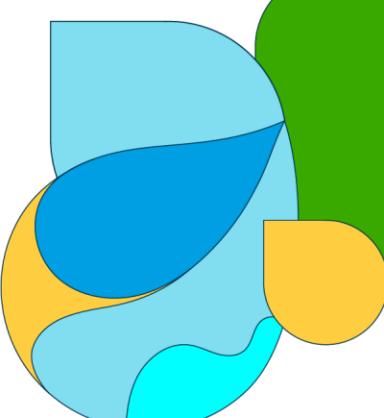


Herencia

```
class A:  
    def __init__(self, nombre):  
        self.nombre = nombre  
  
    def intro(self):  
        print('Hola, mi nombre es: {}'.format(self.nombre))  
  
    def outro(self):  
        print('Adios.')  
  
class B(A):  
    def intro(self):  
        print('Hi, mi nombre es: {}'.format(self.nombre))  
  
a = A('George')  
b = B('Ringo')
```

```
# intro()  
a.intro()  
b.intro()  
print()  
  
# Outro  
a.outro()  
b.outro()
```

```
Hola, mi nombre es: George.  
Hi, mi nombre es: Ringo.  
  
Adios.  
Adios.
```



Sobreescritura

```
class MyList(list):

    "Subclase de list, con nueva funcionalidad"

    def prepend(self, obj):
        """ Siempre agrega un elemento al inicio de la lista"""
        self.insert(0, obj)

lista = MyList()
lista.append('A'); lista.append('B'), lista.append('C')
print (lista)

lista.prepend('X'); lista.prepend('Y'); lista.prepend('Z');
print (lista)
```

```
['A', 'B', 'C']
['Z', 'Y', 'X', 'A', 'B', 'C']
```

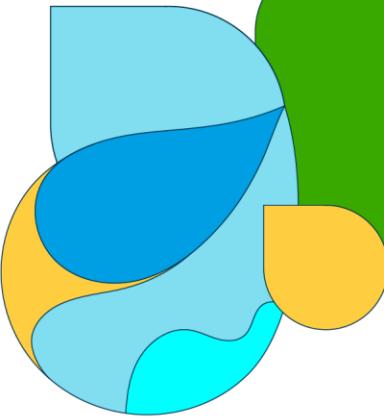
Sobreescritura | Extensión

```
class A:  
    def __init__(self, nombre):  
        self.nombre = nombre  
  
    def intro(self):  
        print('Hola, mi nombre es {}.'.format(self.nombre))  
  
    def outro(self):  
        print('Adios.')  
  
class B(A):  
    def intro(self):  
        super().intro()      # Llamar al heredado super()  
        print('Es un placer conocerte')  
  
a = A('George')  
b = B('Ringo')  
  
a.intro()  
print('-----')  
b.intro()  
print('-----')  
a.outro()  
b.outro()
```

Hola, mi nombre es George.

Hola, mi nombre es Ringo.
Es un placer conocerte

Adios.
Adios.

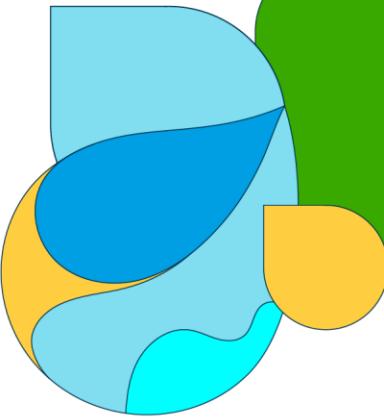


Métodos Estáticos

<clase>.<método_estático>()

```
class Triangulo:  
    def __init__(self, lados):  
        if not self.is_triangulo(lados):  
            raise Exception('No se puede crear un triángulo.')  
        self._lados = lados  
  
    @property  
    def perimetro(self):  
        return sum(self._lados)  
  
    @property  
    def area(self): # Fórmula de Herón  
        p = self.perimetro/2 # Semi-Perímetro  
        a = self._lados[0]  
        b = self._lados[1]  
        c = self._lados[2]  
        return ( p * (p-a) * (p-b) * (p-c) ) ** .5
```

```
@staticmethod  
def is_triangulo(lados):  
    if len(lados) != 3:  
        return False  
    lados.sort()  
    if lados[0] + lados[1] < lados[2]:  
        return False  
    return True
```



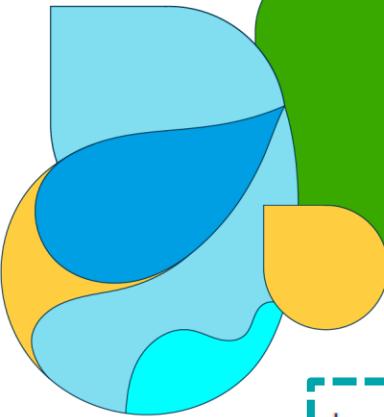
Métodos Estáticos

<clase>.<método_estático>()

```
bueno = [3, 3, 5]
malo = [3, 3, 9]
print ( Triangulo.is_triangulo(bueno) )      # Invocación estática
print ( Triangulo.is_triangulo(malo) )      # Invocación estática

t1 = Triangulo(bueno)
print ("Area:", t1.area)

t2 = Triangulo(malo)
```



Métodos Estáticos

```
bueno = [3, 3, 5]
malo = [3, 3, 9]
print ( Triangulo.is_triangulo(bueno) )    # Invocación estática
print ( Triangulo.is_triangulo(malo) )    # Invocación estática

t1 = Triangulo(bueno)
print ("Area:", t1.area)

t2 = Triangulo(malo)
```

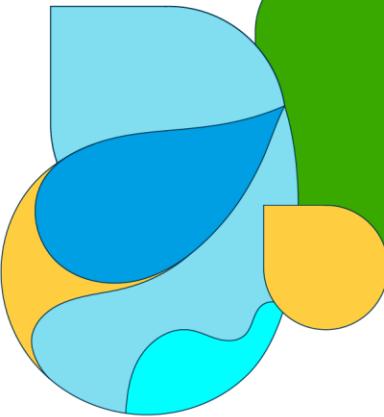
```
True
False
Area: 4.14578098794425
Traceback (most recent call last):

  File "<ipython-input-5-8b2f92b7e463>", line 1, in <module>
    runfile('C:/Material_Python_A/Codigos/05_00P/Demos/Triangle.py', wdir='C:/Material_Python_A/Codigos/05_00P/Demos')

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder_kernels\customize\spydercustomize.py", line 827, in runfile
    execfile(filename, namespace)

  File "C:\ProgramData\Anaconda3\lib\site-packages\spyder_kernels\customize\spydercustomize.py", line 110, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

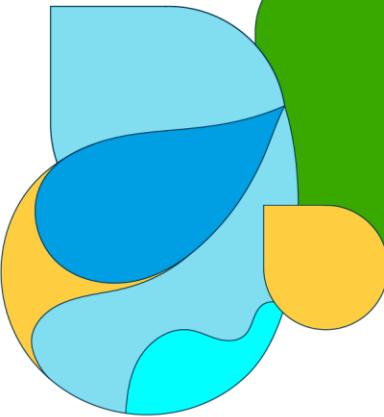
  File "C:/Material_Python_A/Codigos/05_00P/Demos/Triangle.py", line 39, in <module>
    t2 = Triangulo(malo)
```



Métodos & Atributos Clase

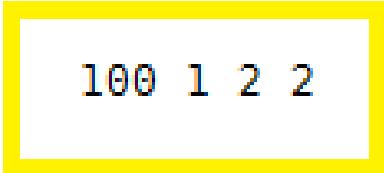
- Un atributo de *clase* es un atributo que se define afuera de todos los métodos de *clase*.

```
class A:  
    prop=1  
    def __init__(self):  
        pass
```

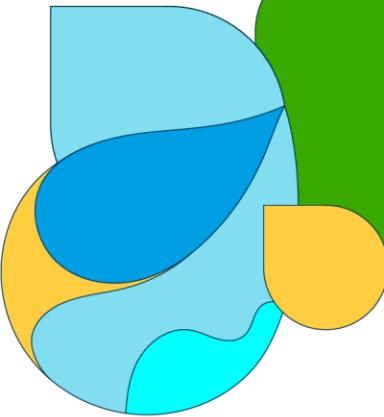


Métodos & Atributos Clase

```
class A:  
  
    prop=1  
    otra=2  
  
    def __init__(self):  
        self.prop = 100  
  
a = A()  
print (a.prop, A.prop, a.otra, A.otra)
```



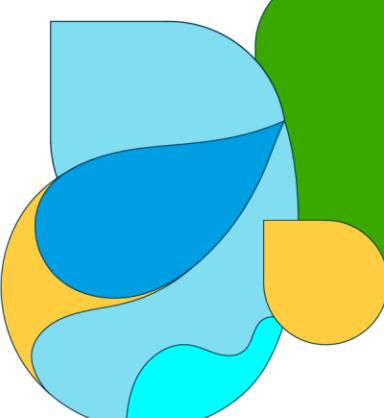
100 1 2 2



Métodos & Atributos Clase

```
class A:  
  
    lista = ['a','e','i' ]  
  
    def __init__(self):  
        self.lista.append('o')  
  
a = A()  
  
print (a.lista, A.lista, sep="\n")  
a.lista.append('u')  
  
print()  
print (a.lista, A.lista, sep="\n")
```

```
['a', 'e', 'i', 'o']  
['a', 'e', 'i', 'o']  
  
['a', 'e', 'i', 'o', 'u']  
['a', 'e', 'i', 'o', 'u']
```



Métodos Clase

```
class A:  
    prop= 1  
  
    @classmethod  
    def metodo_clase (cls, val=10): # cls es convención  
        cls.prop += val  
        return cls.prop  
  
x = A.metodo_clase(100)  
a = A()  
y= a.metodo_clase()  
print (x,y)
```

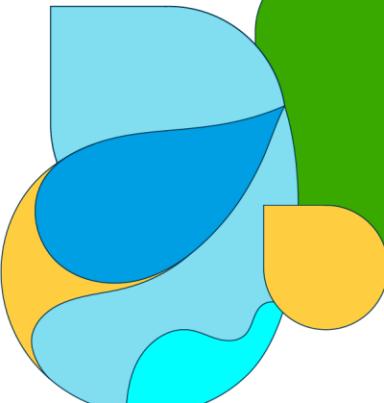
101 111

Métodos Clase

```
class Plane:  
  
    planes = []  
  
    def __init__(self):  
        self._in_air = False  
        type(self).planes.append(self)  
  
    def take_off(self):  
        self._in_air = True  
  
    def land(self):  
        self._in_air = False  
  
    @classmethod  
    def num_planes(cls):  
        return len(cls.planes)  
  
    @classmethod  
    def num_planes_in_air(cls):  
        return len([plane for plane in cls.planes if plane._in_air])
```

```
# Test Plane class:  
  
from Plane import Plane  
  
p1 = Plane()  
p2 = Plane()  
p3 = Plane() # tres instancias  
  
p1.take_off()  
p2.take_off()  
  
p1.land()  
  
print(Plane.num_planes(), Plane.num_planes_in_air())
```

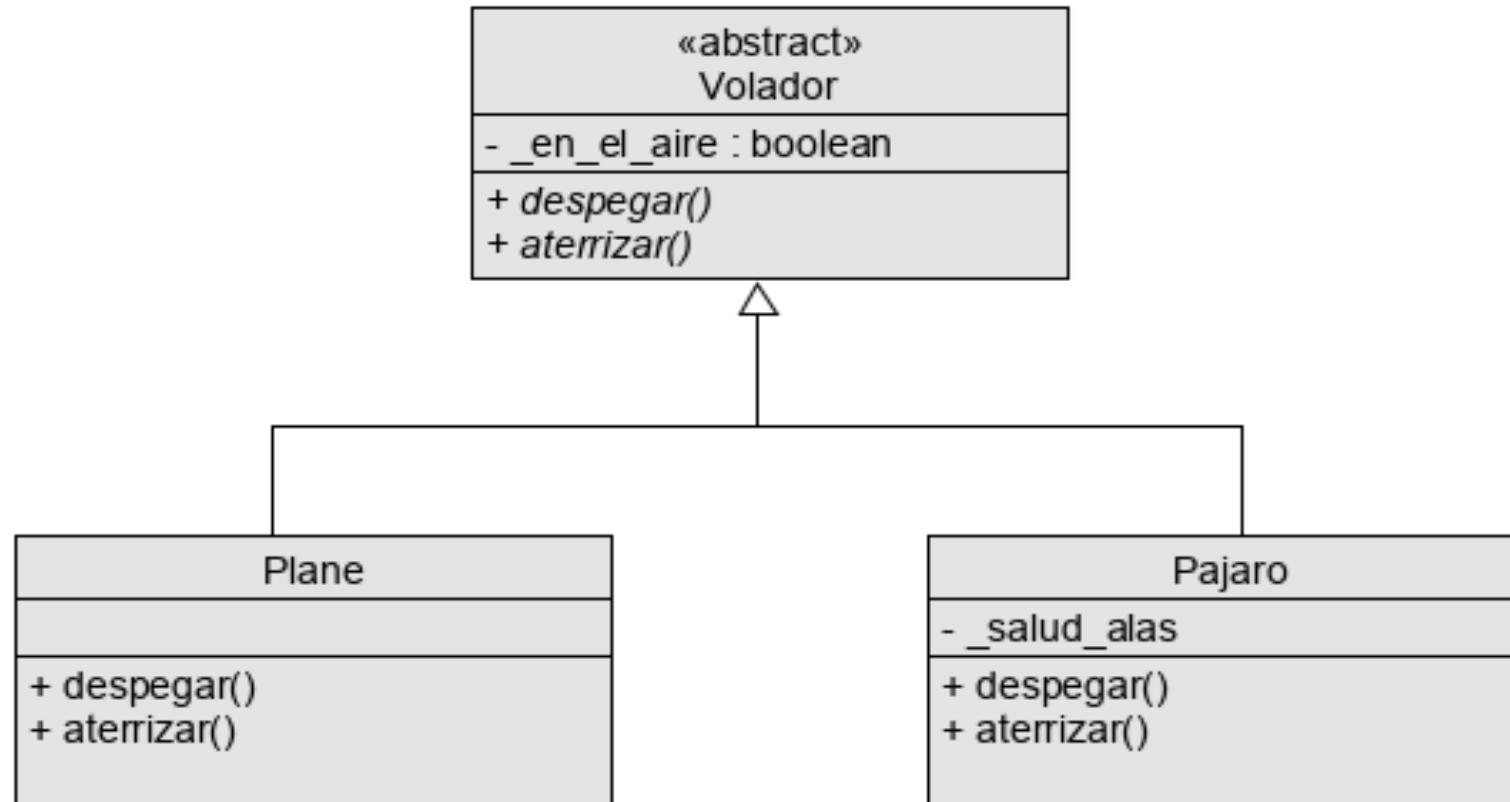
3 1



Clases Abstractas

- Una clase **abstracta** es una clase que no se puede instanciar, pero se crea con el propósito de subclásificar.
- Por ejemplo, imagina que estás creando un juego con varios objetos voladores, incluidos aviones y pájaros. Algunas cosas que se deben tener en cuenta son:
 - Los aviones sólo pueden despegar cuando el piloto está despierto.
 - Los aviones sólo pueden aterrizar cuando están sobre aeropuertos.
 - Las aves sólo pueden despegar cuando sus alas están sanas.
 - Las aves pueden aterrizar en cualquier lugar.

Clases Abstractas



Clases Abstractas

```
class Volador():

    objetos_voladores = []

    def __init__(self):
        self._en_el_aire = False
        type(self).objetos_voladores.append(self)

    @abc.abstractmethod
    def despegar(self):
        self._in_air = True

    @abc.abstractmethod
    def aterrizar(self):
        self._en_el_aire = False

    @classmethod
    def num_objectos(cls):
        return len(cls.objetos_voladores)

    @classmethod
    def num_objectos_en_aire(cls):
        return len([fo for fo in cls.objetos_voladores if fo._en_aire])
```

```
ufo = Volador()
print (Volador.num_objectos())
```

Clases Abstractas

```
import abc    # abc - Abstract Base Classes

class Volador(metaclass=abc.ABCMeta):      # metaclass=abc.ABCMeta

    objetos_voladores = []

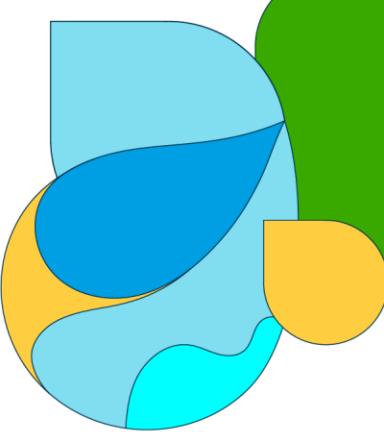
    def __init__(self):
        self._en_el_aire = False
        type(self).objetos_voladores.append(self)

    @abc.abstractmethod                      # Decorador abstracto
    def despegar(self):
        self._in_air = True

    @abc.abstractmethod
    def aterrizar(self):
        self._en_el_aire = False

    @classmethod
    def num_objectos(cls):
        return len(cls.objetos_voladores)

    @classmethod
    def num_objectos_en_aire(cls):
        return len([fo for fo in cls.objetos_voladores if fo._en_aire])
```

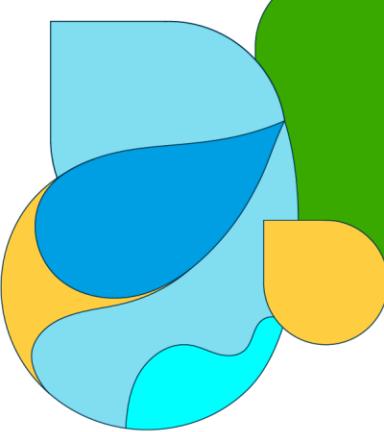


Clases Abstractas

```
ufo = Volador()  
print(Volador.num_objectos())
```

```
ufo = Volador()
```

```
TypeError: Can't instantiate abstract class Volador with abstract methods  
aterrizar, despegar
```



Funciones Preconstruidas

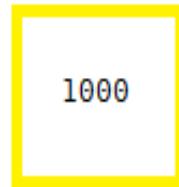
```
print (issubclass(Pajaro,Volador))  
print (issubclass(Pajaro,Pajaro))  
print (issubclass(Pajaro,Plane))  
print (issubclass(Pajaro,str))
```

True
True
False
False

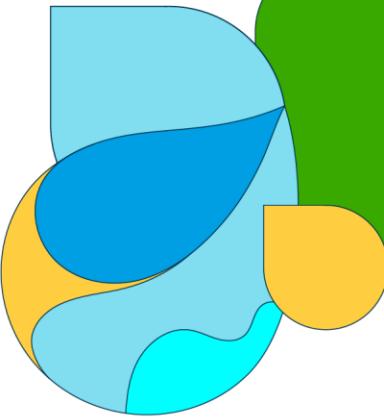
Herencia Múltiple

```
class A:  
    prop=1  
  
    def __init__(self, val):  
        self._prop= val  
  
    def met(self):  
        return 1000  
  
class B:  
    prop= 1  
  
    def __init__(self, val):  
        self._prop= val  
  
    def met(self):  
        return 2000
```

```
class C(A,B):   
    prop= 1  
  
    def __init__(self, val):  
        self._prop= val  
  
c = C(5)  
print (c.met())
```



1000



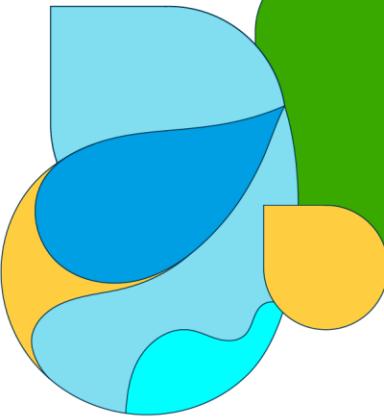
Herencia Múltiple

```
class A:  
    prop=1  
  
    def __init__(self, val):  
        self._prop= val  
  
    def met(self):  
        return 1000  
  
class B:  
    prop= 1  
  
    def __init__(self, val):  
        self._prop= val  
  
    def met(self):  
        return 2000
```

```
class C(A,B):  
    prop= 1  
  
    def __init__(self, val):  
        self._prop= val
```

```
c = C(5)  
print (c.met())
```

1000



Métodos Dunder

- Los métodos **Dunder**, también llamados métodos mágicos.
- Los métodos Dunder inician con dos guiones bajos de prefijo y posfijo en el nombre del método.
- Dunder: Double Under.
 - En español doble subrayado.
- Se usan por lo general en la sobrecarga de operadores.
- `__init__`, `__add__`, `__len__`, `__str__`, etc.

Métodos Dunder | Ejemplo

```
class Mascota:  
    def __init__(self, nombre="solo vino"):  
        self._nombre= nombre  
  
    def __str__():  
        return "El nombre de la mascota es: " + self._nombre  
  
    def __add__(self, otro):  
        return self._nombre + " mascota preferida"  
  
    def __eq__(self, otro):  
        return self._nombre == otro._nombre  
  
  
m = Mascota ('Gasper')  
print (m)  
print (m + " perrijo")  
  
n = Mascota ('Gasper')  
  
print ( m == n )  
  
  
print("\n----- Métodos Mágicos -----")  
i= 0  
for x in dir(Mascota):  
    i+=1  
    print('{:10s} {}'.format(x, "\n" if i%2==0 else "\t"), end="")
```

El nombre de la mascota es: Gasper
Gasper mascota preferida
True

----- Métodos Mágicos -----

__add__	__class__
__delattr__	__dict__
__dir__	__doc__
__eq__	__format__
__ge__	__getattribute__
__gt__	__hash__
__init__	__init_subclass__
__le__	__lt__
__module__	__ne__
__new__	__reduce__
__reduce_ex__	__repr__
__setattr__	__sizeof__
__str__	__subclasshook__
__weakref__	

Resumen del capítulo

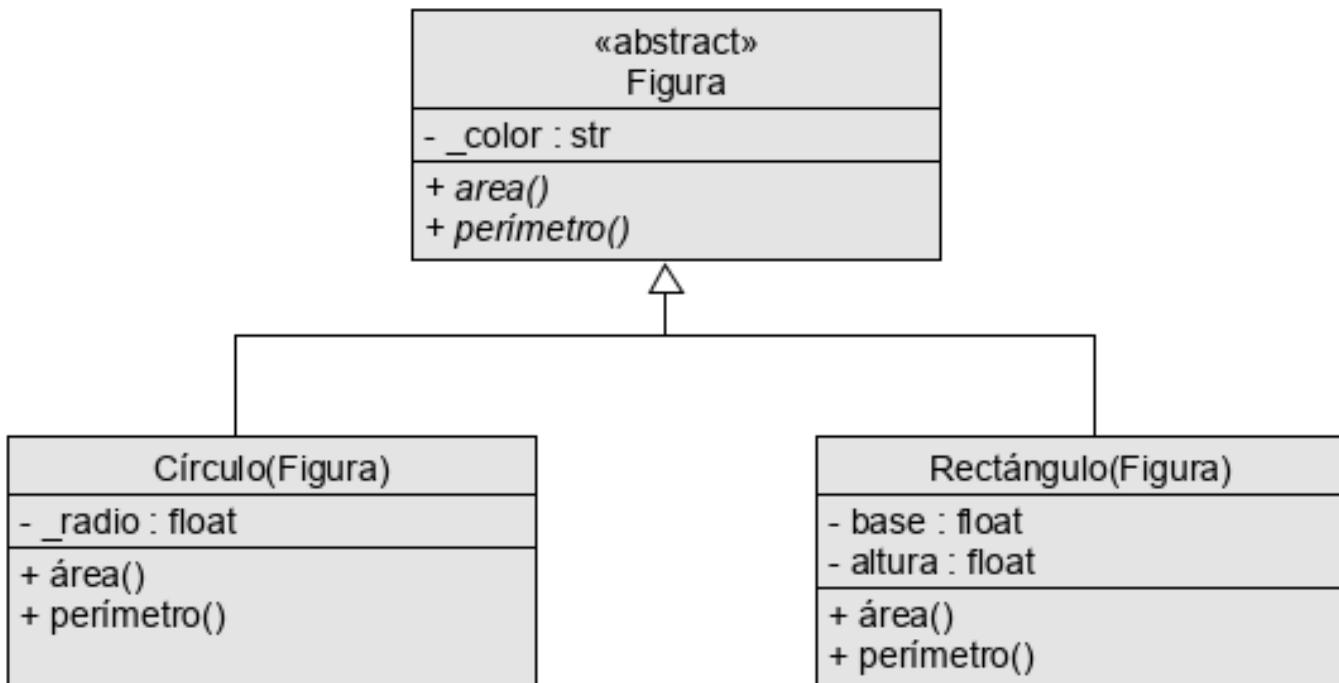
- La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de la programación, esto hace referencia al conjunto de teorías, estándares, modelos y métodos que permiten organizar el conocimiento, proporcionando un medio bien definido para visualizar el dominio del problema e implementar en un lenguaje de programación la solución a ese problema.
- En Python la POO se expresa de manera simple y fácil de escribir, pero se debe tener en cuenta que para programar se requiere entender cómo funciona la teoría de la POO y llevarla a código.

Resumen del capítulo

- La teoría de la POO dice que todos los objetos deben pertenecer a una clase, ya que esta es la base para diferenciarse unos de otros, teniendo atributo y comportamiento que los distingan de otros objetos que pertenezcan a otras clases.
- Python maneja a su manera muy peculiar los cuatro pilares de la programación orientada a objetos: la abstracción, el encapsulamiento, la herencia y el polimorfismo.

Práctica: Áreas & Perímetros

Dado el siguiente diagrama de clases implementar el diagrama en Python.



Práctica: Áreas & Perímetros

- Crea un archivo de nombre `p7_1.py`
- Se puede en ese archivo crear un módulo con las clases Figura, Círculo y Rectángulo.
- Crea un archivo de nombre `Test.py`.
- Importa las clases del módulo del punto dos.
- Captura el siguiente código de prueba:

```
from p7_1 import Figura, Círculo, Rectángulo

figuras= [ Círculo(10), Círculo(1), Rectángulo(1,1), Rectángulo(2,2)]

for f in figuras:
    print (f, f.área(), f.perímetro())
```



Práctica: Dados

Actualmente se tiene una clase Dado para crear listas de objetos con algún número de lados, pero no se tiene ninguna manera de tirar el dado.

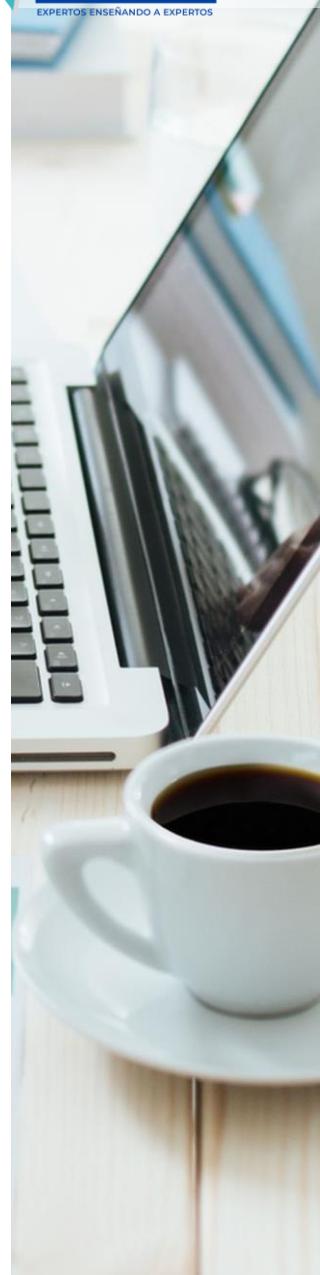
En esta práctica, se agregará el método **roll()** a la clase Dado.

1. Crea el archivo **p7_2.py** con el siguiente código:

```
import random

class Dado:
    def __init__(self, caras=6):
        self.caras = caras

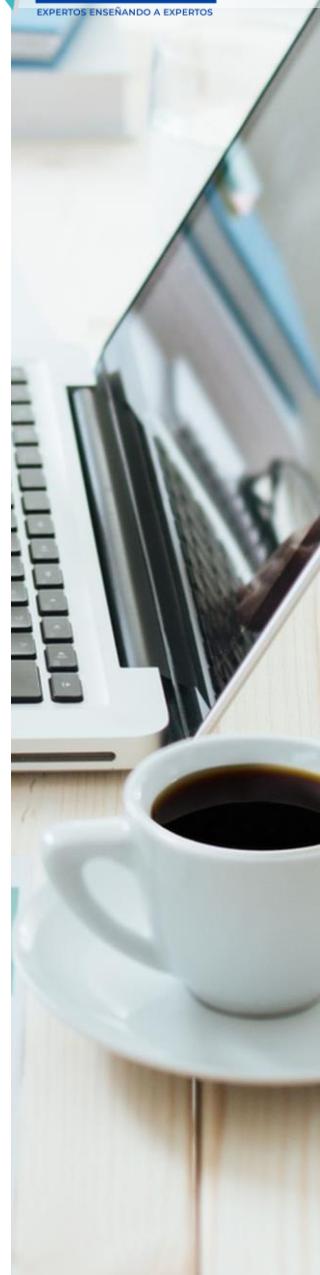
    # Agrega el método roll
```



Práctica: Dados

- Agrega el método **roll()** que devuelva un número entero entre 1 y lados. Usa el módulo random.
- Prueba la solución creando una instancia de Dado e invoca el método roll() varias veces.
- Opcionalmente, escribe código para tirar el dado 100,000 veces y luego usa un objeto Counter para crear una lista que muestre cuantas veces se lanzó cada lado. Debería de generar algo como lo siguiente:

```
[(1, 16422), (2, 16596), (3, 16567), (4, 16761), (5, 16951), (6, 16703)]
```



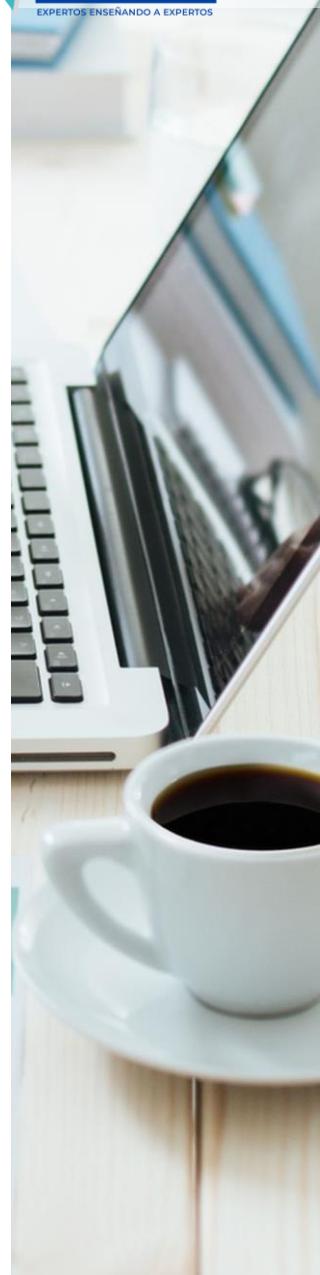
Práctica: Propiedades

En esta práctica, se convertirán los métodos `get_` en propiedades.

1. Crea el archivo `p7_3.py` con el siguiente código:

```
import statistics as stats

class Simulation:
    def __init__(self, fnct_to_run, iterations):
        self._fnct_to_run = fnct_to_run
        self._iterations = iterations
        self._results = []
        self.run()
```



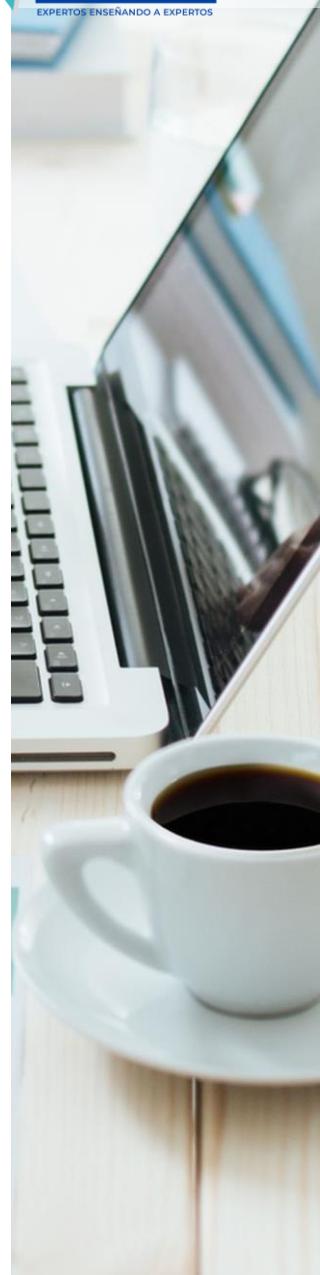
Práctica: Propiedades

```
def run(self):
    for i in range(self._iterations):
        result = self._fnct_to_run()
        self._results.append(result)

def get_mean(self):
    return stats.mean(self._results)

def get_median(self):
    return stats.median(self._results)

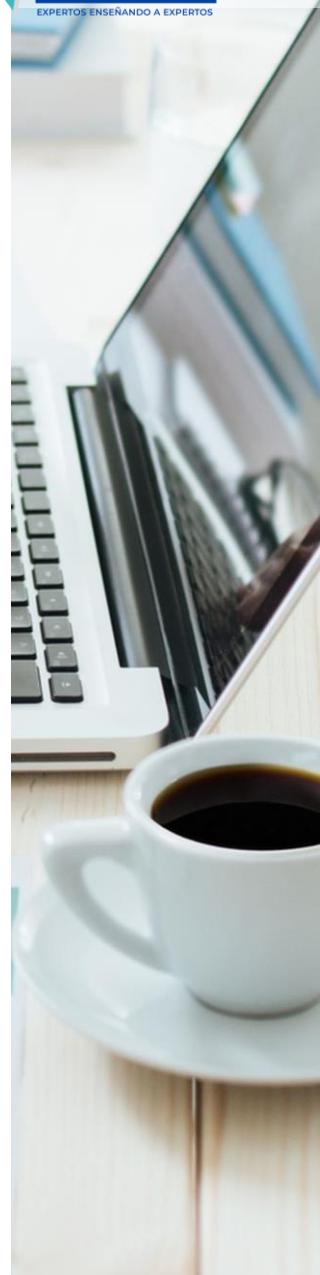
def get_mode(self):
    try:
        return stats.mode(self._results)
    except:
        return None
```



Práctica: Propiedades

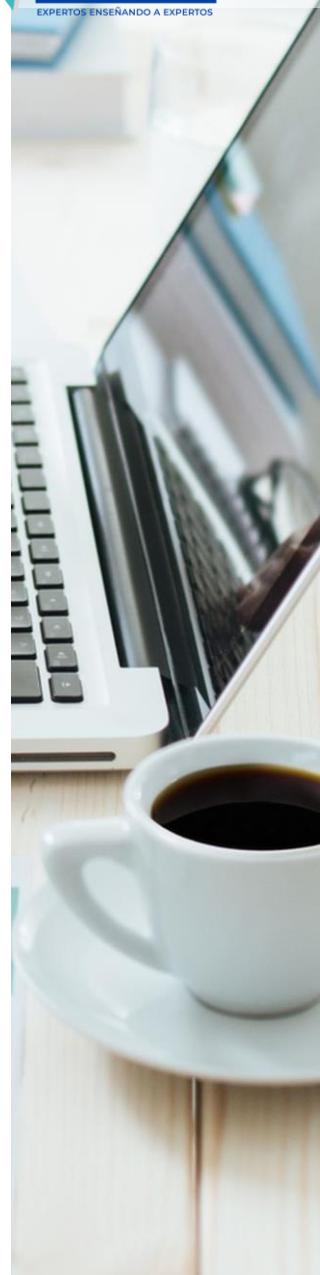
2. Observa los métodos **get_mean()**, **get_median()** y **get_mode()**, que aprovechan el módulo **statistics** para calcular los resultados del promedio, media y moda de un dado lanzado varias veces.
3. Ejecuta el código para ver cómo funciona. Aquí hay una secuencia de instrucciones que puedes ejecutar, usando la clase Dado con el método roll() asignado en la práctica del punto 7.2.

```
from p7_2 import Dado
from p7_3 import Simulation
d= Dado()
sim=Simulation(d.roll,1000)
print (sim.get_mean(), sim.get_median(), sim.get_mode())
```



Práctica: Propiedades

```
Administrator: Command Prompt - python -q
D:\MaterialPythonDeveloper\códigos\07_OOP\soluciones>
D:\MaterialPythonDeveloper\códigos\07_OOP\soluciones>python -q
>>>
>>> from p7_2 import Dado
>>> from p7_3 import Simulation
>>>
>>> d=Dado()
>>> sim= Simulation(d.roll,1000)
>>>
>>> print (sim.get_mean(), sim.get_median(), sim.get_mode())
3.583 4.0 5
>>> -
```



Práctica: Propiedades

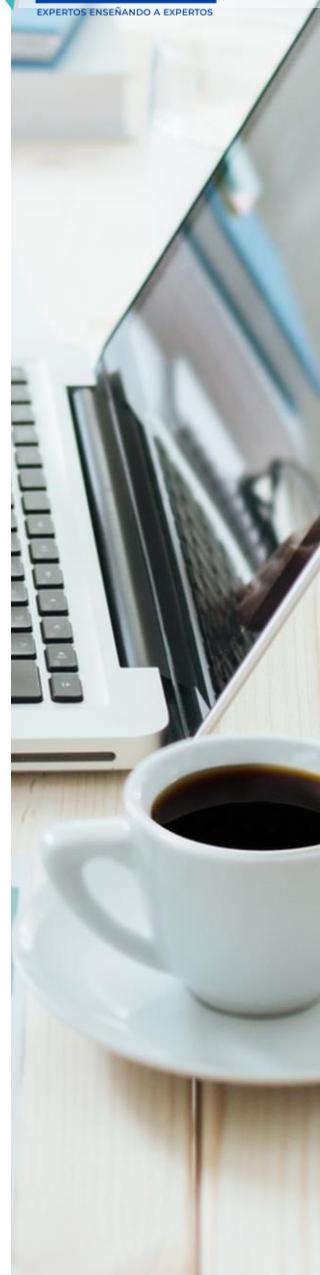
4. Convierte los tres métodos `get_` en propiedades y prueba su solución con el mismo código que el anterior, pero reemplazando la declaración de impresión con referencias de propiedades como se muestra a continuación:

```
from p7_2 import Dado
from p7_3 import Simulation
d= Dado()
sim=Simulation(d.roll,1000)
print (sim.mean, sim.median, sim.mode)
```



Práctica: Propiedades

```
Administrator: Command Prompt - python -q
D:\MaterialPythonDeveloper\códigos\07_OOP\soluciones>
D:\MaterialPythonDeveloper\códigos\07_OOP\soluciones>python -q
>>>
>>> from p7_2 import Dado
>>> from p7_3 import Simulation
>>>
>>> d=Dado()
>>> sim= Simulation(d.roll,1000)
>>> print (sim.mean, sim.median, sim.mode)
3.545 4.0 6
>>> -
```



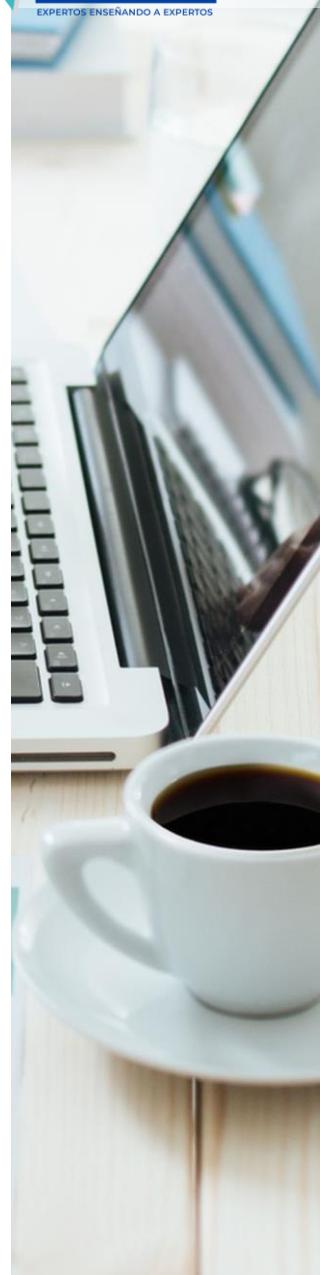
Práctica: Dados Alterados

1. Crea el archivo `p7_4.py` con el siguiente código.

```
import random
from collections import Counter
from p7_2 import Dado

class DadoAlterado(Dado):

    def __init__(self, weights, sides=6):
        if len(weights) != sides:
            raise Exception('Los pesos deben ser del mismo tamaño que el dado de {} caras.'.format(sides))
        super().__init__(sides)
        self._weights = weights
```



Práctica: Dados Alterados

```
def roll(self):
    """ Devuelve un valor entre 1 y el número de lados del dado."""
    # Su código debe ir aquí
```

2. Analiza el código de la clase Dado.
3. Revisa el método `__init__()` de la clase DadoAlterado. Observa que crea un atributo pseudo-privado `_weights` que contiene una lista de pesos, cada uno correspondiente a un lado del dado. Un dado de seis lados con los siguientes pesos:

[1, 1, 1, 1, 1, 5]



Práctica: Dados Alterados

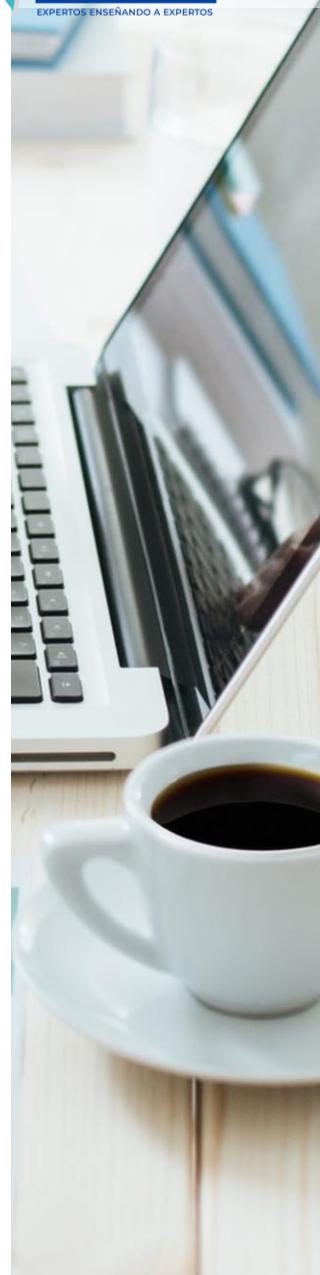
4. Completa el método `roll()` en la clase de `DadoAlterado`. Una vez más, las probabilidades de devolver un valor deben correlacionarse con el peso del en `self._weights`. Una forma de hacerlo es crear una nueva lista que contenga cada lanzamiento, donde `n` es el peso asociado. Por ejemplo, `self._weights` anterior, la nueva lista se vería así:

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

5. Prueba su solución, el resultado debe de ser algo similar a:

[(1, 9899), (2, 10012), (3, 10083), (4, 10133), (5, 10011), (6, 49869)]

6. Observa que la instancia de `DadoAlterado` tiene una propiedad `rolls`, que heredó de la clase `Dado`.



Práctica: Dado Ponderado

1. Crea el archivo `p7_5.py` con el siguiente contenido:

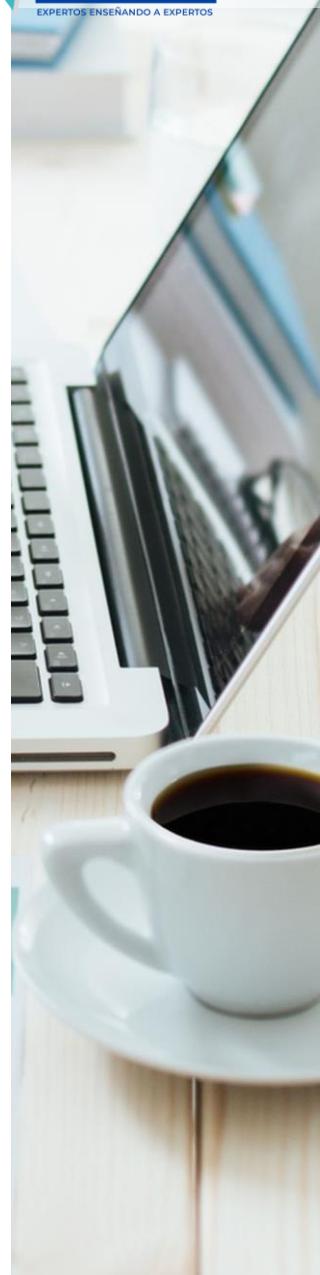
```
import random
from collections import Counter

from p7_4 import DadoAlterado

class DadoPonderado(DadoAlterado):

    def __init__(self, sides=6):
        self._weights = [1] * sides
        super().__init__(self._weights, sides)

    def roll(self):
        # Su código
```

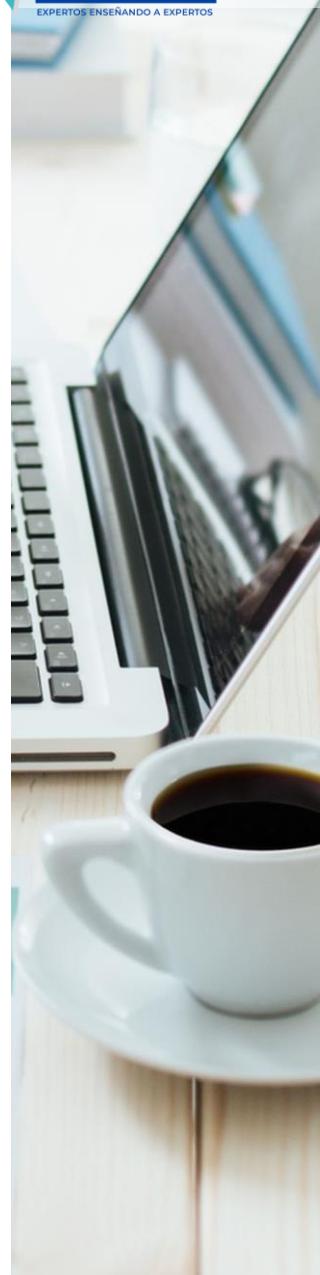


Práctica: Dado Ponderado

2. Revisa el método `__init__()` de la clase: Ten en cuenta que no toma un parámetro de pesos como lo hace la superclase. Más bien, establece todos los valores en `_weights` a 1 usando:

```
self._weights = [1] * sides
```

3. Completa el método `roll()` para que realice lo siguiente:
- Invoque al método `roll()` de la superclase y almacene el resultado en una variable local.
 - Modifique `self._weights` para que el peso del lanzamiento que acaba de rodar se incremente en 1.
 - Devuelva el lanzamiento.



Práctica: Dado Ponderado

4. Prueba la solución. El resultado debería ser similar a lo siguiente:

$[(1, 40), (2, 117), (3, 129), (4, 578), (5, 108), (6, 28)]$



Referencias Bibliográficas

- Magic Methods: <https://medium.com/python-features/magic-methods-demystified-3c9e93144bf7>
- Data Model, Objects, values and types:
<https://docs.python.org/3/reference/datamodel.html>
- Classes: <https://docs.python.org/3/tutorial/classes.html>
- Object-oriented programming: <https://python-textbook.readthedocs.io/en/1.0/Object Oriented Programming.html>
- Implementing an Interface in Python:
<https://realpython.com/python-interface/>



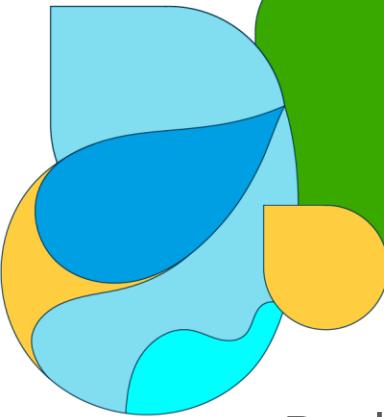
Unidad temática 10

Archivos, directorios y closures

10.1 Archivos y directorios

Objetivos:

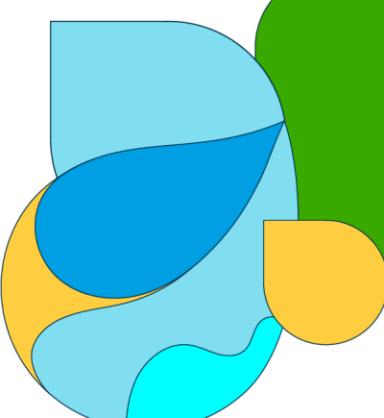
- Leer archivos en el sistema de archivos.
- Crear y escribir en archivos en el sistema de archivos.
- Acceder y trabajar con directorios en el sistema de archivos.
- Conocer los módulos os y os.Path.



Introducción

Python puede entre otras cosas:

- Abrir archivos existentes, crear nuevos archivos.
- Leer el contenido del archivo.
- Añadir al contenido del archivo.
- Sobreescribir el contenido del directorio.
- Listar el contenido del directorio.
- Renombrar archivos y directorios.



Introducción



Función open()

- La función preconstruida open() intentará abrir un archivo en una ruta determinada y devolver un objeto de tipo archivo.
- Sintaxis:
 - **open(ruta, modo)**
- La ruta puede ser relativa o absoluta.
- Los modos de apertura pueden ser:
 - **r, r+, w, w+, x, a, b, t**

Función open()

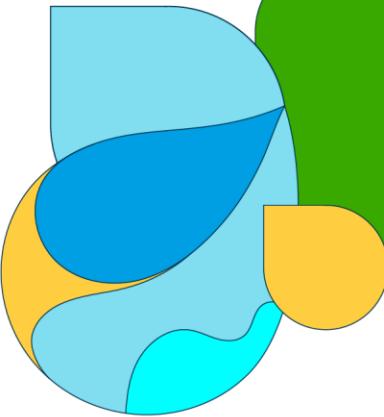
Modo	Descripción
r	Abre para lectura. FileNotFoundException si no existe.
w	Abre para escritura. Si el archivo existe, elimina el contenido, en caso contrario lo crea.
x	Crea y abre para escritura. FileExistsError si el archivo ya existe.
a	Abre para agregar al final del contenido o crea uno nuevo.
r+	Abre para lectura & escritura.
w+	Abre para escritura y lectura. Si el archivo existe borra el contenido.
b	Abre en modo binario.
t	Abre en modo texto.

Función open() | Ejemplo

```
open('poema.txt')
open('poema.txt', 'w')
open('poema.txt', 'r+')
open('logo_netec.jpg', 'r+b')
```

```
read(tam)
readline()
readlines()
list(f)
```

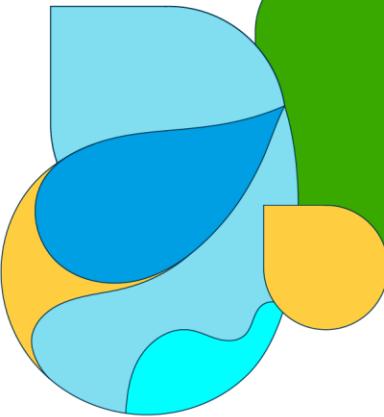
```
archivo = open("poema.txt", "r")
for line in archivo:
    print(line)
```



Función open() | Ejemplo

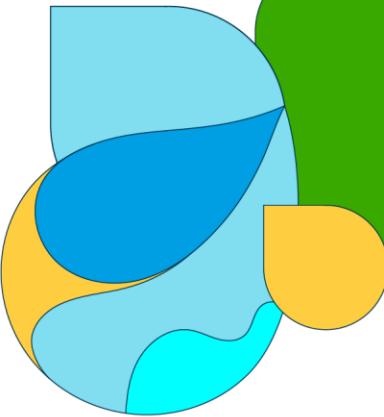
```
archivo = open('mis_archivos/demo.txt')
for linea in archivo:
    print(linea, end='')
archivo.close()
```

```
C:\Users\Administrator\ws_python\demos>python looplines2.py
Linea 1
Linea 2
Linea 3
Linea 4
Linea 5
```



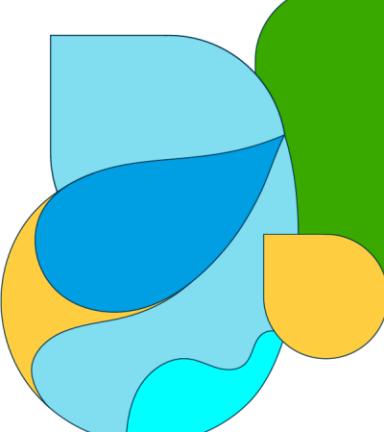
Sentencia with/as

```
with open('mis_archivos/demo.txt') as archivo: # Abrir el recurso
    for linea in archivo: # Procesar el recurso
        print(linea,end='')
    # Cierre automático del recurso
```



Escritura de Archivos

```
with open('my_files/salida.txt','w') as f:  
    f.write('Python es un lenguaje elegante y poderoso')
```



Módulo os

chdir

getcwd

listdir

mkdir

makedirs

remove

removedir

removedirs

unlink

rename

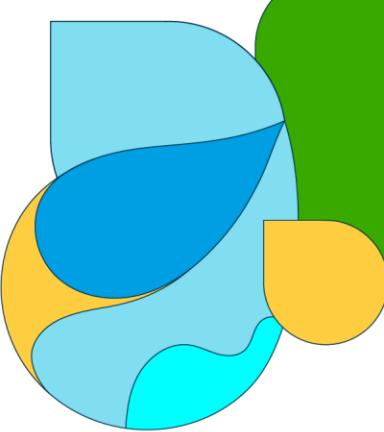
renames

walk

Función listdir()

```
import os

dir_cont = os.listdir(r"C:\Users\Administrator\ws_python\w")
for archivo in dir_cont:
    print(archivo)
```



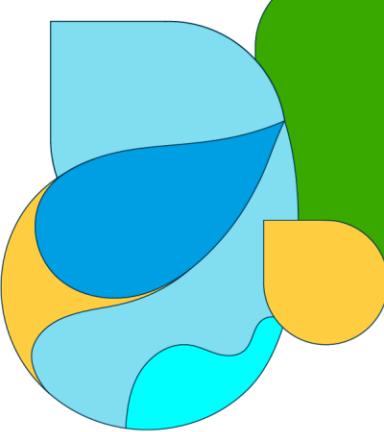
Función renames()

```
import os

postres = 'mis_archivos/postres.txt'
comida = 'nuevos/comidas.txt'

def postres2comida():
    os.renames(postres, comida)
    print('Renombrado:', postres, 'to', comida)

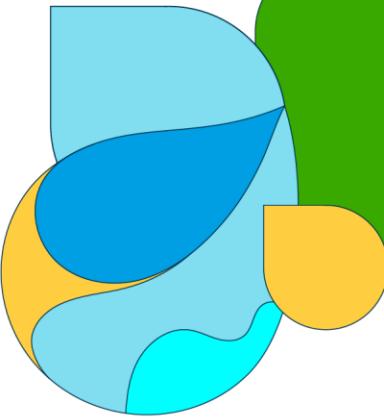
def comida2postres():
    os.renames(comida, postres)
    print('Renombrado:', comida, 'to', postres)
```



Función walk()

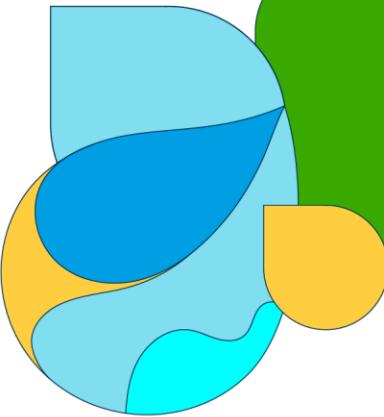
- La estructura/firma de la función walk() es:

```
walk(top, topdown=True, onerror=None, followlinks=False)
```



Función walk | Ejemplo

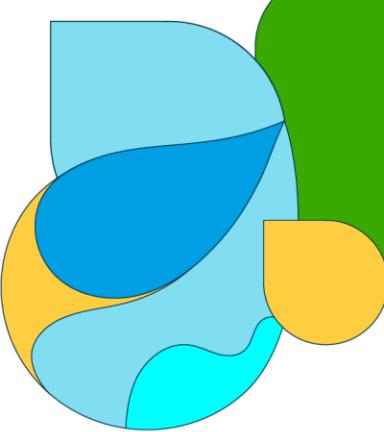
```
def spaces2dashes():
    for dirpath, dirnames, filenames in os.walk('..\mis_demos'):
        for fname in filenames:
            if ' ' in fname:
                oldname = dirpath+'/'+fname
                newname = dirpath+'/'+fname.replace(' ', '-')
                print("Replacing",oldname,"with",newname)
                os.rename(oldname,newname)
```



Función walk | Ejemplo

```
def dashes2spaces():
    for dirpath, dirnames, filenames in os.walk('..\mis_demos'):
        for fname in filenames:
            if '-' in fname:
                oldname = dirpath+'/'+fname
                newname = dirpath+'/'+fname.replace('-', ' ')
                print("Replacing",oldname,"with",newname)
                os.rename(oldname,newname)

#spaces2dashes()
dashes2spaces()
```



Módulo os.path

abspath

basename

dirname

exists

getatime

getmtime

getctime

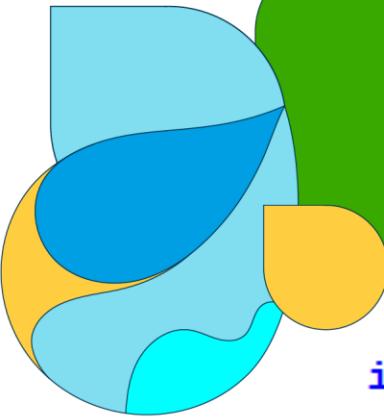
getsize

.isfile

.isdir

join

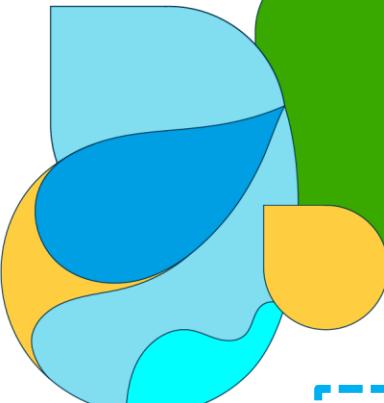
...



Módulo os.path

```
import os.path

print('1.', os.path.abspath('.'))
print('2.', os.path.basename('mis_archivos/netec.jpg'))
print('3.', os.path.dirname('mis_archivos/netec.jpg'))
print('4.', os.path.exists('mis_archivos/netec.jpg'))
print('5.', os.path.getatime('mis_archivos/netec.jpg'))
print('6.', os.path.getmtime('mis_archivos/netec.jpg'))
print('7.', os.path.getctime('mis_archivos/netec.jpg'))
print('8.', os.path.getsize('mis_archivos/netec.jpg'))
print('9.', os.path.isabs('mis_archivos/netec.jpg'))
print('10.', os.path.isfile('mis_archivos/netec.jpg'))
print('11.', os.path.isdir('mis_archivos/netec.jpg'))
print('12.', os.path.join('mis_archivos', 'netec.jpg'))
print('13.', os.path.relpath('mis_archivos/netec.jpg'))
print('14.', os.path.split(os.path.abspath('mis_archivos/netec.jpg')))
print('15.', os.path.splitext('mis_archivos/netec.jpg'))
```



Módulo os.path

- ```
1. C:\Users\Administrator\ws_python\demos
2. netec.jpg
3. mis_archivos
4. True
5. 1654190801.3778338
6. 1654190801.3778338
7. 1654190801.3778338
8. 3890
9. False
10. True
11. False
12. mis_archivos\netec.jpg
13. mis_archivos\netec.jpg
14. ('C:\\\\Users\\\\Administrator\\\\ws_python\\\\demos\\\\mis_archivos', 'netec.jpg')
15. ('mis_archivos/netec', '.jpg')
```

# Resumen del capítulo

En este capítulo, se ha visto como trabajar con archivos y directorios en el sistema operativo.

Python no depende de la noción subyacente del sistema operativo de archivos de texto, Python realiza todo el procesamiento y, por lo tanto, es independiente de la plataforma.

- Python permite leer, escribir y eliminar archivos.
- Usa la función **open("nombre de archivo", "w+")** para crear un archivo.
- En la función open, el carácter + en el modo le dice a Python que abra el archivo con permisos de lectura y escritura.

# Resumen del capítulo

- Para agregar datos a un archivo existente, usa la instrucción: `open("nombre de archivo", "a")`.
- Usa la función de `read()` para leer todo el contenido de un archivo.
- Usa la función `readlines()` para leer el contenido del archivo línea por línea.
- `os.getcwd()` - Obtiene la ruta actual del directorio de trabajo como una cadena - `pwd`
- `os.listdir()` - Obtiene el contenido del directorio actual como una lista de cadenas - `ls`
- `os.walk("dir")` - Devuelve un generador con información de nombre y ruta para directorios y archivos en el directorio actual y todos los subdirectorios - `ls -1R`

# Resumen del capítulo

- **os.chdir("ruta")** - Cambia el directorio de trabajo actual - **cd**
- **os.path.join()** - Crea una ruta para su uso posterior - **no hay equivalente.**
- **os.makedirs("dir1/dir2")** - Crea el directorio - **mkdir -p**
- **os.remove("archivo")** - Elimina un archivo - **rm**

# Práctica: Encontrar Texto

En esta práctica, se va a crear un programa para buscar texto en archivo.

## 1. Procesamiento de archivos p8\_1.py

```
Práctica 8.1 Encontrar texto en un archivo
def search(word, text):
 """Regresa una tupla con el número de línea y la línea de texto(o nada)"""
 pass

def main():

 # Abra el archivo mis_archivos/datos.txt
 # Cree una lista con el contenido del archivo.
 # Guarde la lista con el nombre de azop.
```



# Práctica: Encontrar Texto

```
def main():

 # Abra el archivo mis_archivos/datos.txt
 # Cree una lista con el contenido del archivo.
 # Guarde la lista con el nombre de azop.

 palabra = input('Ingrese una palabra: ')

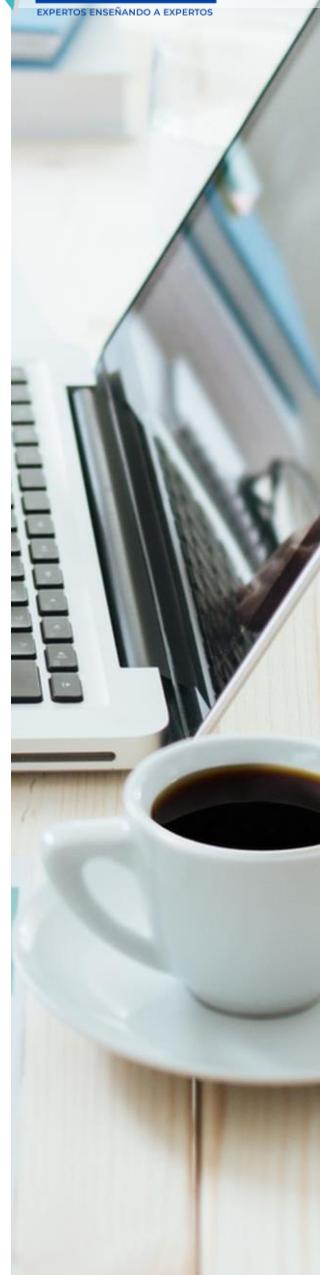
 result = search(palabra, azop)
 if(result):
 print(palabra, 'aparece en', result[0], ':', result[1])
 else:
 print('La palabra {} no fue encontrada'.format(palabra))

main()
```



# Práctica: Encontrar Texto

- En la función `main()`, abre el archivo `mis_archivos\datos.txt`
- Crea una lista a partir del contenido. Guarda la lista como azop.
- El resto de la función `main()` ya está escrito.
- Reemplaza en la función `search()` el código que devuelve una tupla de dos elementos que contiene el número de línea y el texto de la línea. Puedes encontrar útil la función `enumerate()`.
- La ejecución del programa debería ser similar a la siguiente captura de ventana:



# Práctica: Encontrar Texto

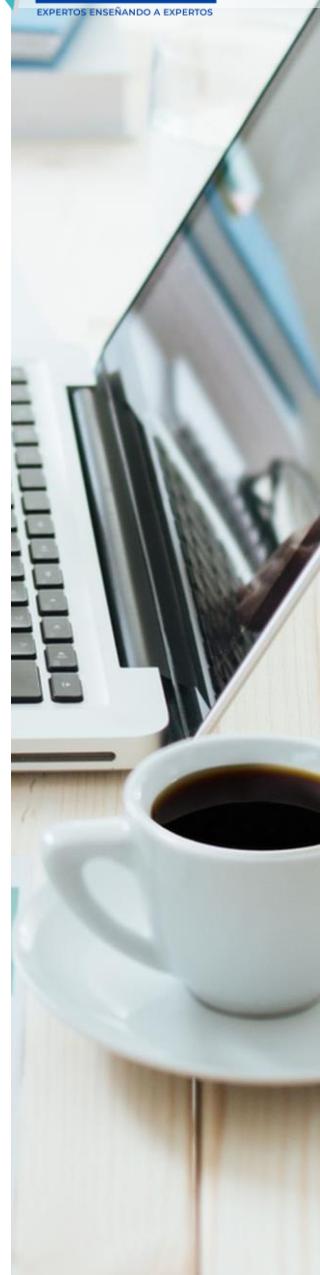
```
Administrator: Command Prompt
D:\MaterialPythonDeveloper\nuevos_codigos\08_archivos\soluciones>python p8_1.py
Ingrese una palabra: python
La palabra 'python' aparece en la posición 3:'https://www.python.org/dev/peps/pep-0020/'

D:\MaterialPythonDeveloper\nuevos_codigos\08_archivos\soluciones>python p8_1.py
Ingrese una palabra: of
La palabra 'of' aparece en la posición 1:'The Zen of Python '

D:\MaterialPythonDeveloper\nuevos_codigos\08_archivos\soluciones>python p8_1.py
Ingrese una palabra: the
La palabra 'the' aparece en la posición 12:'Special cases aren't special enough to break the rules.'

D:\MaterialPythonDeveloper\nuevos_codigos\08_archivos\soluciones>python p8_1.py
Ingrese una palabra: xxx
La palabra 'xxx' no fue encontrada

D:\MaterialPythonDeveloper\nuevos_codigos\08_archivos\soluciones>
```



# Práctica: Escribir en un archivo

1. Captura el código en un archivo de nombre `p8_2.py`
2. Antes de ejecutar este archivo, busca el `mis_archivos\archivo zen.txt`, si existe elimínalo.
3. Ahora ejecuta el programa `p8_2.py`
4. Ahora mira nuevamente en la carpeta `mis_archivos\zen.py`, deberá contener un `zen.txt`
5. Abre el archivo con un editor de textos o muestra el contenido en la terminal.

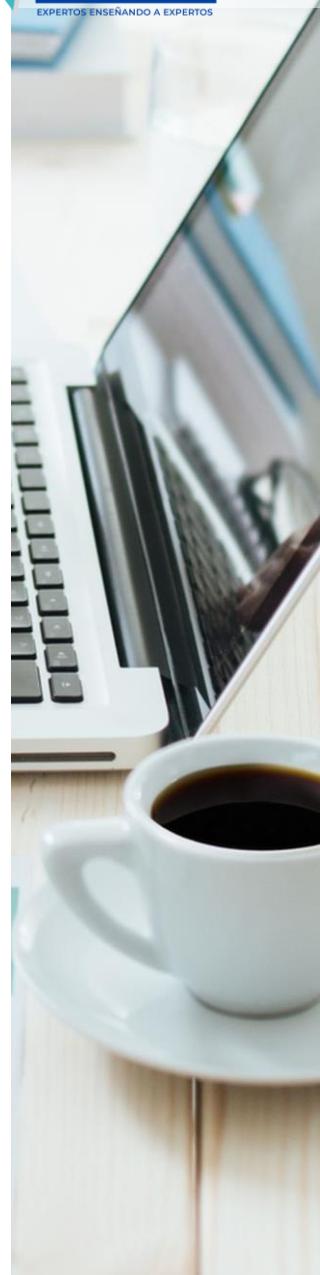
```
with open('mis_archivos/zen.txt','w') as f:
 f.write('La fuerza este contigo.')

```



# Práctica: Escribir en un archivo

- En `p8_2.py`, cambiar la cadena de caracteres que pasó al método `write()` y ejecutar el programa nuevamente. ¿Se reescribió nuevamente el archivo?
- Ahora cambia el modo de escritura '`w`' a agregar '`a`' y ejecuta nuevamente el programa.
- ¿Qué puede observar? Comenta tu resultado.
- El método `seek()` se usa para cambiar la posición del cursor en el archivo. Es muy común volver al comienzo del archivo pasando al método un 0. Si se imprimen los resultados con una invocación `read()`, probablemente obtendrás solo un cambio de línea.
- Esto es porque el cursor está en el extremo del archivo. Utiliza el método `seek()` para solucionar esto, e imprime el contenido del archivo `zen.txt`.



# Práctica: Creador de Listas de Artículos

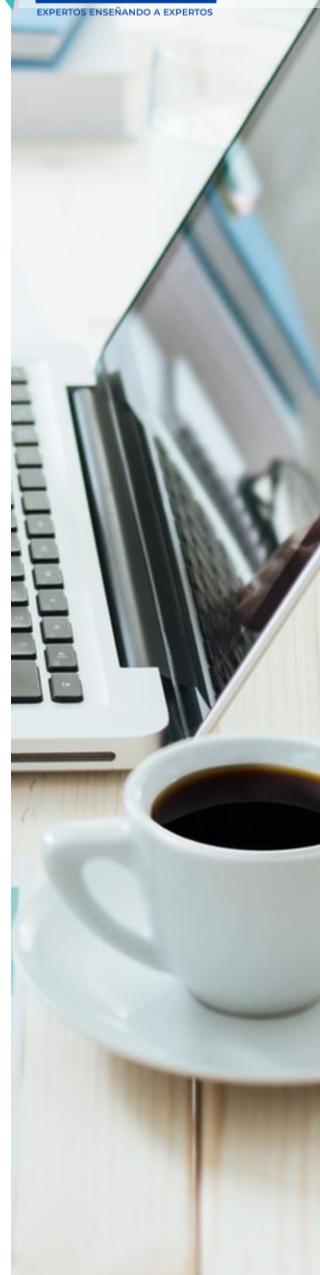
1. Captura el código siguiente en el archivo p8\_3.py.

```
def add_item(item):
 """ Agrega un item despues de quitar los espacios en blanco del archivo list.txt"""
 pass # Ingresa tu código aquí

def remove_item(item):
 """ Remueve la primera instancia de item en la list.txt, si el producto no existe en la
lista, avisa al usuario"""
 pass # Ingresa tu código aquí.

def delete_list():
 """Borra el contenido del archivo list.txt """
 pass # Ingresa tu código aquí.

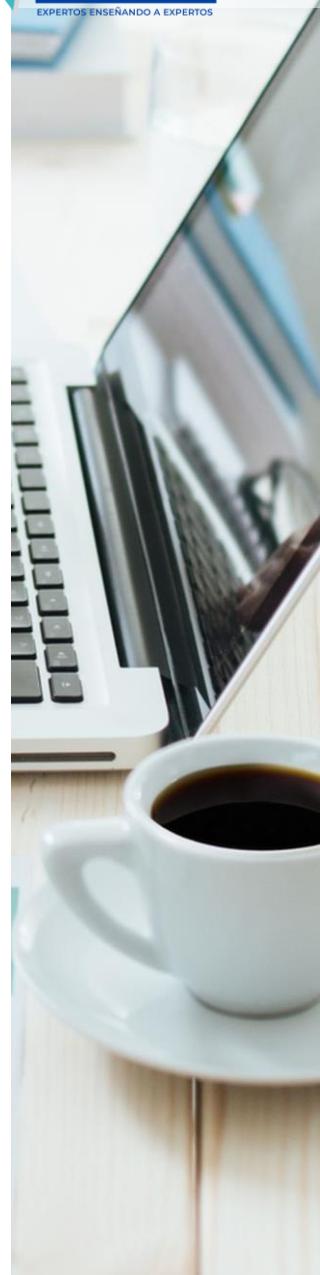
def print_list():
 """Despliega el contenido de la lista"""
 pass # Ingresa tu código aquí.
```



# Práctica: Creador de Listas de Artículos

```
def show_instructions():
 """Imprime las instrucciones"""

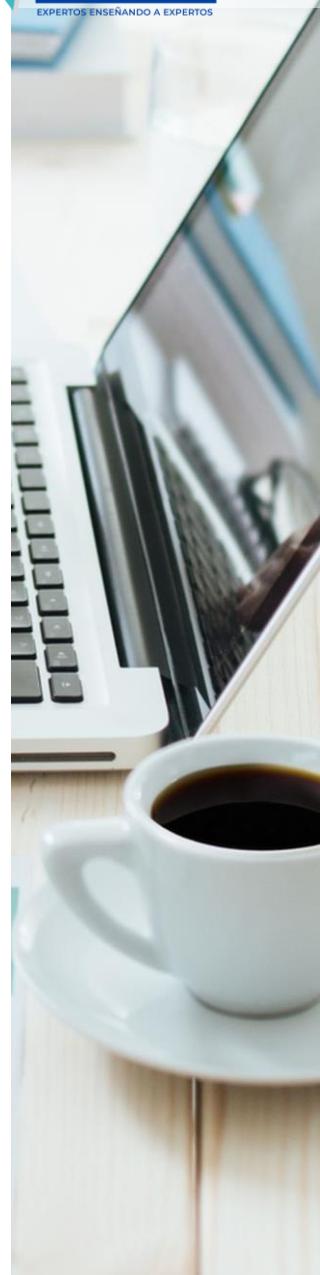
 print("""Opciones:
P -- Imprime la lista.
+abc -- Agrega 'abc' a la lista.
-abc -- Remove 'abc' de la lista.
--all -- Borra la lista entera.
Q -- Salir\n""")
```



```
def main():
 show_instructions()

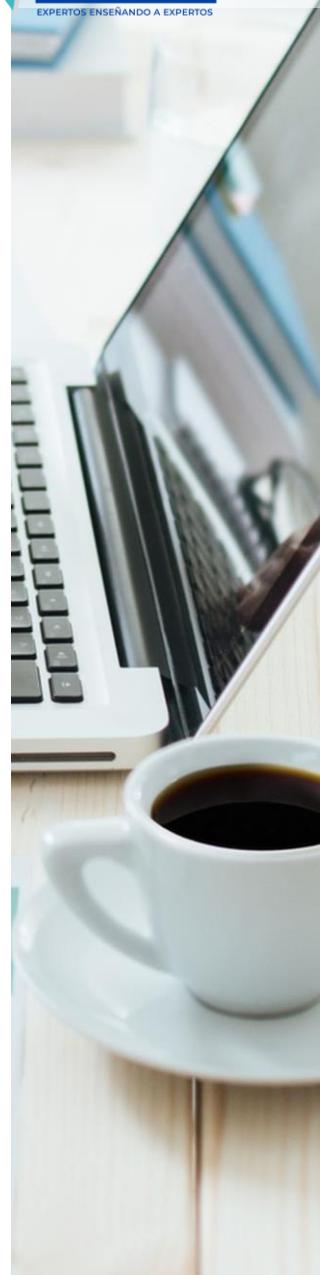
 while True:
 choice = input('>> ')
 if choice.lower() == 'q':
 print('Adios!')
 break
 elif choice.lower() == 'p':
 print_list()
 elif choice.lower() == '--all':
 delete_list()
 elif len(choice) and choice[0] == '+':
 add_item(choice[1:])
 elif len(choice) and choice[0] == '-':
 remove_item(choice[1:])
 else:
 print("No se entiende.")
 show_instructions()

if __name__ == '__main__':
 main()
```



# Práctica: Creador de Listas de Artículos

2. Las funciones main() y show\_instructions() ya están escritas.
3. Tu trabajo es escribir las siguientes funciones, las cuales están documentadas en el archivo:
  - a) add\_item(elemento)
  - b) remove\_item(elemento)
  - c) delete\_list()
  - d) print\_list()



```
C:\> Administrator: Command Prompt - python p8_3.py

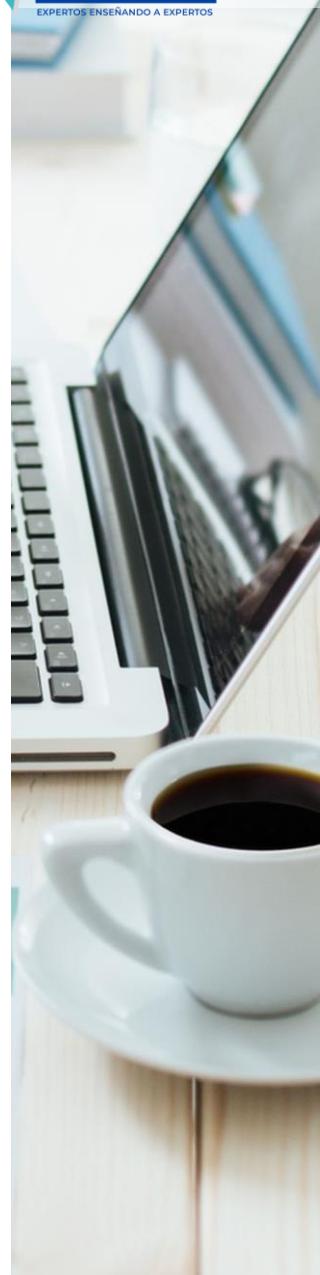
Opciones:
P -- Imprime la lista.
+abc -- Agrega 'abc' a la lista.
-abc -- Remove 'abc' de la lista.
--all -- Borra la lista entera.
Q -- Salir

>> P
piÃ±as
naranjas
mandarinas
duraznos

>> --all
La lista se ha borrado.
>> P

>> +limones
>> +naranjas
>> -limones
>> -zarzamoras
"zarzamoras" elemento no encontrado en la lista.
>> P
naranjas

>>
```





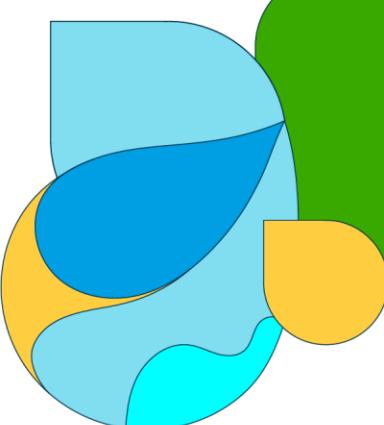
## Referencias Bibliográficas

- Windows 1252: <https://es.wikipedia.org/wiki/Windows-1252>
- File handling:  
[https://www.w3schools.com/python/python\\_file\\_handling.asp](https://www.w3schools.com/python/python_file_handling.asp)
- Python Delete File:  
[https://www.w3schools.com/python/python\\_file\\_remove.asp](https://www.w3schools.com/python/python_file_remove.asp)
- Input and Output:  
<https://docs.python.org/3/tutorial/inputoutput.html>
- Open function:  
<https://docs.python.org/3/library/functions.html#open>
- shutil — High-level file operations:  
<https://docs.python.org/3/library/shutil.html>

## 10.2 Closures

### Objetivos:

- Comprender el uso de funciones anidadas.
- Escribir funciones lambda.
- Usar funciones lambda.
- Escribir funciones auxiliares para facilitar la reutilización de código.
- Crear funciones que retengan el estado entre llamadas.
- Codificar funciones decoradoras para agregar comportamiento a las funciones existentes.
- Definir Closures en Python.



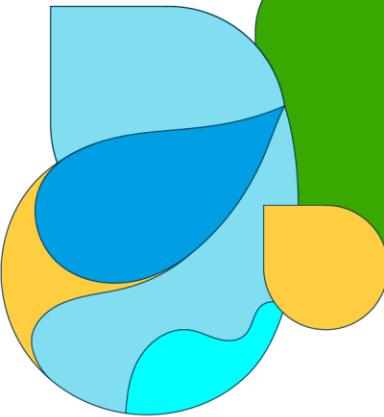
# Introducción

Closure: objeto de tipo función que recuerda valores en ámbitos adjuntos  
utiliza funciones anidadas.

```
def func_out(): # Función contenedora
 men= 'Variable en la función out'

 def func_in(): # Función anidada
 print (men)

 return func_in
```



# Funciones Lambda

- Las funciones lambda son funciones anónimas que generalmente se utilizan para completar pequeñas tareas y después de realizarlas ya no son necesarias.

lambda parámetros: expresión

# Funciones Lambda

```
ff = lambda n: n*2

print (ff(4))
print (ff(2))
print (ff(5))
print (ff("-----"))
```



8  
4  
10  
-----

# Funciones Lambda | Ejemplo

```
gg= lambda a,b : a + b
print (gg(1,2)) # 3

hh= lambda val : val[0] + val[1]
print (hh((10,30))) # 40
```

```
Maps & Lambdas
def cuadrado(n):
 return n**2

ss = list(map(cuadrado, range(10)))

print (ss)
```



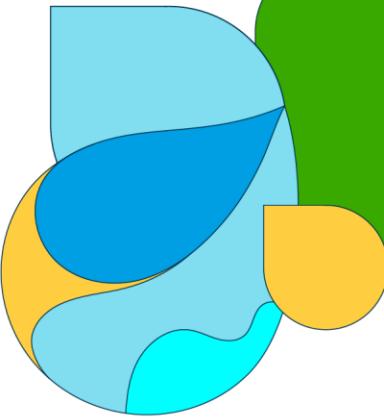
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Funciones Lambda | Ejemplo

```
cuadrados = list(map(lambda n: n**2, range(10)))
print(cuadrados)
```



```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```



# Funciones Lambda | Ejemplo

```
List comprehension
```

```
cuadrados = [n**2 for n in range(10)]
print(cuadrados)
```

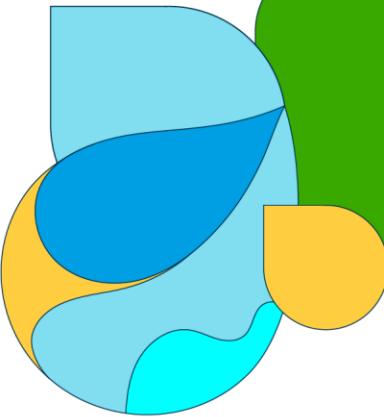


```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

# Funciones Lambda | Ejemplo

```
f = lambda n: n**.5 == int(n**.5)
print(f(25))
print(f(7))
```

True  
False

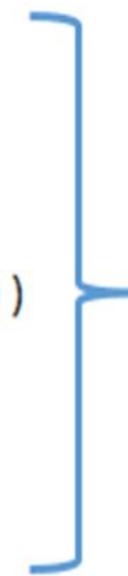


# Funciones Lambda | Ejemplo

```
def es_cuadrado_perfecto(n):
 return n**.5 == int(n**.5)

ss = list(filter(es_cuadrado_perfecto, range(100)))

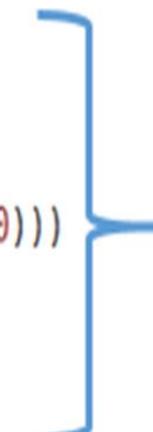
print (ss)
```



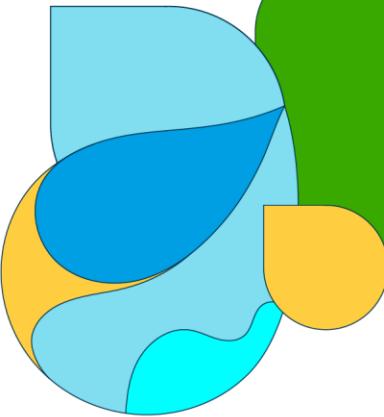
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Funciones Lambda | Ejemplos

```
cc = list(filter(lambda n: n**.5 == int(n**.5), range(100)))
print(cc)
```



```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

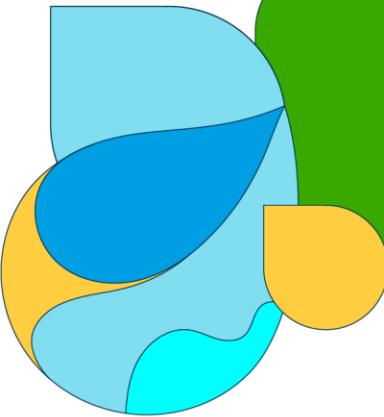


# Funciones Anidadas

```
def outer_func():
 #
 def inner_func():
 print ("Inner Function")

 inner_func()

outer_func()
```



# Funciones Anidadas

```
def outer_fun(men):
 def inner_fun():
 print(f'Hola, {men}!!')
 inner_fun()

outer_fun('Everardo')
```

# Encapsulamiento

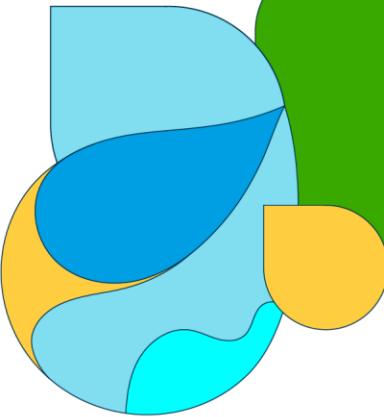
```
def incrementar(valor):
 def inner_increment():
 return valor + 1

 return inner_increment()

print(incrementar(5)) # 6

print (inner_increment()) #
```

NameError: name 'inner\_increment' is not defined



# Closures

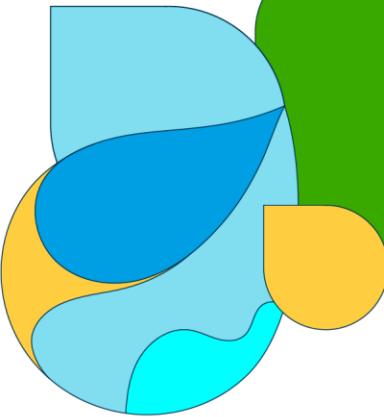
**Para definir un closure debes realizar lo siguiente:**

1. Crear una función anidada.
2. Consultar las variables de la función contenedora.
3. Devolver una función anidada.

```
def gen(exponente):
 def power(base): # Definición
 return base ** exponente #
 return power # Devuelve la función

f=gen(2) # exponente toma el valor

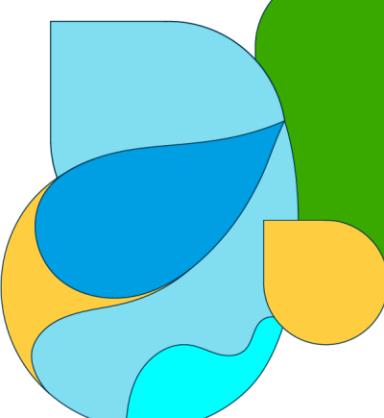
print (f(5)) # 25
```



# Closures | Ejemplo

```
1
2 def gen(exponente):
3 def power(base): # Definición de función interna o anidada
4 return base ** exponente # Hacer referencia a las variables de la función contenedora
5 return power # Devuelve la función interna
6
7 f=gen(2) # exponente toma el valor de 2
8
9 print (f(5)) # 25
10
```

```
>>> from power import gen
25
>>> a=gen(2)
>>> b=gen(3)
>>>
>>> a(5)
25
>>> a(3)
9
>>> b(5)
125
>>> b(3)
27
```

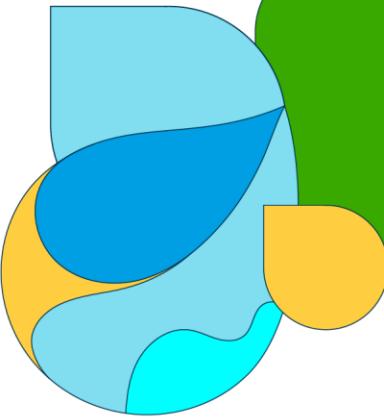


## Closures | Ejemplo

```
def fun(documento):
 def permisos(usuario):
 if usuario.lower() == 'root':
 return f'{usuario} tiene permisos sobre {documento}.'
 return f'{usuario} no tiene permisos sobre {documento}.'
 return permisos

aa=fun('Admin Page')

print(aa('root')) # 'root' tiene permisos sobre Admin Page.
print(aa('john')) # 'john' no tiene permisos sobre Admin Page.
```



# Closures | Ejemplo

```
def promedio():
 lista= []
 def inner(numero):
 lista.append(numero)
 return sum(lista) / len (lista)
 return inner
```

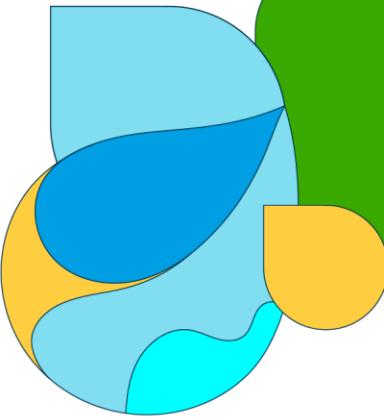
```
>>> from promedios import promedio
>>>
>>> aa=promedio()
>>>
>>> aa(1)
1.0
>>> aa(3)
2.0
>>> aa(5)
3.0
>>> aa(100)
27.25
>>> aa(101)
42.0
```

# Closures | Modificación del Estado

```
def fun(x,y):
 def point():
 print(f"Point ({x},{y})")
 def get_x():
 return x
 def get_y():
 return y
 def set_x(valor):
 nonlocal x
 x= valor
 def set_y(valor):
 nonlocal y
 y= valor
```

```
Getters & Setters
point.get_x = get_x
point.get_y = get_y
point.set_x = set_x
point.set_y = set_y
return point
```

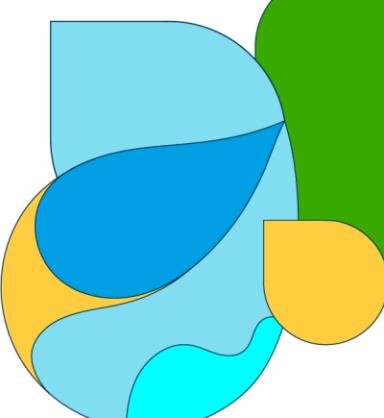
```
>>> from closures3 import fun
>>>
>>> point= fun(1,1)
>>> point.get_x()
1
>>> point.get_y()
1
>>> point.set_x(100)
>>> point.set_y(200)
>>>
>>> point.get_x()
100
>>> point.get_y()
200
```



# Decoradores

- Un decorador es el nombre de un patrón de diseño.
- Los decoradores alteran de manera dinámica la funcionalidad de una función, método o clase sin tener que hacer subclases o cambiar el código fuente de la clase o función decorada.
- Los decoradores forman parte de Python desde la versión 2.4
- Aportan lo siguiente:
  - Reducen el código común y repetitivo conocido como código boilerplate.
  - Favorecen la separación de responsabilidades del código.
  - Aumentan la legibilidad y la mantenibilidad.
  - Los decoradores son explícitos.





# Decoradores

```
@net_decorador
def fun():
 print ("Función con un propósito específico")
```

"""

El código anterior es equivalente a:

```
def fun():
 print ("Función con un propósito específico")

fun= net_decorador(fun)
"""
```

# Decoradores | Ejemplo

```
def net_decorador(función):

 # inner es una función wrapper en la cual el argumento es invocado, pues es una función
 # Las funciones anidadas (inner) pueden accesar a las funciones locales como en este caso.
 def inner():
 print ("Hola, esto mensaje es previo a la ejecución de la función")

 # Invocando a la función dentro de la función anidadas
 función()

 print ("Adios, esto mensaje es después de la ejecución de la función")

 return inner # Regresa referencia a función anidada inner

Definiendo una función outer con un propósito específico
def f_u():
 print ("Esta es una función outer o top level")

Pasando "f_u" a la función decorador para agregar o cambiar su funcionalidad
ff = net_decorador(f_u)

Llamando a la función
ff()
```

# Decoradores | Ejemplo

```
def net_decorador(función):
 # inner es una función wrapper en la cual el argumento es invocado, pues es una función
 # Las funciones anidadas (inner) pueden accesar a las funciones locales como en este caso.
 def inner():
 print ("Hola, esto mensaje es previo a la ejecución de la función")

 # Invocando a la función dentro de la función anidadas
 función()

 print ("Adios, esto mensaje es después de la ejecución de la función")

 return inner # Regresa referencia a función anidada inner

Definiendo una función outer con un propósito específico
def f_u():
 print ("Esta es una función outer o top level")

Pasando "f_u" a la función decorador para agregar o cambiar su funcionalidad
ff = net_decorador(f_u)

Llamando a la función
ff()
```

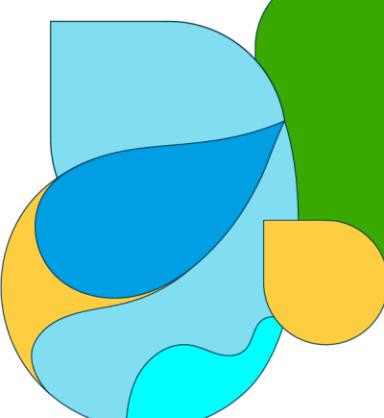
1

3

2

4

5



# Decoradores

```
inner es una función wrapper en la cual el argumento es invocado, pues es una función
Las funciones anidadas (inner) pueden accesar a las funciones locales como en este caso.
def inner():
 print ("Hola, esto mensaje es previo a la ejecución de la función")

 # Invocando a la función dentro de la función anidadas
 función()

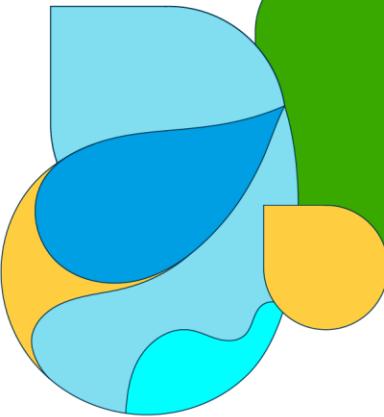
 print ("Adios, esto mensaje es después de la ejecución de la función")

return inner # Regresa referencia a función anidada inner
```

1

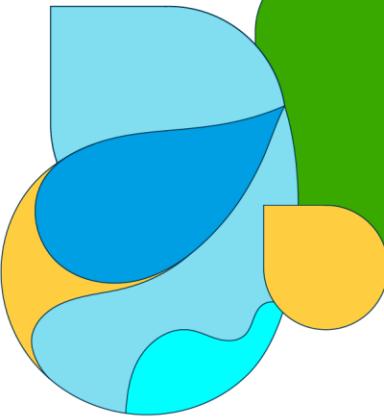
2

3



# Decoradores

Hola, este mensaje es previo a la ejecución de la función  
Esta es una función outer o top level  
Adios, este mensaje es después de la ejecución de la función



# Decoradores | Ejemplo

```
from datetime import datetime

def format_report(f):
 def inner(text): # Función anidada
 print('Reporte iniciado a las: {}'.format(datetime.now()))
 print('-' * 50)
 f(text)
 print('-' * 50)
 print('Reporte completetado a las: {}'.format(datetime.now()))
 return inner # Devuelve una referencia a la función anidada

def report(text): # Función outer con argumentos
 print(text)

rr = format_report(report) # Invocación del decorador

rr('Este es un proyecto con estos datos') # Invocación con argumentos
```

# Decoradores

```
D:\MaterialPythonDeveloper\MA\11_Decoradores\Demos>python Decorators.py
Reporte iniciado a las: 2022-05-31 13:11:06.915933

```

```
Este es un proyecto con estos datos

```

```
Reporte completetado a las: 2022-05-31 13:11:06.916933.
```

```
D:\MaterialPythonDeveloper\MA\11_Decoradores\Demos>
D:\MaterialPythonDeveloper\MA\11_Decoradores\Demos>python Decorators.py
Reporte iniciado a las: 2022-05-31 13:13:05.310012

```

```
Este es un proyecto con estos datos

```

```
Reporte completetado a las: 2022-05-31 13:13:05.311013.
```

```
D:\MaterialPythonDeveloper\MA\11_Decoradores\Demos>python Decorators.py
Reporte iniciado a las: 2022-05-31 13:16:32.646904

```

```
Este es un proyecto con estos datos

```

```
Reporte completetado a las: 2022-05-31 13:16:32.647906.
```

# Resumen del capítulo

- Las funciones en Python son también objetos.
- Las funciones pueden ser asignadas a variables.
- Las funciones se pueden pasar como argumentos a una función determinada.
- Las funciones pueden regresar funciones.
- Recuerda que en los capítulos previos se ha usado el decorador `@property`, `@staticmethod`, `@classmethod` y `@abc.abstractmethod`.
- Si se define una función interna dentro de otra función, entonces se está creando una **función interna**, también conocida como **función anidada**.

# Resumen del capítulo

- En Python, las funciones internas tienen acceso directo a las variables y nombres que se definen en la función contenedora. Esto proporciona un mecanismo para crear funciones **auxiliares, closures y decoradores**.
- Con las funciones anidadas se puede proporcionar encapsulamiento.
- Al implementar funciones de fábrica de closures que mantienen el estado entre llamadas.
- Los decoradores permiten proporcionar nuevas funcionalidades.

# Resumen del capítulo

- Además, se reafirmó el uso de las funciones preconstruidas de mapeo y filtrado, con funciones lambda.

| Función  | Descripción                                                                                                   |
|----------|---------------------------------------------------------------------------------------------------------------|
| filter() | Construye un iterador a partir de aquellos elementos del iterable para los que la función devuelve verdadero. |
| map()    | Devuelve un iterador que aplica la función a cada elemento del iterable.                                      |

# Práctica: Decoradores

1. Crea un archivo de nombre `p9_1.py` con el siguiente contenido:

```
Uso de decoradores
def decorador (fun):
 def inner (*args, **kwargs):
 pass # Aquí podría ir tu código
 return inner

Agrega el decorador a la función
def suma(a,b):
 pass # Aquí podría ir tu código

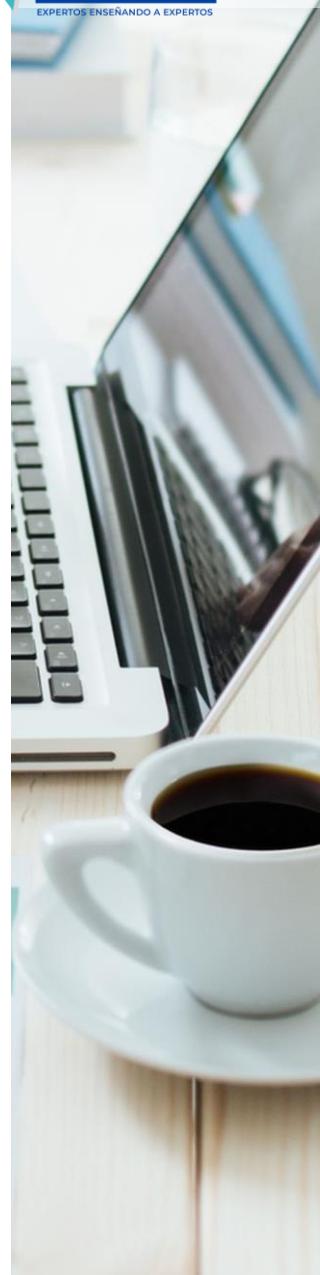
a, b = 30, 50

Invocación a la funcionalidad normal
print ("{} + {} = {}".format(a,b,suma(a,b)))
```



# Práctica: Decoradores

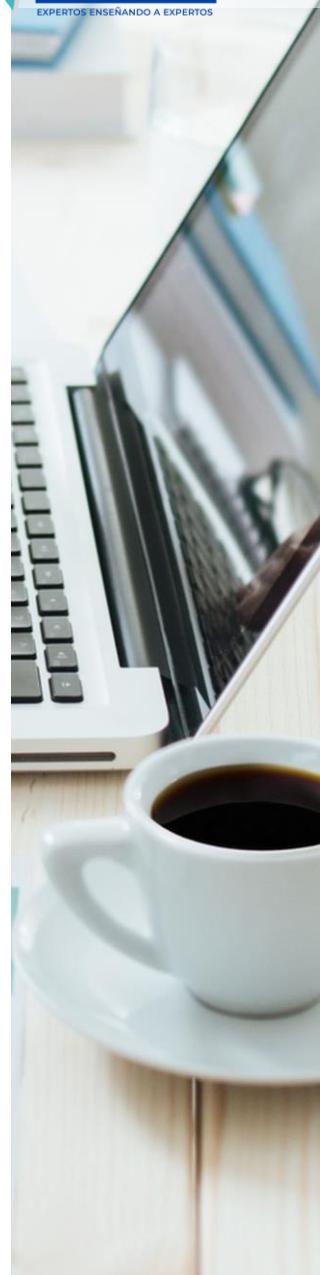
2. Define el decorador para recibir los argumentos de la función suma.
3. Imprime un mensaje antes de invocar a la función outer.
4. Almacena el resultado de la suma.
5. Imprime un mensaje después de invocar a la función outer.
6. Haz que la función inner devuelva el resultado almacenado en el punto 4.
7. Agrega el decorador a la función outer suma().
8. Agrega la funcionalidad a la función suma(). Sugerencia: despliega un mensaje y devuelve el resultado de suma a más b.



# Práctica: Decoradores

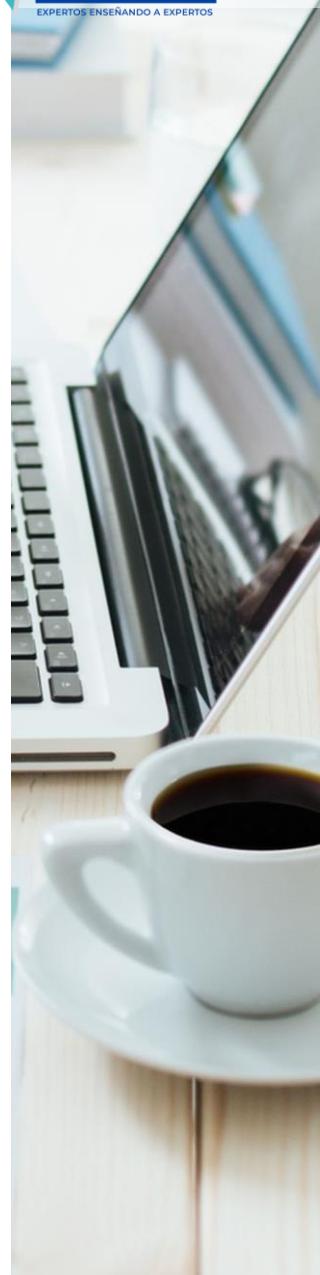
```
C:\Users\Administrator\ws_python>python p9_1.py
Antes de ejecutar la función ''
En la función outer ''
Después de ejecutar la función
30 + 50 = 80

C:\Users\Administrator\ws_python>
```



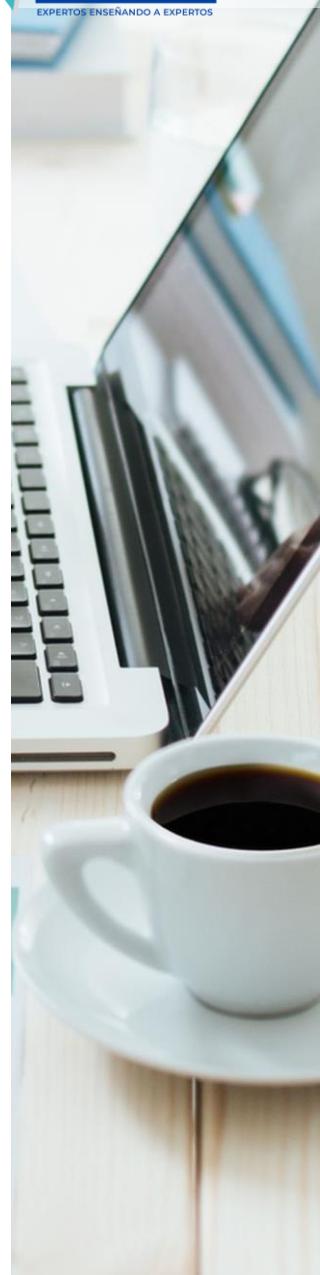
# Práctica: Closures

1. Crea un archivo de nombre `p9_2.py` con las instrucciones siguientes:
2. Haz un ciclo anidado y un closure Python para hacer funciones que obtengan múltiples funciones de multiplicación usando cierres.
3. Crea un closure para multiplicar por 3 y multiplicar por 5.
4. La salida debería ser similar a la siguiente captura de pantalla:



# Práctica: Closures

```
C:\> Select Administrator: Command Prompt - python -q
C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>
C:\Users\Administrator\ws_python>python -q
>>>
>>> from p9_2 import multiplicador
>>> mul3=multiplicador(3)
>>> mul5=multiplicador(5)
>>>
>>> print (mul3(3), mul5(3))
9 15
>>> -
```





## Referencias Bibliográficas

- Build-in Functions:  
<https://docs.python.org/3/library/functions.html>
- Let's just \*keep\*lambda:  
<https://mail.python.org/pipermail/python-dev/2006-February/060415.html>
- Simple tool for simulating classes using closures and nested scopes: <https://code.activestate.com/recipes/578091-simple-tool-for-simulating-classes-using-closures-/>

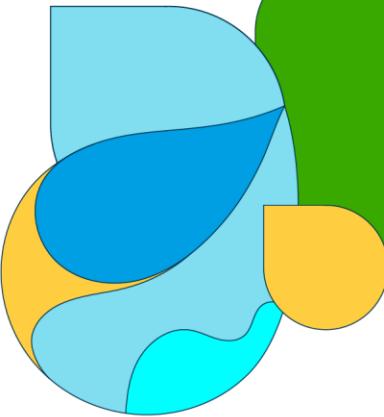
# Unidad temática 11

Bases de datos y Archivo CSV

## 11.1 Bases de Datos

### Objetivos:

- Acceder y trabajar con información almacenada en una base de datos relacional.
- Consultar la base de datos y recuperar registros a través de MySQL para Python.
- Implementar la inserción de datos en una base de datos MySQL en Python.



# Introducción

ORACLE®  
DATABASE



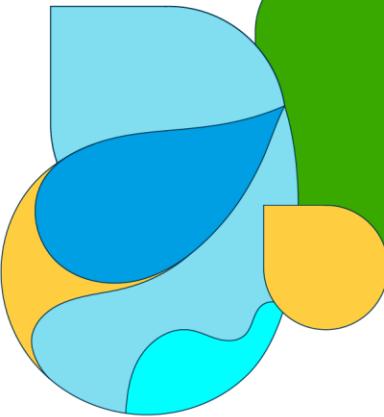
MySQL



PostgreSQL



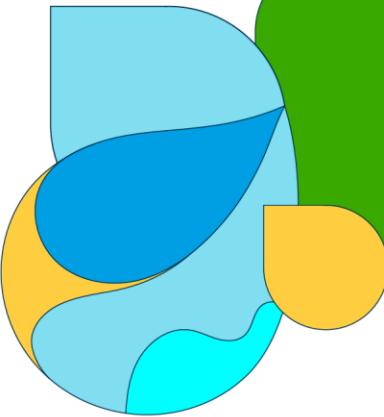
Microsoft®  
SQL Server®



# MySQL

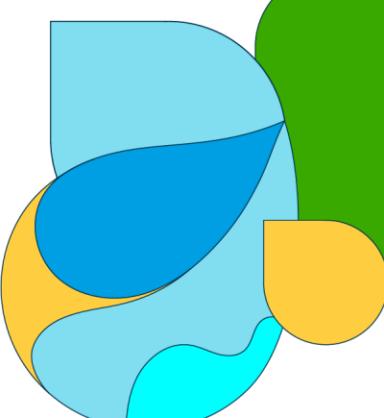
- Base de datos relacional y multiusuario.
- Más de 6 millones de instalaciones.
- Licencia GNU GPL.
- Desarrollada en ANSI-C & C++
- Forma parte de LAM & WAMP.
- Versión Community.
- Multiplataforma.
- MultiAPI.





# MySQL | Ventajas

- Software con Licencia GPL.
- Bajo costo en requerimientos para la elaboración y ejecución del programa.
- No se necesita disponer de hardware o software de alto rendimiento para la ejecución del programa.
- Velocidad al realizar las operaciones y buen rendimiento.
- Facilidad de instalación y configuración.
- Soporte en casi el 100% de los sistemas operativos actuales.
- Soporte de conectividad para la mayoría de los lenguajes de programación.
- Baja probabilidad de corrupción de datos.
- Entorno con seguridad y encriptación.
- Cuenta con varios motores.
- Cuenta con versiones empresariales.

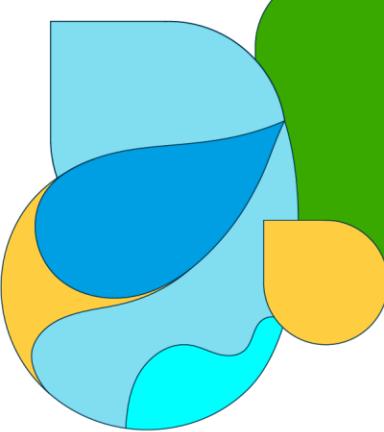


# MySQL

```
C:\> mysql -h localhost -u root -p
```

```
mysql> show databases;
```

```
mysql> select version(), user(), now(), current_date;
+-----+-----+-----+-----+
| version() | user() | now() | current_date |
+-----+-----+-----+-----+
| 8.0.26 | root@localhost | 2022-05-31 20:17:53 | 2022-05-31 |
+-----+-----+-----+-----+
1 row in set (0.00 sec)
```



# MySQL | DDL

```
mysql> create database <nombre_db>;
```

```
mysql> use <nombre_db>;
```

```
mysql> create table <nombre_tb> (
```

```
 nombre_col tipo [constraints],
```

```
 ...
```

```
);
```

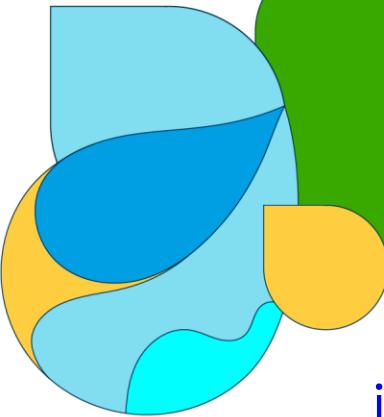
```
mysql> show tables;
```

```
mysql> desc nombre_tb;
```

# MySQL | DDL

```
mysql> show tables;
+-----+
| Tables_in_netecdb |
+-----+
| tb_data |
+-----+
1 row in set (0.02 sec)
```

```
mysql>
mysql> desc tb_data;
+-----+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
id	int	NO	PRI	NULL	auto_increment
name	varchar(35)	YES		NULL	
value	int	YES		NULL	
+-----+-----+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```



# MySQL | DML

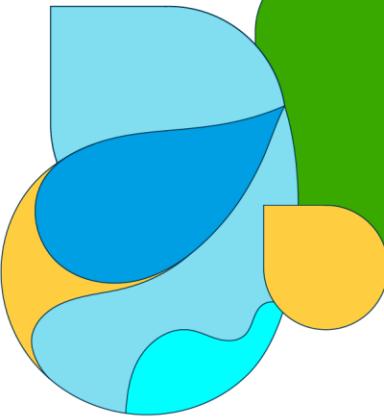
```
insert into products (id, description, price) values (456, 'DISCO', 450);
```

```
update products set price = 1000 where precio > 500};
```

```
delete from productos where precio < 100;
```

```
mysql> desc products;
```

| Field       | Type          | Null | Key | Default | Extra |
|-------------|---------------|------|-----|---------|-------|
| id          | int           | NO   | PRI | NULL    |       |
| description | varchar(50)   | YES  |     | NULL    |       |
| price       | decimal(10,0) | YES  |     | NULL    |       |



# MySQL | Consultas

```
select description from product where price>=350 and price<=1000;
```

```
select count(*) from products;
```

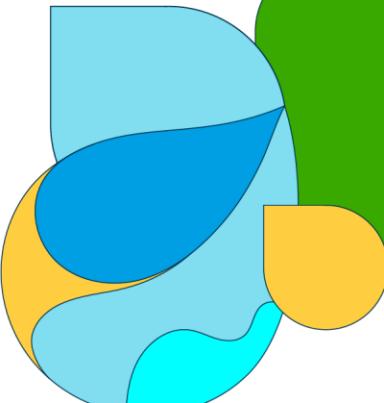
```
select id, description, price from products;
```

```
select * from products;
```

```
select id, description, price * 1.25 'nuevo precio' from products;
```

```
select * from products order by price desc;
```

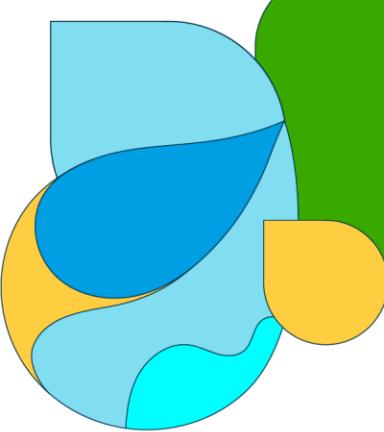
```
select * from products limit 5, 10;
```



# Python & MySQL

```
C:\Users\Administrator>pip --version
pip 22.1.2 from C:\Program Files\Python310\lib\site-packages\pip (python 3.10)
```

```
C:\Users\Administrator>
C:\Users\Administrator>pip install mysql-connector-python
Collecting mysql-connector-python
 Downloading mysql_connector_python-8.0.29-cp310-cp310-win_amd64.whl (7.7 MB)
 ███ 7.7/7.7 MB 3.1 MB/s eta 0:00:00
Collecting protobuf>=3.0.0
 Downloading protobuf-4.21.1-cp310-abi3-win_amd64.whl (524 kB)
 ███ 525.0/525.0 kB 2.1 MB/s eta 0:00:00
Installing collected packages: protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.29 protobuf-4.21.1
```



# Python | MySQL

1. Importar Módulo

6. Cerrar el cursor

7. Cerrar la conexión

2. Abrir una Conexión

5. Obtener los resultados

3. Escribir la consulta SQL

4. Crear un Cursor

1. Importar el módulo compatible con Python Database API-2.0.

```
Módulo mysql.connector
import mysql.connector
```

2. Abrir una conexión a la base de datos.

```
connection = mysql.connector.connect(
 host='localhost',
 user='root',
 password='pancho123',
 database='classicmodels')
```

### 3. Escribir la consulta SQL en una cadena de caracteres.

```
query = '''SELECT employeeNumber, firstName, lastName, jobTitle
FROM Employees
ORDER BY lastname DESC
LIMIT 10'''
```

### 4. Crear un cursor para ejecutar una o más consultas.

```
cursor = connection.cursor() # con dictionary = True para devolver
 # registros como dicts
```

5. Obtener los resultados de la consulta desde el cursor.

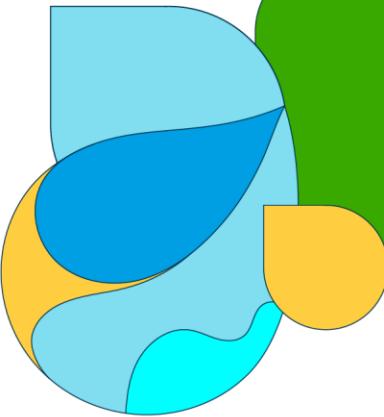
```
cursor.execute(query)
results = cursor.fetchall()
```

6. Cerrar el cursor.

```
cursor.close()
```

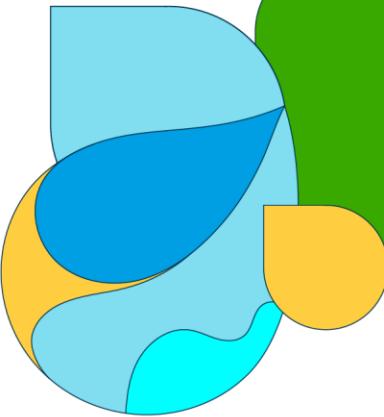
7. Cerrar la conexión a la base de datos.

```
connection.close()
```



# Python | MySQL

| Método             | Descripción                                                    |
|--------------------|----------------------------------------------------------------|
| cursor.execute     | Prepara y ejecuta una consulta a la base de datos.             |
| cursor.executemany | Prepara una consulta y la ejecuta una vez para cada secuencia. |
| cursor.fetchone    | Obtiene la siguiente fila.                                     |
| cursor.fetchmany   | Obtiene las siguientes n filas.                                |
| cursor.fetchall    | Obtiene todas las filas de datos.                              |

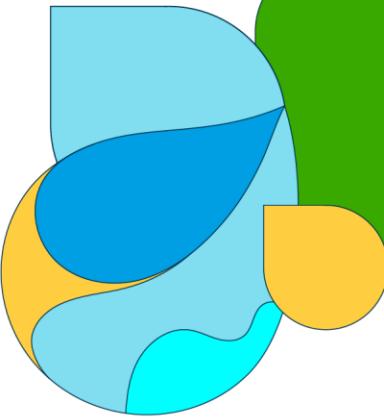


# PyMySQL

```
import pymysql # módulo PyMySQL

connection = pymysql.connect(host='localhost',
 user='root',
 password='pancho123',
 database='classicmodels'
)

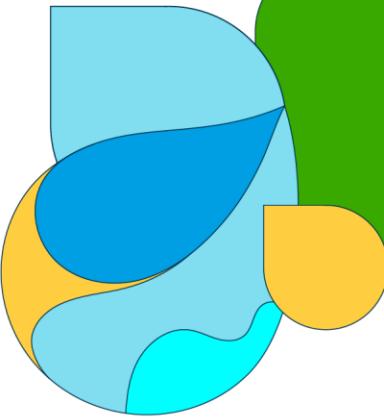
query = '''SELECT employeeNumber, firstName, lastName, reportsTo
FROM Employees
ORDER BY 1 ASC
LIMIT 10'''
```



# PyMySQL

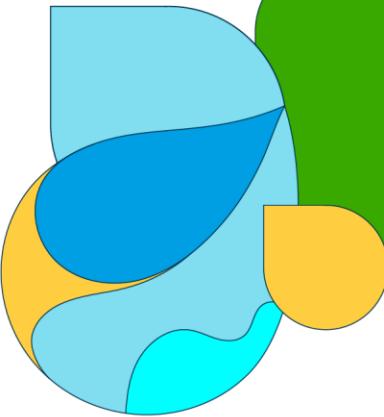
```
with connection.cursor() as cursor:
 cursor.execute(query)
 results = cursor.fetchall()
connection.close()

print ("Num firstName lastName reportsTo")
for ren in results:
 print ('{} {:10s} {:10s} {}'.format(ren[0], ren[1], ren[2], ren[3]))
```



# PyMySQL

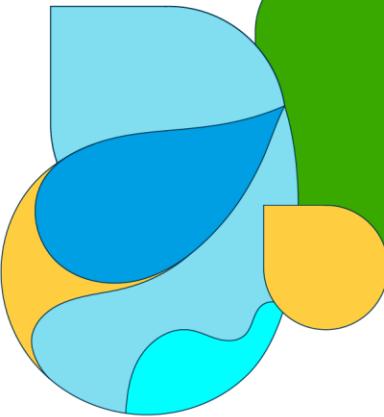
```
D:\demos>python pymysql_connect.py
Num firstName lastName reportsTo
1002 Diane Murphy None
1056 Mary Patterson 1002
1076 Jeff Firrelli 1002
1088 William Patterson 1056
1102 Gerard Bondur 1056
1143 Anthony Bow 1056
1165 Leslie Jennings 1143
1166 Leslie Thompson 1143
1188 Julie Firrelli 1143
1216 Steve Patterson 1143
```



# Diccionarios vs. Tuplas

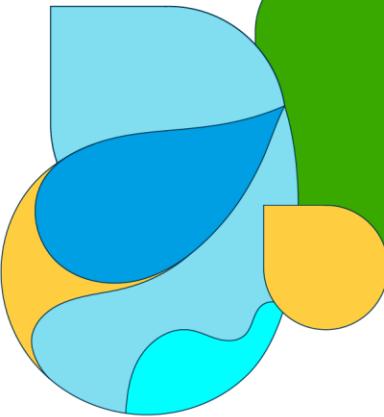
```
cursor = connection.cursor(dictionary=True)
```

```
connection = pymysql.connect(host='localhost',
 user='root',
 password='pancho123',
 database='classicmodels',
 cursorclass=pymysql.cursors.DictCursor
)
```



# Diccionarios vs. Tuplas

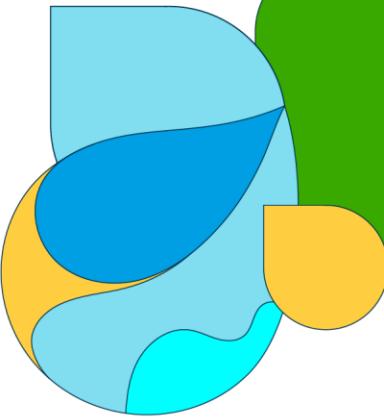
```
D:\>python pymysql_connect.py
{'employeeNumber': 1002, 'firstName': 'Diane', 'lastName': 'Murphy', 'reportsTo': None}
{'employeeNumber': 1056, 'firstName': 'Mary', 'lastName': 'Patterson', 'reportsTo': 1002}
{'employeeNumber': 1076, 'firstName': 'Jeff', 'lastName': 'Firrelli', 'reportsTo': 1002}
{'employeeNumber': 1088, 'firstName': 'William', 'lastName': 'Patterson', 'reportsTo': 1056}
{'employeeNumber': 1102, 'firstName': 'Gerard', 'lastName': 'Bondur', 'reportsTo': 1056}
{'employeeNumber': 1143, 'firstName': 'Anthony', 'lastName': 'Bow', 'reportsTo': 1056}
{'employeeNumber': 1165, 'firstName': 'Leslie', 'lastName': 'Jennings', 'reportsTo': 1143}
{'employeeNumber': 1166, 'firstName': 'Leslie', 'lastName': 'Thompson', 'reportsTo': 1143}
{'employeeNumber': 1188, 'firstName': 'Julie', 'lastName': 'Firrelli', 'reportsTo': 1143}
{'employeeNumber': 1216, 'firstName': 'Steve', 'lastName': 'Patterson', 'reportsTo': 1143}
```



# CRUD

- **CRUD** es un acrónimo para “Create, Retrieve, Update y Delete”.
- Crear o insertar, recuperar , actualizar y para borrar datos.
- Se usa para referirse a las funciones básicas de bases de datos comúnmente llamada la capa de persistencia.





# Creación

1. Importar Módulo

2. Abrir una Conexión

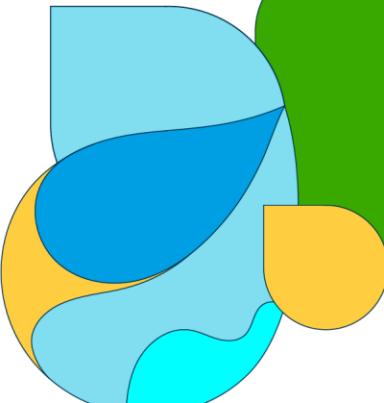
3. Crear un cursor

6. Aplicar los cambios,  
commit()

5. Ejecutar la instrucción

4. Preparar la  
instrucción

7. Cerrar la  
conexión

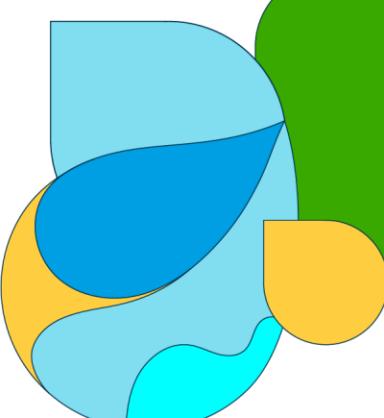


# Creación

```
def demo_insertar():
 try:
 conexion = pymysql.connect (host='localhost',
 user='root',
 password ='pancho123',
 db='classicmodels')
 try:
 with conexion.cursor() as cursor:
 ins = "INSERT INTO emp(employeeNumber, firstName, lastName, jobTitle) VALUES (%s,%s,%s,%s);"
 cursor.execute (ins, (1800, 'Gabriel', 'Guerrero', 'Director'))
 cursor.execute (ins, (1801, 'Jesús', 'Díaz', 'Técnico'))
 cursor.execute (ins, (1802, 'Fernando', 'Jaimes', 'Instructor'))
 conexion.commit()
 except:
 print ("No pudieron insertarse")
 raise
 finally:
 conexion.close()

 except (pymysql.err.OperationalError, pymysql.err.InternalError) as exc:
 print ("Se generó un error al conectarse: " , e)

demo_insertar()
```



# Actualización

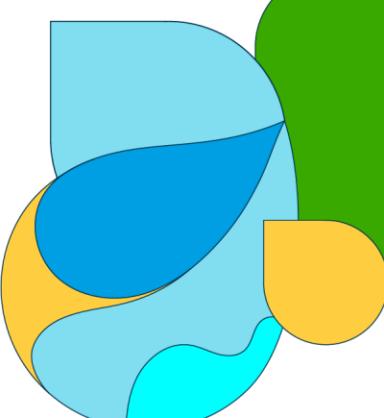
```
import pymysql

def actualiza():
 try:
 conexion = pymysql.connect (host='localhost',
 user='root',
 password ='pancho123',
 db='classicmodels')

 try:
 with conexion.cursor() as cursor:
 stm = "UPDATE emp set jobTitle = %s where employeeNumber = %s;"
 job = "Gerente"
 id = 1801
 cursor.execute(stm, (job,id))

 # Aplicar los cambios a la base

 conexion.commit()
 finally:
 conexion.close()
```



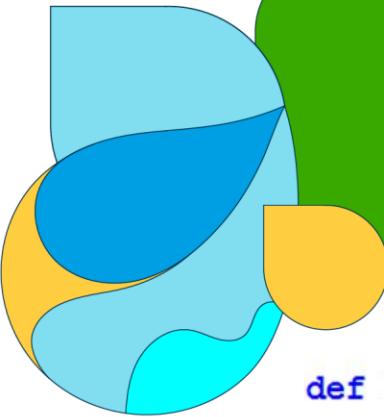
# Actualización

```
Aplicar los cambios a la base

 conexion.commit()
finally:
 conexion.close()

except (pymysql.err.OperationalError, pymysql.err.InternalError) as exc:
 print ("Se generó un error al conectarse: " , exc)

actualiza()
```



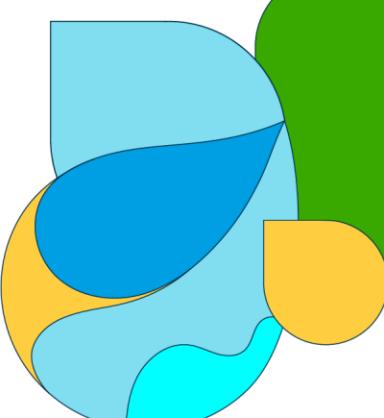
# Eliminación

```
def borrado():
 try:
 conexion = pymysql.connect (host='localhost',
 user='root',
 password ='pancho123',
 db='classicmodels')
 try:
 with conexion.cursor() as cursor:
 stm = "DELETE FROM emp where employeeNumber >= %s;" # Sentencia SQL para eleminar registros
 ids= 1800
 cursor.execute(stm, (ids))

 # Aplicar los cambios a la base
 conexion.commit()
 finally:
 conexion.close()

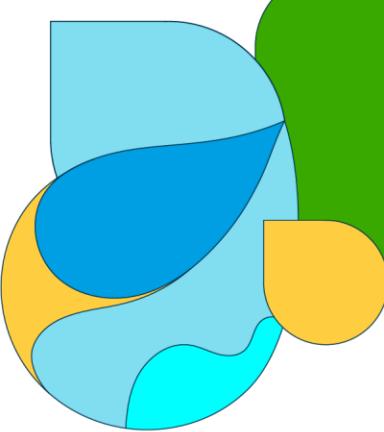
 except (pymysql.err.OperationalError, pymysql.err.InternalError) as exc:
 print ("Se generó un error al conectarse: " , exc)

borrado()
```



# Errores Comunes

- La dirección IP del servidor de la base de datos, asegura que es la dirección correcta o nombre correcto, y que desde el sistema operativo puede llegar al servidor de la base de datos, una sugerencia es probar con la instrucción ping y la IP del servidor.
- Nombre o contraseña del usuario incorrecta.
- Permisos, que el usuario no vea los datos porque no tiene los permisos adecuados en la base de datos.



## Errores Comunes (Cont.)

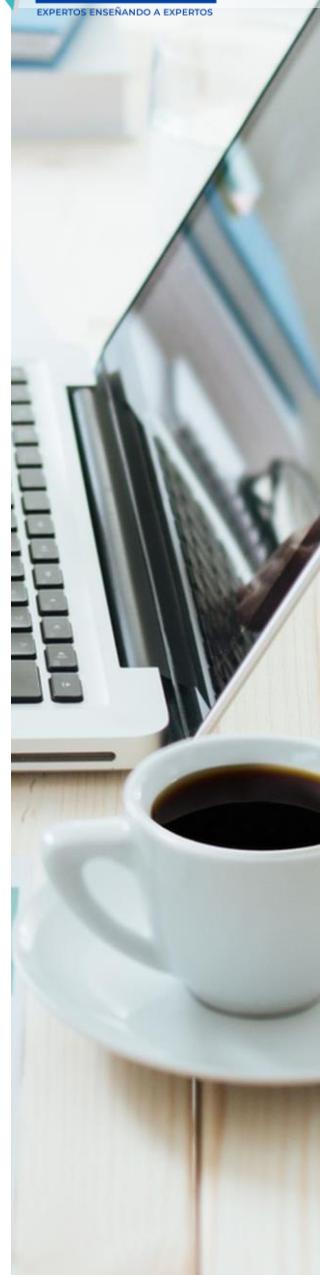
- El nombre de la base de datos no existe.
- El esquema de base de datos no sea el correcto, tablas y/o columnas.
- Los privilegios para ver consultar, insertar, actualizar y borrar la información.

# Resumen del capítulo

- En este capítulo se vio dónde encontrar MySQL para Python, ya que no forma parte de Python de forma predeterminada.
- Se vio cómo instalar MySQL en sistemas Windows.
- Los autores de MySQL para Python han eliminado el problema al proporcionar una forma muy fácil de instalar a través de pip.
- Se vio un flujo básico para conectarse a una base de datos, ejecutar sentencias SQL y procesar los resultados usando PyMySQL.

# Práctica: MySQL Consultas

1. Verifica la versión del servidor de MySQL instalado. En caso de no tener el software descarga la versión de la siguiente liga <https://dev.mysql.com/downloads/installer/>
2. Verifica que el servicio se encuentre levantado. Sugerencia:  
[c:\> net \[start|stop\] mysql80](#)
3. Verifica que el puerto **3306** se encuentre en estado de **LISTENING**.
4. Inicia la consola de MySQL. Sugerencia: **mysql -hlocalhost -uroot -p<contraseña>**



# Práctica: MySQL Consultas

5. Lista las bases de datos que se tienen instaladas.

Sugerencia: [show databases](#)

6. Usa la base de datos classicmodels. Sugerencia: [use classicmodels](#)

7. Ejecuta el script [mysqlsampledatabase.sql](#) en caso de no contar con la base de datos del punto anterior.

8. Verifica las tablas y cardinalidad existentes en la base de datos classicmodels. Sugerencia, usa el siguiente código:



# Práctica: MySQL Consultas

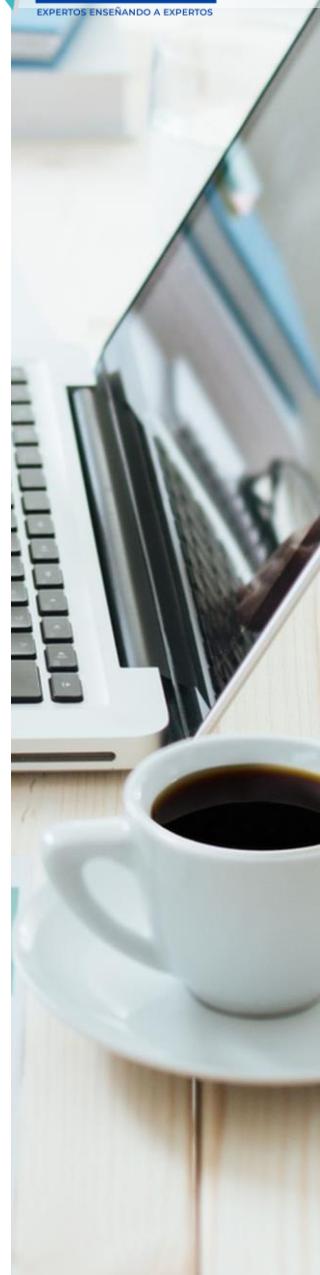
SELECT

```
table_schema as `Database`,
table_name AS `Table`,
round(((data_length + index_length) / 1024 / 1024), 2) `Size in
MB`,table_rows
```

FROM information\_schema.TABLES

WHERE table\_schema = 'classicmodels'

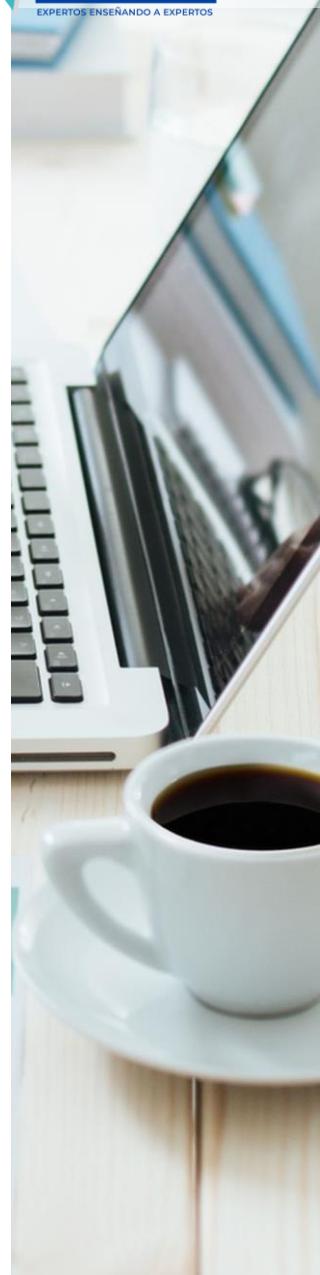
ORDER BY 3 DESC;



# Práctica: MySQL Consultas

```
Administrator: Command Prompt - mysql -uroot -pppancho123 -hlocalhost
-> ORDER BY 3 DESC;
+-----+-----+-----+-----+
| Database | Table | Size in MB | TABLE_ROWS |
+-----+-----+-----+-----+
classicmodels	orderdetails	0.23	2996
classicmodels	products	0.08	110
classicmodels	orders	0.06	326
classicmodels	employees	0.05	23
classicmodels	customers	0.03	122
classicmodels	offices	0.02	7
classicmodels	payments	0.02	273
classicmodels	productlines	0.02	7
+-----+-----+-----+-----+
8 rows in set (0.00 sec)

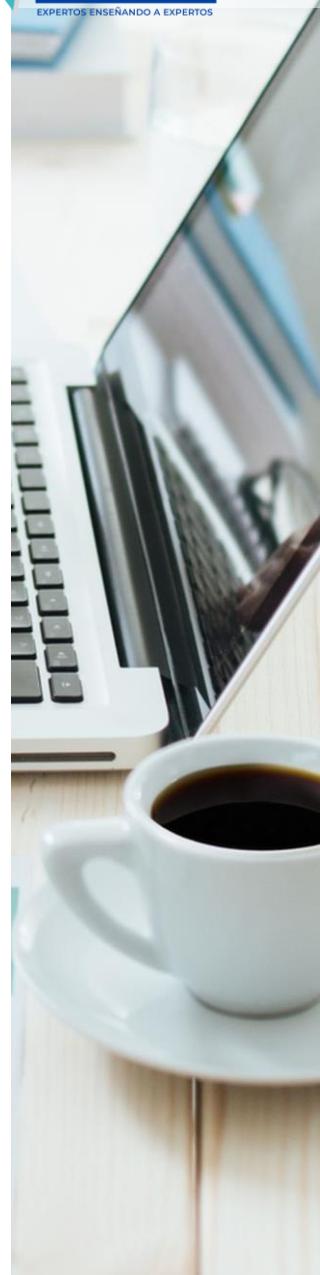
mysql>
```



# Práctica: MySQL Consultas

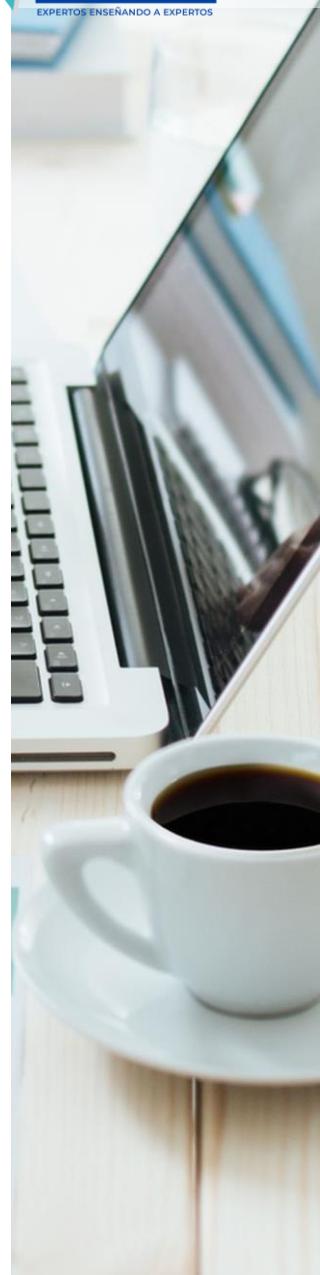
En esta práctica se usará Python para conectarse y realizar consultas a la base de datos `classicmodels`.

1. Instala el conector de MySQL:
  - `pip install mysql-connector-python`
  - `pip install pymysql`.
2. Prueba el conector MySQL. Crea un archivo `p10_2.py` con la siguiente instrucción:
  - `import mysql.connector.`
3. Crea una conexión a la base de datos, usa el nombre, la contraseña, y servidor local, como se vio en el módulo.



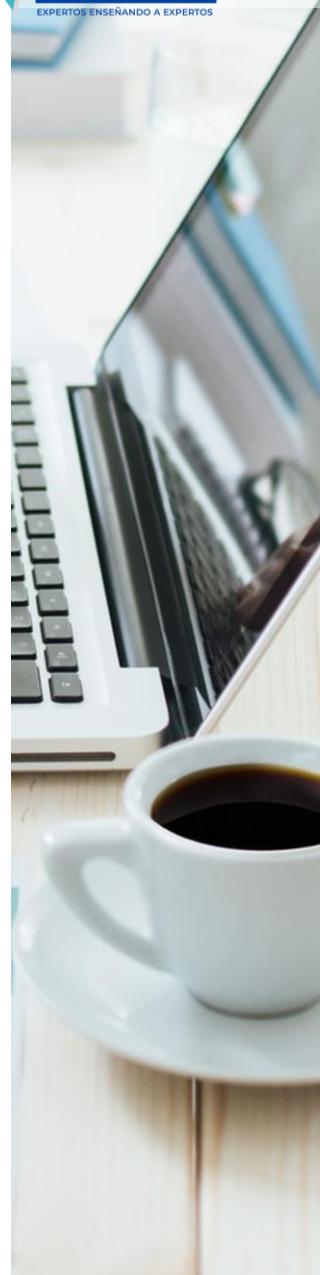
# Práctica: MySQL Consultas

4. Comprueba usando Python si la base de datos existe. Sugerencia:  
`cursor.execute ("show databases")`.
5. Cambia tu conexión agregando la base de datos `classicmodels`.  
Sugerencia: Agrega en el punto 3 la clave `database="classicmodels"`.
6. Comprueba que tablas tiene la base de datos. Sugerencia:  
`cursor.execute ("show tables")`.
7. Crea una función en Python que despliegue el `JobTitle`,  
`employeeNumber`, `firstName` y `lastName` de los registros en la tabla  
`Employees`.



# Práctica: MySQL Consultas

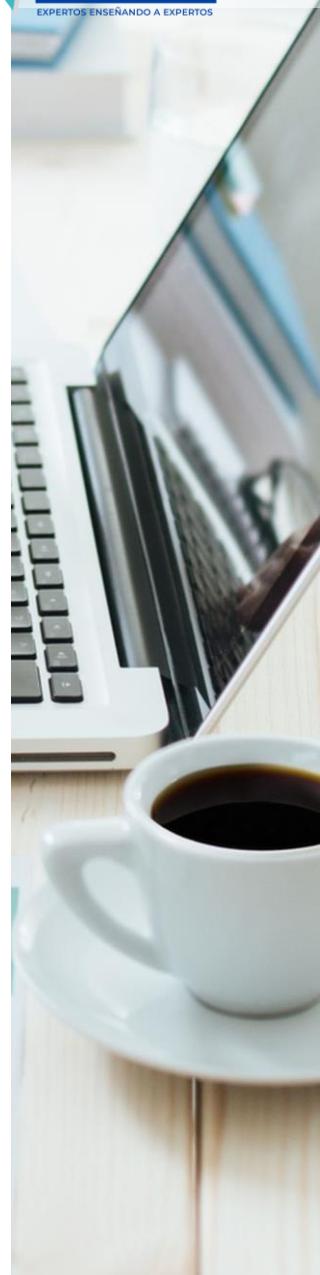
8. Crea una función en Python para desplegar el departamento con la cantidad máxima de empleados.
9. ¿A quién se dejó la responsabilidad de resolver el punto anterior, cliente (Python) o al servidor de la base de datos?
10. Comenta tu experiencia en la práctica.



# Práctica: Python MySQL DML

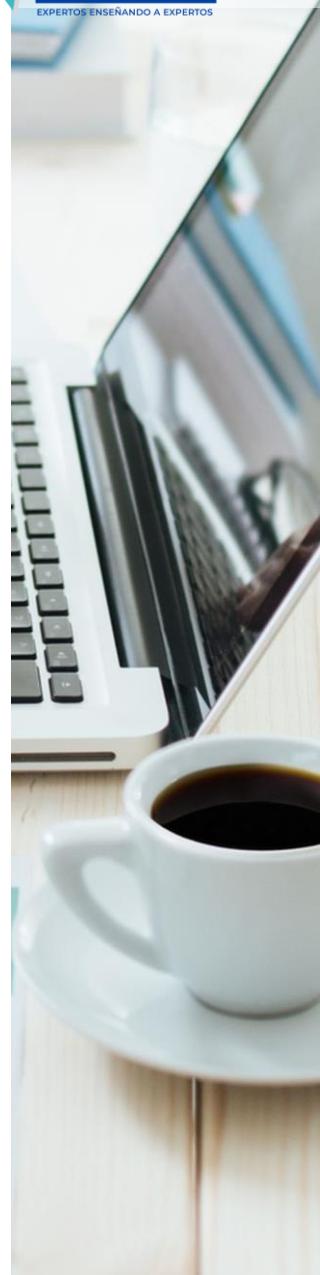
1. Conéctate a la base de datos `classicmodels` desde el cliente de MySQL.
2. Crea una copia de la tabla `employees` con las columnas de `employeeNumber`, `lastName`, `firstName` y `jobTitle`.
3. Verifica la estructura de la tabla y la cantidad de datos insertados.
4. Crea un archivo `p10_3.py`, escribe el código necesario para conectarse a la base de datos `classicmodels`.
5. Crea una función para insertar los siguientes registros:

10000,Garcilaso, de la Vega, Escritor  
10001,Tirso, de Molina, Escritor  
10002,Juan, Ruiz de Alarcón, Escritor  
10003,Miguel, de Cervantes, Escritor



# Práctica: Python MySQL DML

8. Verifica que la información fue agregada a la base de datos.
9. Actualiza el registro 10001, con los datos siguientes:
10. 10001, Federico, García Lorca
11. Comprueba los cambios directamente en la base de datos.



## Referencias Bibliográficas

- World's Most Popular Open Source Database  
<https://www.oracle.com/mysql/>
- DB-Engines Ranking  
<https://www.oracle.com/mysql/>
- MySQL  
<https://dev.mysql.com/>



## Referencias Bibliográficas

- MySQL 8.0 Reference Manual  
<https://dev.mysql.com/doc/refman/8.0/en/>
- MySQL Documentation  
<https://dev.mysql.com/doc/>
- MySQL  
<https://www.mysql.com/>
- MySQL Connector/Python Developer Guide  
<https://dev.mysql.com/doc/connector-python/en/>



## Referencias Bibliográficas

- Forks de MySQL

<https://web.archive.org/web/20120618194905/http://investigacionit.com.ar/forks-de-mysql/>

- PEP 0249 – Python Database API Specification v2.0

<https://www.python.org/dev/peps/pep-0249/>

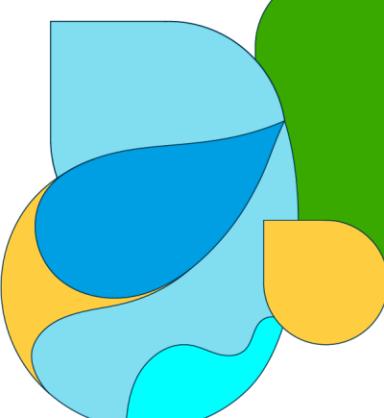
- PyMySQL

<https://github.com/PyMySQL/PyMySQL>

## 11.2 Archivo CSV

### Objetivos:

- Realizar operaciones de lectura en archivos CSV.
- Realizar operaciones de escritura en archivos CSV.



# Lectura de CSV

```
import csv

Lecturas: reader()

with open('../csvs/us-population-2010-2014.csv', newline='') as archivo:
 lineas = csv.reader(archivo)
 for i, linea in enumerate(lineas, 1):
 print(','.join(linea))
 if i >= 5:
 break
print('*'*70)
```

# Lectura de CSV

```
C:\> Administrator: Command Prompt
D:\MaterialPythonDeveloper\codigos\modulo11>python csv_examples.py
Lecturas: reader()
SEX, AGE, POPESTIMATE2010, POPESTIMATE2011, POPESTIMATE2012, POPESTIMATE2013, POPESTIMATE2014
A, 0, 3,951,330, 3,963,071, 3,926,665, 3,945,610, 3,948,350
A, 1, 3,957,888, 3,966,510, 3,978,006, 3,943,077, 3,962,123
A, 2, 4,090,862, 3,971,573, 3,979,952, 3,992,690, 3,957,772
A, 3, 4,111,920, 4,102,501, 3,983,049, 3,992,425, 4,005,190

D:\MaterialPythonDeveloper\codigos\modulo11>
```

# DictReader

```
Lecturas con DictReader()

mapa = {'A':'Both', 'M':'Male', 'F':'Female'} # Diccionario

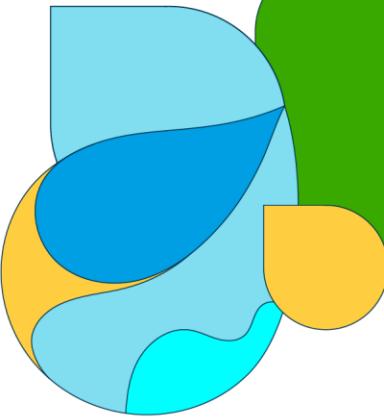
with open('../csvs/us-population-2010-2014.csv', newline='') as csvfile:

 lineas = csv.DictReader(csvfile)
 encabezado = ','.join(lineas.fieldnames) # fieldnames

 print(encabezado)
 print('-' * len(encabezado))

 for linea in lineas:
 sex = mapa[linea['SEX']]
 print('sex,
 linea['AGE'],
 linea['POPESTIMATE2010'],
 linea['POPESTIMATE2011'],
 linea['POPESTIMATE2012'],
 linea['POPESTIMATE2013'],
 linea['POPESTIMATE2014'])

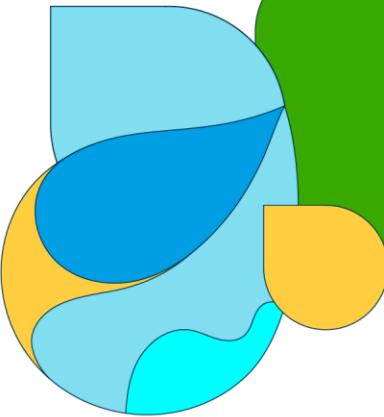
 print('-'*70)
```



# DictReader

```
Administrator: Command Prompt
D:\MaterialPythonDeveloper\codigos\modulo11>python Demo_DictReader.py
SEX, AGE, POPESTIMATE2010, POPESTIMATE2011, POPESTIMATE2012, POPESTIMATE2013, POPESTIMATE2014
A, 0, 3,951,330, 3,963,071, 3,926,665, 3,945,610, 3,948,350
A, 1, 3,957,888, 3,966,510, 3,978,006, 3,943,077, 3,962,123
A, 2, 4,090,862, 3,971,573, 3,979,952, 3,992,690, 3,957,772
A, 3, 4,111,920, 4,102,501, 3,983,049, 3,992,425, 4,005,190

D:\MaterialPythonDeveloper\codigos\modulo11>
```



# DictReader

```
import csv

Lecturas con DictReader()
mapa = {'A':'Both', 'M':'Male', 'F':'Female'} # Diccionario
tt = ['SEX', 'AGE', '2010', '2011', '2012', '2013', '2014']

with open('../csvs/us-population-2010-2014_ver2.csv', newline='') as csvfile:

 lineas = csv.DictReader(csvfile, tt)
 print(tt)
 print('-' * 70)
 i= 0
 for linea in lineas:
 sex = mapa [linea['SEX']]
 print(sex,
 linea['AGE'],
 linea['2010'],
 linea['2011'],
 linea['2012'],
 linea['2013'],
 linea['2014'])
 if i>8:
 break
 i+=1
 print('-'*70)
```

# DictReader

```
Lecturas con DictReader()

mapa = {'A':'Both', 'M':'Male', 'F':'Female'} # Diccionario

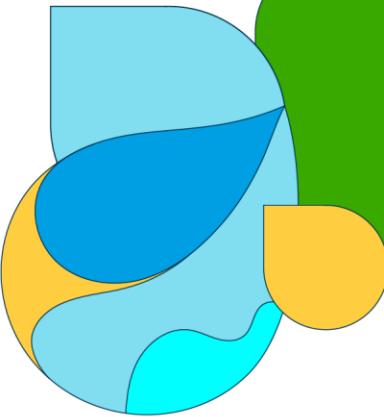
with open('../csvs/us-population-2010-2014.csv', newline='') as csvfile:

 lineas = csv.DictReader(csvfile)
 encabezado = ','.join(lineas.fieldnames) # fieldnames

 print(encabezado)
 print('-' * len(encabezado))

 for linea in lineas:
 sex = mapa[linea['SEX']]
 print('sex,
 linea['AGE'],
 linea['POPESTIMATE2010'],
 linea['POPESTIMATE2011'],
 linea['POPESTIMATE2012'],
 linea['POPESTIMATE2013'],
 linea['POPESTIMATE2014'])

print('*70)
```



# DictReader

```
Administrator: Command Prompt
D:\MaterialPythonDeveloper\codigos\modulo11>python Demo_DictReader.py
SEX, AGE, POPESTIMATE2010, POPESTIMATE2011, POPESTIMATE2012, POPESTIMATE2013, POPESTIMATE2014
A, 0, 3,951,330, 3,963,071, 3,926,665, 3,945,610, 3,948,350
A, 1, 3,957,888, 3,966,510, 3,978,006, 3,943,077, 3,962,123
A, 2, 4,090,862, 3,971,573, 3,979,952, 3,992,690, 3,957,772
A, 3, 4,111,920, 4,102,501, 3,983,049, 3,992,425, 4,005,190

D:\MaterialPythonDeveloper\codigos\modulo11>
```

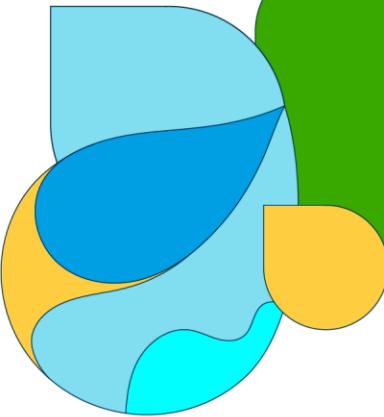
# DictReader

```
import csv

Lecturas con DictReader()
mapa = {'A':'Both', 'M':'Male', 'F':'Female'} # Diccionario
tt = ['SEX', 'AGE', '2010', '2011', '2012', '2013', '2014']

with open('../csvs/us-population-2010-2014_ver2.csv', newline='') as csvfile:

 lineas = csv.DictReader(csvfile, tt)
 print(tt)
 print('-' * 70)
 i= 0
 for linea in lineas:
 sex = mapa [linea['SEX']]
 print(sex,
 linea['AGE'],
 linea['2010'],
 linea['2011'],
 linea['2012'],
 linea['2013'],
 linea['2014'])
 if i>8:
 break
 i+=1
 print('-'*70)
```



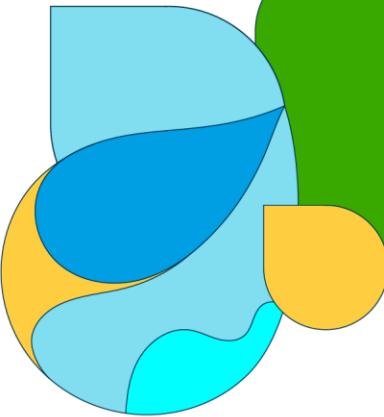
# Búsqueda de Información

```
import csv

Búsqueda de datos en el archivo CSV
with open('../csvs/us-population-2010-2014.csv', newline='') as cvs_archivo:
 lineas = csv.DictReader(cvs_archivo)

 for linea in lineas:
 if (linea['AGE'] == '30' and linea['SEX'] == 'F'):
 popu = linea['POPESTIMATE2011']
 break
 else:
 popu = None

print(popu)
print('*'*50)
```



# Búsqueda de Información

```
función reutilizable
def buscar(lineas, age, sex, year):
 for linea in lineas:
 if (linea['AGE'] == str(age) and linea['SEX'] == sex):
 return linea['POPESTIMATE' + str(year)]
 return None

print('Uso de la función buscar(): ')

with open('../csvs/us-population-2010-2014.csv', newline='') as csvfile:
 lineas = csv.DictReader(csvfile)
 pop1 = buscar(lineas, 30, 'F', 2011)
 pop2 = buscar(lineas, 30, 'F', 2011)

print(pop1, pop2)
print('*'*50)
```

# Creación de Archivos CSV

```
import pymysql, csv

connection = pymysql.connect(host='localhost',
 user='root',
 password='panchol123',
 database='classicmodels'
)

query = '''SELECT productCode code, avg(buyPrice) precio
FROM products WHERE productName is NOT NULL GROUP BY productLine ORDER BY productLine'''

with connection.cursor() as cursor:
 cursor.execute(query)
 results = cursor.fetchall()

connection.close()

with open('../csvs/salida.csv', 'w', newline='') as csvfile:
 writer = csv.writer(csvfile)
 writer.writerow(['Code', 'Price'])
 writer.writerows(results)

print('*'*70)
```



# DictWriter

```
import pymysql, csv

connection = pymysql.connect(host='localhost',
 user='root',
 password='pancho123',
 database='classicmodels'
)

query = '''SELECT productCode code, avg(buyPrice) precio
FROM products WHERE productName is NOT NULL GROUP BY productLine ORDER BY productLine'''

with connection.cursor() as cursor:
 cursor.execute(query)
 results = cursor.fetchall()

connection.close()

with open('../csvs/salida2.csv', 'w', newline='') as csvfile:
 fieldnames= ['Code','Promedio'] # fieldnames
 writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
 writer.writeheader()
 writer.writerows(results)

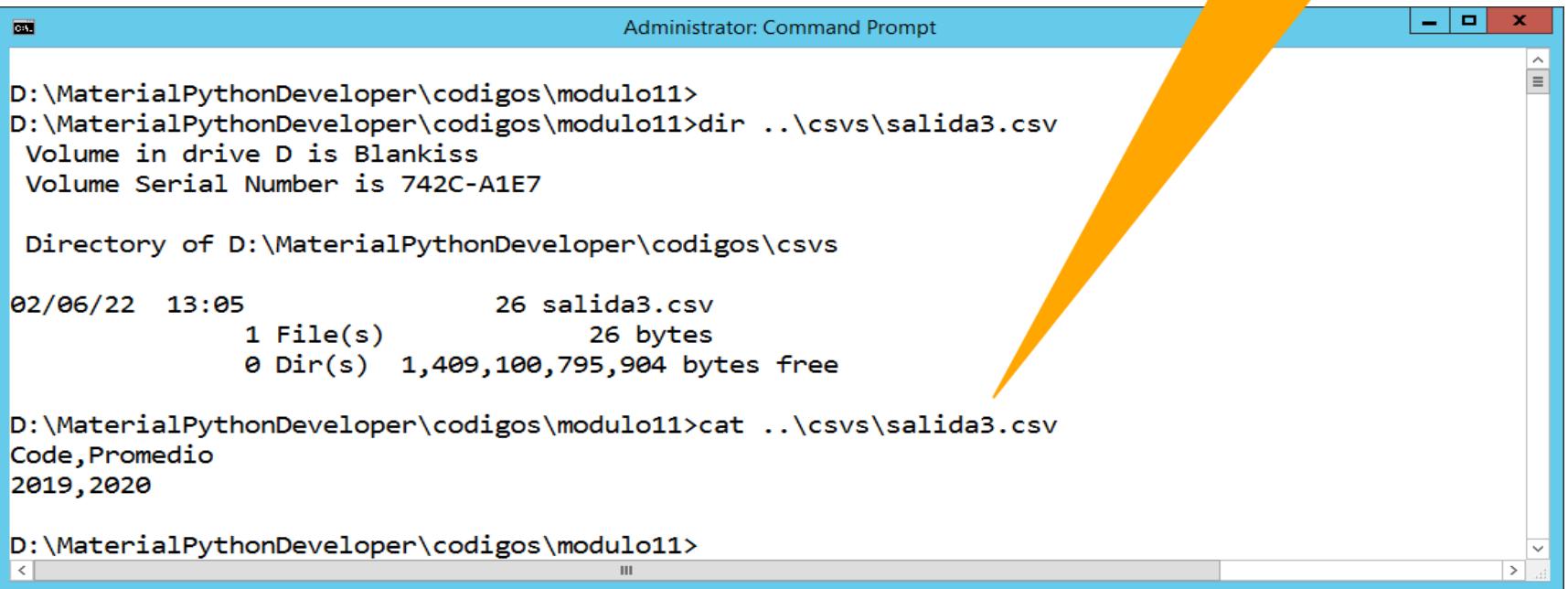
print('*'*50)
```

Escribe los encabezado con los valores de fieldnames

# DictWriter

```
with open('../csvs/salida3.csv', 'a', newline='') as csvfile:
 fieldnames= ['Code','Promedio'] # fieldnames
 writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
 writer.writeheader()
 writer.writerow({'Code":2019,"Promedio":2020})
```

Diccionario -> filas



```
Administrator: Command Prompt

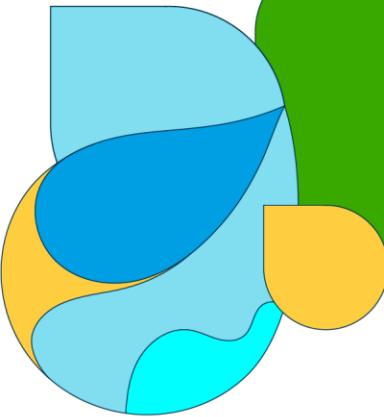
D:\MaterialPythonDeveloper\codigos\modulo11>
D:\MaterialPythonDeveloper\codigos\modulo11>dir ..\csvs\salida3.csv
Volume in drive D is Blankiss
Volume Serial Number is 742C-A1E7

Directory of D:\MaterialPythonDeveloper\codigos\csvs

02/06/22 13:05 26 salida3.csv
 1 File(s) 26 bytes
 0 Dir(s) 1,409,100,795,904 bytes free

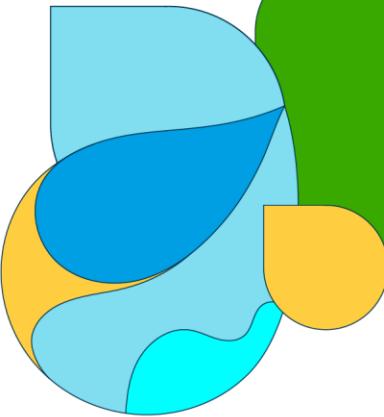
D:\MaterialPythonDeveloper\codigos\modulo11>cat ..\csvs\salida3.csv
Code,Promedio
2019,2020

D:\MaterialPythonDeveloper\codigos\modulo11>
```



# Dialectos CSV

- El dialecto predeterminado en el módulo CSV es 'Excel'.
- Otras opciones reconocidas por el módulo son:
  - 'excel-tab'
  - 'unix'



# Python CSV

`csv.reader`

`csv.writer`

`csv.register_dialect`

`csv.get_dialect`

`csv.list_dialect`

`csv.unregister_dialect`

`csv.field_size_limit`

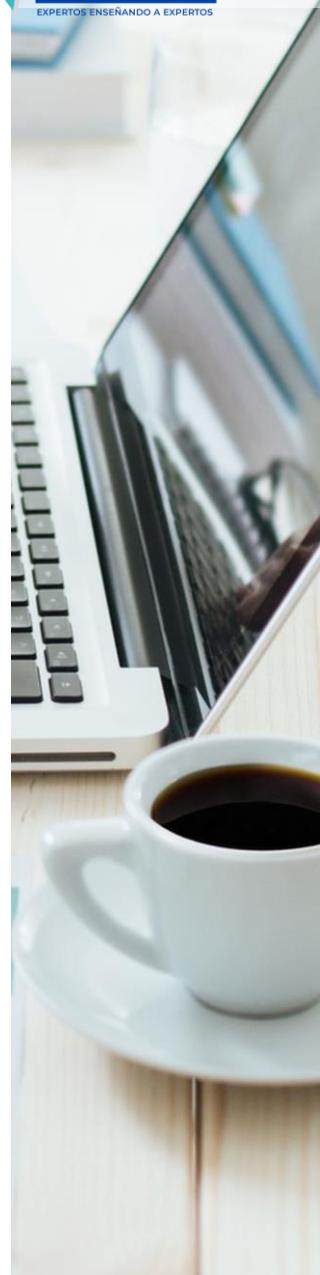
# Resumen del capítulo

- En el capítulo se ha aprendido a trabajar con información almacenada en archivos CSV.
- La mayoría de las tareas de lectura, procesamiento y escritura CSV pueden ser gestionadas fácilmente con el módulo básico CSV de Python.
- Para grandes volúmenes de datos considera usar Pandas.
- Existen otras formas de analizar archivos de texto, los módulos ANTLR, PLY y PlyPlus, pueden manejar análisis pesado y si la manipulación simple no funciona, siempre estará el módulo re para expresiones regulares.

# Práctica

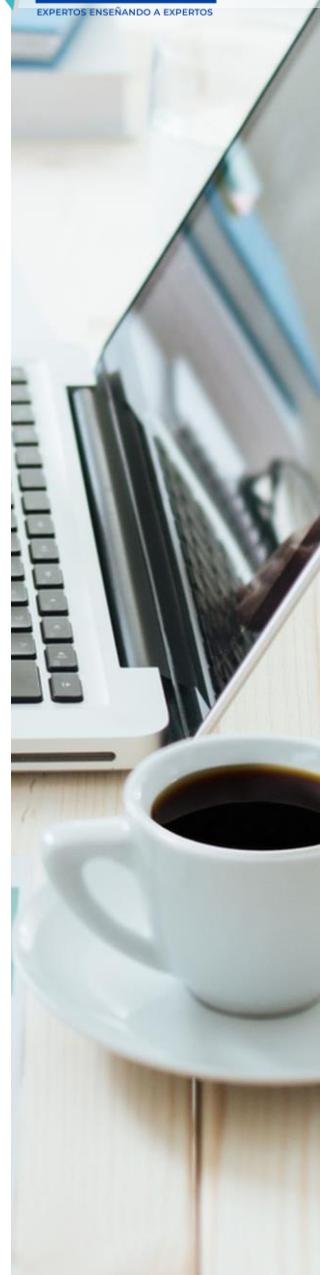
En esta práctica, se utilizará una función auxiliar para comparar datos en un archivo CSV.

1. Abre el archivo [info.csv](#) y estúdialo.
2. Utilizarás este archivo para encontrar la diferencia de población entre hombres y mujeres en el año 2011.
3. Crea un archivo con el nombre de [p11\\_1.py](#) con el contenido de la imagen de la siguiente página.



# Práctica

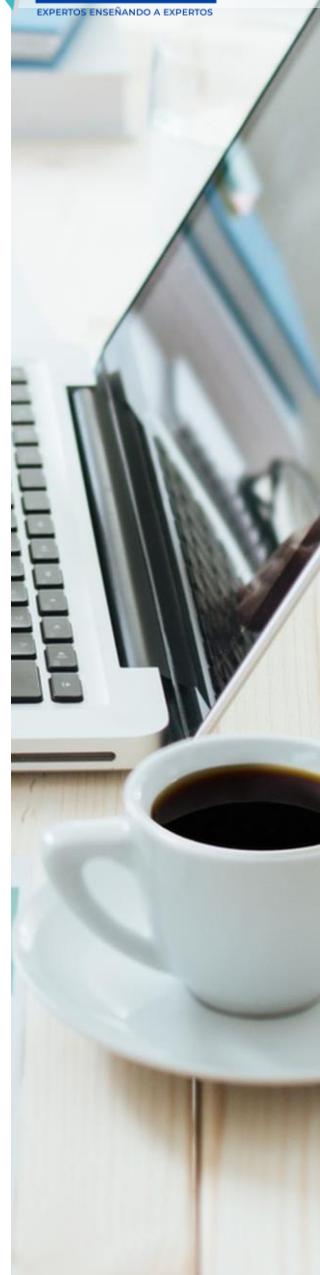
4. Estudia la función `comparar()`. Esta calcula la diferencia entre dos poblaciones y el porcentaje de la diferencia.
5. Crea dos tuplas: `edad_gen_año`, una para mujeres de 30 años en 2011 y la otra para hombres de 30 años en 2010.
6. Llama a la función `comparar()`, pasando los datos apropiados.
7. `(10353, 1.004910053492218)`
8. Imprime el resultado.



# Práctica

```
import csv

def compar(lineas, edad_gen_año1, edad_gen_año2):
 pop1, pop2 = -1, -1
 for linea in lineas:
 if (linea['AGE'] == str(edad_gen_año1[0]) and linea['SEX'] == edad_gen_año1[1]):
 pop1 = linea['POPESTIMATE' + str(edad_gen_año1[2])]
 pop1 = int(pop1.replace(',', '')) # , ->
 if (linea['AGE'] == str(edad_gen_año2[0]) and linea['SEX'] == edad_gen_año2[1]):
 pop2 = linea['POPESTIMATE' + str(edad_gen_año2[2])]
 pop2 = int(pop2.replace(',', '')) # , ->
 if pop1 > 0 and pop2 > 0:
 return (pop2 - pop1, pop2/pop1)
```





## Referencias Bibliográficas

- CSV File Reading and Writing.

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

- ANTLR

<https://www.antlr.org/>

- PLY Python Lex-Yacc

<http://www.dabeaz.com/ply/>

- PlyPlus

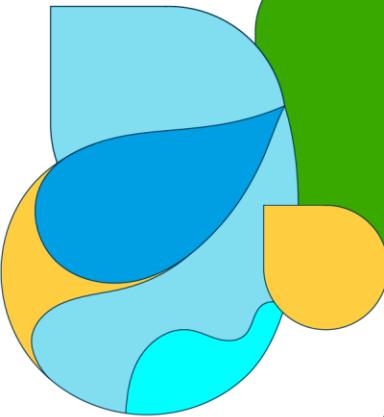
<https://pypi.org/project/PlyPlus/>

# Unidad temática 12

Librerías para Ciencia de Datos: NumPy

## Objetivos:

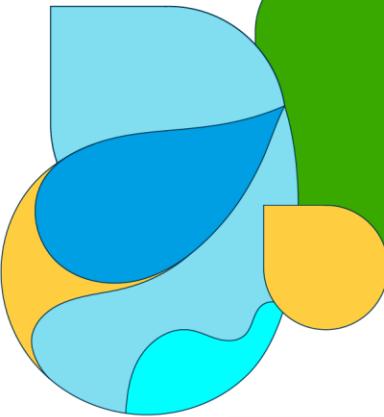
- Generar Arrays multidimensionales.
- Obtener Arrays de fuentes alternas como son APIs, archivos de texto plano, archivos csv y excel.
- Modificar y obtener nuevos arrays de uno específico adaptado a las necesidades requeridas.



# Introducción Numpy

## ¿QUÉ ES NUMPY?

NumPy es el paquete fundamental para la computación científica en Python. Es una biblioteca de Python que proporciona un objeto de matriz multidimensional, varios objetos derivados (como matrices y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en arreglos, incluidas matemáticas, lógica, manipulación de formas, clasificación, selección, E/S, transformada discreta de Fourier, álgebra lineal básica, operaciones estadísticas básicas, simulación aleatoria y mucho más.



# Introducción Numpy

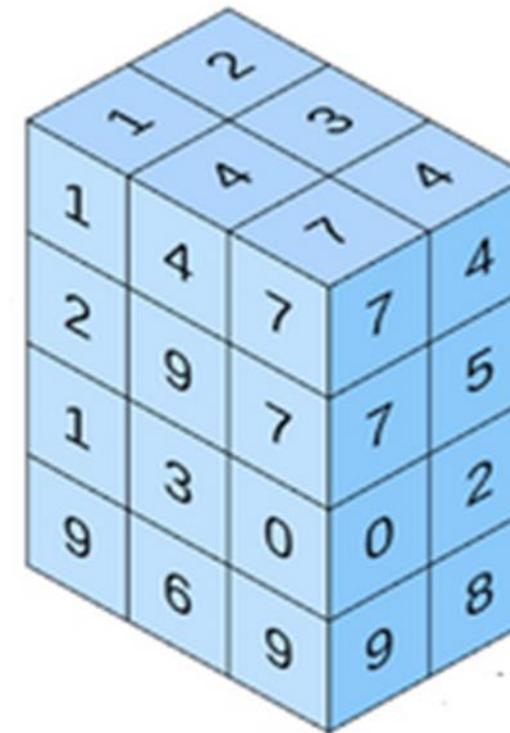
Array 1D

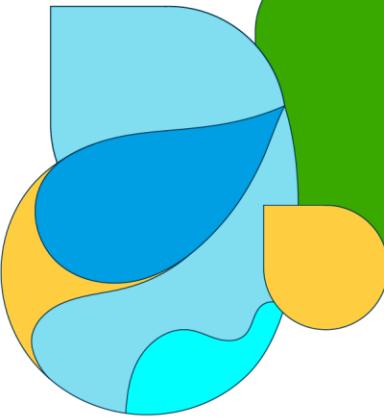
|   |   |   |    |
|---|---|---|----|
| 7 | 2 | 9 | 10 |
|---|---|---|----|

Array 2D

|     |     |     |
|-----|-----|-----|
| 5.2 | 3.0 | 4.5 |
| 9.1 | 0.1 | 0.3 |

Array 3D





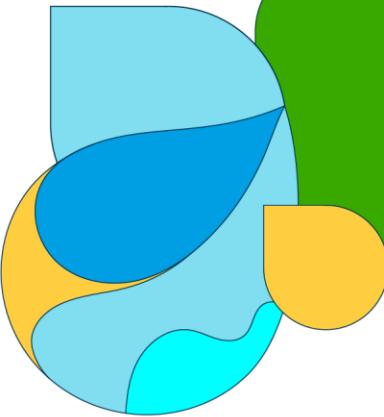
# Introducción Numpy

## Beneficios:

- Más velocidad.
- Menos Bucles.
- Código más claro.
- Mejor calidad.

## Donde se utiliza NumPy:

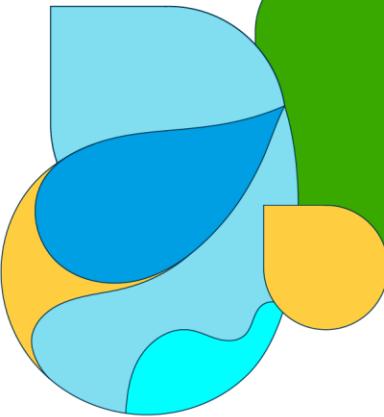
- Matemáticas (MATLAB).
- Machine Learning.
- Ciencia de datos.
- Estadística.



# Creación de arrays

**Para crear un array se utiliza la siguiente función de NumPy.**

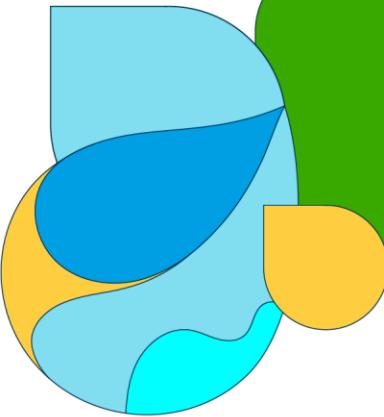
- `np.array(lista)` : Crea un array a partir de la lista o tupla lista y devuelve una referencia a él. El número de dimensiones del array dependerá de las listas o tuplas anidadas en lista:
- Para una lista de valores se crea un array de una dimensión, también conocido como vector.
- Para una lista de listas de valores se crea un array de dos dimensiones, también conocido como matriz.
- Para una lista de listas de listas de valores se crea un array de tres dimensiones, también conocido como cubo.
- Y así sucesivamente. No hay límite en el número de dimensiones del array más allá de la memoria disponible en el sistema.



# Creación de arrays

## Ejemplos de creación de arrays

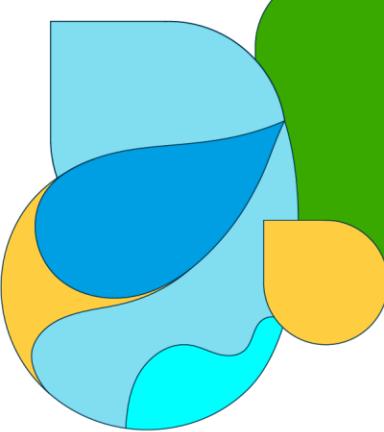
```
>>> import numpy as np
>>> # Array de una dimensión
>>> a1 = np.array([1, 2, 3])
>>> print(a1)
[1 2 3]
>>> # Array de dos dimensiones
>>> a2 = np.array([[1, 2, 3], [4, 5, 6]])
>>> print(a2)
[[1 2 3]
 [4 5 6]]
>>> # Array de tres dimensiones
>>> a3 = np.array([[[1, 2, 3], [4, 5, 6]], [[7, 8, 9], [10, 11, 12]]])
>>> print(a3)
[[[1 2 3]
 [4 5 6]]
 [[7 8 9]
 [10 11 12]]]
```



## Copiado y edición

NumPy hace uso de un concepto llamado "Referencia de arreglos" (array referencing) que es un concepto muy confuso para personas que son nuevas en la librería.

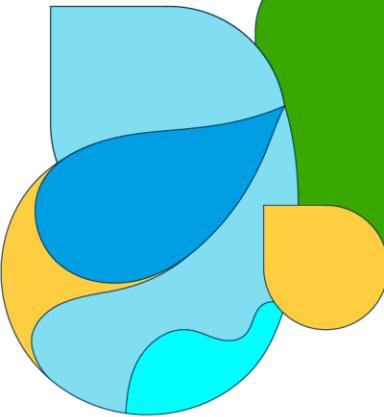
Veamos ejemplos en la siguiente lámina:



# Copiado y edición

## Algunos ejemplos

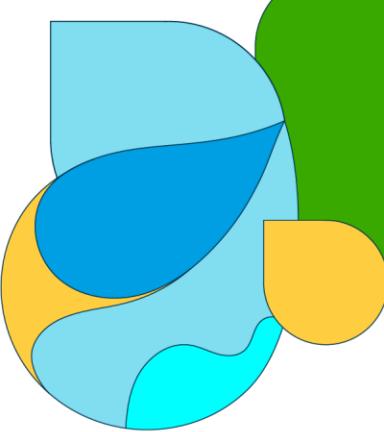
```
>>> a = np.array([6, 7, 8, 9])
>>> a
array([6, 7, 8, 9])
>>> b = a[0:2]
>>> b
array([6, 7])
>>> b[1] = 4
>>> b
array([6, 4])
>>> a
array([6, 4, 8, 9])
```



## Copiado y edición

Por defecto, NumPy no crea una copia de un arreglo cuando hace referencia a la variable del arreglo original usando el operador de asignación =.

En cambio, simplemente apunta la nueva variable a la anterior, lo que permite que la segunda variable realice modificaciones en la variable original, incluso si esta no es tu intención.



# Copiado y edición

```
>>> a = np.array([1, 2, 3])

>>> a

array([1, 2, 3])

>>> copia_a = a.copy()

>>> copia_a

array([1, 2, 3])

>>> copia_a[0] = 9

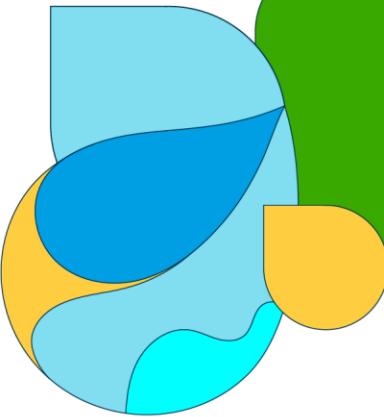
>>> copia_a

array([9, 2, 3])

>>> a

array([1, 2, 3])

>>>
```



# Apilamiento y reestructuración

Hay veces que requerimos combinar dos matrices (arrays en general).

Uno de los mecanismos que nos proporciona NumPy es el apilado.

## Apilado vertical:

```
>>> a = np.random.randint(1, 100, size=(3, 2))
>>> b = np.random.randint(1, 100, size=(1, 2))
>>> a
array([[93, 57],
 [34, 1],
 [46, 8]])
>>> b
array([[47, 3]])
>>> np.vstack((a, b))
array([[93, 57],
 [34, 1],
 [46, 8],
 [47, 3]])
```

# Apilado horizontal:

```
>>> c = np.random.randint(1, 100, size=(3, 2))
>>> d = np.random.randint(1, 100, size=(3, 1))
>>> c
array([[39, 11],
 [88, 1],
 [42, 1]])
>>> d
array([[66],
 [82],
 [48]])
>>> np.hstack((c, d))
array([[39, 11, 66],
 [88, 1, 82],
 [42, 1, 48]])
```

# Repetición de elementos por ejes:

**El parámetro de repetición indica el número de veces que repetimos el array completo por cada eje:**

```
>>> e = np.array([[1, 2],
... [3, 4],
... [5, 6]])
>>> e
array([[1, 2],
 [3, 4],
 [5, 6]])
>>> np.tile(e, 3) # x3 en columnas
array([[1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4],
 [5, 6, 5, 6, 5, 6]])
>>> np.tile(e, (2, 3)) # x2 en filas; x3 en columnas
array([[1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4],
 [5, 6, 5, 6, 5, 6],
 [1, 2, 1, 2, 1, 2],
 [3, 4, 3, 4, 3, 4],
 [5, 6, 5, 6, 5, 6]])
```

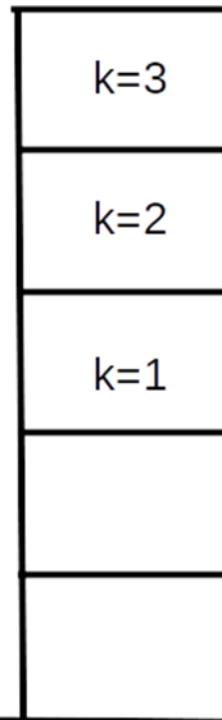
# Repetición por elementos:

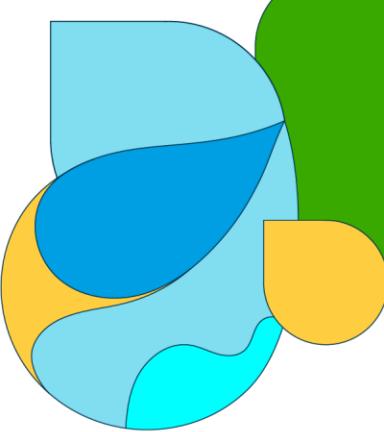
**El parámetro de repetición indica el número de veces que repetimos cada elemento del array:**

```
>>> f = np.array([[1, 2], [3, 4], [5, 6]])
>>> f
array([[1, 2],
 [3, 4],
 [5, 6]])
>>> np.repeat(f, 2)
array([1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6])
>>> np.repeat(f, 2, axis=0)
array([[1, 2],
 [1, 2],
 [3, 4],
 [3, 4],
 [5, 6],
 [5, 6]])
>>> np.repeat(f, 3, axis=1) # x3 en columnas
array([[1, 1, 1, 2, 2, 2],
 [3, 3, 3, 4, 4, 4],
 [5, 5, 5, 6, 6, 6]])
```

# Acceso por diagonal

|    |    |    |    |
|----|----|----|----|
| 73 | 86 | 90 | 20 |
| 96 | 55 | 15 | 48 |
| 38 | 63 | 96 | 48 |
| 13 | 87 | 32 | 96 |



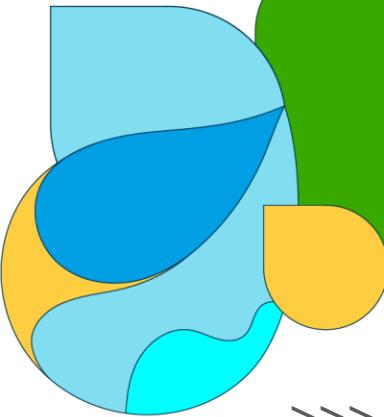


# Acceso por diagonal

```
>>> g = np.array([[73, 86, 90, 20], [96, 55, 15, 48], [38, 63, 96, 95], [13, 87, 32, 96]])
>>> g
array([[73, 86, 90, 20],
 [96, 55, 15, 48],
 [38, 63, 96, 95],
 [13, 87, 32, 96]])
>>> np.diag(g)
array([73, 55, 96, 96]) # k = 0

>>> for k in range(1, g.shape[0]):
... print(f'k={k}', np.diag(g, k=k))
...
k=1 [86 15 95]
k=2 [90 48]
k=3 [20]

>>> for k in range(1, g.shape[0]):
... print(f'k={-k}', np.diag(g, k=-k))
...
k=-1 [96 63 32]
k=-2 [38 87]
k=-3 [13]
```



# Modificación de elementos por diagonal

```
>>> h = np.array([[73, 86, 90, 20], [96, 55, 15, 48], [38, 63, 96, 95], [13, 87, 32, 96]])
>>> di = np.diag_indices(h.shape[0])
>>> di
(array([0, 1, 2, 3]), array([0, 1, 2, 3]))
>>> h[di] = 0
>>> h
array([[0, 86, 90, 20],
 [96, 0, 15, 48],
 [38, 63, 0, 95],
 [13, 87, 32, 0]])
```

# Comparando arrays

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

>>> b = np.array([[1, 2], [3, 4]])

>>>

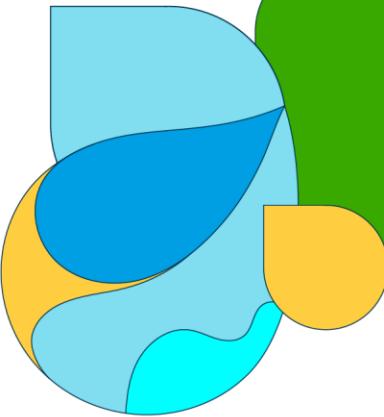
>>> a == b

array([[True, True],
 [True, True]])
```

Para comparar arrays en su totalidad, podemos hacer uso de la siguiente función:

```
>>> np.array_equal(a, b)

True
```



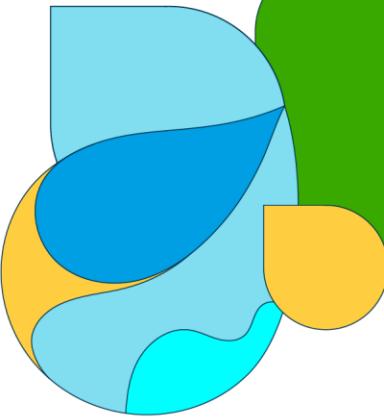
# Unión de arrays

```
>>> x = np.array([9, 4, 11, 3, 14, 5, 13, 12, 7, 14])

>>> y = np.array([17, 9, 19, 4, 18, 4, 7, 13, 11, 10])

>>> np.union1d(x, y)

array([3, 4, 5, 7, 9, 10, 11, 12, 13, 14, 17, 18, 19])
```



# Intersección de arrays

$x \cap y$

```
>>> x = np.array([9, 4, 11, 3, 14, 5, 13, 12, 7, 14])

>>> y = np.array([17, 9, 19, 4, 18, 4, 7, 13, 11, 10])

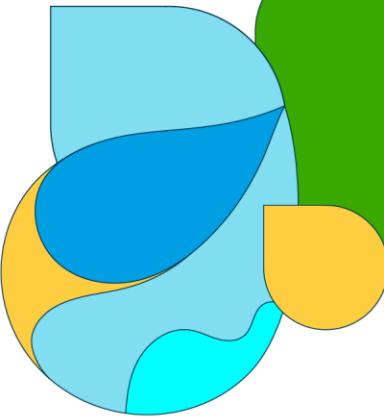
>>> np.intersect1d(x, y)

array([4, 7, 9, 11, 13])
```

# Diferencia de arrays

x\y

```
>>> x = np.array([9, 4, 11, 3, 14, 5, 13, 12, 7, 14])
>>> y = np.array([17, 9, 19, 4, 18, 4, 7, 13, 11, 10])
>>> np.setdiff1d(x, y)
array([3, 5, 12, 14])
```



# Ordenación sobre arrays unidimensionales

```
>>> a = np.array([23, 24, 92, 88, 75, 68, 12, 91, 94, 24, 9, 21, 42, 3, 66])

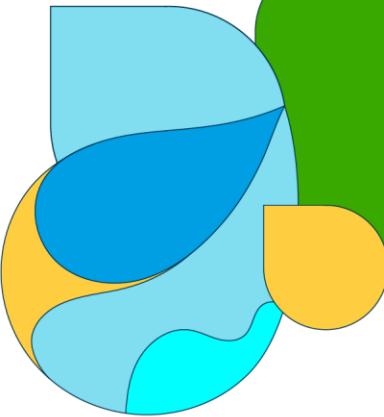
>>> np.sort(a)

array([3, 9, 12, 21, 23, 24, 24, 42, 66, 68, 75, 88, 91, 92, 94])

>>> a.sort()

>>> a

array([3, 9, 12, 21, 23, 24, 24, 42, 66, 68, 75, 88, 91, 92, 94])
```

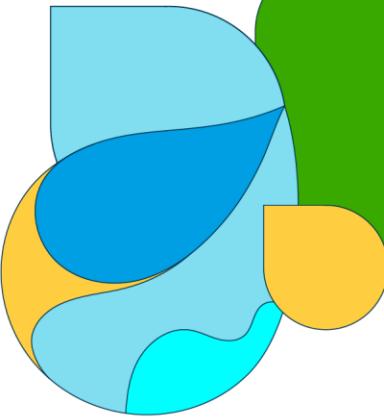


# Ordenación sobre arrays multidimensionales

```
>>> b = np.array([[52, 23, 90, 46],
... [61, 63, 74, 59],
... [75, 5, 58, 70],
... [21, 7, 80, 52]])

>>> np.sort(b, axis=1)
array([[23, 46, 52, 90],
 [59, 61, 63, 74],
 [5, 58, 70, 75],
 [7, 21, 52, 80]])

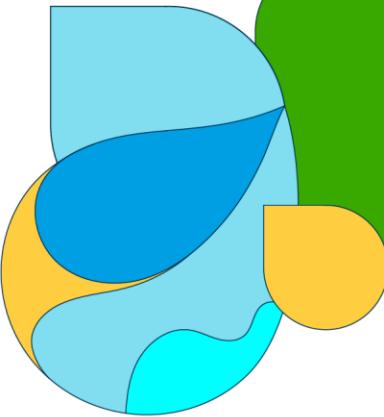
>>> np.sort(b, axis=0)
array([[21, 5, 58, 46],
 [52, 7, 74, 52],
 [61, 23, 80, 59],
 [75, 63, 90, 70]])
```



## Suma con array «fila»:

```
>>> a = np.array([[9, 8, 1],[7, 6, 7]])
>>> b = np.array([[2, 3, 6]])
>>> a + b # broadcasting
array([[11, 11, 7],
 [9, 9, 13]])
```

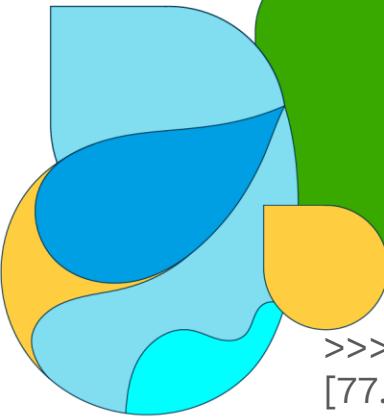
```
>>> c = np.array([[9, 8, 1],
... [7, 6, 7]])
>>> d = np.array([[1],
... [6]])
>>> c + d # broadcasting
array([[10, 9, 2],
 [13, 12, 13]])
```



# Operación entre arrays escalares

```
>>> e = np.array([[9, 8, 1],
... [7, 6, 7]])
>>> e + 5
array([[14, 13, 6],
 [12, 11, 12]])
>>> e - 5
array([[4, 3, -4],
 [2, 1, 2]])
>>> e * 5
array([[45, 40, 5],
 [35, 30, 35]])

>>> e / 5
array([[1.8, 1.6, 0.2],
 [1.4, 1.2, 1.4]])
>>> e // 5
array([[1, 1, 0],
 [1, 1, 1]])
>>> e ** 5
array([[59049, 32768, 1],
 [16807, 7776, 16807]])
```



# Funciones universales

```
>>> f = np.array([[48.32172375, 24.89651106, 77.49724241],
[77.81874191, 22.54051494, 65.11282444], [5.54960482,
59.06720303, 62.52817198]])

>>> f

array([[48.32172375, 24.89651106, 77.49724241],
[77.81874191, 22.54051494, 65.11282444],
[5.54960482, 59.06720303, 62.52817198]])

>>> np.sqrt(f)

array([[6.95138287, 4.98964037, 8.80325181],
[8.82149318, 4.74768522, 8.06925179],
[2.35575992, 7.68551905, 7.9074757]])

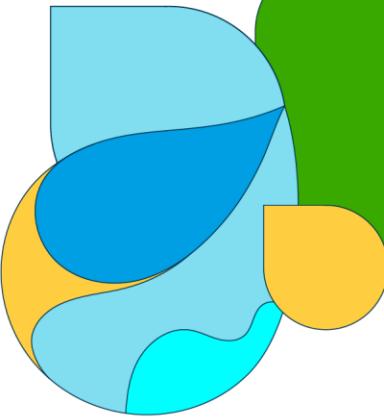
>>> np.sin(f)

array([-0.93125201, -0.23403917, 0.86370434],
[0.66019205, -0.52214693, 0.75824777],
[-0.66953344, 0.58352078, -0.29903488]])
```

```
>>> np.ceil(f)
array([[49., 25., 78.],
[78., 23., 66.],
[6., 60., 63.]])

>>> np.floor(f)
array([[48., 24., 77.],
[77., 22., 65.],
[5., 59., 62.]])

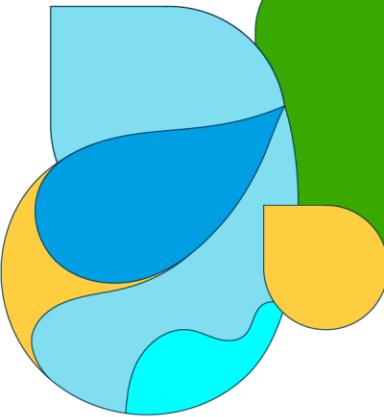
>>> np.log(f)
array([[3.87788123, 3.21472768, 4.35024235],
[4.3543823 , 3.11531435, 4.17612153],
[1.71372672, 4.07867583, 4.13561721]])
```



# Reduciendo resultados

```
>>> g = np.array([[8, 2, 7], [2, 0, 6], [6, 3, 4]])
>>> g
array([[8, 2, 7],
 [2, 0, 6],
 [6, 3, 4]])
>>> np.sum(g, axis=0)
array([16, 5, 17])
```

```
>>> np.sum(g, axis=1)
array([17, 8, 13])
>>> np.prod(g, axis=0)
array([96, 0, 168])
>>> np.prod(g, axis=1)
array([112, 0, 72])
```



# Funciones estadísticas

```
>>> h = np.array([[-6.79006504, -0.01579498, -0.29182173, 0.3298951 , -5.30598975], [3.10720923, -4.09625791, -7.60624152, 2.3454259 , 9.23399023], [-7.4394269 , -9.68427195, 3.04248586, -5.9843767 , 1.536578], [3.33953286, -8.41584411, -9.530274 , -2.42827813, -7.34843663], [7.1508544 , 5.51727548, -3.20216834, -5.00154367, -7.15715252]])

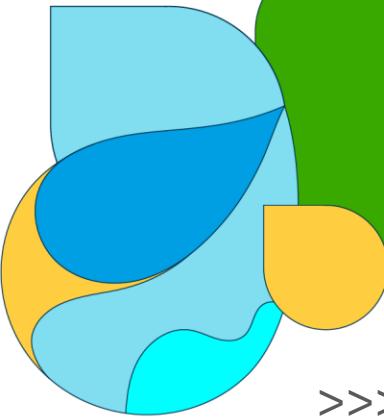
>>> h

array([[-6.79006504, -0.01579498, -0.29182173, 0.3298951 , -5.30598975],
 [3.10720923, -4.09625791, -7.60624152, 2.3454259 , 9.23399023],
 [-7.4394269 , -9.68427195, 3.04248586, -5.9843767 , 1.536578],
 [3.33953286, -8.41584411, -9.530274 , -2.42827813, -7.34843663],
 [7.1508544 , 5.51727548, -3.20216834, -5.00154367, -7.15715252]])

>>> np.mean(h)
-2.1877878728

>>> np.std(h)
5.393254993673597

>>> np.median(h)
-3.20216834
```



# Máximos y Mínimos

```
>>> i = np.array([[66, 54, 33, 15, 58], [55, 46, 39, 16, 38], [73, 75, 79, 25, 83], [81, 30, 22, 32, 8], [92, 25, 82, 10, 90]])
>>> i
array([[66, 54, 33, 15, 58],
 [55, 46, 39, 16, 38],
 [73, 75, 79, 25, 83],
 [81, 30, 22, 32, 8],
 [92, 25, 82, 10, 90]])
>>> np.min(i)
8
>>> np.max(i)
92
>>> np.max(i, axis=0)
array([92, 75, 82, 32, 90])
>>> np.max(i, axis=1)
array([66, 55, 83, 81, 92])
```

# Resumen del capítulo

- NumPy es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.
- Incorpora una nueva clase de objetos llamados arrays que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.
- La ventaja de Numpy frente a las listas predefinidas en Python es que el procesamiento de los arrays se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.

# Actividad del capítulo

## CALCULAR EL PRODUCTO EXTERNO DE DOS VECTORES USANDO NUMPY EN PYTHON

En Python, se tiene la función exterior() del paquete NumPy para encontrar el producto exterior de dos matrices.

Sintaxis: `numpy.outer(a, b, out = None)`

Parámetros:

a: [array\_like] Primer vector de entrada. La entrada se aplana si aún no es unidimensional.

b: [array\_like] Segundo vector de entrada. La entrada se aplana si aún no es unidimensional.

out: [ndarray, opcional] Una ubicación donde se almacena el resultado.

Devuelve: [ndarray] Devuelve el producto exterior de dos vectores. fuera  $[i, j] = a[i] * b[j]$



# Actividad del capítulo

Caso 1:

$$a = [6 \ 2]$$

$$b = [2 \ 5]$$

Caso 2:

$$a = [1 \ 3]$$

$$[2 \ 6]$$

$$b = [0 \ 1]$$

$$[1 \ 9]$$

Caso 3:

$$a = [2 \ 8 \ 2]$$

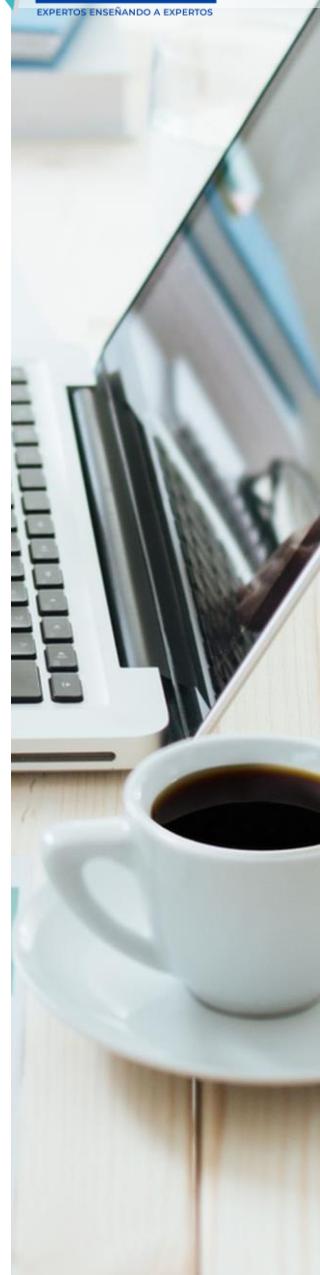
$$[3 \ 4 \ 8]$$

$$[0 \ 2 \ 1]$$

$$b = [2 \ 1 \ 1]$$

$$[0 \ 1 \ 0]$$

$$[2 \ 3 \ 0]$$



# Solución caso 1:

```
import numpy as np

arraya = np.array([6,2])

arrayb = np.array([2,5])

print("Arrays 1-D:")

print(arraya)

print(arrayb)

print("El producto exterior de las dos
matrices es:")

result = np.outer(arraya, arrayb)

print(result)
```

Arrays 1-D:

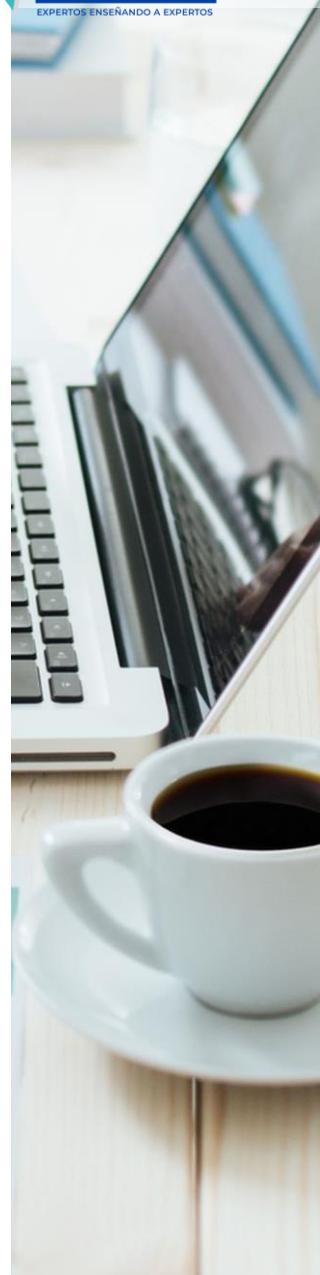
[6 2]

[2 5]

El producto exterior de las dos matrices es:

[[12 30]

[ 4 10]]



## Solución caso 2:

```
import numpy as np

matrixa = np.array([[1, 3], [2, 6]])

matrixb = np.array([[0, 1], [1, 9]])

print("Arrays 2-D:")

print(matrixa)

print(matrixb)

print("El producto exterior de las dos
matrices es:")

result = np.outer(matrixa, matrixb)

print(result)
```

Arrays 2-D:

[[1 3]

[2 6]]

[[0 1]

[1 9]]

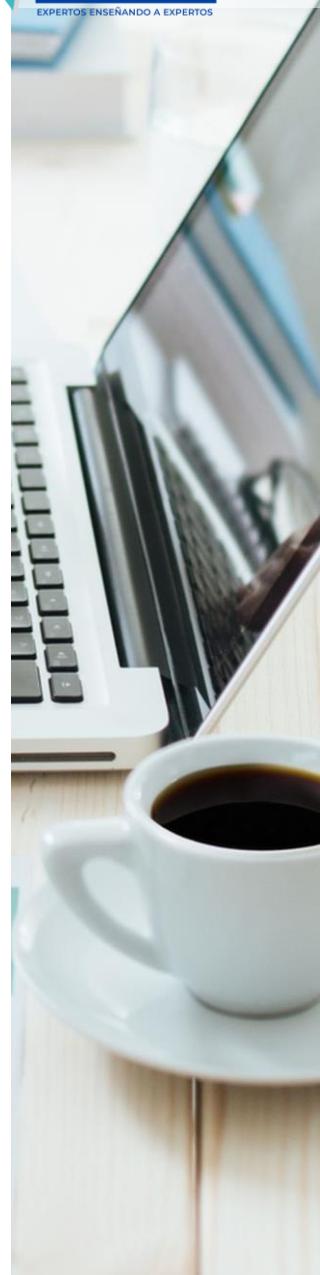
El producto exterior de las dos matrices  
es:

[[ 0 1 1 9]

[ 0 3 3 27]

[ 0 2 2 18]

[ 0 6 6 54]]



## Solución caso 3:

```
import numpy as np

matrixa = np.array([[2, 8, 2], [3, 4, 8], [0, 2, 1]])

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

print("Arrays 3-D:")

print(matrixa)

print(matrixb)

print("El producto exterior de las dos matrices
es:")

result = np.outer(matrixa, matrixb)

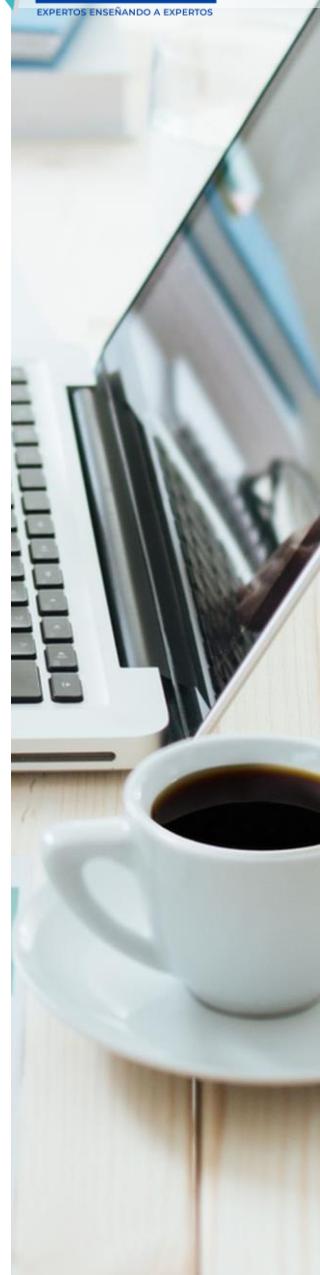
print(result)
```

Arrays 3-D:

```
[[2 8 2]
[3 4 8]
[0 2 1]]
[[2 1 1]
[0 1 0]
[2 3 0]]
```

El producto exterior de las dos matrices es:

```
[[4 2 2 0 2 0 4 6 0]
[16 8 8 0 8 0 16 24 0]
[4 2 2 0 2 0 4 6 0]
[6 3 3 0 3 0 6 9 0]
[8 4 4 0 4 0 8 12 0]
[16 8 8 0 8 0 16 24 0]
[0 0 0 0 0 0 0 0 0]
[4 2 2 0 2 0 4 6 0]
[2 1 1 0 1 0 2 3 0]]
```



## Referencias Bibliográficas

- <https://aprendeconalf.es/docencia/python/manual/numpy/>
- <https://numpy.org/doc/>
- [https://www.wikiwand.com/es/Producto\\_exterior](https://www.wikiwand.com/es/Producto_exterior)



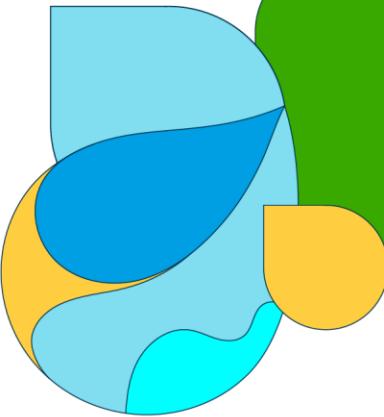
# Unidad temática 13

Librerías para Ciencia de Datos: Pandas

## Objetivos:

- Obtener información de diferentes medios (y formatos) y manipularlos a través del manejo de dataframes.
- Generar información nueva y guardarla en formatos distintos según sus necesidades.

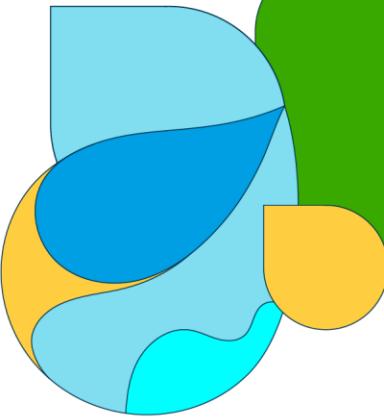
# ¿Qué es Pandas?



Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que el trabajo con datos "relacionales" o "etiquetados" sea fácil e intuitivo. Pretende ser el elemento fundamental de alto nivel para realizar análisis de datos prácticos y del mundo real en Python. La estructura más utilizada es el DataFrame.

Su nombre se deriva de PANel DATA, los pasos típicos que se requieren seguir son los siguientes:

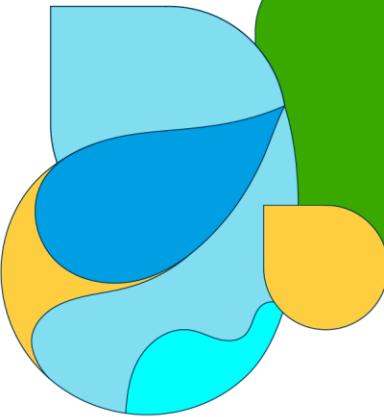
- Cargar
- Preparar
- Modelar
- Manipular
- Analizar



# Dataframe

Los DataFrames son la estructura principal de pandas y tienen las siguientes características:

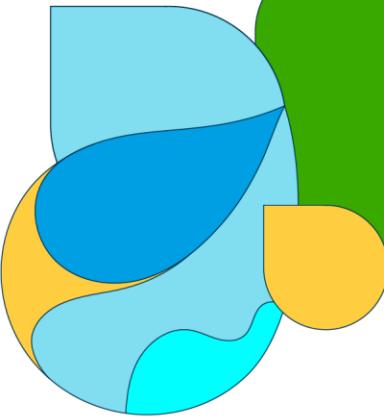
- Son estructuras de datos etiquetados y bidimensionales.
- Constan de tres componentes: Los datos, los índices y las columnas.
- Se pueden especificar los nombres de los índices y las columnas.



# Dataframe

## DataFrame

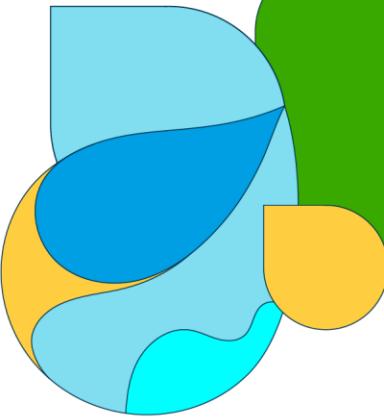
|   | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| A | 2 | 6 | 7 | 1 |
| B | 4 | 3 | 0 | 5 |
| C | 9 | 0 | 8 | 9 |



# Características de Pandas.

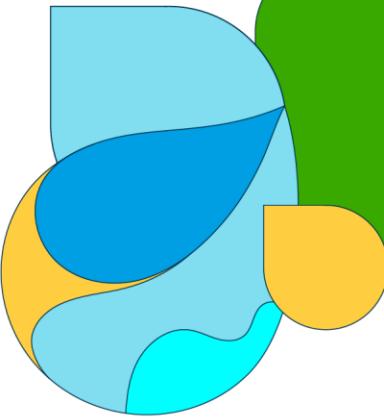
- Objeto DataFrame rápido y eficiente.
- Herramientas para cargar datos en objetos de datos.
- Etiquetado, corte, indexación de grandes conjuntos de datos.
- Alineación de datos y manejo de datos faltantes.
- Remodelación y giro de conjuntos de fechas.
- Las columnas se pueden eliminar o insertar.
- Agrupar datos para agregación y transformaciones.
- Alto rendimiento de fusión y unión de datos.
- Funcionalidad de la serie de tiempo.

# ¿Qué es un DataFrame?



Un DataFrame es una estructura de datos con dos dimensiones en la cual es posible guardar datos de distintos tipos en columnas (como caracteres, enteros, valores de punto flotante, factores, entre otros). Es similar a una hoja de cálculo o una tabla de SQL o el data.frame de R.

Un DataFrame siempre tiene un índice (con inicio en 0), el índice refiere a la posición de un elemento en la estructura de datos.



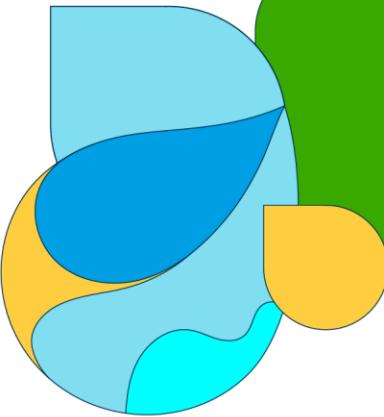
## Creación de un DataFrame a partir de un diccionario

Si tenemos los siguientes diccionarios:

```
unidades_2015 = {"Ag":2, "Au":5, "Cu":3, "Pt":2}
```

```
unidades_2016 = {"Ag":4, "Au":6, "Cu":7, "Pt":2}
```

```
unidades_2017 = {"Ag":3, "Au":2, "Cu":4, "Pt":1}
```



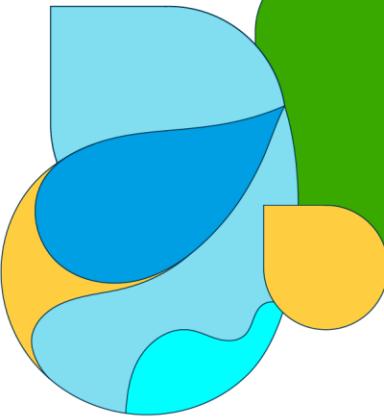
# Creación de un DataFrame a partir de un diccionario

```
>>> import pandas as pd
>>> unidades_2015 = {"Ag":2, "Au":5, "Cu":3, "Pt":2}
>>> unidades_2016 = {"Ag":4, "Au":6, "Cu":7, "Pt":2}
>>> unidades_2017 = {"Ag":3, "Au":2, "Cu":4, "Pt":1}

>>> unidades = pd.DataFrame([unidades_2015, unidades_2016, unidades_2017], index = [2015,
2016, 2017])

>>> unidades

 Ag Au Cu Pt
2015 2 5 3 2
2016 4 6 7 2
2017 3 2 4 1
```



# Cómo importar datos desde un fichero de texto plano

Si tenemos con.txt con encabezado y con el siguiente contenido:

col1 col2

1 2

2 4

3 6

4 8

5 10

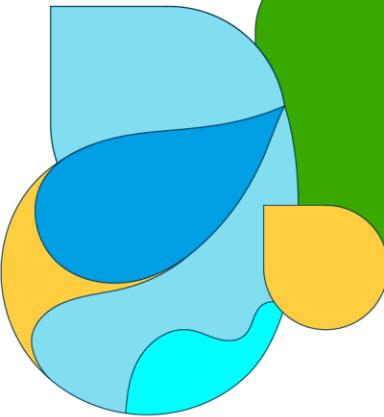
6 12

7 14

8 16

9 18

10 20



# Cómo importar datos desde un fichero de texto plano

## Con encabezado

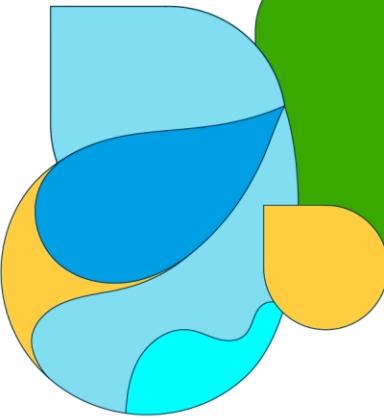
```
>>> import pandas as pd
>>> df = pd.read_csv("con.txt", sep = " ")
>>> df

 col1 col2
0 1 2
1 2 4
2 3 6
3 4 8
4 5 10
5 6 12
6 7 14
7 8 16
8 9 18
9 10 20
```

## Sin encabezado

```
>>> df = pd.read_csv("sin.txt", sep = " ", header = None)
>>> df

 0 1
0 1 2
1 2 4
2 3 6
3 4 8
4 5 10
5 6 12
6 7 14
7 8 16
8 9 18
9 10 20
```



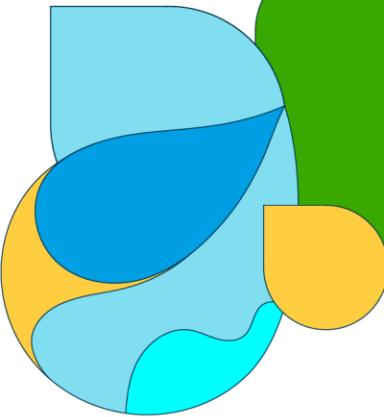
# Como importar datos desde un fichero de texto plano

## Archivos CSV

```
>>> df.to_csv('nombre_archivo.csv')
```

## Archivos Excel

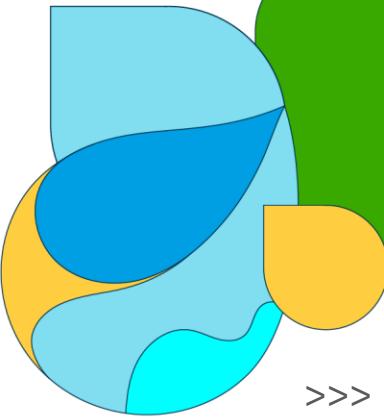
```
df.to_excel('nombre_archivo.xlsx')
```



# Métodos útiles

Para este punto, usaremos el DataFrame para los siguientes ejercicios.

Estamos trabajando con un DataFrame catálogo.xlsx de 229 renglones x 12 columnas, las columnas las podemos conocer con la siguiente instrucción:

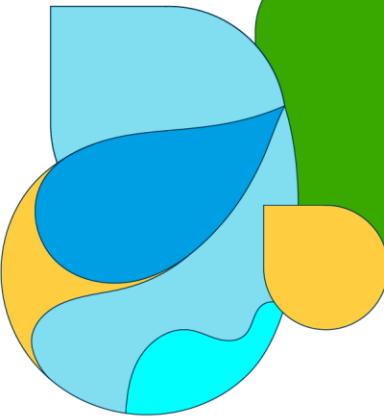


# Métodos útiles

```
>>> df.columns
Index(['PRODUCTO', 'EMPAQUE', 'Familia', 'Codigo Familia', 'CLAVE VENTA',
 'Clave FABRICA', 'WATTS', 'CCT', 'PF', 'LM', 'V', 'MEDIDA FOCO (mm)'],
 dtype='object')
```

Para obtener los tipos de datos contenidos en el DataFrame:

```
>>> df.dtypes
PRODUCTO float64
EMPAQUE float64
Familia object
Codigo Familia object
CLAVE VENTA object
Clave FABRICA object
WATTS object
CCT object
PF float64
LM object
V object
```

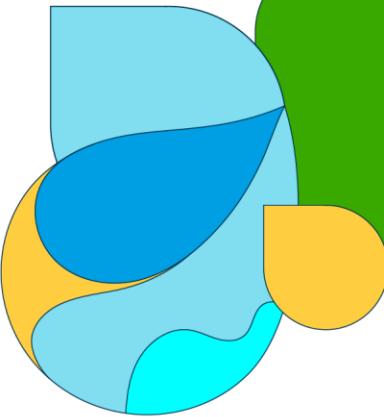


# Métodos útiles

## Fuciones:

df.iloc[0] # Primera fila

```
>>> df.iloc[0] # Primera fila
PRODUCTO 7.50228e+12
EMPAQUE 1.75023e+13
Familia A19
Codigo Familia 1
CLAVE VENTA YKA1901
Clave FABRICA YGA03A41
WATTS 5W
CCT 3000K
PF 0.5
LM 470LM
V 100-240V
MEDIDA FOCO (mm) 112x60
Name: 0, dtype: object
```



# Métodos útiles

## Fuciones:

df.iloc[1] # Segunda fila

```
>>> df.iloc[1] # Segunda fila
PRODUCTO 7.50228e+12
EMPAQUE 1.75023e+13
Familia A19
Codigo Familia 1
CLAVE VENTA YKA1902
Clave FABRICA YGA03A41
WATTS 5W
CCT 6500K
PF 0.5
LM 470LM
V 100-240V
MEDIDA FOCO (mm) 112x60
Name: 1, dtype: object
```

# Resumen del capítulo

A lo largo de este tema exploramos que Pandas:

- Es una librería de código abierto que nos hace más fácil todo el ciclo de vida de cualquier dato, desde que este es generado hasta que es aprovechado.
- Permite, de forma fácil e intuitiva realizar operaciones capaces de gestionar y manipular cualquier tipo de información sin importar el formato, y sobre todo de una forma rápida y eficaz. Esto hace que Pandas se haya convertido en el mejor amigo de cualquier curioso por los datos.
- Ofrece funcionalidades que permiten realizar operaciones matemáticas, calcular estadísticas y generar gráficos a partir del contenido de una Serie o DataFrame.

# Actividad del módulo

caso 1:

$$a = [6 \ 2]$$

$$b = [2 \ 5]$$

caso 2:

$$a = [1 \ 3]$$

$$[2 \ 6]$$

$$b = [0 \ 1]$$

$$[1 \ 9]$$

caso 3:

$$a = [2 \ 8 \ 2]$$

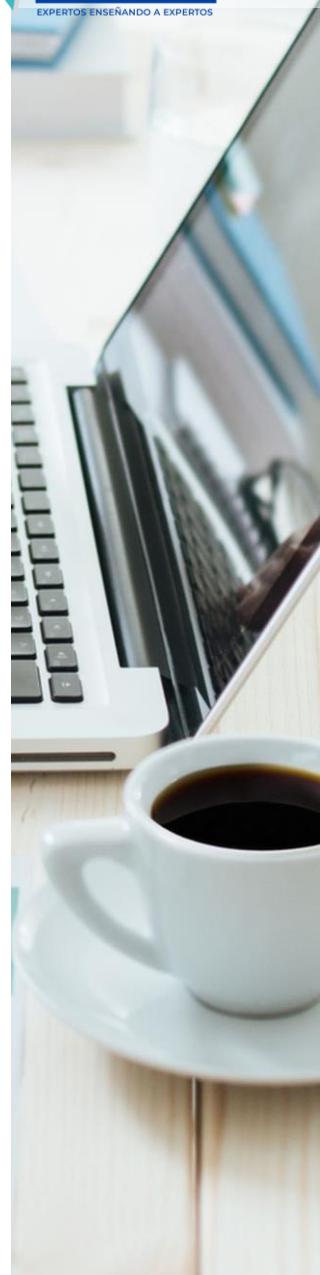
$$[3 \ 4 \ 8]$$

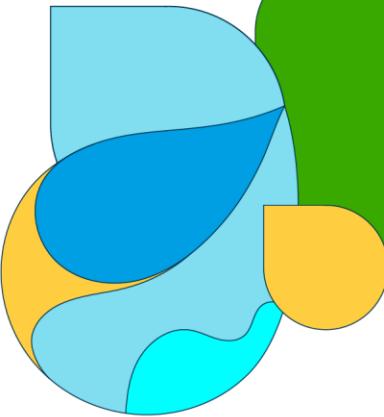
$$[0 \ 2 \ 1]$$

$$b = [2 \ 1 \ 1]$$

$$[0 \ 1 \ 0]$$

$$[2 \ 3 \ 0]$$





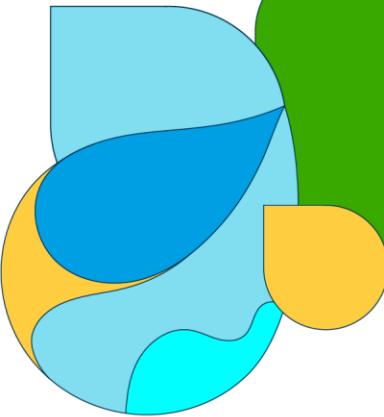
# Métodos útiles

## Fuciones:

```
df.iloc[-1] # Última fila
```

```
>>> df.iloc[-1] # Última fila
PRODUCTO NaN
EMPAQUE NaN
Familia NaN
Codigo Familia Sin Familia
CLAVE VENTA FE258-3
Clave FABRICA FE258
WATTS 15W
CCT 6400K

Name: PRODUCTO, Length: 229, dtype:
float64
```



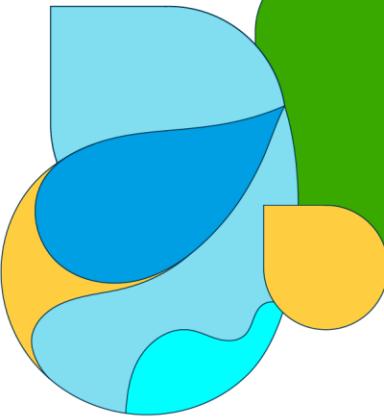
# Métodos útiles

```
>>> df.iloc[:, 1] # Segunda columna
```

```
0 1.750228e+13
1 1.750228e+13
2 1.750228e+13
3 1.750228e+13
4 1.750228e+13
5 NaN
6 1.750228e+13
7 1.750228e+13
...
...
```

```
>>> df.iloc[:, -1] # Última columna
```

```
0 112x60
1 112x60
2 112x60
3 112x60
4 112x60
5 NaN
6 112x60
7 112x60
...
...
```



# Métodos útiles

Hagamos los siguientes ejemplos:

```
df.iloc[0:5] # Primeras cinco filas
```

```
df.iloc[:, 0:5] # Primeras cinco columnas
```

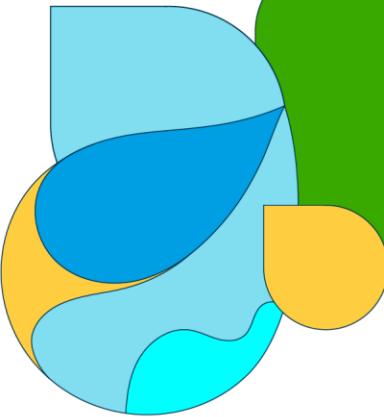
```
df.iloc[[0,2,1]] # Primera, tercera y segunda filas
```

```
df.iloc[:, [0,2,1]] # Primera, tercera y segunda
columnas
```

```
>>> df.iloc[0:5] # Primeras cinco filas
```

|   | PRODUCTO         | EMPAQUE      | ... | V    |
|---|------------------|--------------|-----|------|
| 0 | MEDIDA FOCO (mm) |              |     |      |
| 1 | 240V             | 112x60       |     | 100- |
| 2 | 7.502281e+12     | 1.750228e+13 |     | ...  |
| 3 | 240V             | 112x60       |     | 100- |
| 4 | 7.502281e+12     | 1.750228e+13 |     | ...  |
|   | 240V             | 112x60       |     | 100- |
|   | 112x60           |              |     | 127V |

[5 rows x 12 columns]



# Métodos útiles

```
>>> df.iloc[:, 0:5] # Primeras cinco columnas
```

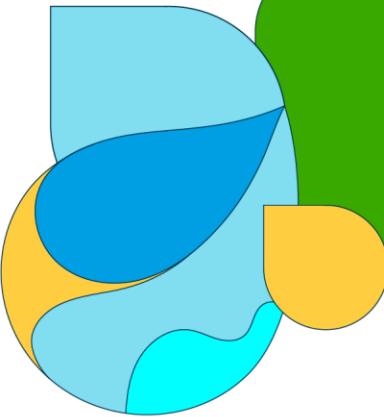
|       | PRODUCTO     | EMPAQUE      | ... | Codigo | Familia | CLAVE |
|-------|--------------|--------------|-----|--------|---------|-------|
| VENTA |              |              |     |        |         |       |
| 0     | 7.502281e+12 | 1.750228e+13 | ... | 1      | YKA1901 |       |
| 1     | 7.502281e+12 | 1.750228e+13 | ... | 1      | YKA1902 |       |
| 2     | 7.502281e+12 | 1.750228e+13 | ... | 1      | YKA1909 |       |
| 3     | 7.502281e+12 | 1.750228e+13 | ... | 1      | YKA1910 |       |
| 4     | 7.502281e+12 | 1.750228e+13 | ... | 1      | YKA1933 |       |
| ...   |              |              |     |        |         |       |

[229 rows x 5 columns]

```
>>> df.iloc[[0,2,1]] # Primera, tercera y segunda filas
```

|      | PRODUCTO     | EMPAQUE      | ... | V MEDIDA FOCO |
|------|--------------|--------------|-----|---------------|
| (mm) |              |              |     |               |
| 0    | 7.502281e+12 | 1.750228e+13 | ... | 100-          |
| 240V |              | 112x60       |     |               |
| 2    | 7.502281e+12 | 1.750228e+13 | ... | 100-          |
| 240V |              | 112x60       |     |               |
| 1    | 7.502281e+12 | 1.750228e+13 | ... | 100-          |
| 240V |              | 112x60       |     |               |

[3 rows x 12 columns]



# Métodos útiles

```
>>> df.iloc[:, [0,2,1]] # Primera, tercera y segunda columnas
```

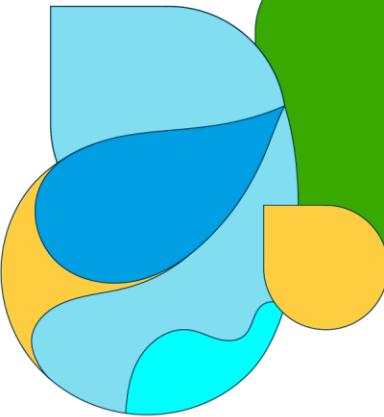
|     | PRODUCTO     | Familia | EMPAQUE      |
|-----|--------------|---------|--------------|
| 0   | 7.502281e+12 | A19     | 1.750228e+13 |
| 1   | 7.502281e+12 | A19     | 1.750228e+13 |
| 2   | 7.502281e+12 | A19     | 1.750228e+13 |
| 3   | 7.502281e+12 | A19     | 1.750228e+13 |
| 4   | 7.502281e+12 | A19     | 1.750228e+13 |
| ... |              |         |              |

[229 rows x 3 columns]

```
>>> df_sub = df.loc[1:5] # sub_DataFrame de las filas 1 a 4
```

```
>>> df_sub.loc[1] # fila 1
```

|                  |                  |
|------------------|------------------|
| PRODUCTO         | 7.50228e+12      |
| EMPAQUE          | 1.75023e+13      |
| Familia          | A19              |
| Codigo Familia   | 1                |
| CLAVE VENTA      | YKA1902          |
| Clave FABRICA    | YGA03A41         |
| WATTS            | 5W               |
| CCT              | 6500K            |
| PF               | 0.5              |
| LM               | 470LM            |
| V                | 100-240V         |
| MEDIDA FOCO (mm) | 112x60           |
| Name:            | 1, dtype: object |



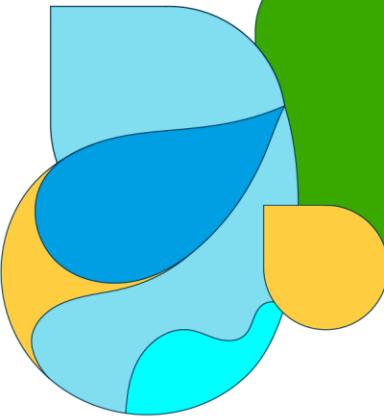
# Métodos útiles

```
>>> df_sub.loc[2] # fila 2
```

|                        |             |
|------------------------|-------------|
| PRODUCTO               | 7.50228e+12 |
| EMPAQUE                | 1.75023e+13 |
| Familia                | A19         |
| Codigo Familia         | 1           |
| CLAVE VENTA            | YKA1909     |
| Clave FABRICA          | YGA03A41    |
| WATTS                  | 9W          |
| CCT                    | 3000K       |
| PF                     | 0.5         |
| LM                     | 810LM       |
| V                      | 100-240V    |
| MEDIDA FOCO (mm)       | 112x60      |
| Name: 2, dtype: object |             |

```
>>> df_sub.loc[3] # fila 3
```

|                        |             |
|------------------------|-------------|
| PRODUCTO               | 7.50228e+12 |
| EMPAQUE                | 1.75023e+13 |
| Familia                | A19         |
| Codigo Familia         | 1           |
| CLAVE VENTA            | YKA1910     |
| Clave FABRICA          | YGA03A41    |
| WATTS                  | 9W          |
| CCT                    | 6500K       |
| PF                     | 0.5         |
| LM                     | 810LM       |
| V                      | 100-240V    |
| MEDIDA FOCO (mm)       | 112x60      |
| Name: 3, dtype: object |             |

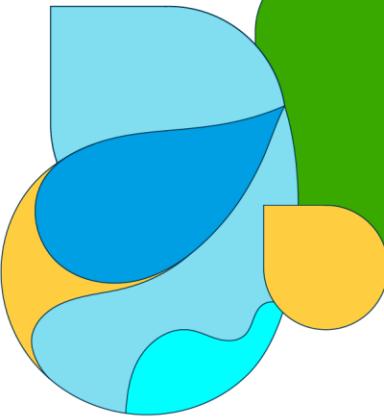


# Métodos útiles

```
>>> df.describe()
 PRODUCTO EMPAQUE PF
count 1.810000e+02 1.810000e+02 185.000000
mean 7.502281e+12 1.730889e+13 0.789946
std 2.195108e+03 1.834681e+12 0.636112
min 7.502281e+12 0.000000e+00 0.500000
25% 7.502281e+12 1.750228e+13 0.500000
50% 7.502281e+12 1.750228e+13 0.900000
75% 7.502281e+12 1.750228e+13 0.900000
max 7.502281e+12 1.750228e+13 9.000000
```

Transformar DataFrames

```
>>> df = pd.read_excel('catalogo.xlsx')
>>> df.columns
Index(['PRODUCTO', 'EMPAQUE', 'Familia', 'Codigo
Familia', 'CLAVE VENTA',
 'Clave FABRICA', 'WATTS', 'CCT', 'PF', 'LM',
 'V', 'MEDIDA FOCO (mm)'],
 dtype='object')
```



# Métodos útiles

Borrar Columnas:

```
>>> df=df.drop(columns=['Familia','Codigo
Familia','Clave FABRICA'])
>>> df.columns
Index(['PRODUCTO', 'EMPAQUE', 'CLAVE VENTA',
'WATTS', 'CCT', 'PF', 'LM', 'V',
 'MEDIDA FOCO (mm)'],
 dtype='object')
```

Borrar filas que tengan datos vacíos:

```
>>> df=df.dropna()
```

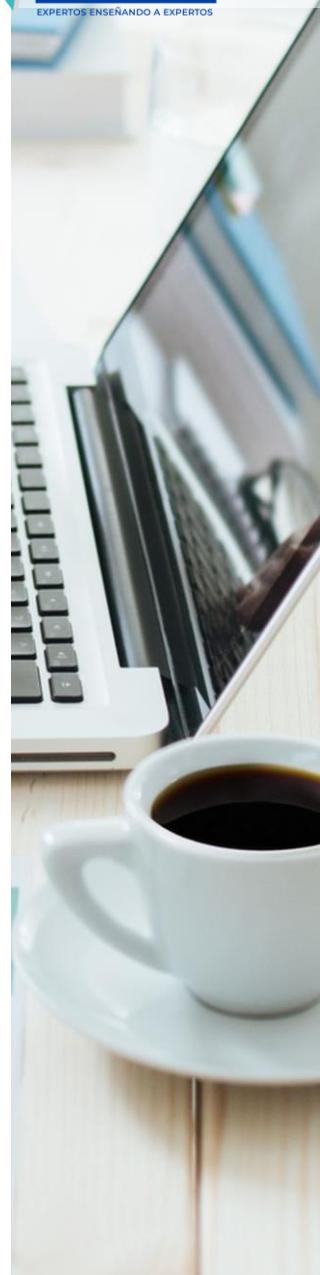
Salvar ficheros:

```
>>> df.to_excel('catalogo-2.xlsx', index=False)
```

# Actividad del capítulo

Probemos el siguiente programa para manipular el archivo techogar.csv:

- import pandas as pd
- datos = pd.read\_csv('techogar.csv',header=0)
- print(datos) # 1a ejecución para verificar lectura y observar resultados
- print(datos['Cable']) # 2a ejecución para ver solo la columna cable
- print(datos.sort\_values(by='Teléfono')) # 4a ejecución aplicamos un filtro para ordenar en relación a la columna Teléfono
- print('\n') # imprimimos un salto de carro para visualizar mejor el resultado
- print(datos.sort\_values(by='Teléfono', ascending=False)) # 5a ejecución, invertimos el despliegue de los datos
- tel = datos['Teléfono'] # para generar una nueva lista que solo contenga la columna referida
- print(tel) # 7a ejecución mostrar la lista tel
- print('\n') # imprimimos un salto de carro para visualizar mejor el resultado
- print(tel[tel>10]) # 7a ejecución mostrar la lista tel para mostrar solo porcentajes mayores al 10



# Actividad del capítulo

## Parte de los resultados:

|   | Año | Computadora  | Internet  | Televisión | Cable     | Teléfono  |
|---|-----|--------------|-----------|------------|-----------|-----------|
| ● | 0   | 2001         | 11.777200 | 6.212087   | 91.897078 | 13.529983 |
| ● | 1   | 2002         | 15.207573 | 7.455098   | 93.592413 | 15.375804 |
| ● | 2   | 2004         | 18.001723 | 8.692415   | 91.689488 | 19.330399 |
| ● |     | ...          |           |            |           |           |
| ● | 13  | 2015         | 44.911120 | 39.177739  | 93.523252 | 43.745243 |
| ● | 0   |              | 13.529983 |            |           |           |
| ● | 1   |              | 15.375804 |            |           |           |
| ● | 2   |              | 19.330399 |            |           |           |
| ● |     | ...          |           |            |           |           |
| ● | 11  |              | 36.730110 |            |           |           |
| ● | 12  |              | 38.109545 |            |           |           |
| ● | 13  |              | 43.745243 |            |           |           |
| ● |     | Name: Cable, |           |            |           |           |



## Referencia Bibliográfica

- <https://aprendeconalf.es/docencia/python/manual/pandas/>
- <https://pandas.pydata.org/docs/index.html>
- [https://www.youtube.com/watch?v=\\_8onVOY2j4E](https://www.youtube.com/watch?v=_8onVOY2j4E)

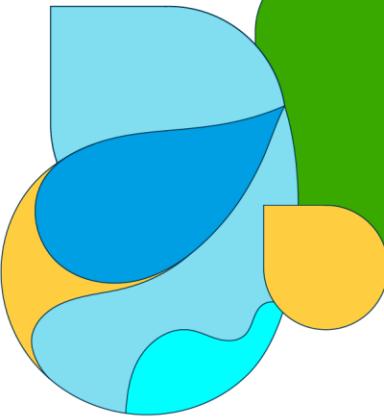


# Unidad temática 14

Librerías para Ciencia de Datos:  
Matplotlib

## Objetivos:

- Crear los diferentes tipos de gráficos que nos ofrece la librería.
- Saber combinar esta librería con las ya estudiadas para representar datos importantes en los diferentes tipos de gráficos.
- Combinar diferentes tipos de gráficos en una sola ilustración para poder generar histogramas.



# Introducción

Matplotlib es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en Python. Matplotlib hace que las cosas fáciles sean fáciles y las difíciles sean posibles, entre otras permite:

- Crear publicaciones de calidad de publicación .
- Hacer figuras interactivas que puedan hacer zoom, desplazarse, actualizar.
- Personalizar el estilo visual y el diseño .
- Exportar a muchos formatos de archivo .
- Incrustar en JupyterLab e interfaces gráficas de usuario .
- Utilizar una gran variedad de paquetes de terceros creados en Matplotlib.

# Introducción

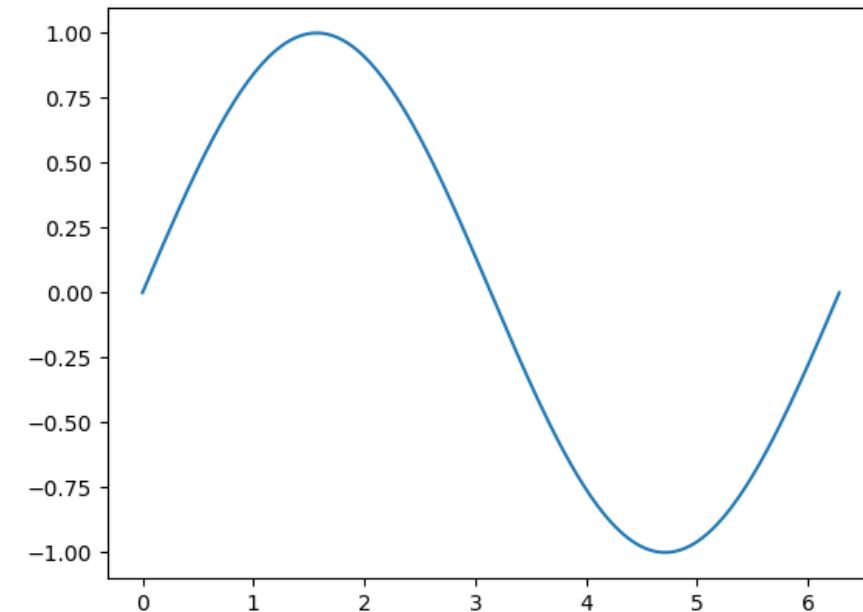
Existe una gran cantidad de paquetes de terceros que amplían y se basan en la funcionalidad de Matplotlib, incluidas varias interfaces de trazado de nivel superior (seaborn, HoloViews, ggplot, ...) y un conjunto de herramientas de proyección y mapeo (Cartopy).

Un pequeño ejemplo:

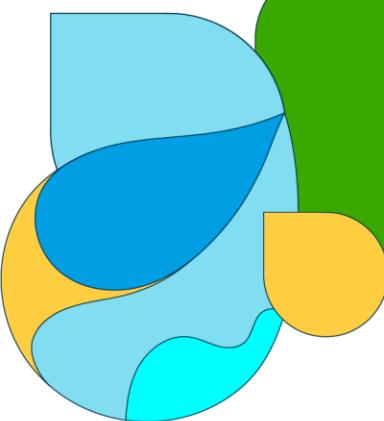
```
import matplotlib.pyplot as plt
import numpy as np
```

```
x = np.linspace(0, 2 * np.pi, 200)
y = np.sin(x)
```

```
fig, ax = plt.subplots()
ax.plot(x, y)
plt.show() .
```



# Gráfica en línea simple

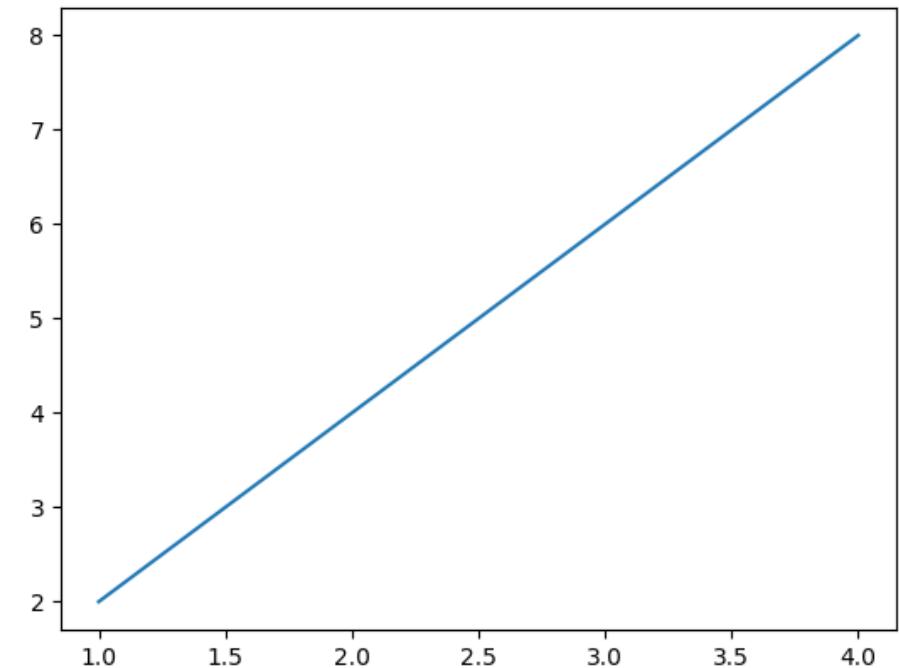


Primero importa la biblioteca Matplotlib.pyplot para trazar funciones, además, importa la biblioteca Numpy según los requisitos. Luego, define los valores de datos x e y.

Partiremos del siguiente código:

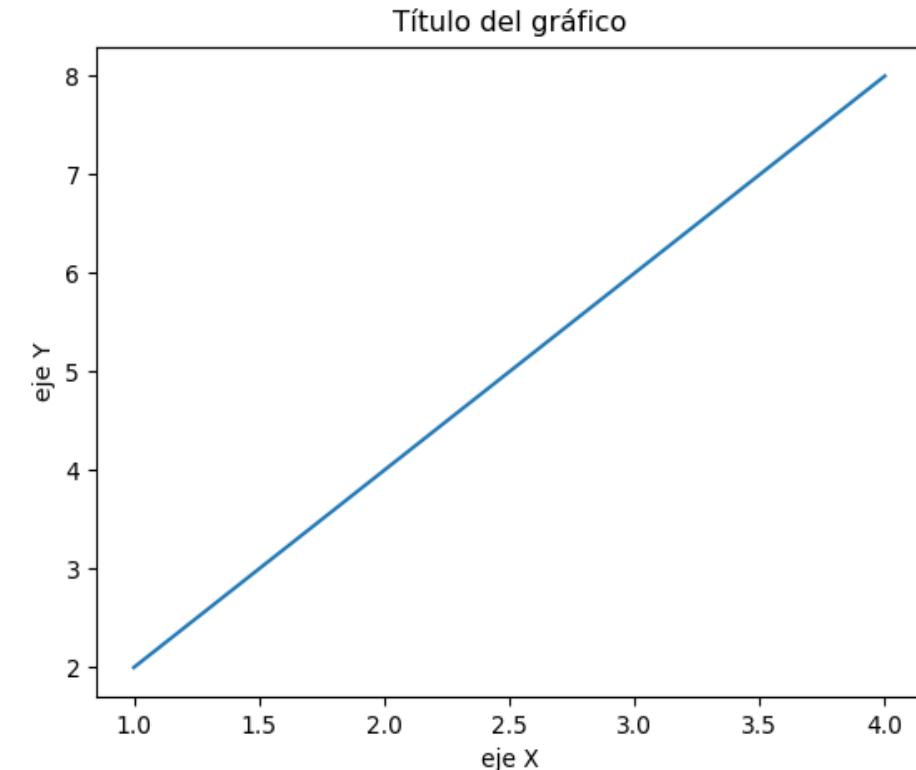
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2
plt.plot(x, y)
plt.show()
```

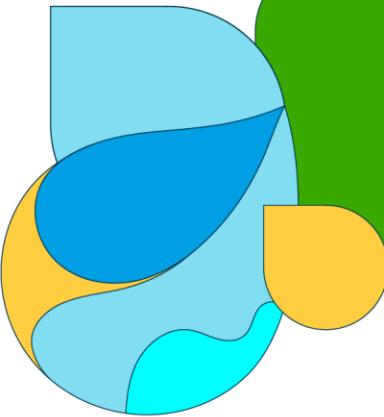
Podemos observar que el gráfico no cuenta con etiqueta en el eje x, ni en el eje y. La librería nos permite añadir las dimensiones del gráfico. Con el siguiente código, agregaremos etiquetas a los gráficos.



# Gráfica en línea simple

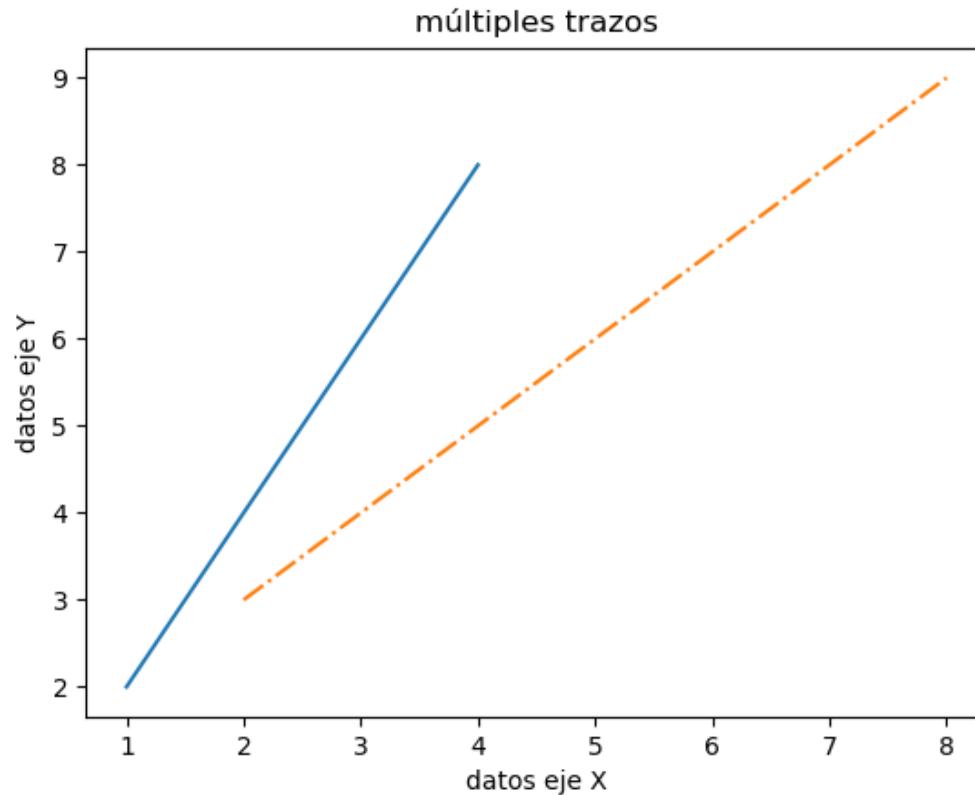
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x**2
plt.plot(x, y)
plt.xlabel("eje Xs")
plt.ylabel("eje Y")
plt.title("Título del gráfico")
plt.show()
```



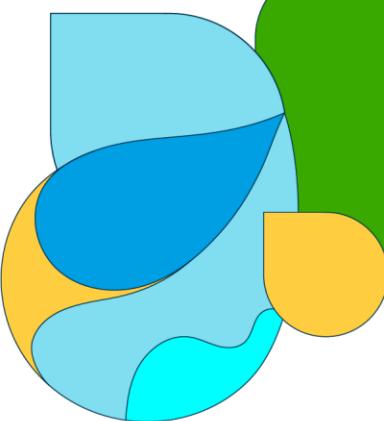


# Múltiples gráficos

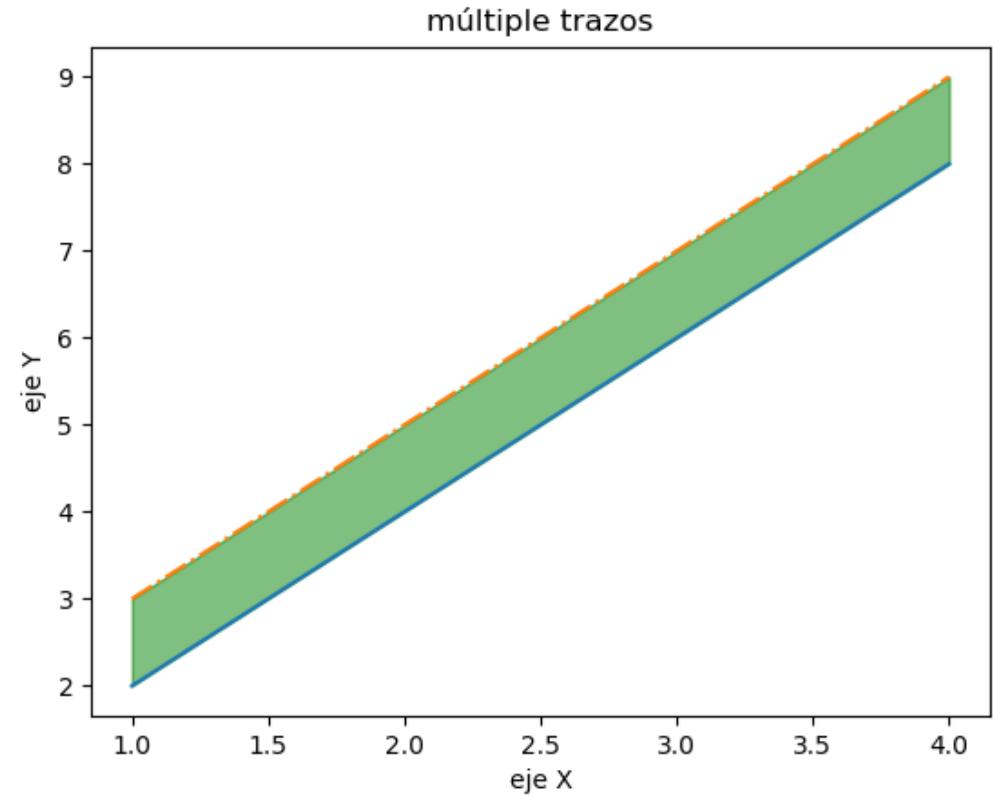
```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x*2
plt.plot(x, y)
x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]
plt.plot(x1, y1, '-.')
plt.xlabel("datos eje X")
plt.ylabel("datos eje Y")
plt.title('múltiples trazos')
plt.show()
```



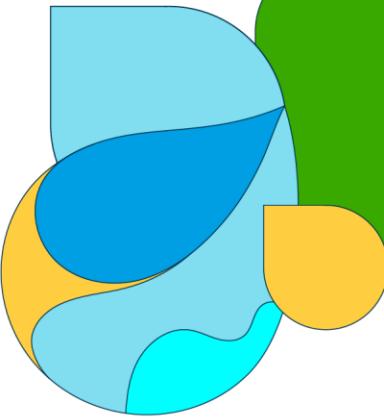
# Relleno de trazos



```
import matplotlib.pyplot as plt
import numpy as np
x = np.array([1, 2, 3, 4])
y = x**2
plt.plot(x, y)
x1 = [2, 4, 6, 8]
y1 = [3, 5, 7, 9]
plt.plot(x, y1, '-.')
plt.xlabel("eje X")
plt.ylabel("eje Y")
plt.title('m ltiple trazos')
plt.fill_between(x, y, y1, color='green', alpha=0.5)
plt.show()
```



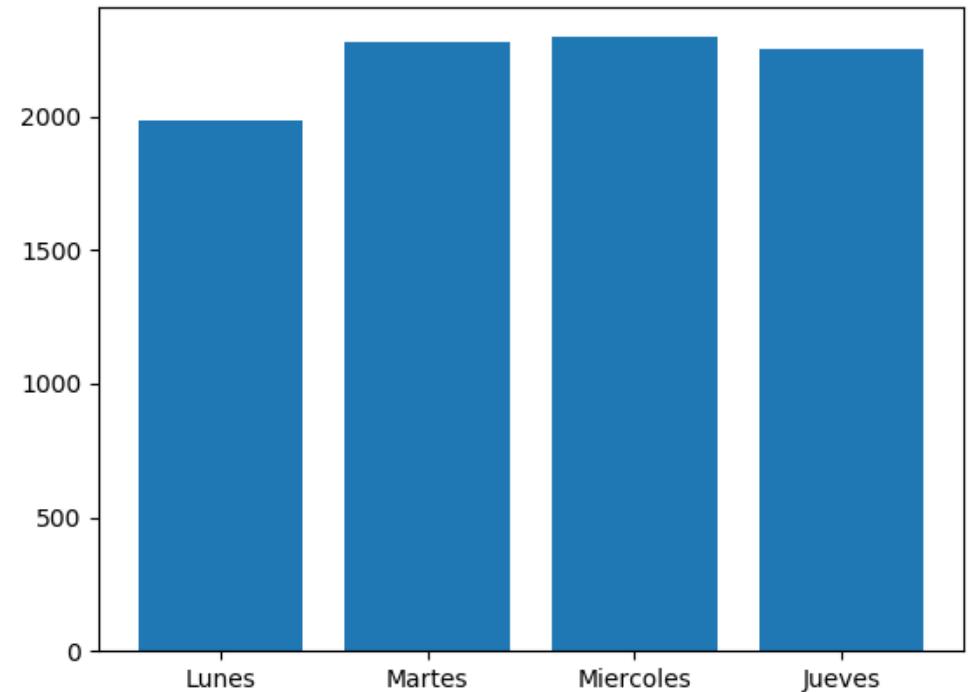
# Gráfica de barra



Las gráficas de barras son una de las más comunes para la representación de datos. Y una muy buena opción es la asociación de valores numéricos a categorías, pudiéndose apilar diferentes categorías en una misma barra.

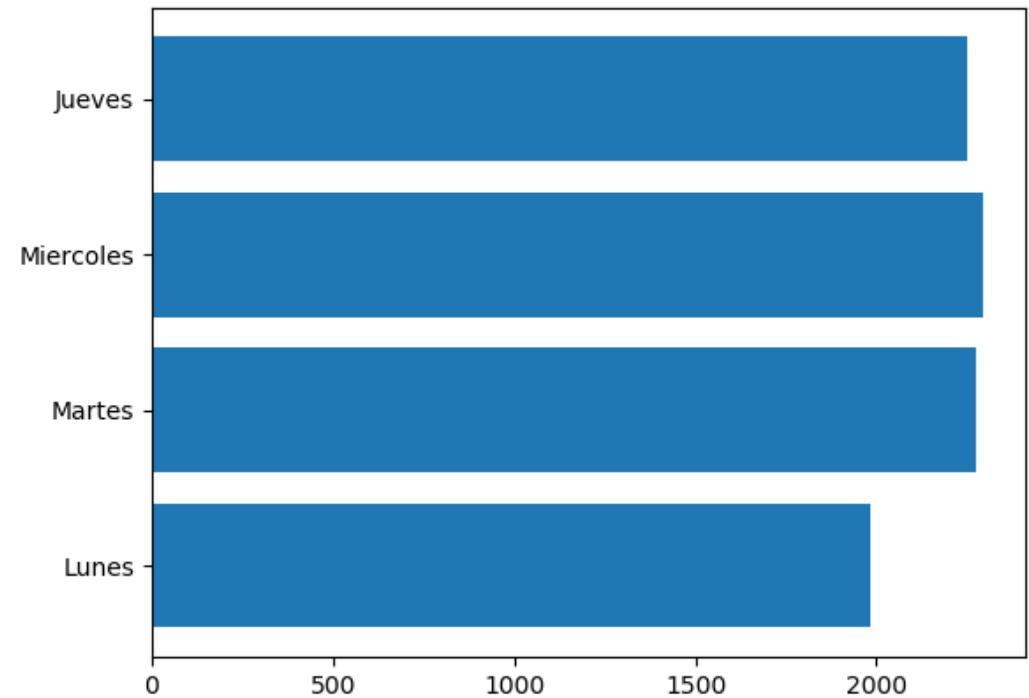
# Relleno de trazos

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.DataFrame({'España' : [826, 943, 942, 901],
 'Colombia': [668, 781, 791, 813],
 'México': [488, 553, 563, 537]},
 index=('Lunes','Martes','Miercoles','Jueves'))
total = data.sum(axis=1)
plt.bar(total.index, total)
plt.show()
```



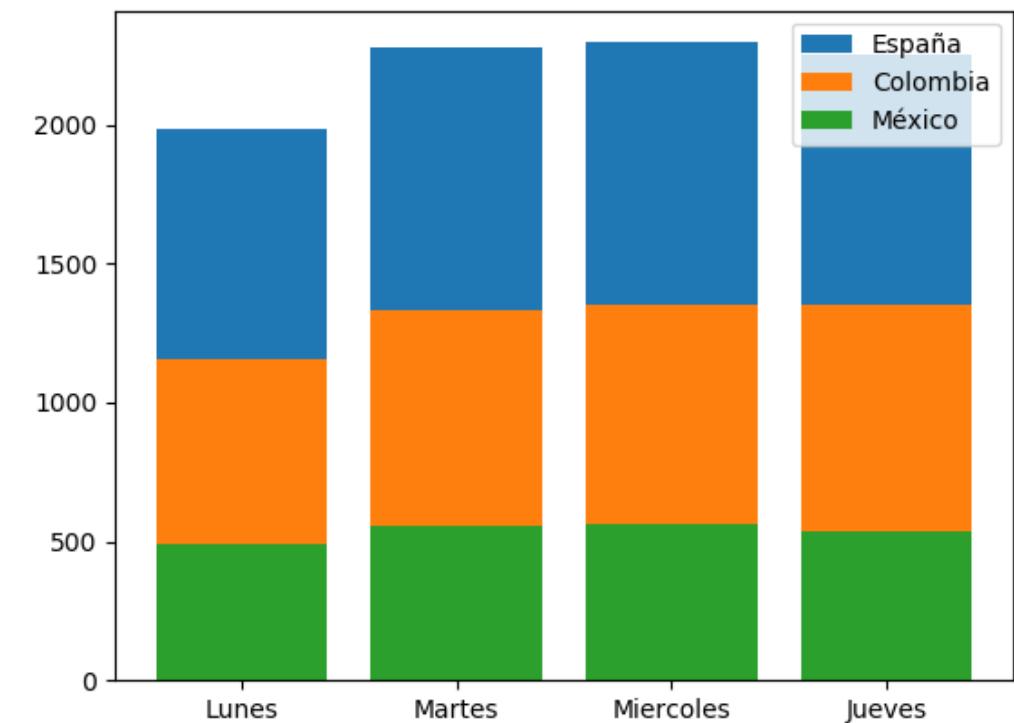
# Gráfica de barras horizontales

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.DataFrame({'España' : [826, 943, 942, 901],
 'Colombia': [668, 781, 791, 813],
 'México': [488, 553, 563, 537]},
 index=('Lunes','Martes','Miercoles', 'Jueves'))
total = data.sum(axis=1)
plt.barh(total.index, total)
plt.show()
```



# Gráficos apilados de barras

```
import pandas as pd
import matplotlib.pyplot as plt
data = pd.DataFrame({'España' : [826, 943, 942, 901],
'Colombia': [668, 781, 791, 813],
'México': [488, 553, 563, 537]},
index=('Lunes', 'Martes', 'Miercoles', 'Jueves'))
plt.bar(data.index, data.España + data.Colombia +
data.México, label='España')
plt.bar(data.index, data.Colombia + data.México,
label='Colombia')
plt.bar(data.index, data.México, label='México')
plt.legend(loc='best')
plt.show()
```

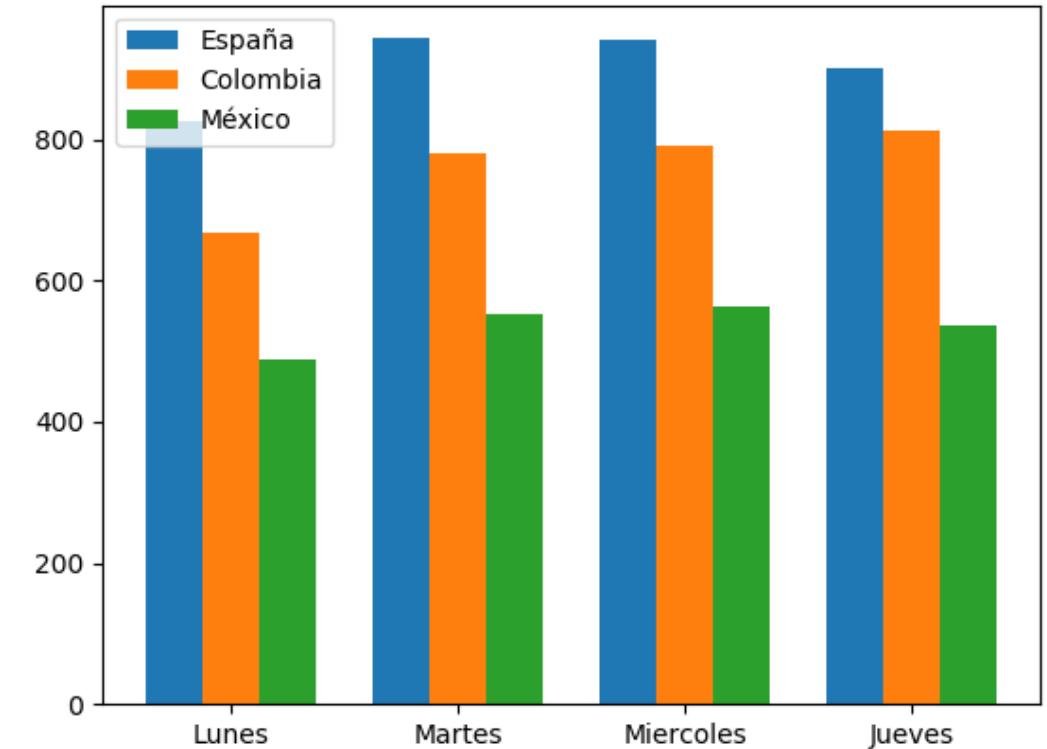


# Gráficos de barras adyacentes

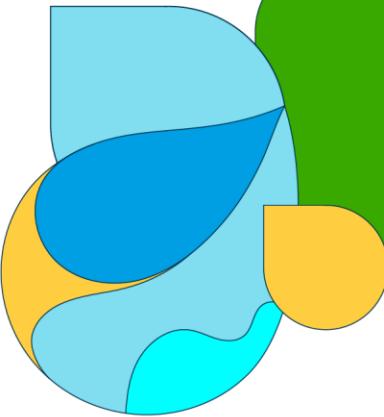
```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data = pd.DataFrame({'España' : [826, 943, 942, 901],
'Colombia': [668, 781, 791, 813],
'México': [488, 553, 563, 537]},
index=('Lunes', 'Martes', 'Miércoles', 'Jueves'))
n = len(data.index)
x = np.arange(n)
width = 0.25
plt.bar(x - width, data.España, width=width, label='España')
plt.bar(x, data.Colombia, width=width, label='Colombia')
plt.bar(x + width, data.Méjico, width=width, label='Méjico')
plt.xticks(x, data.index)
plt.legend(loc='best')
plt.show()

```



# Gráfica de Pastel



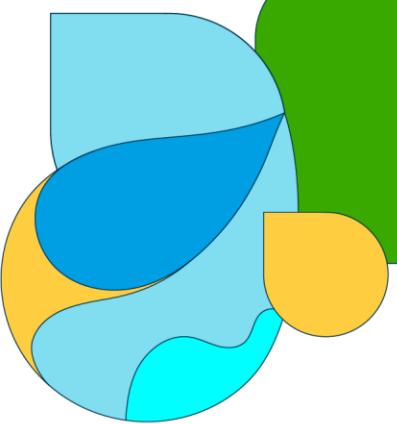
Las gráficas de pastel podemos representar porcentajes y proporciones. En Python podemos utilizar la librería Matplotlib para desarrollar este tipo de gráficas.

Matplotlib dispone de la función pie, cuya sintaxis varía según grado de personalización y control que se requiera sobre la gráfica de pastel a dibujar.

En el siguiente ejemplo de esta función vamos a suponer que se tienen los siguientes datos sobre algunas personas que tienen cierta cantidad de monedas en su poder:

| Nombre   | Monedas |
|----------|---------|
| Juan     | 10      |
| Ana      | 20      |
| Diana    | 25      |
| Catalina | 30      |

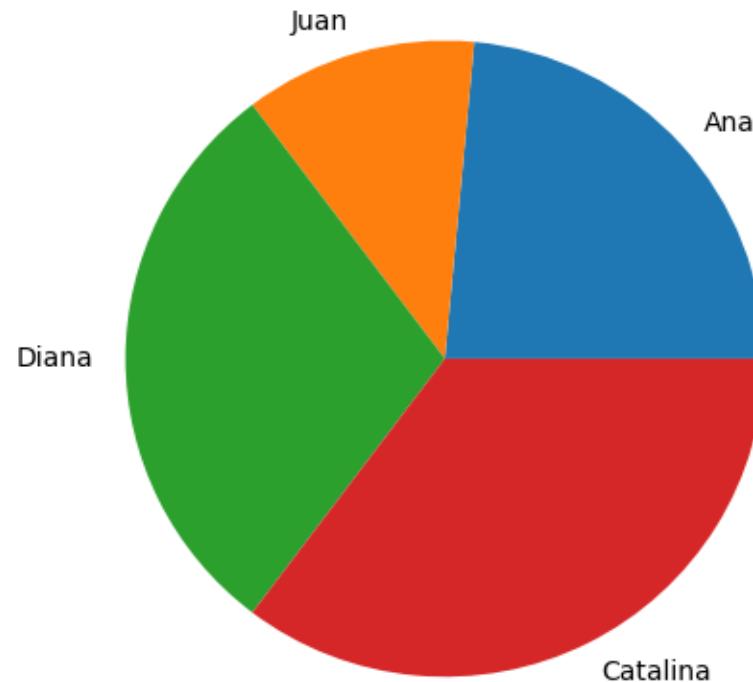
# Gráficos de Pastel



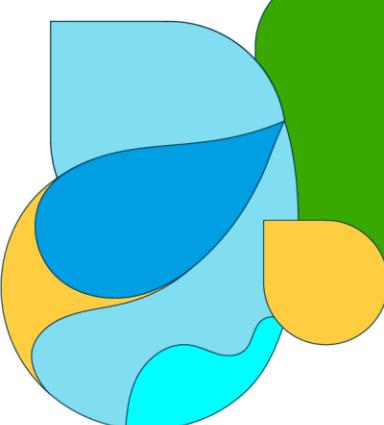
```
import matplotlib.pyplot as plt
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
plt.pie(monedas, labels=nombres)
plt.show()
```

La función pie acepta un primer argumento que contiene los valores absolutos de cada ítem, además, de un keyword argument labels que contiene las etiquetas correspondientes.

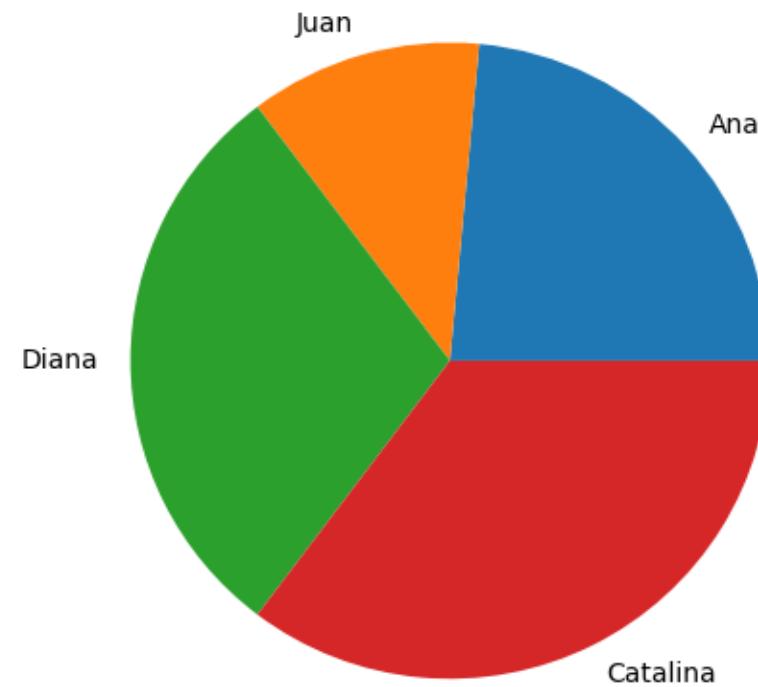
Para corregir el aspecto achatado de la gráfica se puede utilizar la función axis.



# Gráficos de Pastel



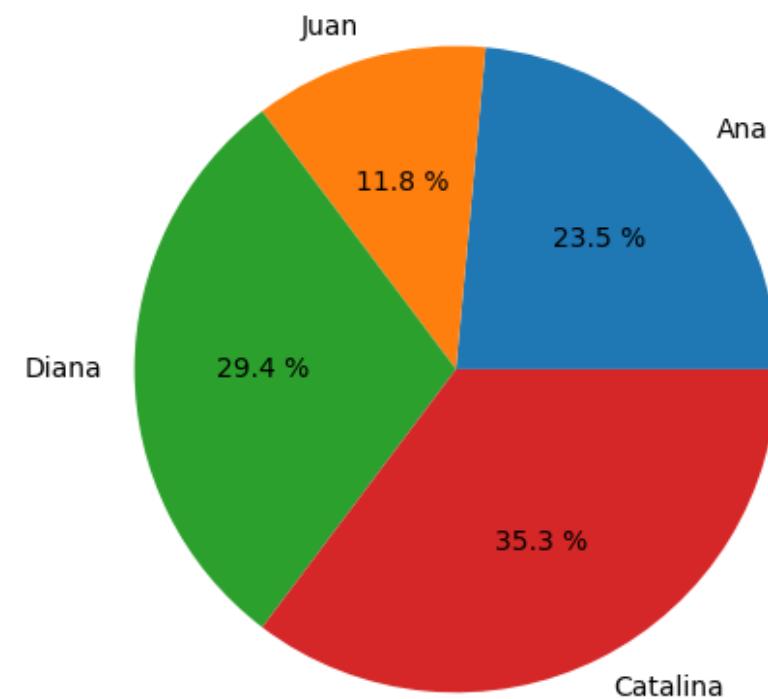
```
import matplotlib.pyplot as plt
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
plt.pie(monedas, labels=nombres)
plt.axis("equal")
plt.show()
```



# Gráficos de Pastel

Para indicar el porcentaje añadimos el argumento  
autopct:

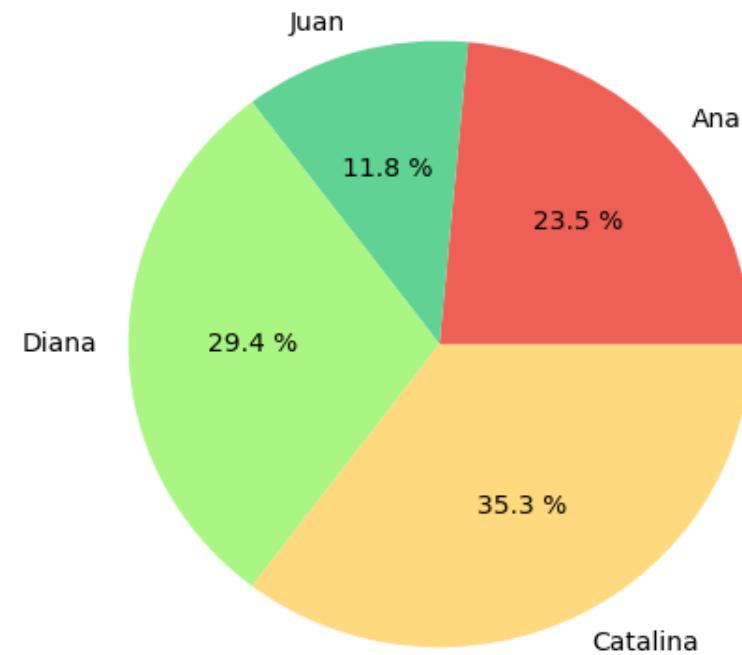
```
import matplotlib.pyplot as plt
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
plt.pie(monedas, labels=nombres, autopct="%0.1f %%")
plt.axis("equal")
plt.show()
```



# Manipulando los colores

Los colores se pueden especificar de manera manual, pasando una lista de color en formato hexadecimal o RGB.

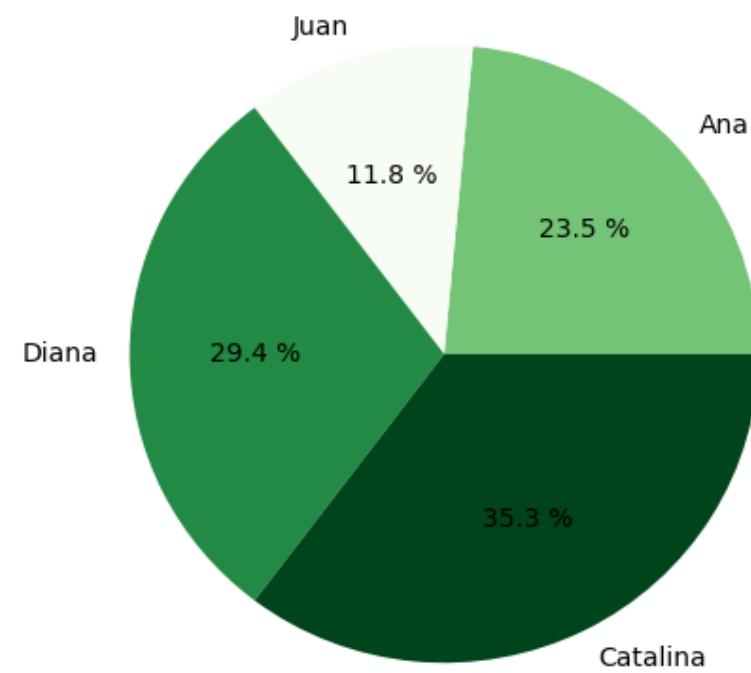
```
import matplotlib.pyplot as plt
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
colores
= ["#EE6055","#60D394","#AAF683","#FFD97D","#FF9
B85"]
plt.pie(monedas, labels=nombres, autopct="%0.1f
%%", colors=colores)
plt.axis("equal")
plt.show()
```



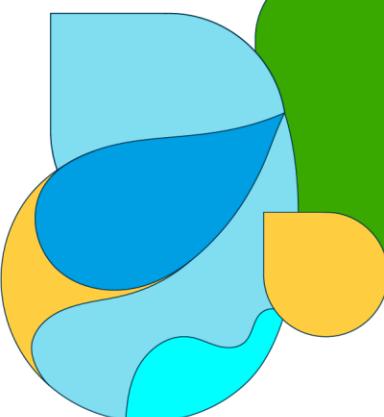
# Manipulando los colores

Los colores también se pueden determinar y auto calcular utilizando un mapa de color específico. Por ejemplo:

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import colors
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
normdata = colors.Normalize(min(moneda
max(moneda))
colormap = cm.get_cmap("Greens")
colores =colormap(normdata(moneda))
plt.pie(moneda, labels=nombres, autopct="%0.1f %%"
colors=colores)
plt.axis("equal")
plt.show()
```

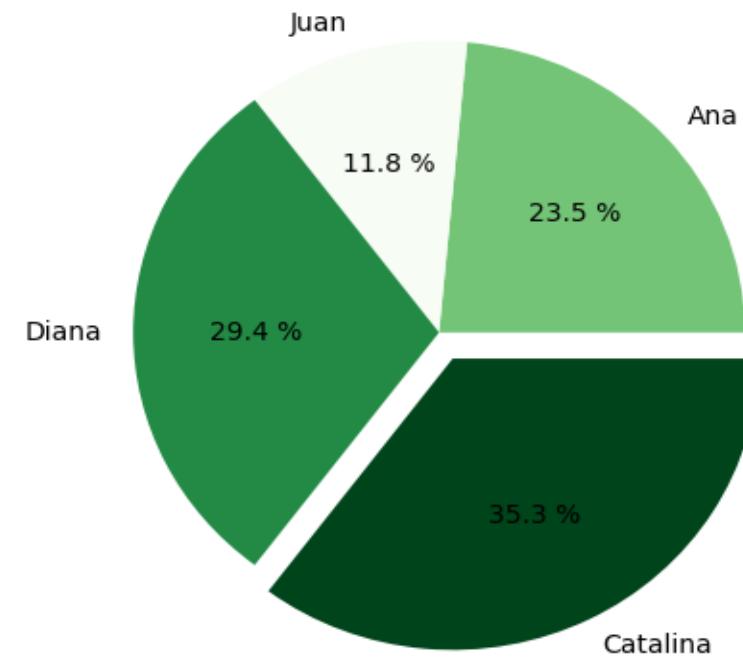


# Extracción de rebanadas

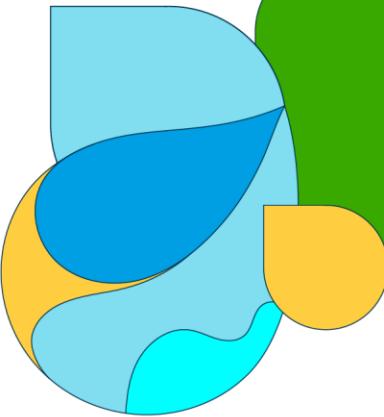


Es posible también segmentar o separar del bloque una o más de las rebanadas de la gráfica de pastel. Para ello se debe pasar una lista o tupla con valores entre 0 y n que indican el desfase respecto al centro, 0 indica ningún desfase y n un desfase equivalente a  $n^*$  donde r es el radio de la gráfica de pastel.

```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import colors
monedas = [20,10,25,30]
nombres = ["Ana","Juan","Diana","Catalina"]
normdata = colors.Normalize(min(monedas), max(monedas))
colormap = cm.get_cmap("Greens")
colores =colormap(normdata(monedas))
desfase = (0, 0, 0, 0.1)
plt.pie(monedas, labels=nombres, autopct="%0.1f" %%
 colors=colores, explode=desfase)
plt.axis("equal")
plt.show()
```



# Definición



El histograma es entonces un gráfico que muestra cómo se distribuyen los datos de una muestra estadística o de una población. Esto, respecto a alguna variable numérica. Los histogramas suelen usar barras, cuya altura dependerá de la frecuencia de los datos, que corresponde al eje Y. En tanto, en el eje X podemos observar la variable de estudio.

Pasos necesarios para la construcción de un histograma.

- a.** Fijar los intervalos a representar. Debemos agrupar las datos en intervalos consecutivos y que no se solapen entre sí. Así podremos representar la frecuencia de cada uno de estos intervalos. En un histograma cada una de las barras corresponde a un intervalo diferente. Por ejemplo, si nuestra muestra de datos contiene los valores 2, 4, 5, 5, 8, 9 y 9, podríamos dividir el conjunto en dos intervalos: de 1 a 5 y de 6 a 10.
- b.** Determinar la frecuencia de cada intervalo. Una vez que los intervalos están determinados es necesario contar cuántos valores pertenecen a cada intervalo. Esa cuenta es lo que llamamos la frecuencia del intervalo. Para el ejemplo anterior, tendríamos que para el intervalo 1-5 tenemos 4 valores (2, 4, 5, 5) y que para el intervalo 6-10 tenemos 3 valores (8, 9, 9). Es decir, 4 y 3 son las frecuencias de los intervalos propuestos.
- c.** Una vez que determinados los intervalos y las frecuencias calculadas solo resta graficar. Para ello podemos hacer alguna representación textual o utilizar una librería de visualización de datos.

# Ejemplo

Tenemos un conjunto de datos con las edades de un grupo de personas: 12, 15, 13, 12, 18, 20, 19, 20, 13, 12, 13, 17, 15, 16, 13, 14, 13, 17, 19. Como ves el rango está entre las edades 12 y 20 años.

En este caso, los intervalos serían los siguientes: de 12 a 13 (sin incluir el último), de 13 a 14, de 14 a 15, etc.

En este caso, el número de intervalos son  $20 - 12 + 1 = 9$ .

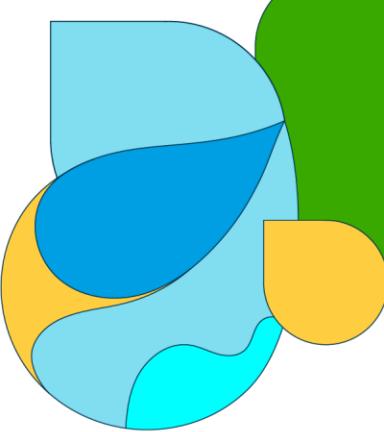
Si tenemos el siguiente código:

```
edades = [12, 15, 13, 12, 18, 20, 19, 20, 13, 12, 13, 17, 15, 16, 13, 14, 13, 17, 19]
```

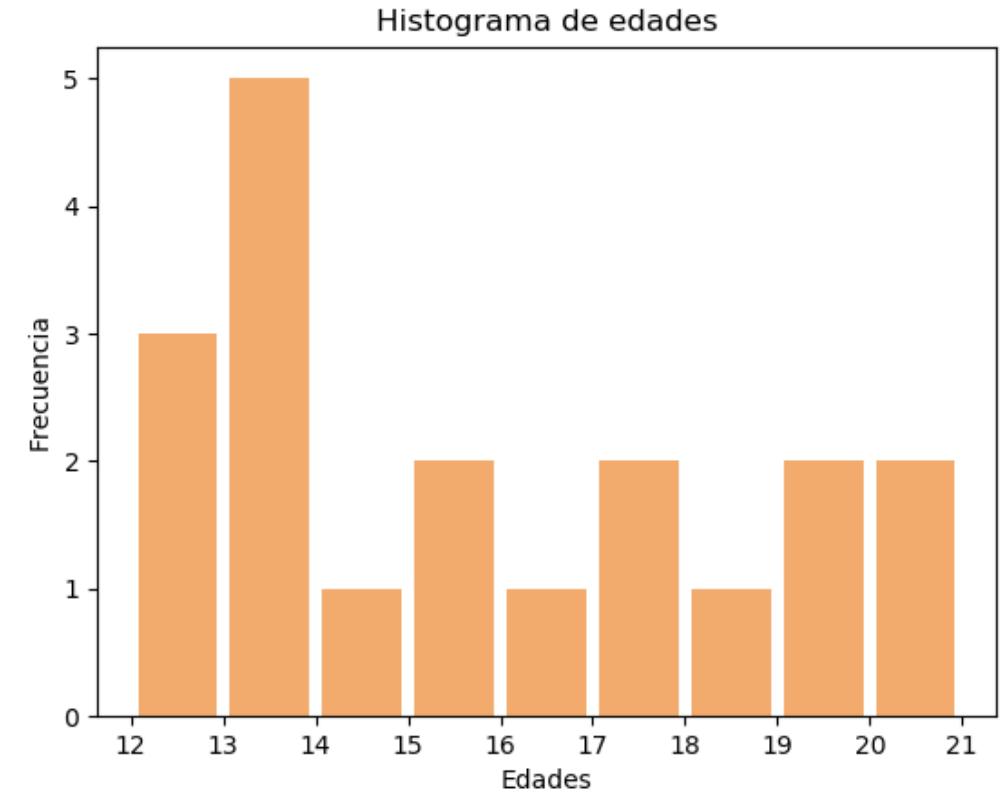
Es una representación textual de las frecuencias todavía no es un histograma.

Para que sea un histograma tenemos que hacer una representación con barras (o algo similar) donde las barras son proporcionales a las frecuencias.

# Histograma



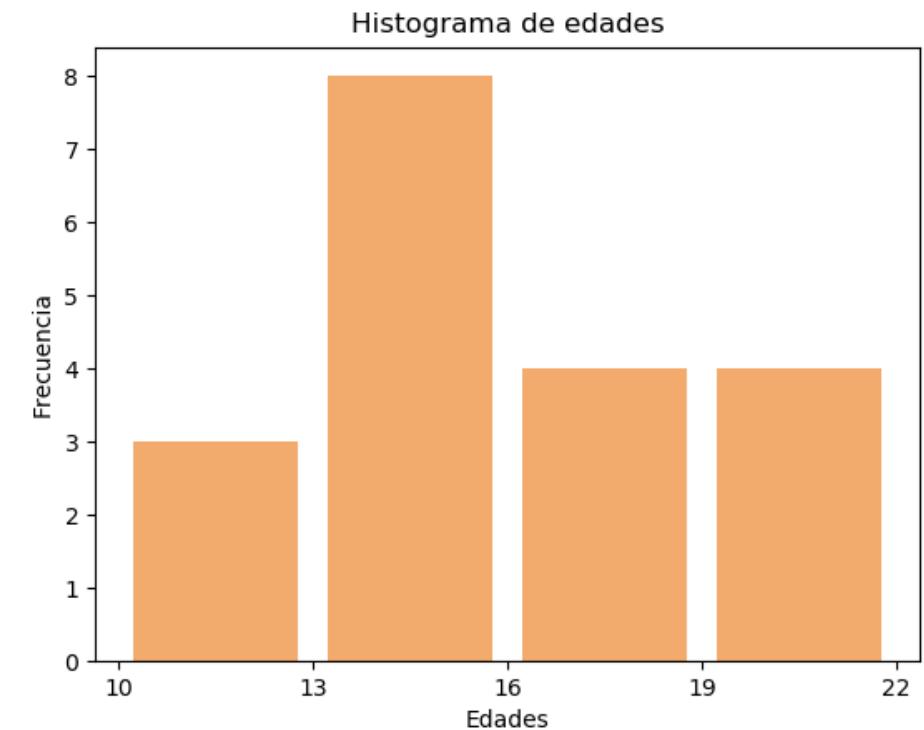
```
import matplotlib.pyplot as plot
edades = [12, 15, 13, 12, 18, 20, 19, 20, 13, 12, 13, 17, 15, 16, 13,
14, 13, 17, 19]
intervalos = range(min(edades), max(edades) + 2)
#calculamos los extremos de los intervalos
plot.hist(x=edades, bins=intervalos, color='#F2AB6D',
rwidth=0.85)
plot.title('Histograma de edades')
plot.xlabel('Edades')
plot.ylabel('Frecuencia')
plot.xticks(intervalos)
plot.show() #dibujamos el histograma
```



# Histograma

Si queremos calcular las frecuencias para otros intervalos solo tenemos que proporcionar mediante el parámetro bins una lista con los extremos de los intervalos deseados de la siguiente manera:

```
import matplotlib.pyplot as plot
edades = [12, 15, 13, 12, 18, 20, 19, 20, 13, 12, 13, 17, 15, 16, 13,
14, 13, 17, 19]
intervalos = [10, 13, 16, 19, 22] #indicamos los extremos de
los intervalos
plot.hist(x=edades, bins=intervalos, color='#F2AB6D',
rwidth=0.85,)
plot.title('Histograma de edades')
plot.xlabel('Edades')
plot.ylabel('Frecuencia')
plot.xticks(intervalos)
plot.show() #dibujamos el histograma
```

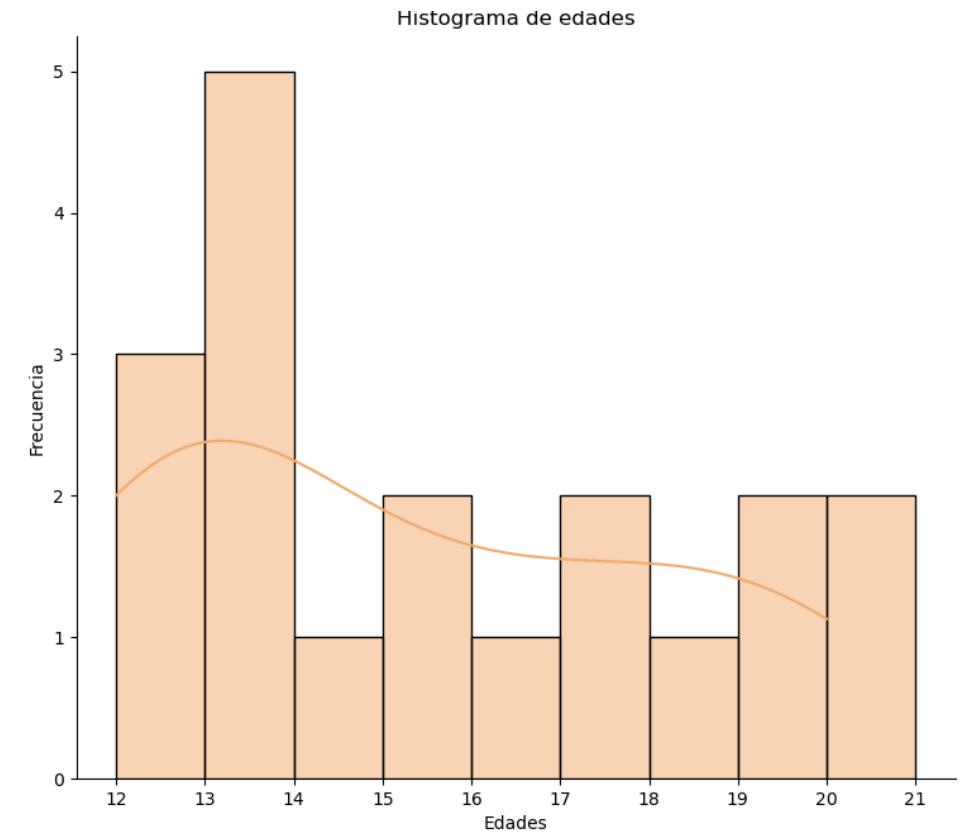


# Histograma

```

import matplotlib.pyplot as plot
import seaborn as sb
edades = [12, 15, 13, 12, 18, 20, 19, 20, 13, 12, 13, 17, 15, 16, 13,
14, 13, 17, 19]
intervalos = range(min(edades), max(edades) + 2)
sb.displot(edades, color='#F2AB6D', bins=intervalos,
kde=True) #Creamos el gráfico en Seaborn
#Configuramos en Matplotlib
plot.xticks(intervalos)
plot.ylabel('Frecuencia')
plot.xlabel('Edades')
plot.title('Histograma de edades')
plot.show()

```

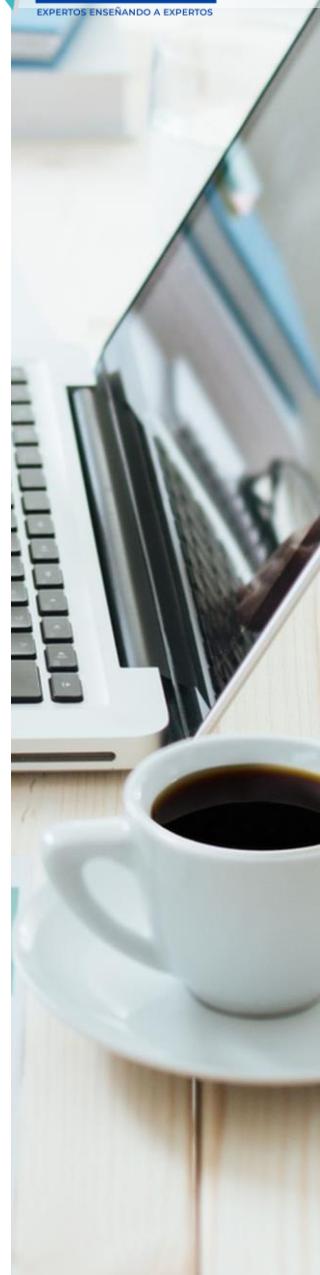


# Resumen del capítulo

- Es una librería ideal para estudiantes de ciencias e ingeniería.
- Nos permite desarrollar aplicaciones para el ámbito científico y tecnológico.

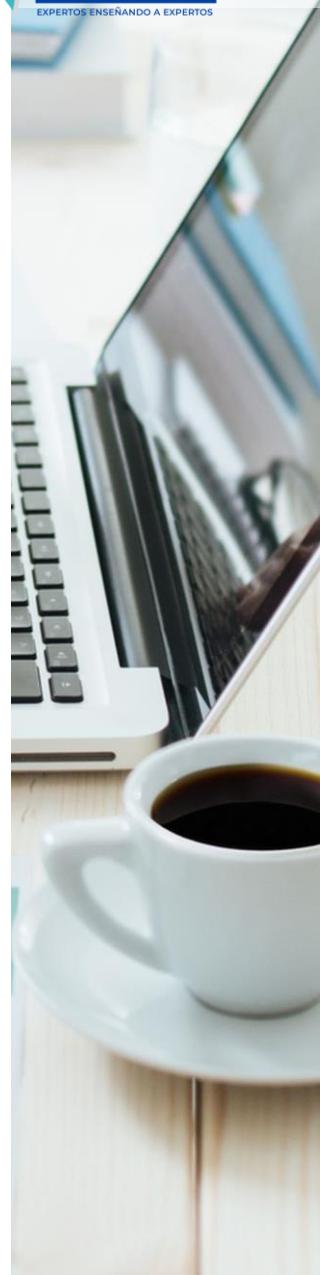
# Actividad del capítulo

- Ejercicio 1. Escribir un programa que pregunte al usuario por las ventas de un rango de años y muestre por pantalla un diagrama de líneas con la evolución de las ventas.
- Ejercicio 2. Escribir una función que reciba un diccionario con las notas de las asignaturas de un curso y una cadena con el nombre de un color y devuelva un diagrama de barras de las notas en el color dado.
- Ejercicio 3. Escribir una función que reciba una serie de Pandas con las notas de los alumnos de un curso y devuelva un diagrama de cajas con las notas. El diagrama debe tener el título “Distribución de notas”.



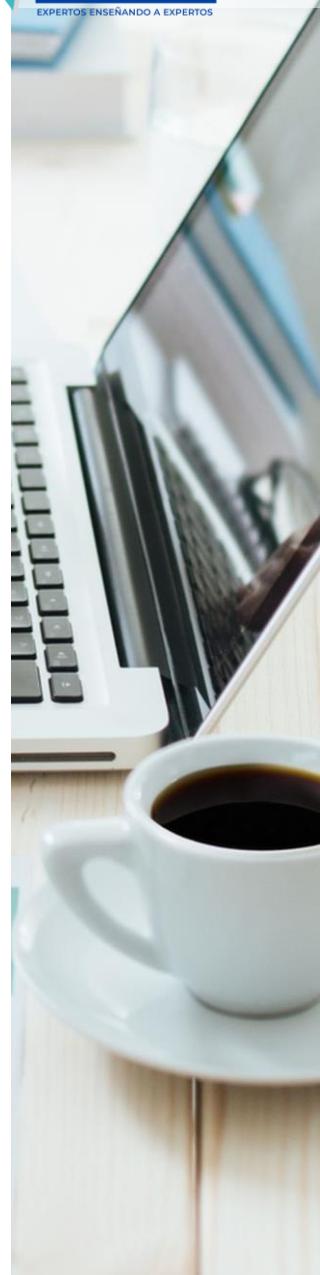
# Actividad del capítulo

- Solución 1
- import matplotlib.pyplot as plt
- inicio = int(input('Introduce el año inicial: '))
- fin = int(input('Introduce el año final: '))
- ventas = {}
- for i in range(inicio, fin+1):
  - ventas[i] = float(input('Introduce las ventas del año ' + str(i) + ': '))
- fig, ax = plt.subplots()
- ax.plot(ventas.keys(), ventas.values())
- plt.show()



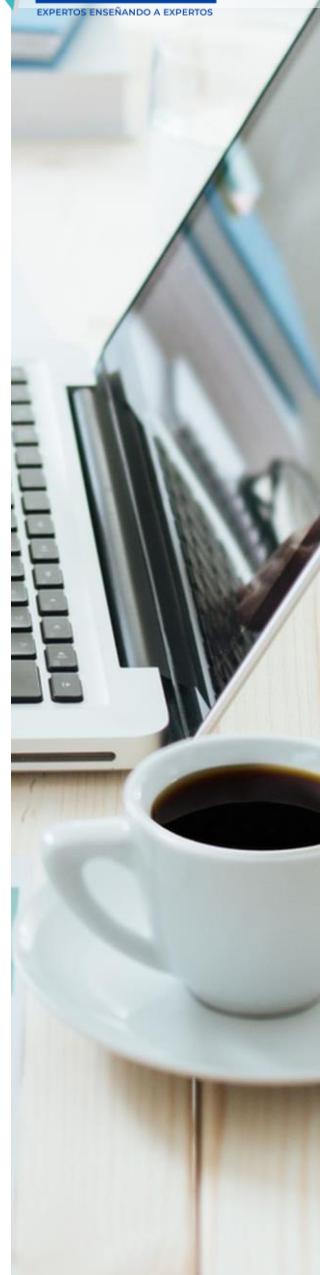
# Actividad del capítulo

- Solución 2
- import matplotlib.pyplot as plt
- def diagrama\_barras\_notas(notas, color):
- fig, ax = plt.subplots()
- ax.bar(notas.keys(), notas.values(), color = color)
- return ax
- notas = {'Programación':9, 'Mates':6.5, 'Economía':4, 'Historia': 8}
- diagrama\_barras\_notas(notas, 'orange')
- plt.show()



# Actividad del capítulo

- Solución 3
- import pandas as pd
- import matplotlib.pyplot as plt
- def diagrama\_caja\_notas(notas):
  - fig, ax = plt.subplots()
  - notas.plot(kind = 'box', ax = ax)
  - plt.xticks([])
  - plt.title('Distribución de notas')
  - return ax
- notas = [4, 8, 7.5, 6, 5.5, 5.2, 3.5, 7.7, 3.2, 9, 6.8]
- s\_notas = pd.Series(notas)
- diagrama\_caja\_notas(s\_notas)
- plt.show()



## Referencia Bibliográfica

- <https://docs.scipy.org/doc/>
- <https://www.youtube.com/watch?v=cJelxUK-oDQ>
- <https://riptutorial.com/es/home>

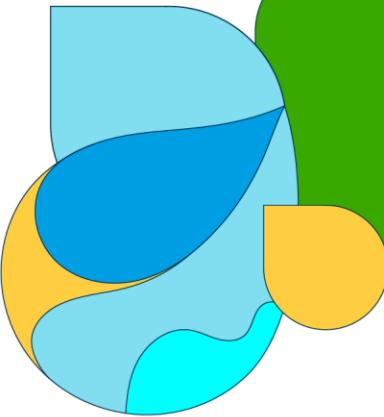


# Unidad temática 15

Librerías para Ciencia de Datos: SciPy

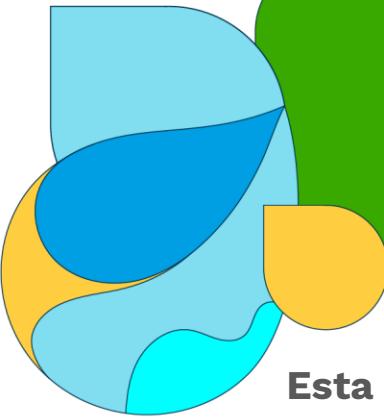
## Objetivos:

- Manejar matemáticas avanzadas al igual que con MATLAB, IDL, Octave, R-Lab y SciLab.
- Ser Introducido a la ciencia de datos y hacer uso de la librería con conocimiento de causa.
- Poder generar aplicaciones científicas donde se haga uso de los métodos numéricos e interactuar con software de álgebra lineal como octave.



# Introducción

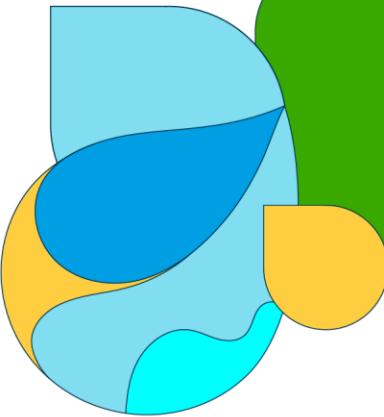
SciPy es un software de código abierto para matemáticas, ciencias e ingeniería. La biblioteca SciPy depende de NumPy, que proporciona una manipulación de matrices N-dimensional conveniente y rápida. La biblioteca SciPy está diseñada para trabajar con matrices NumPy y proporciona muchas rutinas numéricas fáciles de usar y eficientes, como rutinas para integración y optimización numérica. Juntos, se ejecutan en todos los sistemas operativos populares, se instalan rápidamente y son gratuitos. NumPy y SciPy son fáciles de usar, pero lo suficientemente potentes como para que algunos de los principales científicos e ingenieros del mundo dependan de ellos. Si necesita manipular números en una computadora y mostrar o publicar los resultados, ¡pruebe SciPy!



# Introducción

**Esta librería está formada por subpaquetes donde cada uno ejecuta cálculos específicos:**

- Algebra lineal -> linalg
- Procesamiento de señales -> signal
- Funciones estadísticas -> stats
- Funciones especiales -> special
- Integración -> integrate
- Herramientas de interpolación -> interpolate
- Herramientas de optimización -> optimize
- Algoritmos de transformada de Fourier -> fftpack –
- Entrada y salida de datos -> io
- Wrappers a la librería LAPACK -> lib.lapack
- Wrappers a la librería BLAS -> lib.blas
- Wrappers a librerías externas -> lib
- Matrices sparse -> sparse
- otras utilidades -> misc
- Vector Quantization / Kmeans -> cluster
- Ajuste a modelos con máxima entropía -> maxentropy



## linalg.solve

$$7x + 2y = 8$$

$$4x + 5y = 10$$

La integramos al siguiente código:

```
from scipy import linalg
import numpy as np
a = np.array([[7, 2], [4, 5]])
b = np.array([8, 10])
res = linalg.solve(a, b)
print(res)
```

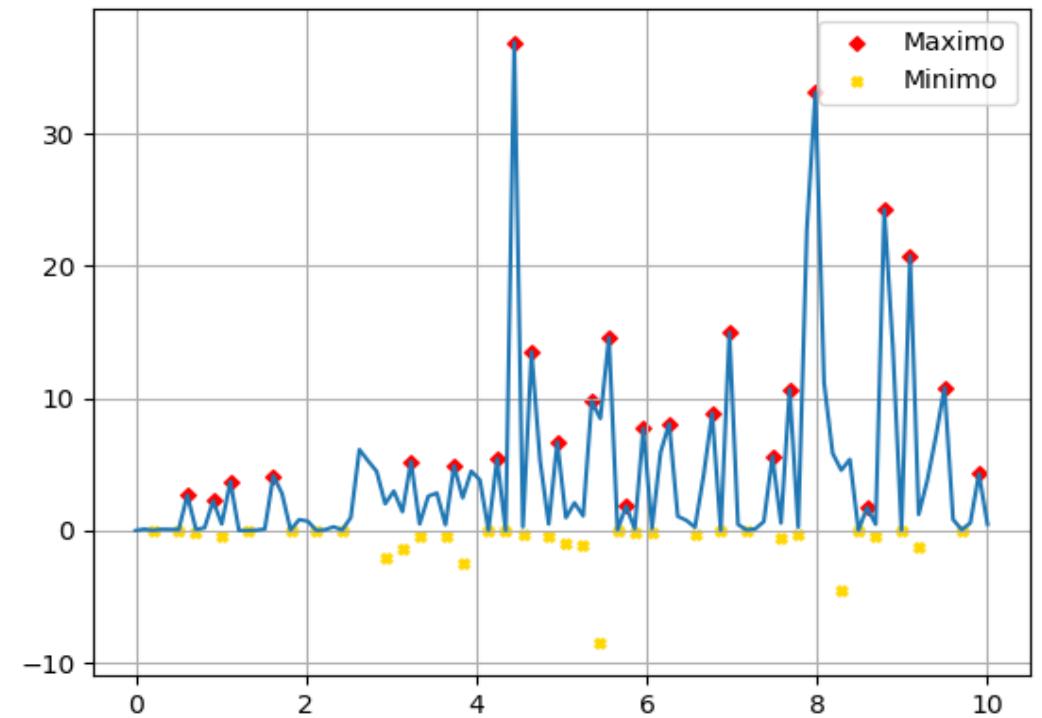
Al ejecutar: \$ python3 scipy01.py  
[0.74074074 1.40740741]

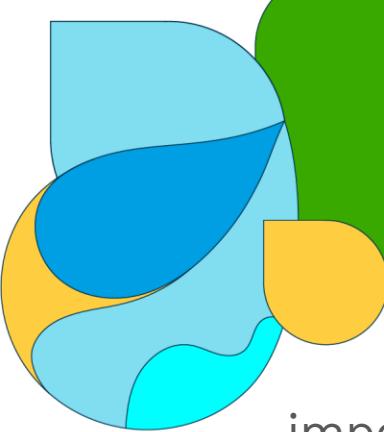
# scipy.signal

```

from cProfile import label
from turtle import color
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import find_peaks
#Creamos una función aleatoria para analizar
x = np.linspace(0, 10, 100)
y = x*np.random.randn(100)**2
#determinar picos
picos = find_peaks(y, height=1, threshold=1,distance=1)
#print(piclos)
altura = picos[1]['peak_heights']
posicion_piclos = x[picos[0]]
#determinar minimo
y2 = y*-1 # y la multiplicamos por -1 para obtener una imagen reflejada y hacer el ejercicio mas interesante.
minimos = find_peaks(y2)
posicion_minimos = x[minimos[0]] # lista de posiciones minimo
altura_minimos = y2[minimos[0]] # lista de altura minima reflejados
trazado
fig = plt.figure()
ax = fig.subplots()
ax.plot(x,y)
ax.scatter(posicion_piclos,altura , color='r', s = 15, marker = 'D', label='Maximo')
ax.scatter(posicion_minimos,altura_minimos, color='gold', s = 15, marker = 'X', label='Minimo')
ax.legend()
ax.grid()
plt.show()

```





# scipy.stats

```
import numpy as np
import scipy
from scipy import stats
arr = np.array([[2, 5, 6, 8],
[3, 7, 3, 0],
[1, 1, 4, 4],
[9, 5, 0, 5],
[6, 4, 2, 2]])
result = scipy.stats.sem(arr)
print("El error estándar de los datos dados es:\n", result)
```

El error estándar de los datos dados es:

[1.46287388 0.9797959 1. 1.356466 ]

# scipy.integrate

```
from scipy.integrate import quad
def f(x):
 return 3.0*x*x + 1.0
I, err = quad(f, 0, 1)
print(I)
print(err)
```

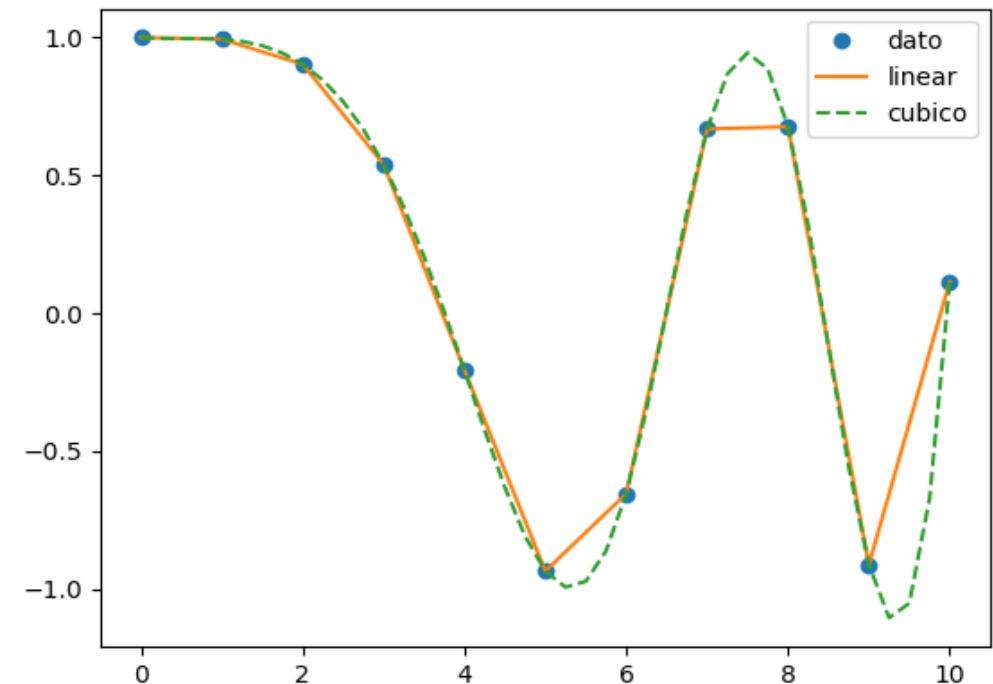
El resultado que nos arroja es:

2.0

2.220446049250313e-14

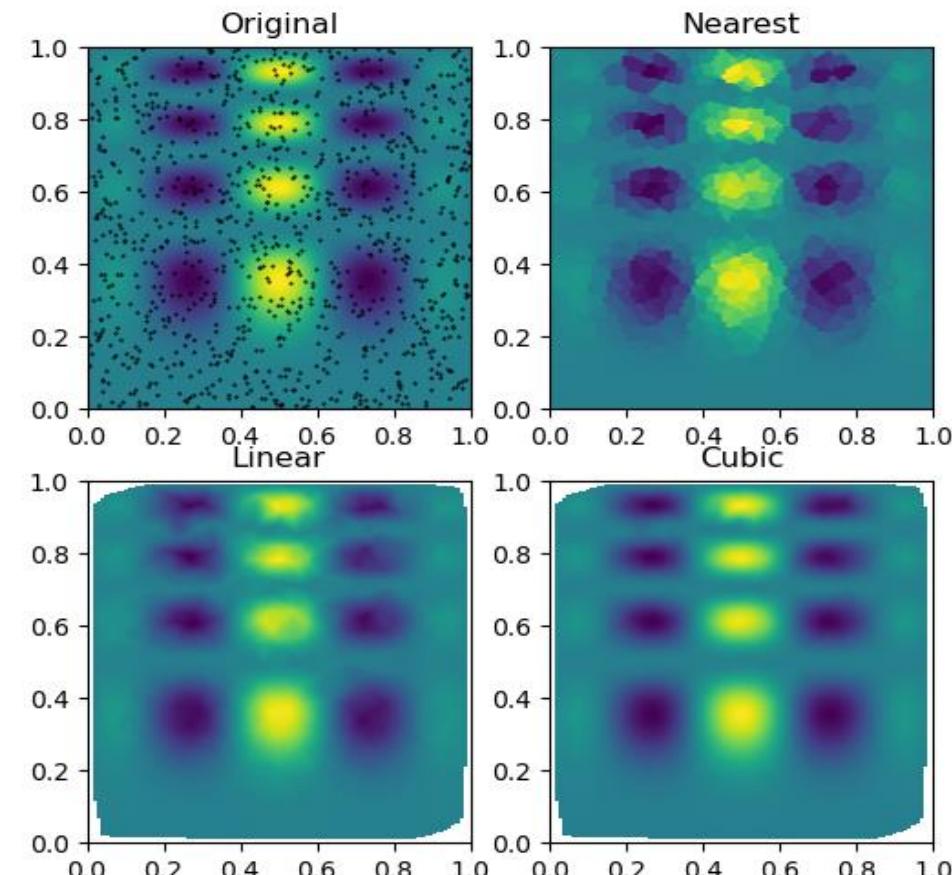
# scipy.interpolate

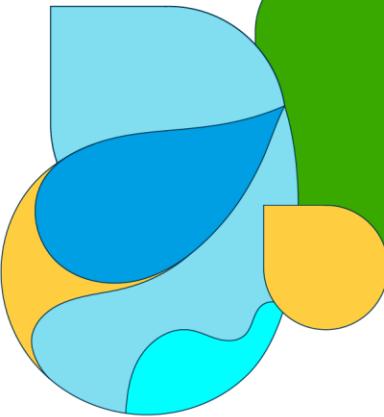
```
import numpy as np
from scipy.interpolate import interp1d
x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/9.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 10, num=41, endpoint=True)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), '-',
 xnew, f2(xnew), '--')
plt.legend(['dato', 'linear', 'cubico'], loc='best')
plt.show()
```



# interpolar en 2-D

```
import numpy as np
from scipy.interpolate import griddata
import matplotlib.pyplot as plt
def func(x, y):
 return x*(1-x)*np.cos(4*np.pi*x) * np.sin(4*np.pi*y**2)**2
grid_x, grid_y = np.mgrid[0:1:100j, 0:1:200j]
rng = np.random.default_rng()
points = rng.random((1000, 2))
values = func(points[:,0], points[:,1])
grid_z0 = griddata(points, values, (grid_x, grid_y), method='nearest')
grid_z1 = griddata(points, values, (grid_x, grid_y), method='linear')
grid_z2 = griddata(points, values, (grid_x, grid_y), method='cubic')
plt.subplot(221)
plt.imshow(func(grid_x, grid_y).T, extent=(0,1,0,1), origin='lower')
plt.plot(points[:,0], points[:,1], 'k.', ms=1)
plt.title('Original')
plt.subplot(222)
plt.imshow(grid_z0.T, extent=(0,1,0,1), origin='lower')
plt.title('Nearest')
plt.subplot(223)
plt.imshow(grid_z1.T, extent=(0,1,0,1), origin='lower')
plt.title('Linear')
plt.subplot(224)
plt.imshow(grid_z2.T, extent=(0,1,0,1), origin='lower')
plt.title('Cubic')
plt.gcf().set_size_inches(6, 6)
plt.show()
```

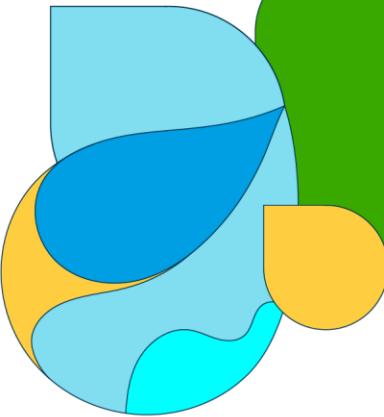




# scipy.optimize

```
import numpy as np
from scipy.optimize import minimize
def rosen(x):
 """La función Rosenbrock"""
 return sum(100.0*(x[1:]-x[:-1]**2.0)**2.0 +
 (1-x[:-1])**2.0)
x0 = np.array([1.3, 0.7, 0.8, 1.9, 1.2])
res = minimize(rosen, x0,
 method='nelder-mead', options={'xatol':
 1e-8, 'disp': True})
print(res.x)
```

Optimization terminated successfully.  
Current function value: 0.000000  
Iterations: 339  
Function evaluations: 571  
[1. 1. 1. 1. 1.]

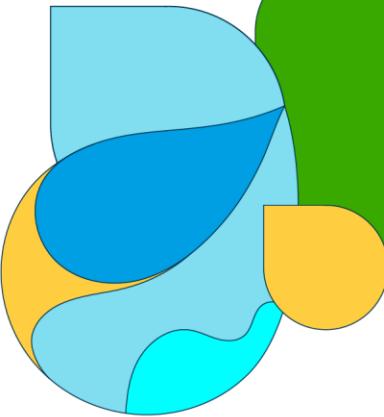


# scipy.fftpack

```
import numpy as np
from scipy.fftpack import fft , ifft
x = np.array([1.0 , 2.0 , 1.0 , - 1.0 , 1.5])
y = fft (x)
print(y)
print(np.sum(x))
```

Resultado:

```
[4.5 -0.j 2.08155948-
 1.65109876j -1.83155948+1.60822041j
 -1.83155948-
 1.60822041j 2.08155948+1.65109876j]
4.5
np.sum(x) es 4.5
```

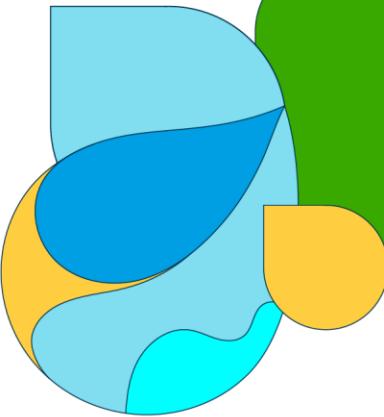


# scipy.io

SciPy tiene muchos módulos, clases y funciones disponibles para leer y escribir datos en una variedad de formatos de archivo. Ejemplo octave

```
octave:1> a = 1:12
a =
 1 2 3 4 5 6 7 8 9 10 11 12
octave:2> a = reshape(a, [1 3 4])
a =
ans(:,:,1) =
 1 2 3
ans(:,:,2) =
 4 5 6
ans(:,:,3) =
 7 8 9
ans(:,:,4) =
 10 11 12
octave:3> save -6 octave_a.mat a % MATLAB 6 compatible
octave:4> ls octave_a.mat
octave_a.mat
octave:5> quit
```

```
>>> import scipy.io as sio
>>> mat_contents = sio.loadmat('octave_a.mat')
>>> mat_contents
{'__header__': b'MATLAB 5.0 MAT-file, written by Octave 6.4.0,
2022-09-13 17:02:57 UTC', '__version__': '1.0', '__globals__': [],
'a': array([[[1., 4., 7., 10.,
 2., 5., 8., 11.,
 3., 6., 9., 12.]]])}
>>> oct_a = mat_contents['a']
>>> oct_a
array([[[1., 4., 7., 10.,
 2., 5., 8., 11.,
 3., 6., 9., 12.]]])
```



# Funciones especiales

Las funciones especiales de scipy se utilizan para realizar operaciones matemáticas con los datos dados. La función especial en scipy es un módulo disponible en el paquete scipy. Dentro de esta función especial, los métodos disponibles son:

cbrt - da la raíz cúbica del número dado.

peine - da las combinaciones de los elementos.

exp10 - da el número con aumento a la potencia 10 del número dado.

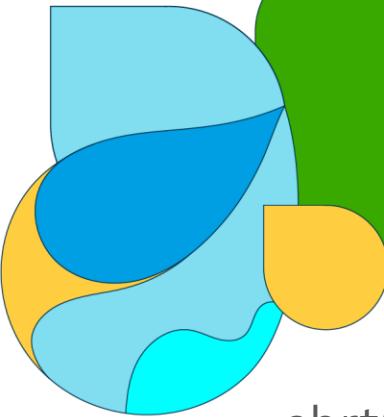
exprl - da el error relativo exponencial,  $(\exp(x) - 1) / x$ .

gamma : devuelve el valor calculando  $z * \text{gamma}(z) = \text{gamma}(z + 1)$  y  $\text{gamma}(n + 1) = n !$ , para un número natural 'n'.

lambertw : calcula  $W(z) * \exp(W(z))$  para cualquier número complejo z, donde W es la función lambertw.

logsumexp - da el logaritmo de la suma de exponencial de un número dado.

perm - da las permutaciones de los elementos.



# cbrt()

cbrt()

**Se utiliza para devolver la raíz cúbica del número dado.**

Sintaxis: cbrt (número)

Ejemplo: programa para encontrar la raíz cúbica

```
>>>
```

```
>>> from scipy.special import cbrt
```

```
>>> print(cbrt(64))
```

```
4.0
```

```
>>> print(cbrt(78))
```

```
4.272658681697917
```

```
>>> print(cbrt(128))
```

```
5.039684199579493
```

```
>>>
```

# peine()

**Se conoce como combinaciones y devuelve la combinación de un valor dado.**

Sintaxis: `scipy.special.comb (N, k)`

Donde, N es el valor de entrada y k es el número de repeticiones.

Ejemplo:

```
>>>
>>> from scipy.special import comb
>>> print(comb(4,1))
4.0
>>>
```

# exp10()

**Este método da el número con una potencia de 10 del número dado.**

Sintaxis: exp10 (valor)

Donde valor es el número que se da como entrada.

Ejemplo: programa para encontrar la potencia de 10

```
>>>
>>> from scipy.special import exp10
>>> print(exp10(2))
100.0
>>>
```

# exprel()

**Se conoce como función exponencial de error relativo.**

Devuelve el valor de error para una variable dada. Si x está cerca de cero, entonces  $\exp(x)$  está cerca de 1.

Sintaxis: `scipy.special.exprel (input_data)`

Ejemplo:

```
>>>
>>> from scipy.special import exprel
>>> print(exprel(0))
1.0
>>>
```

# gamma()

**Se conoce como función Gamma. Es el factorial generalizado**

ya que  $z * \text{gamma}(z) = \text{gamma}(z + 1)$  y  $\text{gamma}(n + 1) = n !$ ,  
para un número natural 'n'.

Sintaxis: `scipy.special.gamma (input_data)`

Donde, datos de entrada es el número de entrada.

Ejemplo:

```
>>>
>>> from scipy.special import gamma
>>> print(gamma(56))
1.2696403353658278e+73
>>>
```

# lambertw()

**También se conoce como función de Lambert.**

Calcula que el valor de  $W(z)$  es tal que  $z = W(z) * \exp(W(z))$  para cualquier número complejo  $z$ , donde  $W$  se conoce como la función de Lambert

Sintaxis: `scipy.special.lambertw (input_data)`

Ejemplo:

```
>>>
```

```
>>> from scipy.special import lambertw
>>> print([lambertw(1),lambertw(0),lambertw(56),
lambertw(68),lambertw(10)])
[(0.5671432904097838+0j), 0j, (2.9451813101206707+0j),
(3.0910098540499797+0j), (1.7455280027406994+0j)]
>>>
```

# permanent()

**La permanente representa la permutación. Devolverá la permutación de los números dados.**

Sintaxis: `scipy.special.perm (N, k)` donde N es el valor de entrada y k es el número de repeticiones.

Ejemplo:

```
>>>
>>> from scipy.special import perm
>>> print([perm(4, 1), perm(4, 2), perm(4, 3), perm(4, 4), perm(4, 5)])
[4.0, 12.0, 24.0, 24.0, 0.0]
>>> print([perm(6, 1), perm(6, 2), perm(6, 3), perm(6, 4), perm(6, 5)])
[6.0, 30.0, 120.0, 360.0, 720.0]
>>>
```

## Resumen del capítulo

- Es una librería ideal para estudiantes de ciencias e ingeniería.
- Nos permite desarrollar aplicaciones para el ámbito científico y tecnológico.
- Nos permite resolver problemas de investigación de operaciones.

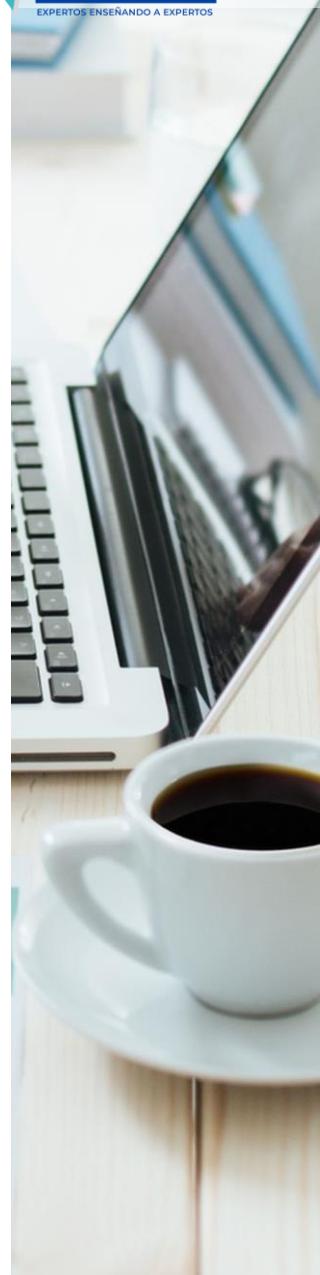
# Actividad del capítulo

## Ajustar una función a los datos de un histograma.

Supongamos que hay un pico de datos distribuidos normalmente (gaussianos) (media: 3.0, desviación estándar: 0.3) en un fondo en descomposición exponencial. Esta distribución se puede ajustar con `curve_fit` en unos pocos pasos:

- 1.) Importar las bibliotecas requeridas.
- 2.) Definir la función de ajuste que se ajustará a los datos.
- 3.) Obtener datos del experimento o generar datos. En este ejemplo, se generan datos aleatorios para simular el fondo y la señal.
- 4.) Añadir la señal y el fondo.
- 5.) Ajustar la función a los datos con `curve_fit`.
- 6.) (Opcionalmente) Graficar los resultados y los datos.

En este ejemplo, los valores de  $y$  observados son las alturas de los intervalos de histogramas, mientras que los valores de  $x$  observados son los centros de los binscenters histogramas ( `binscenters` ). Es necesario pasar el nombre de la función de ajuste, los valores  $x$  y los valores  $y$  a `curve_fit` . Además, con  $p0$  se puede proporcionar un argumento opcional que contiene estimaciones aproximadas para los parámetros de ajuste. `curve_fit` devuelve `popt` y `pcov` , donde `popt` contiene los resultados de ajuste para los parámetros, mientras que `pcov` es la matriz de covarianza, cuyos elementos diagonales representan la varianza de los parámetros ajustados.



# Actividad del capítulo

## Código de la solución:

```

import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
def fit_function(x, A, beta, B, mu, sigma):

 return (A * np.exp(-x/beta) + B * np.exp(-1.0 * (x -
mu)**2 / (2 * sigma**2)))

data = np.random.exponential(scale=2.0, size=100000)

data2 = np.random.normal(loc=3.0, scale=0.3, size=15000)

bins = np.linspace(0, 6, 61)

data_entries_1, bins_1 = np.histogram(data, bins=bins)
data_entries_2, bins_2 = np.histogram(data2, bins=bins)
data_entries = data_entries_1 + data_entries_2

binscenters = np.array([0.5 * (bins[i] + bins[i+1]) for i in
range(len(bins)-1)])

popt, pcov = curve_fit(fit_function, xdata=binscenters,
ydata=data_entries, p0=[20000, 2.0,
2000, 3.0, 0.3])
print(popt)

```

```

xspace = np.linspace(0, 6, 100000)

plt.bar(binscenters, data_entries, width=bins[1] - bins[0],
color='navy', label=r'Histogram entries')

plt.plot(xspace, fit_function(xspace, *popt),
color='darkorange', linewidth=2.5,
label=r'Fitted function')

plt.xlim(0,6)

plt.xlabel(r'x axis')

plt.ylabel(r'Number of entries')

plt.title(r'Exponential decay with gaussian peak')

plt.legend(loc='best')

plt.show()

plt.clf()

```



## Referencia Bibliográfica

- <https://docs.scipy.org/doc/>
- <https://www.youtube.com/watch?v=cJelxUK-oDQ>
- <https://riptutorial.com/es/home>

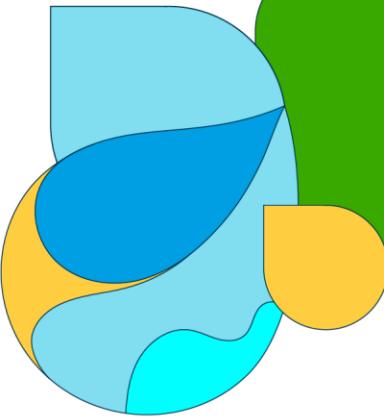


# Unidad temática 16

Librerías para Ciencia de Datos:  
Scikit-learn

## **Objetivos:**

- Comprender el concepto de Machine Learning.
- Conocer y aprovechar las Apis disponibles para el entrenamiento de los proyectos que requiera realizar.
- Conocer y practicar los diferentes métodos de evaluación para tomar la mejor decisión.



# Introducción

Scikit-learn es considerada la librería más útil para Machine Learning en Python, es de código abierto y es reutilizable en varios contextos, fomentando el uso académico y comercial.

Proporciona una gama de algoritmos de aprendizaje supervisados y no supervisados en Python.

Esta librería está construida sobre SciPy (Scientific Python) e incluye las siguientes librerías o paquetes:

- NumPy: librería de matriz n-dimensional base.
- Pandas: estructura de datos y análisis.
- SciPy: librería fundamental para la informática científica.
- Matplotlib: trazado completo 2D.
- Ipython: consola interactiva mejorada.
- SymPy: matemática simbólica.

# Bases de estimaciones

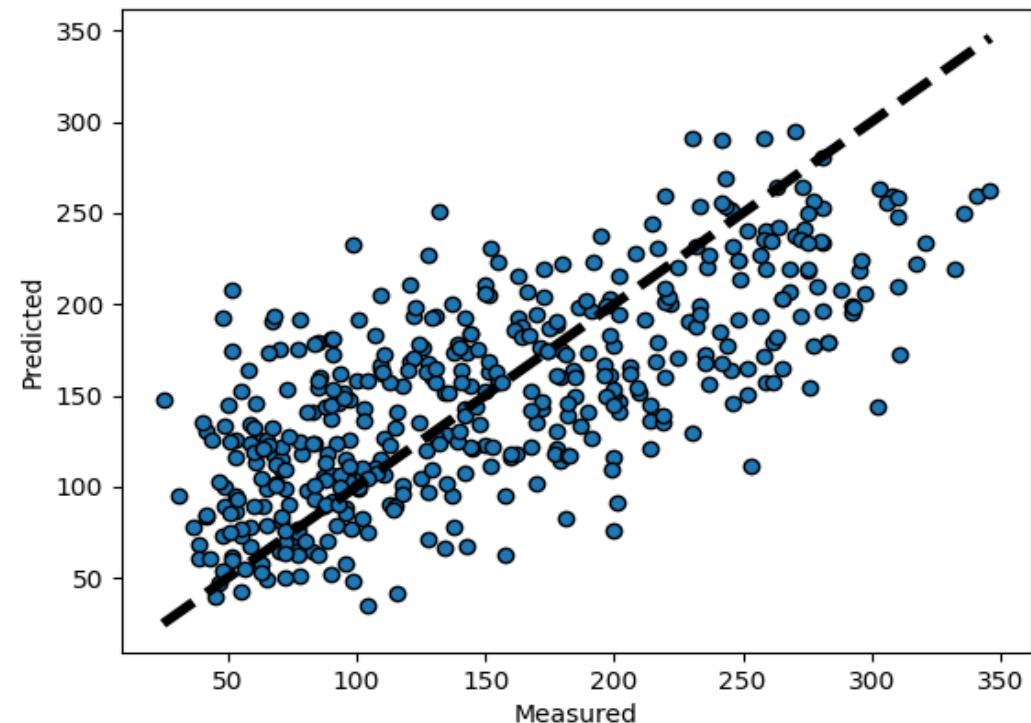
Datos importantes:

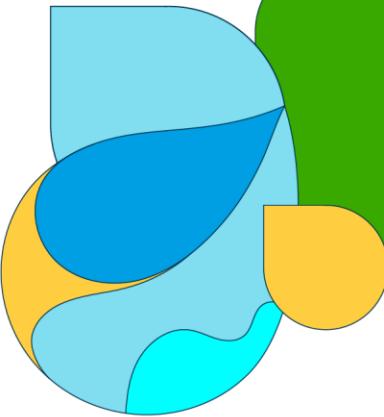
- Como fue diseñado Scikit-Learn, existe una API muy fácil de ocupar y muy unificada. Esta unificación se da por un objeto que se llama “estimador” que tiene en todos los casos y para el algoritmo de Machine Learning que sea, una API que es común y 3 métodos que son clave.
- Scikit-Learn posee muchos modelos, se pueden implementar tanto, regresiones lineales como regresiones regularizadas, árboles de decisión, SDMs, etc.
- Scikit-Learn contiene todos los modelos que son usados hoy en día, y una de las virtudes de esta librería es que sigue muy de cerca lo que pasa en la investigación.
- Scikit-Learn es la librería más usada de Machine Learning General, no de Machine Learning Especializado, para ello está la librería de Tensor Flow y sirve casi exclusivamente para modelos de Deep Learning.

# Un ejemplo

```
from sklearn import datasets
from sklearn.model_selection import cross_val_predict
from sklearn import linear_model
import matplotlib.pyplot as plt

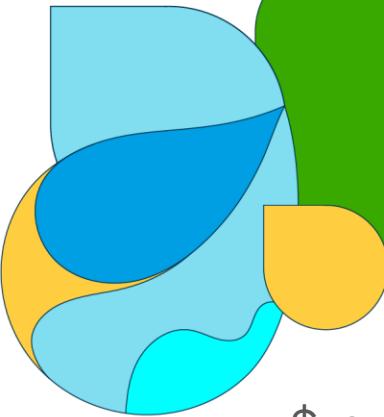
lr = linear_model.LinearRegression()
X, y = datasets.load_diabetes(return_X_y=True)
cross_val_predict devuelve una matriz del mismo tamaño
que `y` donde cada entrada
es una predicción obtenida por validación cruzada:
predicted = cross_val_predict(lr, X, y, cv=10)
fig, ax = plt.subplots()
ax.scatter(y, predicted, edgecolors=(0, 0, 0))
ax.plot([y.min(), y.max()], [y.min(), y.max()], 'k--', lw=4)
ax.set_xlabel('Measured')
ax.set_ylabel('Predicted')
plt.show()
```





# pre-procesadores

- En scikit-learn, los preprocesadores y transformadores siguen la misma API que los objetos del estimador (todos en realidad heredan de la misma clase `BaseEstimator`). Los objetos transformadores no tienen un método `predict` sino un método `transform` que produce una matriz de ejemplo recién transformada `X`:
- En un transformador, transforma la entrada, normalmente sólo `X`, en algún espacio transformado (convencionalmente anotado como `Xt`). La salida es un arreglo o matriz dispersa de longitud `n_samples` y con el número de columnas fijado después del ajuste.



# Un ejemplo

```
$ python3
Python 3.9.14 (main, Sep 7 2022, 23:43:48)
[GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for
more information.

>>> from sklearn.preprocessing import
StandardScaler
>>> X = [[0,15],[1, -10]]
>>> StandardScaler().fit(X).transform(X)
array([-1., 1.,
 [1., -1.]])
```

# Ejemplo

```
>>> import pandas as pd
>>> import sklearn as skl
>>> url =
"https://raw.githubusercontent.com/AprendeConEjemplos/
aprendizaje-automatico-con-scikit-
learn/main/04_Proposito/Stars.csv"
>>> dataframe = pd.read_csv(url)
>>> print(dataframe)
```

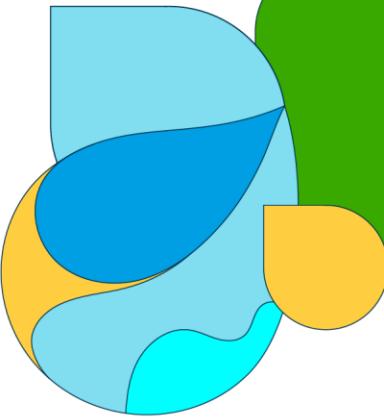
|     |       | Temperature    | L             | R         | A_M   | Color |    |
|-----|-------|----------------|---------------|-----------|-------|-------|----|
|     |       | Spectral_Class | Type          |           |       |       |    |
| 0   | 0     | 3068           | 0.002400      | 0.1700    | 16.12 | Red   | M  |
| 1   | 0     | 3042           | 0.000500      | 0.1542    | 16.60 | Red   | M  |
| 2   | 0     | 2600           | 0.000300      | 0.1020    | 18.70 | Red   | M  |
| 3   | 0     | 2800           | 0.000200      | 0.1600    | 16.65 | Red   | M  |
| 4   | 0     | 1939           | 0.000138      | 0.1030    | 20.06 | Red   | M  |
| ..  | ..    | ..             | ..            | ..        | ..    | ..    | .. |
| 235 | 9.93  | 38940          | 374830.000000 | 1356.0000 | -     |       |    |
|     | Blue  |                | O             | 5         |       |       |    |
| 236 | 10.63 | 30839          | 834042.000000 | 1194.0000 | -     |       |    |
|     | Blue  |                | O             | 5         |       |       |    |
| 237 | 10.73 | 8829           | 537493.000000 | 1423.0000 | -     |       |    |
|     | White |                | A             | 5         |       |       |    |
| 238 | 11.23 | 9235           | 404940.000000 | 1112.0000 | -     |       |    |
|     | White |                | A             | 5         |       |       |    |
| 239 | 7.80  | 37882          | 294903.000000 | 1783.0000 | -     |       |    |
|     | Blue  |                | O             | 5         |       |       |    |

# Ejemplo

```
>>> dataset = dataframe.drop("Type", axis=1)
>>> label = dataframe["Type"].copy()
>>> dataset
```

|                | Temperature | L             | R         | A_M   | Color |     |
|----------------|-------------|---------------|-----------|-------|-------|-----|
| Spectral_Class |             |               |           |       |       |     |
| 0              | 3068        | 0.002400      | 0.1700    | 16.12 | Red   | M   |
| 1              | 3042        | 0.000500      | 0.1542    | 16.60 | Red   | M   |
| 2              | 2600        | 0.000300      | 0.1020    | 18.70 | Red   | M   |
| 3              | 2800        | 0.000200      | 0.1600    | 16.65 | Red   | M   |
| 4              | 1939        | 0.000138      | 0.1030    | 20.06 | Red   | M   |
| ..             | ...         | ...           | ...       | ...   | ...   | ... |
| 235            | 38940       | 374830.000000 | 1356.0000 | -     |       |     |
| 9.93           | Blue        | O             |           |       |       |     |
| 236            | 30839       | 834042.000000 | 1194.0000 | -     |       |     |
| 10.63          | Blue        | O             |           |       |       |     |
| 237            | 8829        | 537493.000000 | 1423.0000 | -     |       |     |
| 10.73          | White       | A             |           |       |       |     |
| 238            | 9235        | 404940.000000 | 1112.0000 | -     |       |     |
| 11.23          | White       | A             |           |       |       |     |
| 239            | 37882       | 294903.000000 | 1783.0000 | -     |       |     |
| 7.80           | Blue        | O             |           |       |       |     |

[240 rows x 6 columns]



# Pipelines

Entubados (pipelines) Scikit-learn.

fuente: <https://scikit-learn.org/stable/modules/compose.html>

Los transformadores generalmente se combinan con clasificadores, regresores u otros estimadores para construir un estimador compuesto. La herramienta más común es un Pipeline . Pipeline se usa a menudo en combinación con FeatureUnion, que concatena la salida de los transformadores en un espacio de características compuesto. TransformedTargetRegressor se ocupa de transformar el objetivo (es decir, log-transform y ). Por el contrario, Pipelines solo transforma los datos observados ( X ).

# Ejemplo

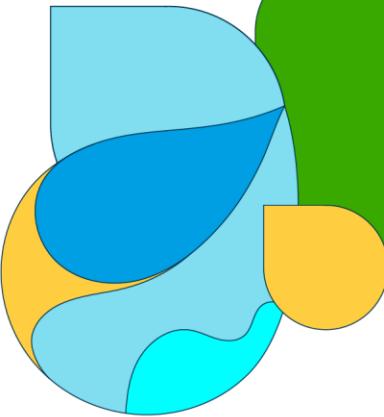
## Construcción

El Pipeline se construye usando una lista de pares, que es una cadena que contiene el nombre que le quieres dar a este paso y es un objeto estimador:(key, value)keyvalue

```
>>> from sklearn.pipeline import Pipeline
>>> from sklearn.svm import SVC
>>> from sklearn.decomposition import PCA
>>> estimators = [('reduce_dim', PCA()), ('clf',
SVC())]
>>> pipe = Pipeline(estimators)
>>> pipe
Pipeline(steps=[('reduce_dim', PCA()), ('clf',
SVC())])
```

La función de utilidad make\_pipeline es una forma abreviada de construir tuberías; toma un número variable de estimadores y devuelve una canalización, completando los nombres automáticamente:

```
>>> from sklearn.pipeline import make_pipeline
>>> from sklearn.naive_bayes import
MultinomialNB
>>> from sklearn.preprocessing import Binarizer
>>> make_pipeline(Binarizer(), MultinomialNB())
Pipeline(steps=[('binarizer', Binarizer()),
('multinomialnb', MultinomialNB())])
```

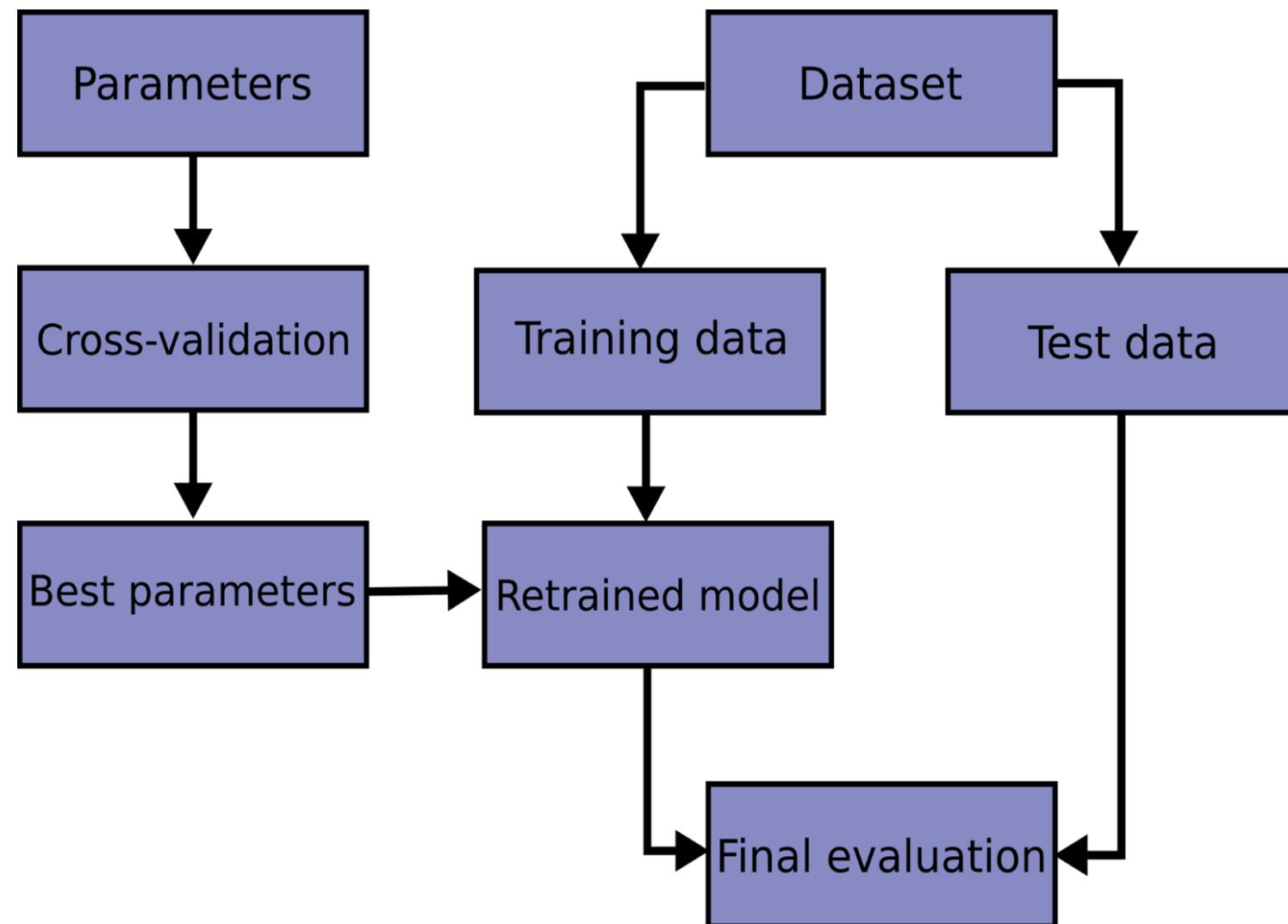


# Evaluación del modelo

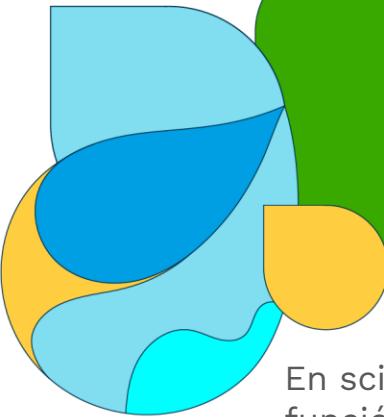
## **Validación cruzada: evaluación del rendimiento del estimador.**

Aprender los parámetros de una función de predicción y probarla sobre los mismos datos es un error metodológico: un modelo que se limitase a repetir las etiquetas de las muestras que acaba de ver tendría una puntuación perfecta, pero no lograría predecir nada útil sobre datos aún no vistos. Esta situación se llama sobreajuste. Para evitarla, es práctica común cuando se realiza un experimento de aprendizaje automático (supervisado), el mantener una parte de los datos disponibles como un conjunto de prueba ```X_test, y_test`''. Ten en cuenta que la palabra «experimento» no pretende denotar un uso académico únicamente, ya que incluso en entornos comerciales el aprendizaje automático suele comenzar de forma experimental. Este es un diagrama de flujo del proceso de trabajo típico de validación cruzada en el entrenamiento de modelos. Los mejores parámetros pueden determinarse mediante las técnicas de búsqueda en cuadrícula.

# Evaluación del modelo



# Evaluación del modelo



En scikit-learn una división aleatoria en conjuntos de entrenamiento y prueba puede ser rápidamente calculada con la función de ayuda `train_test_split`. Carguemos el conjunto de datos del iris para ajustar una máquina de vectores de soporte lineal en él:

```
>>> import numpy as np
>>> from sklearn.model_selection import train_test_split
>>> from sklearn import datasets
>>> from sklearn import svm
>>> X, y = datasets.load_iris(return_X_y=True)
>>> X.shape, y.shape
((150, 4), (150,))
```

Ahora podemos hacer un muestreo rápido de un conjunto de entrenamiento y reservar el 40% de los datos para probar (evaluar) nuestro clasificador:

```
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
>>> X_train.shape, y_train.shape
((90, 4), (90,))
>>> X_test.shape, y_test.shape
((60, 4), (60,))
>>> clf = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
>>> clf.score(X_test, y_test)
0.9666666666666667
```

# Resumen del capítulo

- Se abarcaron los temas necesarios para poder modelar proyectos.
- El uso de APIs se vuelve fundamental para el entrenamiento de modelos.
- Quedaron claros los métodos de evaluación de modelos.

# Actividad del capítulo

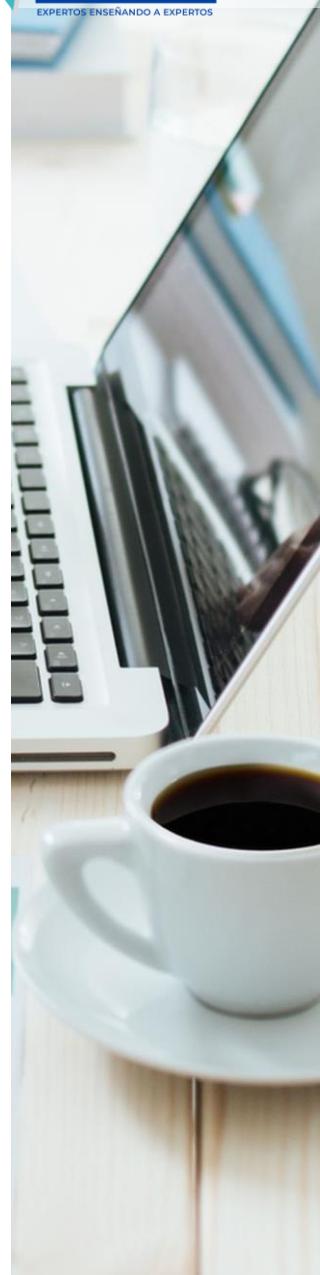
- Con tres ejercicios entrenaremos modelos.



# 1er ejercicio

## Primer ejemplo

El primer ejemplo ilustra cómo el estimador robusto del Determinante de Covarianza Mínimo puede ayudar a concentrarse en un conglomerado relevante cuando existen puntos periféricos. En este caso, la estimación de la covarianza empírica está sesgada por puntos fuera del clúster principal. Por supuesto, algunas herramientas de cribado habrían señalado la presencia de dos conglomerados (Máquinas de vectores de Soporte, modelos de mezclas gaussianas, detección de valores atípicos univariantes, ...). Pero si se tratara de un ejemplo de alta dimensión, ninguna de ellas podría aplicarse tan fácilmente.

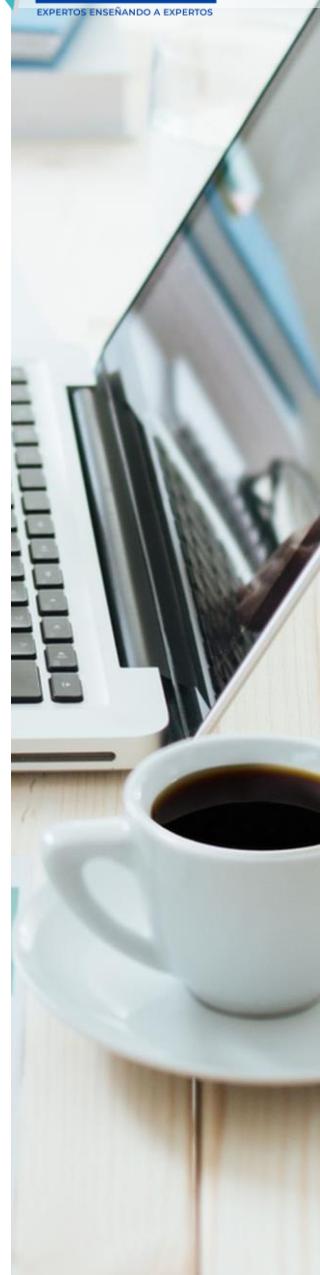
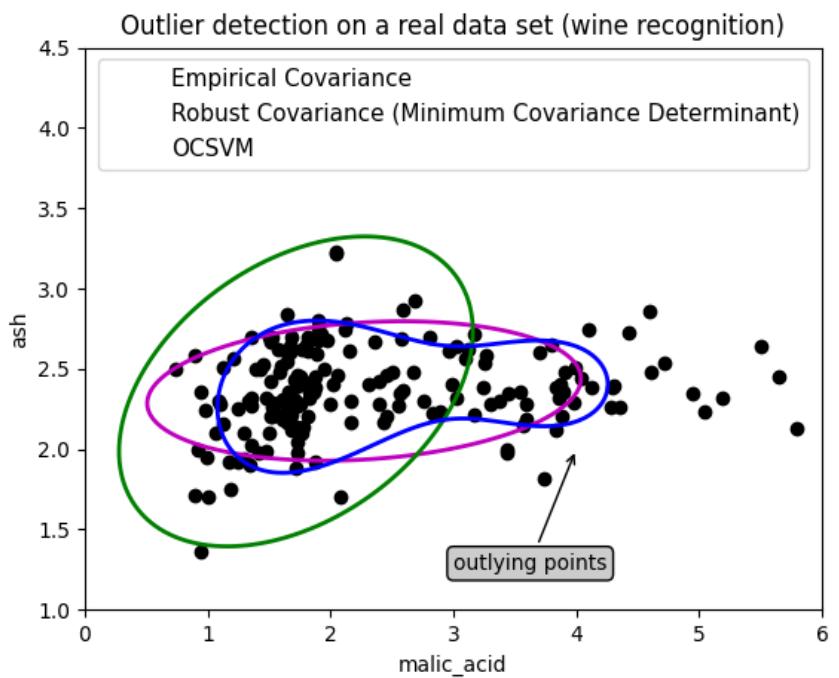


# 1er ejercicio

```

from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.datasets import load_wine
Definir "clasificadores" a utilizar
classifiers = {
 "Empirical Covariance":
 EllipticEnvelope(support_fraction=1.,
 contamination=0.25),
 "Robust Covariance (Minimum Covariance Determinant)": EllipticEnvelope(contamination=0.25),
 "OCSVM": OneClassSVM(nu=0.25, gamma=0.35)}
...

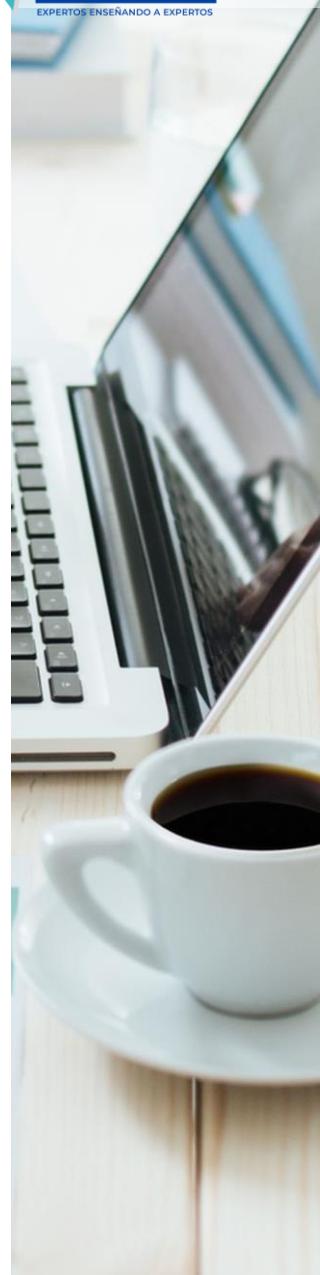
```



## 2do. ejercicio

### Segundo ejemplo

El segundo ejemplo muestra la capacidad del estimador robusto del Determinante Mínimo de la Covarianza para concentrarse en el modo principal de la distribución de los datos: la localización parece estar bien estimada, aunque la covarianza es difícil de estimar debido a la distribución en forma de plátano. De todos modos, podemos deshacernos de algunas observaciones periféricas. La SVM de una clase es capaz de capturar la estructura real de los datos, pero la dificultad estriba en ajustar su parámetro de ancho de banda del núcleo para obtener un buen compromiso entre la forma de la matriz de dispersión de los datos y el riesgo de sobreajuste de los datos.



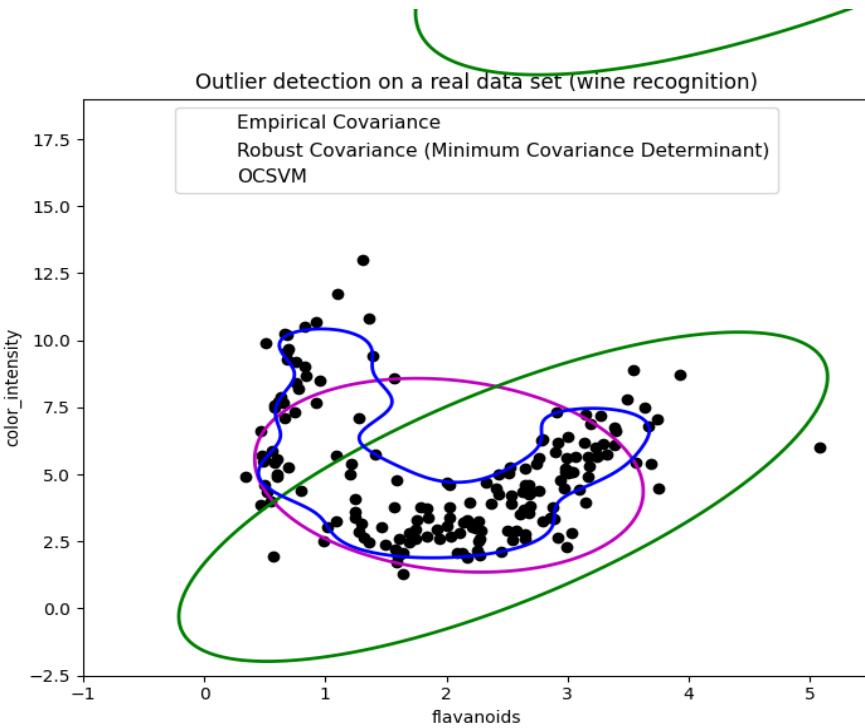
## 2do. ejercicio

```
print(__doc__)

Author: Virgile Fritsch <virgile.fritsch@inria.fr>
License: BSD 3 clause

import numpy as np
from sklearn.covariance import EllipticEnvelope
from sklearn.svm import OneClassSVM
import matplotlib.pyplot as plt
import matplotlib.font_manager
from sklearn.datasets import load_wine
Define "classifiers" to be used

...
...
```



## 3er. ejercicio

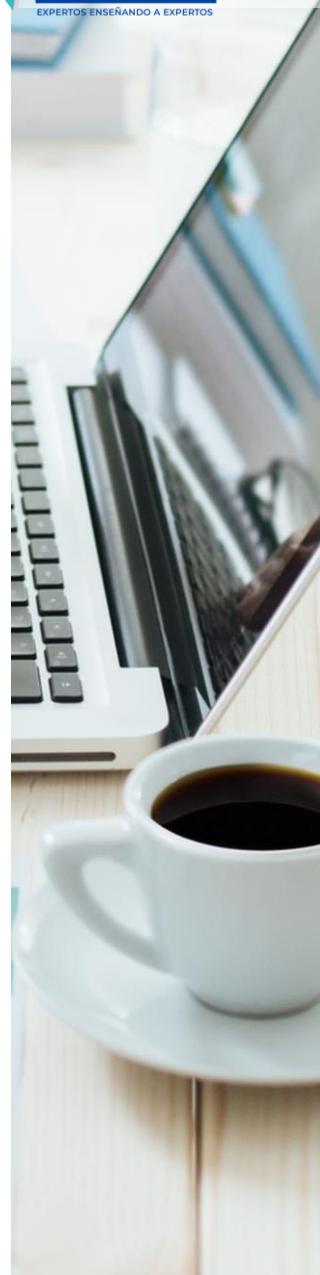
Tercer ejemplo

Ejemplo de reconocimiento de rostros mediante eigenfaces y SVM.

El conjunto de datos utilizado en este ejemplo es un extracto preprocesado de «Labeled Faces in the Wild», también conocido como LFW:

[http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz\(233MB\)](http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz(233MB))

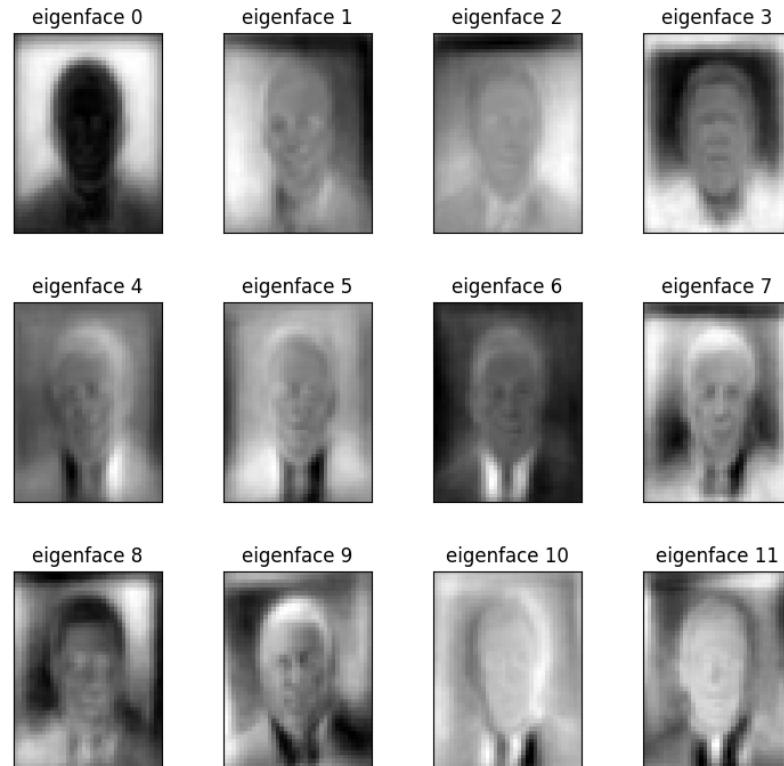
Resultados esperados para las 5 personas más representadas en el conjunto de datos:



## 3er. ejercicio

```
from time import time
import logging
import matplotlib.pyplot as plt

from sklearn.model_selection import
train_test_split
from sklearn.model_selection import
GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```



## Referencia Bibliográfica

- [https://qu4nt.github.io/scikit-learn-doc-es/auto\\_examples/index.html](https://qu4nt.github.io/scikit-learn-doc-es/auto_examples/index.html)
- <https://aprendeia.com/libreria-scikit-learn-de-python/>
- <https://scikit-learn.org/stable/>
- <https://scikit-learn.org/stable/modules/compose.html>
- [https://qu4nt.github.io/scikit-learn-doc-es/auto\\_examples/applications/plot\\_outlier\\_detection\\_wine.html#sphx-glr-auto-examples-applications-plot-outlier-detection-wine-py](https://qu4nt.github.io/scikit-learn-doc-es/auto_examples/applications/plot_outlier_detection_wine.html#sphx-glr-auto-examples-applications-plot-outlier-detection-wine-py)



GOBIERNO DE COLOMBIA

# Gracias



@SENACOMUNICA

[www.sena.edu.co](http://www.sena.edu.co)

Las acciones de formación ejecutadas en el marco de la convocatoria  
DG 0001 - 2023 son gratuitas para los trabajadores beneficiarios