

PYTHON



Inteligencia Artificial

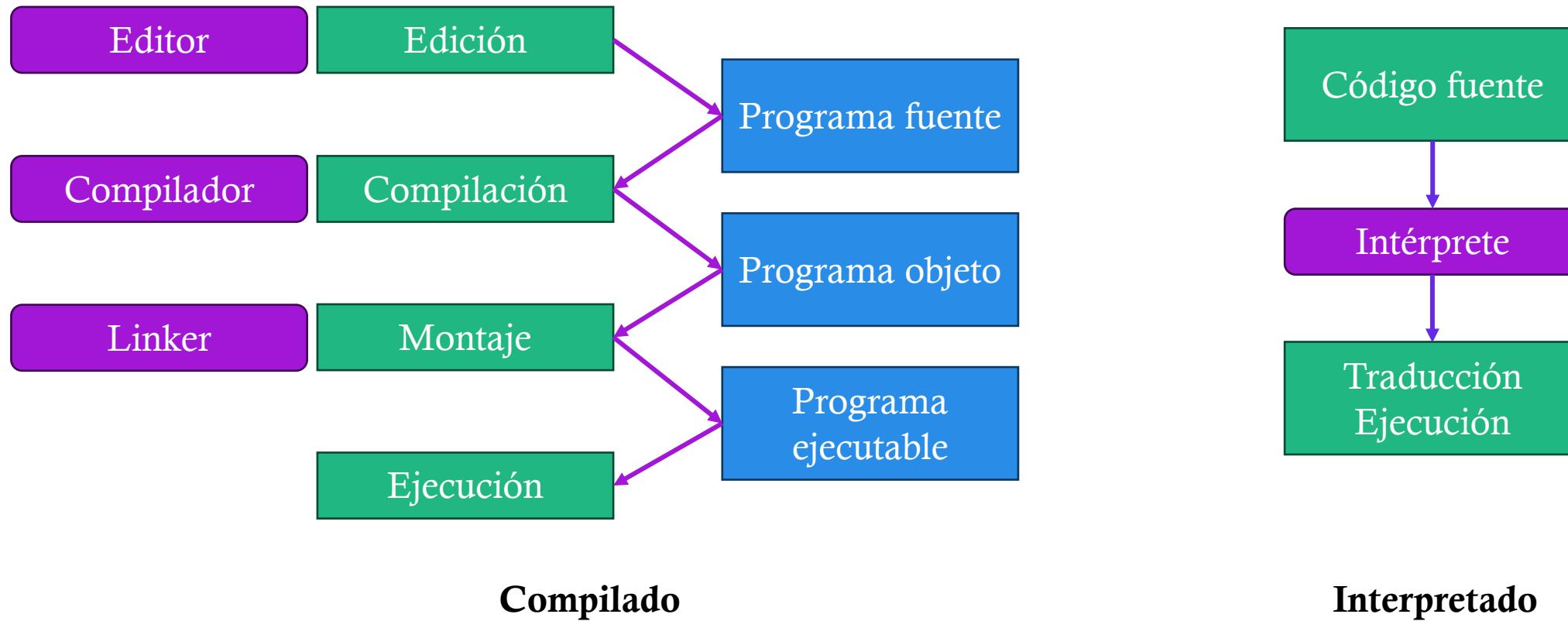
CEIA - FIUBA

Dr. Ing. Facundo Adrián
Lucianna

PYTHON

- Python es un lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.
- Python es un lenguaje de programación **multiparadigma**. Permite varios estilos: **programación orientada a objetos**, **programación imperativa** y **programación funcional**.
- OBS: En el fondo, Python es un lenguaje orientado a objetos, todo, absolutamente todo es un objeto.
- Python usa tipado dinámico y conteo de referencias para la gestión de memoria.
- Python reemplazó en gran medida a LISP en IA, principalmente por ser multiparadigma.

PYTHON



PYTHON

Python es uno de los lenguajes más usados en Ciencia de datos. ¿Por qué?

- Porque tiene una sintaxis simple y es fácil de adaptar para quienes no vienen de ambientes de ingeniería o ciencia de la informática.

PYTHON

Python es famoso por ser lento comparado con lenguajes como C++, por qué se usa en Machine Learning o IA?

- La respuesta es que no se usa librerías hechas Python. Ninguna de las bibliotecas que se utilizan está realmente escrita en Python.
- Casi siempre están escritos en Fortran o C++ y simplemente interactúan con Python a través de algún wrapper.
- La velocidad de Python es irrelevante si solo se interactúa con las librerías escritas en un C++ altamente optimizado.

Fuente: <https://qr.ae/pKrGdr>

PYTHON

Modo interactivo

- Python posee un **modo interactivo**: Se escriben las instrucciones en una especie de intérprete de comandos. Las expresiones pueden ser introducidas una a una.

```
Python 3.10.4 (main, ) [GCC 11.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> var = "Hola, mundo"
>>> print(var)
Hola, mundo
>>>
```

PYTHON

Modo interactivo

- **iPython** (Parte de SciPy): Extiende la capacidad del modo interactivo y provee un kernel para Jupyter

```
:~$ ipython3
Python 3.10.4 (main, ) [GCC 11.2.0]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.31.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: var = "Hola, mundo"

In [2]: print(var)
Hola, mundo

In [3]: 
```

PYTHON

Jupyter Notebook

- Es un entorno computacional interactivo basado en la web para crear documentos de notebook.
- Jupyter Notebook es similar a la interfaz de notebook de otros programas como Maple, Mathematica y SageMath, un estilo de interfaz computacional que se originó con **Mathematica** en la década de 1980.



PYTHON

Jupyter Notebook

jupyter Read Ingredients and products example Last Checkpoint: 09/08/2022 (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

In [1]:

```
import pandas as pd
import numpy as np

pd.set_option("max_colwidth", None)
pd.set_option("max_seq_items", None)
pd.set_option("display.max_columns", None)
pd.set_option('display.max_rows', None)
```

In [2]:

```
def explode_product_df(df: pd.DataFrame, col: str, add_internal_id:bool = False, add_sku_parent: bool = True,
                      multiply:bool = False, quantity_col: str = "quantity", order_column: bool = False) -> tuple:

    if not col in df.columns:
        return -1, pd.DataFrame()

    df_temp = df.copy()

    if add_internal_id:
        df_temp["parent_internal_id"] = 0
        df_temp["internal_id"] = np.arange(1, df_temp.shape[0] + 1)
        counter_id = df_temp.shape[0] + 1

    first_level = df_temp.drop([col], axis=1)
    first_level[col] = np.nan

    df_list = [first_level]

    def add_data_from_parent_to_list(lists, col, data):
        for dic in lists:
            dic[col] = data
        return lists
```



PYTHON

Google Colab

Si por algún motivo no podés correr Python de local o tienes un setup malo, podes usar Google Colab en la nube.

Google Colab permite escribir y ejecutar Python en el navegador:

- Sin configurar
- Fácil de compartir
- Acceso a GPUs sin cargo

Es una Jupyter Notebook que corre en una máquina virtual de Google Cloud:

- Es gratuito
- Ofrece 12 GB de RAM y 100 GB de disco.
- Las notebooks quedan en Google Drive, fácil de compartir.



PYTHON

Google Colab

The screenshot shows the Google Colab interface with a Jupyter notebook titled "1 Ejercicios Python Roncedo.ipynb". The notebook contains three exercises:

- Ejercicio 1:**

```
[ ] print ("Hola, estoy aprendiendo Python")
```

Hola, estoy aprendiendo Python
- Ejercicio 2:**

```
[ ] print ("Sobre el puente de Avignon\n    Todos bailan, todos bailan\n\nSobre el puente de Avignon\n    Todos bailan y yo tambien")
```

Sobre el puente de Avignon
Todos bailan, todos bailan

Sobre el puente de Avignon
Todos bailan y yo tambien
- Ejercicio 3:**

```
[ ] var1 = 1
```

```
[ ] var2 = 3.14
```

```
[ ] var3 = -8
```

```
[ ] var4 = "LINTEC"
```





VARIABLES EN PYTHON

VARIABLES

- Específico y sensible a mayúsculas y minúsculas.
- Llamar al valor a través del nombre de la variable

Tipos de variables:

- Numéricas: Integer, float, complejos
- Caracteres: String
- Lógicas: Bool
- Listas
- Tuplas
- Set
- Diccionarios

VARIABLES

- En Python, con `type()` puedo saber que variable es.
- Además, puedo convertir algunas variables en otra: `int()`, `float()`, `str()`, `bool()`, `list()`, `dict()`
- Preguntar el tipo de variable: `isinstance(a, int)`

VARIABLES

¿Qué pasa cuando asignamos una variable?

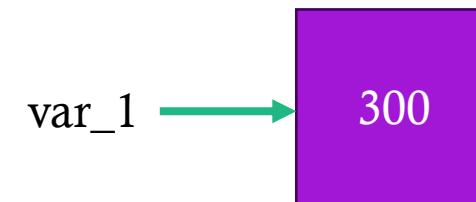
Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.



Código

```
1 var_1 = 300
```

codeboing.com



VARIABLES

¿Qué pasa cuando asignamos una variable?

Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

● ● ● Código

```
1 var_1 = 300
2 var_2 = var_1
```

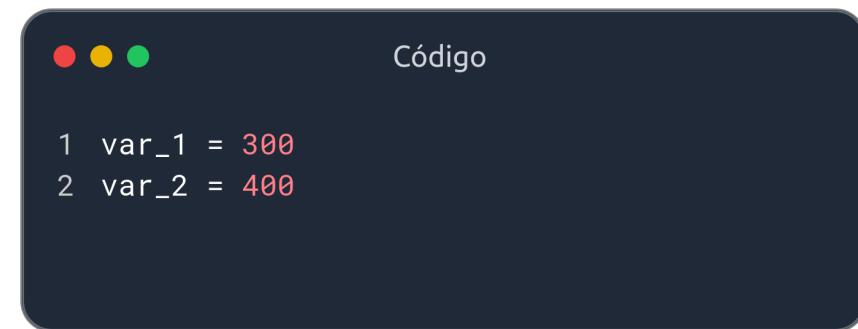
codetobing.com



VARIABLES

¿Qué pasa cuando asignamos una variable?

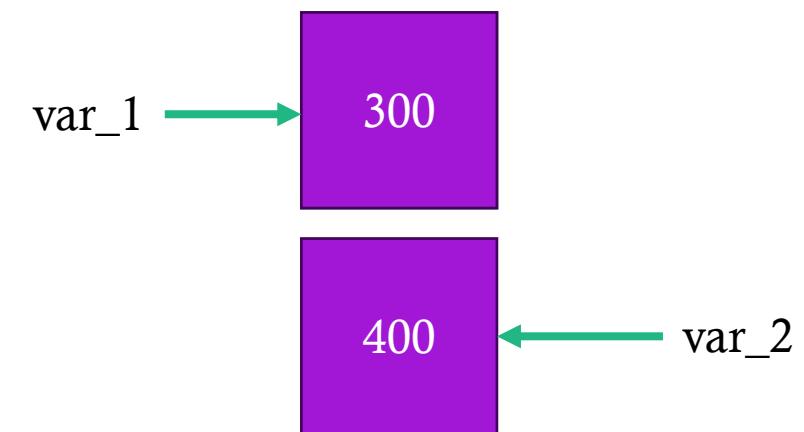
Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.



Código

```
1 var_1 = 300
2 var_2 = 400
```

codetoin.com



VARIABLES

¿Qué pasa cuando asignamos una variable?

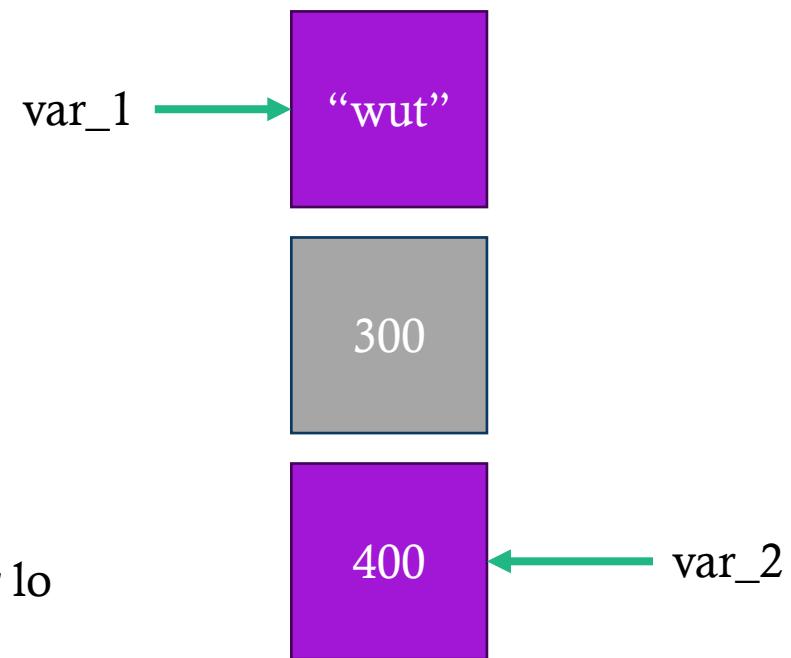
Una variable de Python es un nombre simbólico que es una referencia o puntero a un objeto.

Código

```
1 var_1 = "wut"  
2 var_2 = 400
```

codebtomg.com

300 quedará en memoria RAM hasta que el Garbage Collector lo recolecte o asignamos una nueva variable con 300.



VARIABLES

- **Mutables**: Permiten ser modificadas una vez creados.
- **Inmutables**: No permiten ser modificables una vez creados.

Mutables	Inmutables
Listas	Variables numéricas
Diccionarios	Strings
Sets	Tuplas



OPERADORES

OPERADORES

Aritméticos:

Suma	+	$x + y$
Resta	-	$x - y$
Multiplicación	*	$x * y$
División	/	x / y
Módulo	%	$x \% y$
División entera	//	$x // y$
Exponente	**	$x ** y$

OPERADORES

Comparadores (retornan booleanos):

Mayor	>	$x > y$
Menor	<	$x < y$
Igual	==	$x == y$
Distinto	!=	$x != y$
Mayor o igual	\geq	$x \geq y$
Menor o igual	\leq	$x \leq y$

OPERADORES

Lógicos (solo para booleanos):

and	$x \text{ and } y$
or	$x \text{ or } y$
not	$\text{not } x$

OPERADORES

De asignación:

Asignar	=	$x = y$
Sumar y asigna	$+=$	$x += y \ (x = x + y)$
Resta y asigna	$-=$	$x -= y \ (x = x - y)$
Multiplicación y asigna	$*=$	$x *= y \ (x = x * y)$
Divide y asigna	$/=$	$x /= y \ (x = x / y)$

OPERADORES

Identificadores (devuelven booleanos):

is	x is y
is not	x is not y



FUNCIONES Y LIBRERÍAS

FUNCIONES Y LIBRERÍAS

Llamada a funciones

- Las funciones son llamadas con un nombre y entre paréntesis los argumentos:

```
pow(2,5) # Devuelve 32, es equivalente a 2**5
```

- Esta forma de introducir los argumentos se llama de forma posicional.
- Otra forma de introducir los argumentos es mediante **keys**:

```
pow(exp=5, base=2)
```

FUNCIONES Y LIBRERÍAS

Llamada a funciones

- Algunas funciones tienen argumentosopcionales:

```
pow(2,5, mod=3) # Es equivalente a (2**5) % 3
```

- Los argumentosopcionales siempre son en modo de key.

Built-in Functions de Python: <https://docs.python.org/3/library/functions.html>

FUNCIONES Y LIBRERÍAS

Funciones Built-In importantes

- **print()** - Imprime en pantalla una cadena de strings
- **type()** - Retorna el tipo del objeto/variable.
- **abs()** - Retorna el valor absoluto
- **sorted()** - Retorna una lista ordenada del iterable
- **max()** - Retorna el máximo elemento de un iterable
- **min()** - Retorna el mínimo elemento de un iterable
- **round()** - Retorna un flotante redondeado
- **len()** - Retorna la cantidad de elementos en un objeto/iterable
- **sum()** - Suma todos los elementos de un iterable.
- **help()** - Muestra la documentación del objeto

FUNCIONES Y LIBRERÍAS

Importando librerías externas

La declaración **import** permite hacer visibles identificadores de otros módulos.

- Built-in libraries: <https://docs.python.org/3/library/>

Forma 1

```
● ● ●  
Forma 1  
  
1 # Forma 1  
2 import math  
3  
4 var = math.sqrt(2)  
5 print(math.pi)
```

codemining.com

Forma 2

```
● ● ●  
Forma 2  
  
1 # Forma 2  
2 import math as mt  
3  
4 var = mt.sqrt(2)  
5 print(mt.pi)
```

codemining.com

FUNCIONES Y LIBRERÍAS

Importando librerías externas

La declaración **import** permite hacer visibles identificadores de otros módulos.

- Built-in libraries: <https://docs.python.org/3/library/>

Forma 3

```
● ● ●
Forma 3

1 # Forma 3
2 from math import pi, sqrt
3
4 var = sqrt(2)
5 print(pi)
```

code2img.com

Forma 4

```
● ● ●
Forma 4

1 # Forma 4
2 from math import *
3
4 var = sqrt(2)
5 print(pi)
```

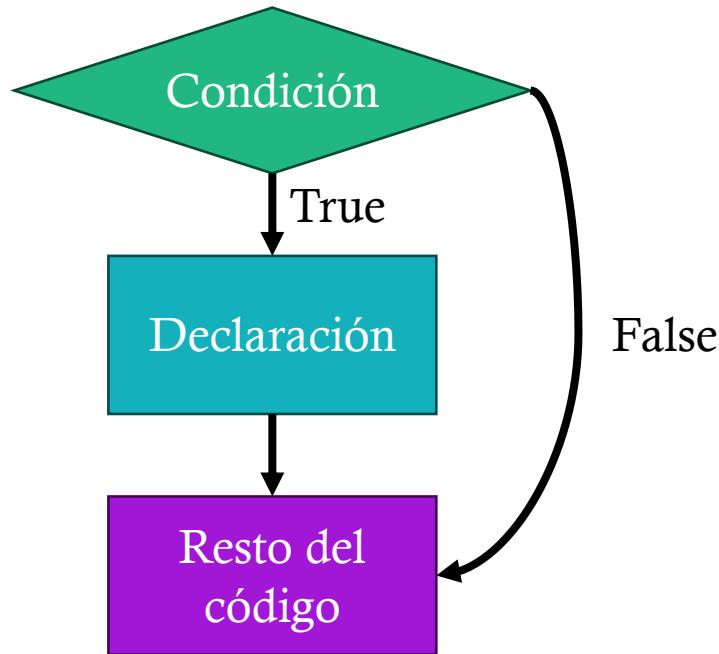
code2img.com



DECLARACIÓN DE CONTROL

FUNCIONES Y LIBRERÍAS

IF



IF

```
1 if <Expresión_booleana>:  
2   <Declaración>  
3  
4 <Resto_del_código>
```

codealong.com

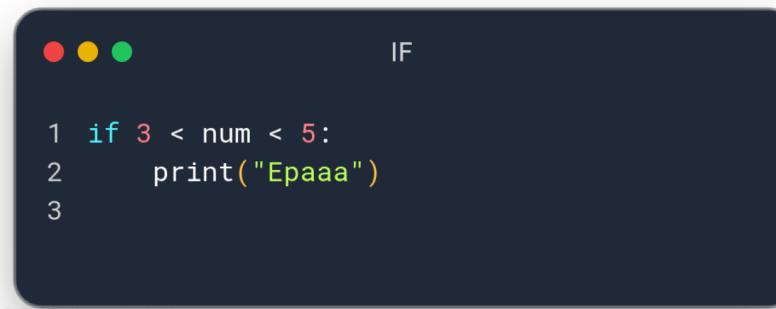
IF

```
1 if num > 3:  
2   print("Epaaa")  
3
```

codealong.com

FUNCIONES Y LIBRERÍAS

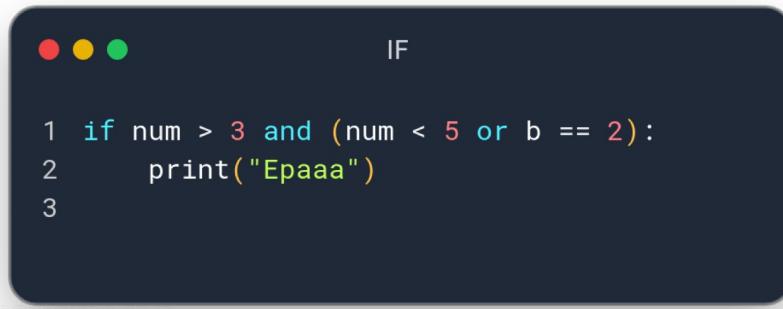
IF – Múltiples condiciones en un IF



IF

```
1 if 3 < num < 5:  
2     print("Epaaa")  
3
```

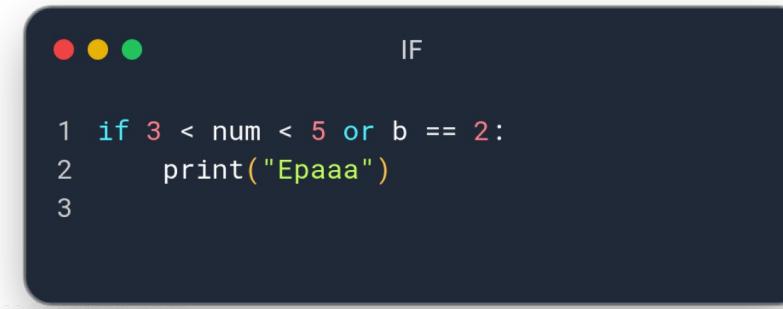
codeturing.com



IF

```
1 if num > 3 and (num < 5 or b == 2):  
2     print("Epaaa")  
3
```

codeturing.com



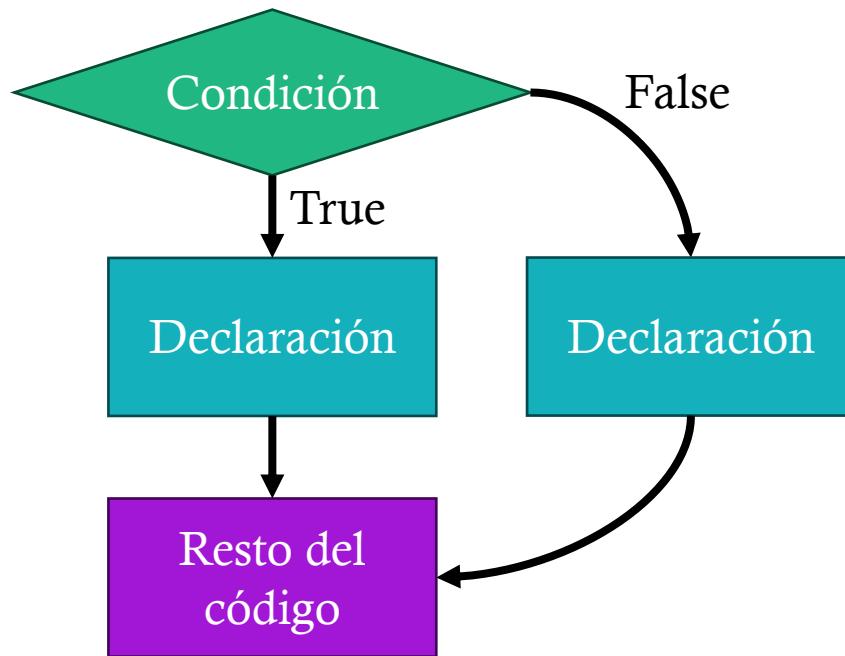
IF

```
1 if 3 < num < 5 or b == 2:  
2     print("Epaaa")  
3
```

codeturing.com

FUNCIONES Y LIBRERÍAS

IF-ELSE



```
● ● ● IF_ELSE  
1 if <Expresión_booleana>:  
2     <Declaración_1>  
3 else:  
4     <Declaración_2>  
5  
6 <Resto_del_código>
```

Codecademy

```
● ● ● IF_ELSE  
1 if num > 3:  
2     print("num es mayor a 3")  
3 else:  
4     print("num es menor o igual a 3")  
5
```

Codecademy

FUNCIONES Y LIBRERÍAS

Nested IF-ELSE

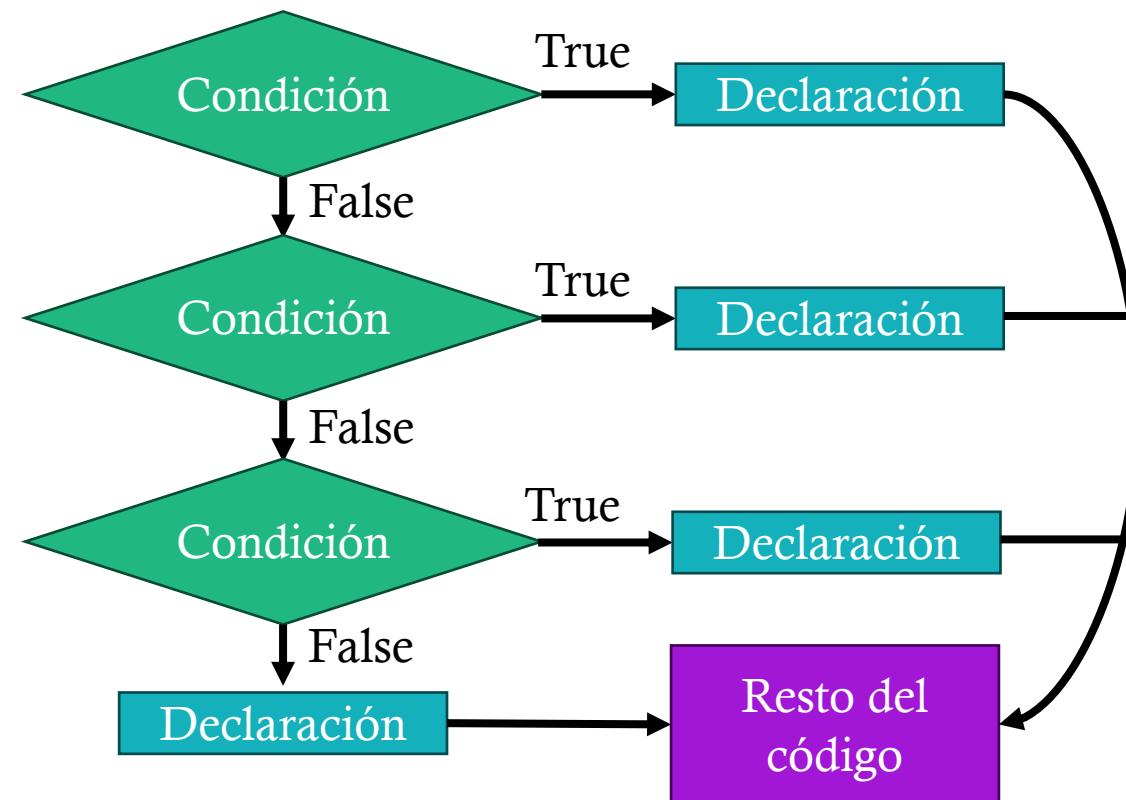
```
● ● ● NESTED_IF_ELSE  
1 if <Expresión_1>:  
2     <Declaración_1>  
3     if <Expresión_2>:  
4         <Declaración_3>  
5 else:  
6     <Declaración_2>  
7     if <Expresión_3>:  
8         <Declaración_4>  
9  
10 <Resto_del_código>
```

Comisión de Programación

```
● ● ● NESTED_IF_ELSE  
1 if num > 3:  
2     print("num es mayor a 3")  
3     if num < 5:  
4         print("num es menor a 3")  
5 else:  
6     print("num es menor o igual a 3")  
7     if num > 1:  
8         print("num es mayor a 1")
```

FUNCIONES Y LIBRERÍAS

ELIF



FUNCIONES Y LIBRERÍAS

ELIF



IF_ELIF_ELSE

```
1 if <Expresión_1>:  
2     <Declaración_1>  
3 elif <Expresión_2>:  
4     <Declaración_2>  
5 elif <Expresión_3>:  
6     <Declaración_3>  
7 else:  
8     <Declaración_4>  
9  
10 <Resto_del_código>
```



IF_ELIF_ELSE

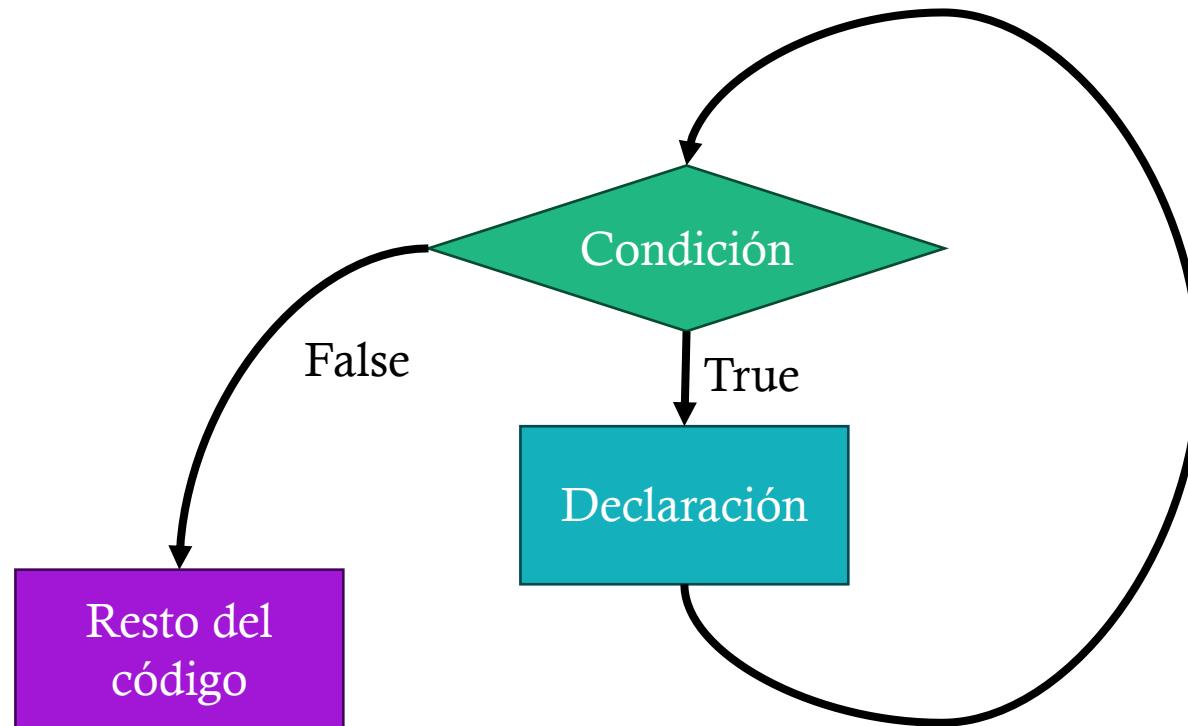
```
1 if num == 3:  
2     print("num es 3")  
3 elif num == 5:  
4     print("num es 5")  
5 elif num == 42:  
6     print("num es 42")  
7 else:  
8     print("num no es 3, 5 o 42")
```



BUCKLES

BUCLLES

While



No es el bucle más popular de Python

BUCLES

While



WHILE

```
1 while <Expresión>:  
2     <Declaración_1>  
3     <Declaración_2>  
4  
5 <Resto_del_codigo>
```



WHILE

```
1 nivel = 0  
2 while nivel <= 9000:  
3     print("Aumentando nivel")  
4     nivel += 1  
5  
6 print("It's over 9000")
```

No es el bucle más popular de Python

BUCLES

Ciclo FOR



FOR

```
1 for <variable> in <objeto_iterable>:  
2     <Declaración_1>  
3     <Declaración_2>  
4  
5 <Resto_del_codigo>
```



FOR

```
1 producto = 1  
2 for value in range(1, 11):  
3     producto *= value  
4
```

BUCLES

Iterables

- Un iterable es un objeto en Python capaz de retornar un miembro a la vez, permitiendo que sea iterable en un **loop FOR**.
- Las listas, tuplas, strings, diccionarios, sets son iterables
- Una versión de **iterables** son los **generadores**, que evitan guardar cada elemento en memoria, sino que se generan en la medida que se necesitan

BUALES

Iterables

- Tenemos la función `range(start, stop, step)` que genera una secuencia de números enteros.



The image shows a screenshot of a Jupyter Notebook cell. At the top left are three colored dots (red, yellow, green). To the right of them is the word "Código". Below that is a block of Python code:

```
1 for i in range(10):  
2     print(i)
```

At the bottom left of the cell is the URL "code.simpq.com".

BUCLES

Iterables

Lista o tupla



Código

```
1 lista = [0, "alice", 3.14]
2 for elemento in lista:
3     print(elemento)
```

codeturing.com

String



Código

```
1 string = "hola, mundo"
2 for char in string:
3     print(char)
```

codeturing.com

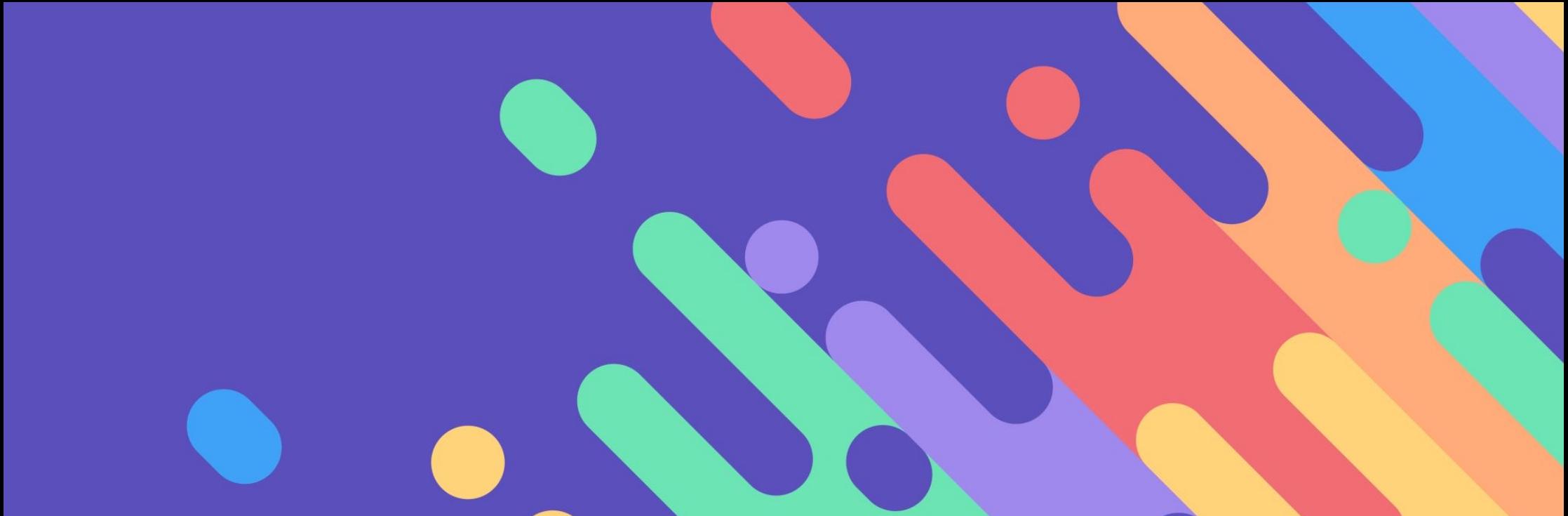
Diccionario



Código

```
1 diccionario = {
2     "nombre": "Aureliano",
3     "apellido": "Buen día",
4     "pais": "Colombia"
5 }
6 for key in diccionario:
7     print(key)
8     print(diccionario[key])
```

codeturing.com



STRINGS

STRINGS Y SUS OPERACIONES

- Strings pueden ser comparados. Se comparan carácter a carácter. El orden es en ASCII (<https://elcodigoascii.com.ar>)
- Es decir 'a' es menor a 'b' , pero 'A' es menor a 'a'.
- También podemos usar el + para concatenar dos caracteres:
“Aureliano” + “Buendia” # Retorna “AurelianoBuendia”
- Si usamos * con un entero, repite el string
“Aureliano” * 2 # Retorna “AurelianoAureliano”

STRINGS Y SUS OPERACIONES

- Podemos cortar un string usando índices
- Cortes se puede determinar en rangos



STRING

```
1 nombre_completo = "Aureliano Buendia"
2
3 nombre_completo[0] # Retorna A
```

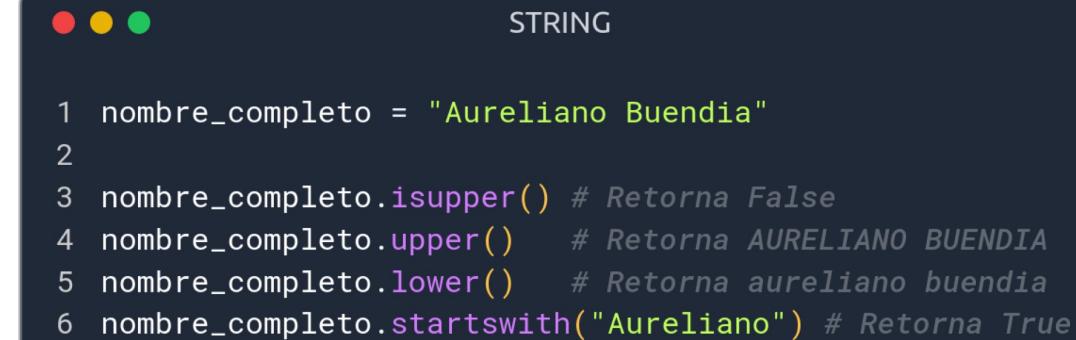


STRING

```
1 nombre_completo[inferior:superior]
2
3 nombre_completo[0:9]    # Retorna Aureliano
4 nombre_completo[:9]     # Retorna Aureliano
5 nombre_completo[10:17]   # Retorna Buendia
6 nombre_completo[10:]    # Retorna Buendia
7 nombre_completo[-7:]    # Retorna Buendia
```

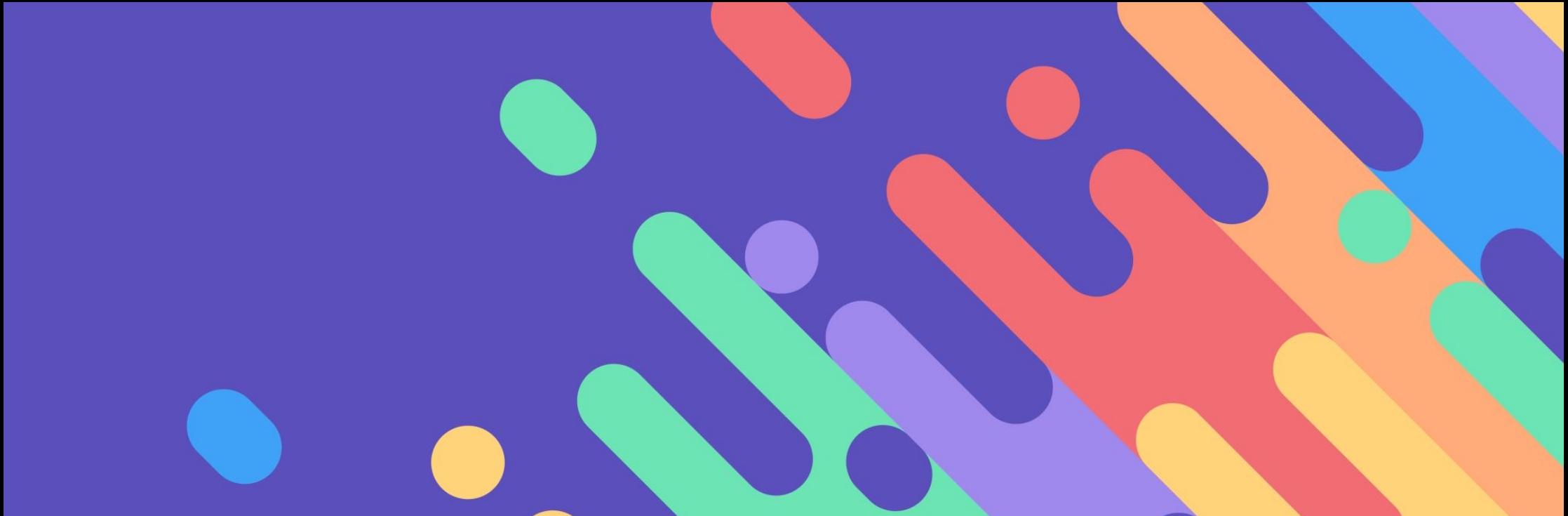
STRINGS Y SUS OPERACIONES

- Recordar que todo en Python es un objeto. Los objetos tienen atributos y métodos.
- Métodos son similares a funciones, toman argumentos, realizan una acción y devuelven algo
`<object>.<nombre del método>(<lista de argumentos>)`
- Strings son objetos, por lo que tienen métodos



The image shows a screenshot of a Jupyter Notebook cell. At the top left are three colored dots (red, yellow, green). To the right of them is the word "STRING". The code in the cell is as follows:

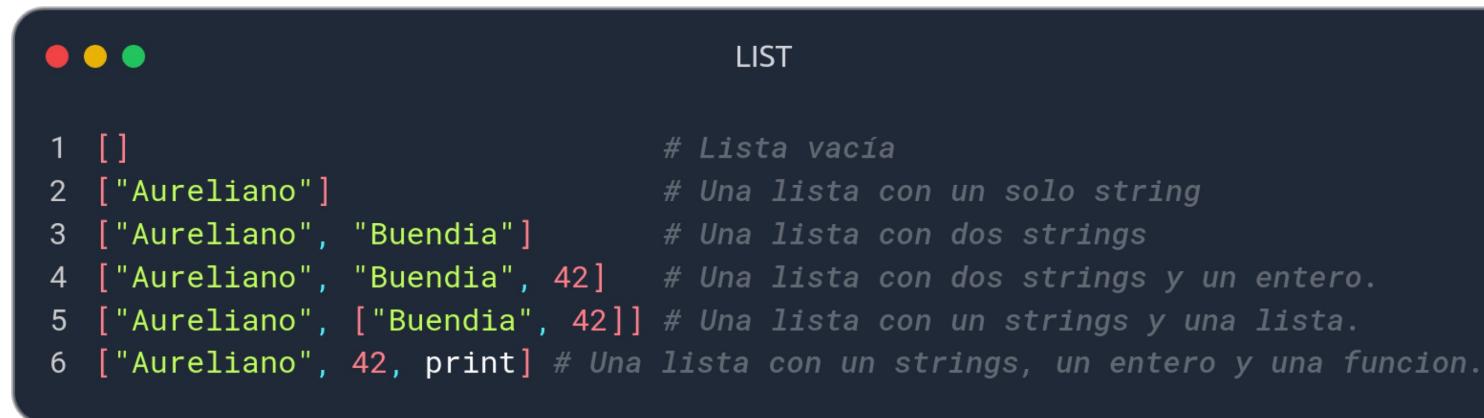
```
1 nombre_completo = "Aureliano Buendia"
2
3 nombre_completo.isupper() # Retorna False
4 nombre_completo.upper()   # Retorna AURELIANO BUENDIA
5 nombre_completo.lower()   # Retorna aureliano buendia
6 nombre_completo.startswith("Aureliano") # Retorna True
```



LISTAS

LISTAS

- Una lista es una secuencia de cero o más objetos en Python normalmente llamados ítems.
- Las listas son **mutables**.
- Se generan usando `[]` y los ítems se separan en coma:



```
● ● ● LIST

1 []                      # Lista vacía
2 ["Aureliano"]           # Una lista con un solo string
3 ["Aureliano", "Buendia"] # Una lista con dos strings
4 ["Aureliano", "Buendia", 42] # Una lista con dos strings y un entero.
5 ["Aureliano", ["Buendia", 42]] # Una lista con un strings y una lista.
6 ["Aureliano", 42, print] # Una lista con un strings, un entero y una funcion.
```

LISTAS

- Las listas también se pueden acceder a ítems mediante índices y cortarlas en sublistas:

```
● ● ● LIST  
1 list_range = list(range(0, 22, 2))  
2  
3 list_range[2] # Retorna 4  
4 list_range[:9] # Retorna [0, 2, 4, 6, 8, 10, 12, 14, 16]  
5 list_range[5:9] # Retorna [10, 12, 14, 16]  
6 list_range[-1] # Retorna 20  
7 list_range[-7:-1] # Retorna [8, 10, 12, 14, 16, 18]
```

codeturing.com

LISTAS

- Métodos de listas:



A screenshot of a dark-themed terminal window titled "LIST". The window shows the following Python code and its output:

```
1 listita = []          # listita es []
2 listita.append(42)    # listita es [42]
3 listita.append(19)    # listita es [42, 19]
4 listita.sort()        # listita es [19, 42]
5 var = listita.pop()   # Guarda en var a 42, listita es [19]
6 listita.insert(0, 22) # listita es [22, 19]
7 listita.insert(-1, 55) # listita es [22, 55, 19]
8 listita.remove(22)    # listita es [55, 19]
9 listita.remove(22)    # Error (ValueError)
```



TUPLAS

TUPLAS

- Una tupla es una secuencia de cero o más objetos Python normalmente llamados ítems.
- Las tuplas son **inmutables**.
- Se generan usando `()` y los ítems se separan en coma:

```
● ● ● TUPLE  
1 ("Aureliano",) # Una tupla con un solo string  
2 ("Aureliano", "Buendia") # Una tupla con dos strings  
3 ("Aureliano", "Buendia", 42) # Una tupla con dos strings y un entero.  
4 ("Aureliano", ["Buendia", 42]) # Una tupla con un strings y una lista.  
5 ("Aureliano", 42, print) # Una tupla con un strings, un entero y una funcion.
```

VOLVAMOS AL CICLO FOR

- FOR es realmente útil para iterar en ítems en secuencias como strings, listas y tuplas, entre otros...

```
● ● ● FOR_LIST  
1 listita = [4, 8, 15, 16, 23, 42]  
2 for item in listita:  
3     print(item)  
4
```

- Es equivalente:

```
● ● ● FOR_LIST  
1 listita = [4, 8, 15, 16, 23, 42]  
2 for index in range(len(listita)):  
3     print(listita[index])  
4
```

VOLVAMOS AL CICLO FOR

- ¿Y si quiero también el index?



FOR_LIST

```
1 listita = [4, 8, 15, 16, 23, 42]
2 for index, item in enumerate(listita):
3     print("Posicion: " + str(index))
4     print("Elemento: " + str(item))
```



DICCIONARIO

DICCIONARIO

- Un diccionario es una secuencia de un key único con un valor.
- Los diccionarios son **mutables**.
- Se generan usando `{}` y los ítems se separan en coma:

```
● ● ● DICTONARY  
1 dictionary1 = {} # Una diccionario vacio  
2 dictionary = {"nombre" : "Aureliano",  
3                 "apellido" : "Buen Dia",  
4                 "edad" : 42,  
5                 "hobbies" : ["tennis", "cocer"]} # Una diccionario con 4 entradas
```

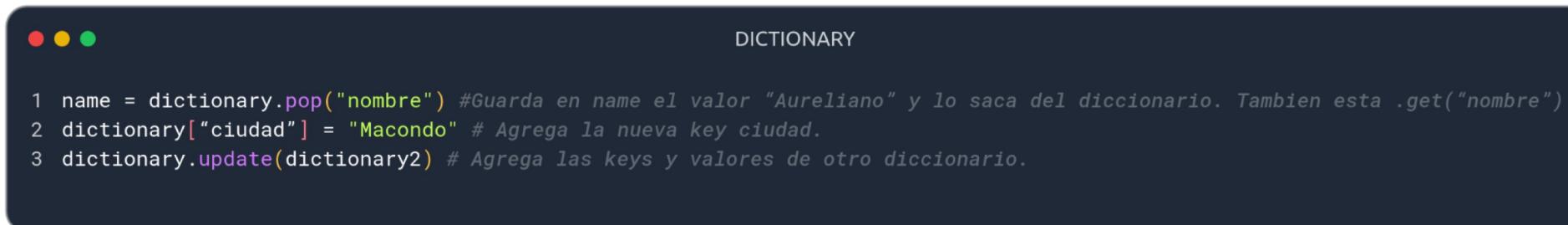
DICCIONARIO

- Accedemos usando las keys



```
● ● ● DICTIONARY  
1 name = dictionary["nombre"] # Guarda en name el valor "Aureliano"  
2
```

- Algunos métodos



```
● ● ● DICTIONARY  
1 name = dictionary.pop("nombre") #Guarda en name el valor "Aureliano" y lo saca del diccionario. Tambien esta .get("nombre")  
2 dictionary["ciudad"] = "Macondo" # Agrega la nueva key ciudad.  
3 dictionary.update(dictionary2) # Agrega las keys y valores de otro diccionario.
```

DICCIONARIO

- Ciclo FOR con el diccionario

```
● ● ● FOR_DICTIONARY  
1 for key in dictionary:  
2     print(key) # Imprime solo las keys  
3
```

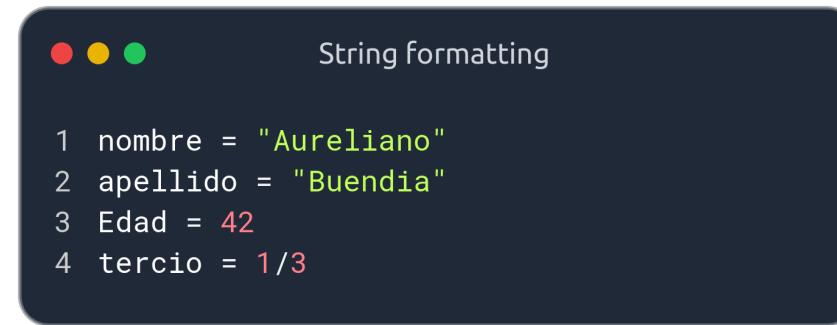
```
● ● ● FOR_DICTIONARY  
1 for key in dictionary:  
2     print(key)  
3     print(dictionary[key]) # Imprimimos tambien los valores de cada key  
4
```



STRING FORMATTING

FORMATO DE STRINGS

- Si queremos formar texto junto a variables, hay al menos 4 formas de hacerlo 😕



The screenshot shows a terminal window with a dark background and three colored dots (red, yellow, green) in the top-left corner. The title bar reads "String formatting". The code area contains the following Python code:

```
1 nombre = "Aureliano"
2 apellido = "Buendia"
3 Edad = 42
4 tercio = 1/3
```

At the bottom left of the terminal window, the URL "code2img.com" is visible.

- Queremos imprimir usando las variables:

"Hola, tu nombre es Aureliano Buendia y tu edad es 42. Un tercio es 0.333"

FORMATO DE STRINGS

- **Modo 1:** Usando el operador %



String formatting

```
1 #Formato 1
2 string = ("Hola, tu nombre es %s %s y tu edad es %d."
3           " Un tercio es %.3f" % (nombre, apellido, Edad, tercio))
4 print(string)
```

DATAFLYING.COM

FORMATO DE STRINGS

- **Modo 2:** Usando el método `.format()`

```
String formatting

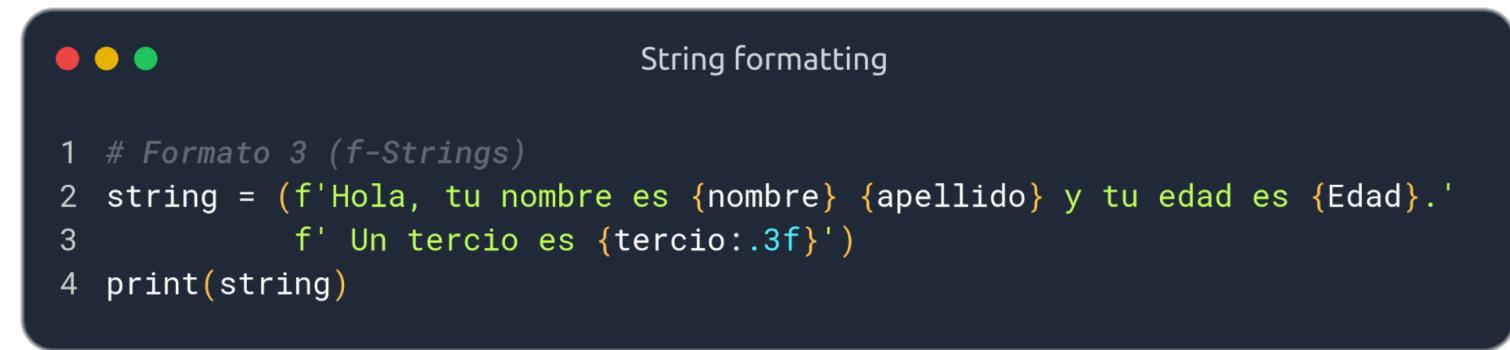
1 #Formato 2 (.format)
2 string = ("Hola, tu nombre es {} {} y tu edad es {}."
3           " Un tercio es {:.3f}").format(nombre, apellido, Edad, tercio)
4 print(string)
```

```
String formatting

1 #Formato 2 (.format)
2 string = ("Hola, tu nombre es {nombre} {apellido} y tu edad es {edad}."
3           " Un tercio es {third:.3f}").format(nombre=nombre,
4                                         apellido=apellido,
5                                         edad=Edad,
6                                         third=tercio)
7 print(string)
```

FORMATO DE STRINGS

- **Modo 3:** Usando **f-strings**



The screenshot shows a terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. The title bar reads "String formatting". The main area contains the following Python code:

```
1 # Formato 3 (f-Strings)
2 string = (f'Hola, tu nombre es {nombre} {apellido} y tu edad es {Edad}.'
3             f' Un tercio es {tercio:.3f}')
4 print(string)
```

FORMATO DE STRINGS

- **Modo 4:** Transformando y concatenando

```
String formatting

1 # Formato 4 (Transformando y concatenando)
2 string = ('Hola, tu nombre es ' + nombre + " " + apellido + " y tu edad es "
3           + str(Edad) + ". Un tercio es " + str(round(tercio,3)))
4 print(string)
```



FUNCIONES

CREACIÓN DE NUEVAS FUNCIONES

- Supongamos que queremos calcular el número combinatorio:

$$C(m, n) = \frac{m!}{(m - n)!n!}$$

- Donde $n!$ (el factorial de n) es el producto de los números enteros de 1 a n .

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (n - 1) \cdot n = \prod_{i=1}^n i$$

CREACIÓN DE NUEVAS FUNCIONES



FACTORIAL

```
1 #Factorial de n!
2 f = 1
3 for i in range(1, n + 1):
4     f *= i
```

codetabimg.com



CONV

```
1 n, m = 3, 5
2
3 #Numerador
4 num = 1
5 for i in range(1, m+1):
6     num *= i
7
8 #Denominador
9 den_a = 1
10 for i in range(1, n+1):
11     den_a *= i
12
13 den_b = 1
14 for i in range(1, m-n+1):
15     den_b *= i
16
17 den = den_a * den_b
18
19 #Resultado
20 num_conv = num / den
```

codetabimg.com

CREACIÓN DE NUEVAS FUNCIONES

- Escribir el mismo código una y otra vez es propenso a errores y difícil de mantener el código. Si creamos una función que haga la multiplicación va a ser mucho más sencillo.

```
● ● ●
```

FUNC

```
1 def <nombre función>(argumentos):  
2     <secuencia de código>
```

codealong.com

```
● ● ●
```

FACTORIAL

```
1 def factorial(n):  
2     """Calcula el factorial de n"""  
3     output = 1  
4     for i in range(1, n+1):  
5         output *= i  
6     return output
```

codealong.com

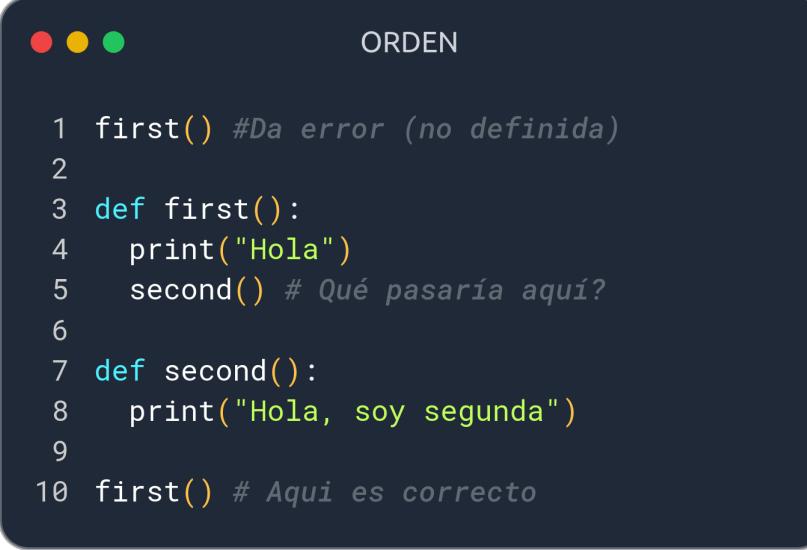
CREACIÓN DE NUEVAS FUNCIONES

```
● ● ● FACTORIAL  
1 def factorial(n):  
2     """Calcula el factorial de n"""  
3     output = 1  
4     for i in range(1, n+1):  
5         output *= i  
6  
7     return output
```

```
● ● ● NUM_CONV  
1 def num_conv(n, m):  
2     """Calcula el numero combinatorio de n y m  
3  
4     n es la cantidad de objetos a seleccionar de un conjunto total de  
5     m objetos.  
6     """  
7  
8     return int(factorial(m) / (factorial(n)*factorial(m-n)))
```

CREACIÓN DE NUEVAS FUNCIONES

- Las variables que están dentro de las funciones existen solamente dentro de las funciones (variables locales).
- Las funciones deben ser definidas antes de ser llamadas



```
ORDEN

1 first() #Da error (no definida)
2
3 def first():
4     print("Hola")
5     second() # Qué pasaría aquí?
6
7 def second():
8     print("Hola, soy segunda")
9
10 first() # Aquí es correcto
```

cocetomg.com

CREACIÓN DE NUEVAS FUNCIONES

- Podemos agregar argumentos opcionales fácilmente en nuestras funciones



FACTORIAL

```
1 def factorial(n, print_output=False):
2     """Calcula el factorial de n"""
3     output = 1
4     for i in range(1, n+1):
5         output *= i
6
7     if print_output:
8         print(output)
9
10    return output
```

codetiming.com

CREACIÓN DE NUEVAS FUNCIONES

- Se pueden retornar muchos valores (que se obtendrán como en una tupla)



CONV_SEGUNDOS

```
1 def conv_segundos(segundos):
2
3     horas = segundos // (60 * 60)
4     rest_horas = segundos % (60 * 60)
5     minutos = rest_horas // 60
6     segundos = minutos % 60
7
8     return horas, minutos, segundos
```

codalima.com

CREACIÓN DE NUEVAS FUNCIONES

- Una función puede llamarse a sí misma:



FIBONACCI

```
1 def recur_fibo(n):
2     if n <= 1:
3         return n
4     else:
5         return recur_fibo(n-1) + recur_fibo(n-2)
```

codetoplay.com

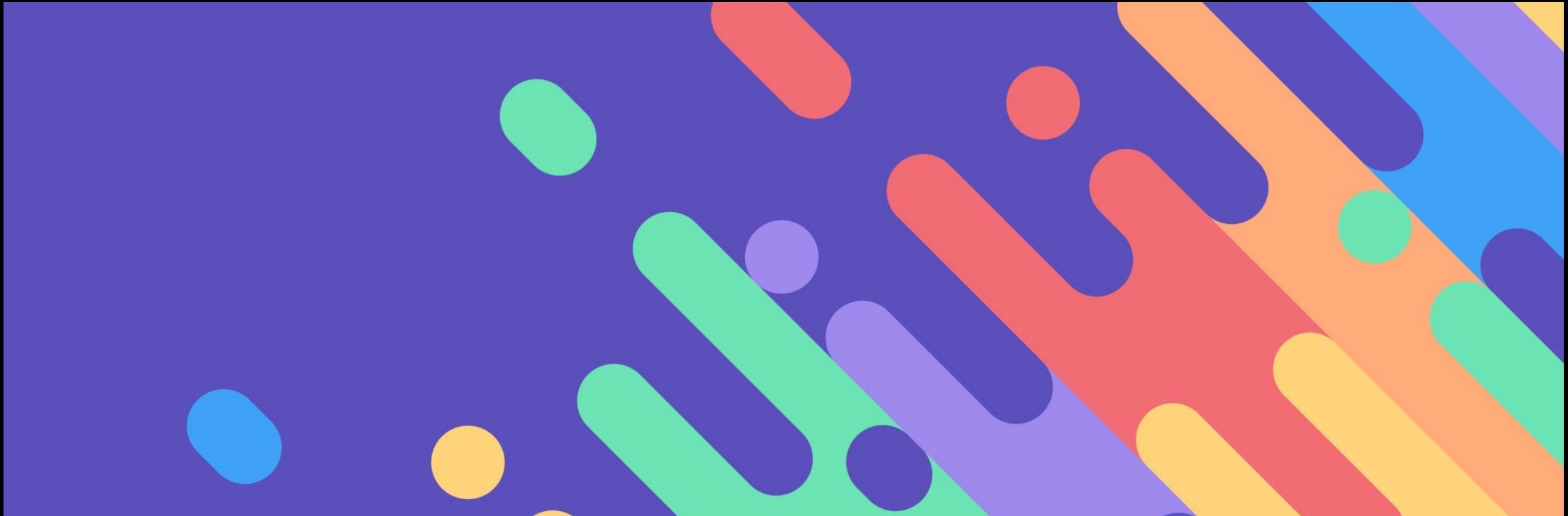
CREACIÓN DE NUEVAS FUNCIONES

- Una función anónima es una función sin nombre. En Python, se crea una función anónima con la palabra clave **lambda**.



LAMBDA

```
1 sum_one = lambda x : x + 1
2 print(sum_one(2))
3 print((lambda x : x + 1 )(2))
4
5 sum_two_numbers = lambda x, y: x + y
6 print(sum_two_numbers(2,4))
7
8 conditional = (lambda x : x if x % 2 else -1)
9 print(conditional(3))
```



LIST COMPREHENSION Y GENERATORS

LIST COMPREHENSION

- Es una expresión que genera una colección basada en otra colección.
- En general produce listas.
- Sintaxis simple y limpia.
- Soporta condicionales.
- Puede ser lazy.
- Es una de las herramientas más importante en Python

LIST COMPREHENSION



List comprehension

```
1 [state for var in iterable if predicate]
2
3 #Ejemplo
4 even_squared = [x**2 for x in range(20) if not(x % 2)]
```

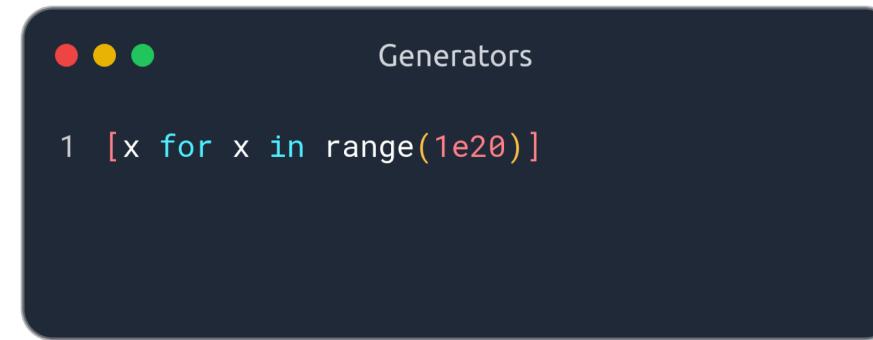
www.cecilio.com

LIST COMPREHENSION

- Computa todos los valores cuando se crea (ocupa memoria).
- Es preferible usar List comprehensivo antes que bucles.
- También existen los:
 - Set comprehension
 - Dictionary comprehension

GENERATORS

- Generan valores de forma lazy (no ocupan memoria) pero se consumen.



No alcanza la memoria RAM para genera esta lista

GENERATORS

- Generan valores de forma lazy (no ocupan memoria) pero se consumen.



Generators

```
1 generator = (x for x in range(int(1e20)))
2 print(next(generator)) # Imprime 0
3 print(next(generator)) # Imprime 1
4 ...
5 print(next(generator)) # Imprime 1e20
6 print(next(generator)) # Da error StopIteration.
```

codetiming.com



CLASES Y OBJETOS

CLASES Y OBJETOS



Class

```
1 class new_class:  
2     '''Documentacion'''  
3     def __init__(self, atr1, atr2):  
4         self.atr1 = atr1  
5         self.atr2 = atr2  
6     def method_1(self, x):  
7         '''Documentacion'''  
8         return x  
9     def method_2(self, x):  
10        return x
```



Object

```
1 obj_1 = new_class(1, 2)  
2 obj_2 = new_class(4, 4)  
3 obj_3 = new_class(23, 4)
```

codetiming.com

codetiming.com

CLASES Y OBJETOS

```
Object

1 class car:
2     '''Es una clase de auto'''
3     ruedas = 4
4
5     def __init__(self, color, brand):
6         self.color = color
7         self.marca = brand
8         self.velocidad = 0
9     def bocina(self):
10        '''Toca la bocina'''
11        print("Piiiii")
12    def acelerar(self, x):
13        self.velocidad += x
14    def frenar(self):
15        self.velocidad = 0
```

codeboing.com

```
Object

1 auto_1 = car("rojo", "Ford")
2 auto_2 = car("verde", "Chevrolet")
3
4 print(auto_1.ruedas) #4
5 print(auto_1.marca) # Ford
6 auto_1.bocina() # Imprime Piiii
7 auto_2.acelerar(120)
8 print(auto_2.velocidad) #Imprime 120
9 auto_2.frenar()
10 print(auto_2.velocidad) #Imprime 0
```

codeboing.com

CLASES Y OBJETOS – HERENCIA



Object

```
1 class car_ford(car):
2     def __init__(self, color, model):
3         super().__init__(color, "Ford")
4         self.modelo = model
5     def cambiar_color(new_color):
6         self.color = new_color
```

codeturing.com



Object

```
1 auto_3 = car_ford("Azul", "Mondeo")
2 print(auto_3.marca) # Imprime Ford
3 print(auto_3.modelo) # Imprime Mondeo
4 auto_3.bocina() #Imprime Piiji
```

codeturing.com