Python Tutorial

¿Qué es Python?

Python es un lenguaje de programación popular. Fue creado por Guido van Rossum y lanzado en 1991.

Se utiliza para:

* desarrollo web (lado del servidor),
* desarrollo de software,
* matemáticas,
* secuencias de comandos del sistema.

¿Qué puede hacer Python?

* Python se puede usar en un servidor para crear aplicaciones web.
* Python se puede usar junto con el software para crear flujos de trabajo.
* Python puede conectarse a sistemas de bases de datos. También puede leer y modificar archivos.
* Python se puede usar para manejar big data y realizar operaciones matemáticas complejas.
* Python se puede utilizar para la creación rápida de prototipos o para el desarrollo de software listo para la producción.

¿Por qué Python?

* Python funciona en diferentes plataformas (Windows, Mac, Linux, Raspberry Pi, etc.).
* Python tiene una sintaxis simple similar al idioma inglés.
* Python tiene una sintaxis que permite a los desarrolladores escribir programas con menos líneas que otros lenguajes de programación.
* Python se ejecuta en un sistema de interpretación, lo que significa que el código se puede ejecutar tan pronto como se escribe. Esto significa que la creación de prototipos puede ser muy rápida.
* Python se puede tratar de forma procesal, orientada a objetos o funcional.

Bueno saber

* La versión principal más reciente de Python es Python 3, que usaremos en este tutorial. Sin embargo, Python 2, aunque no se actualiza con nada más que actualizaciones de seguridad, sigue siendo bastante popular.
* En este tutorial, Python se escribirá en un editor de texto. Es posible escribir Python en un entorno de desarrollo integrado, como Thonny, Pycharm, Netbeans o Eclipse, que son particularmente útiles cuando se administran colecciones más grandes de archivos de Python.

Sintaxis de Python en comparación con otros lenguajes de programación

* Python fue diseñado para facilitar la lectura y tiene algunas similitudes con el idioma inglés con influencia de las matemáticas.
* Python usa nuevas líneas para completar un comando, a diferencia de otros lenguajes de programación que a menudo usan punto y coma o paréntesis.
* Python se basa en la sangría, usando espacios en blanco, para definir el alcance; como el alcance de los bucles, funciones y clases. Otros lenguajes de programación a menudo usan corchetes para este propósito.

Ejemplo

print("Hello, World!")

# Primeros pasos con Python

## **Instalación de Python**

Muchas PC y Mac ya tendrán Python instalado.

Para verificar si tiene Python instalado en una PC con Windows, busque Python en la barra de inicio o ejecute lo siguiente en la línea de comandos (cmd.exe):

C:\Users\Your Name>python --version

Para verificar si tiene Python instalado en Linux o Mac, luego en Linux abra la línea de comando o en Mac abra la Terminal y escriba:

python --version

Si descubre que no tiene Python instalado en su computadora, puede descargarlo de forma gratuita desde el siguiente sitio web: <https://www.python.org/>

## **Inicio rápido de Python**

Python es un lenguaje de programación interpretado, esto significa que, como desarrollador, escribe archivos de Python (.py) en un editor de texto y luego coloca esos archivos en el intérprete de Python para ejecutarlos.

La forma de ejecutar un archivo python es así en la línea de comando:

C:\Users\Your Name>python helloworld.py

Donde "helloworld.py" es el nombre de su archivo python.

Escribamos nuestro primer archivo de Python, llamado helloworld.py, que se puede hacer en cualquier editor de texto.

holamundo.py

print("Hello, World!")

Simple como eso. Guarde su archivo. Abra su línea de comando, navegue hasta el directorio donde guardó su archivo y ejecute:

C:\Users\Your Name>python helloworld.py

La salida debe leer:

Hello, World!

Felicitaciones, ha escrito y ejecutado su primer programa en Python.

## **La línea de comandos de Python**

Para probar una pequeña cantidad de código en python, a veces es más rápido y fácil no escribir el código en un archivo. Esto es posible porque Python se puede ejecutar como una línea de comando.

Escriba lo siguiente en la línea de comandos de Windows, Mac o Linux:

C:\Users\Your Name>python

O, si el comando "python" no funcionó, puede probar con "py":

C:\Users\Your Name>py

Desde allí, puede escribir cualquier python, incluido nuestro ejemplo de hola mundo anterior en el tutorial:

C:\Users\Your Name>python  
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print("Hello, World!")

Que escribirá "¡Hola, mundo!" en la línea de comando:

C:\Users\Your Name>python  
Python 3.6.4 (v3.6.4:d48eceb, Dec 19 2017, 06:04:45) [MSC v.1900 32 bit (Intel)] on win32  
Type "help", "copyright", "credits" or "license" for more information.  
>>> print("Hello, World!")  
Hello, World!

Cuando haya terminado en la línea de comandos de python, simplemente puede escribir lo siguiente para salir de la interfaz de línea de comandos de python:

exit()

## **Ejecutar sintaxis de Python**

Como aprendimos en la página anterior, la sintaxis de Python se puede ejecutar escribiendo directamente en la línea de comandos:

>>> print("Hello, World!")  
Hello, World!

O creando un archivo python en el servidor, usando la extensión de archivo .py y ejecutándolo en la línea de comandos:

C:\Users\Your Name>python myfile.py

## **sangría de Python**

## La sangría se refiere a los espacios al comienzo de una línea de código.

Mientras que en otros lenguajes de programación la sangría en el código es solo para facilitar la lectura, la sangría en Python es muy importante.

Python usa sangría para indicar un bloque de código.

### **Ejemplo**

if 5 > 2:  
  print("Five is greater than two!")

Python le dará un error si omite la sangría:

### **Ejemplo**

Error de sintaxis:

if 5 > 2:  
print("Five is greater than two!")

La cantidad de espacios depende de usted como programador, el uso más común es cuatro, pero tiene que ser al menos uno.

### **Ejemplo**

if 5 > 2:  
 print("Five is greater than two!")   
if 5 > 2:  
        print("Five is greater than two!")

Tienes que usar la misma cantidad de espacios en el mismo bloque de código, de lo contrario, Python te dará un error:

### **Ejemplo**

Error de sintaxis:

if 5 > 2:  
 print("Five is greater than two!")  
        print("Five is greater than two!"

## **Variables de Python**

En Python, las variables se crean cuando le asignas un valor:

### **Ejemplo**

Variables en Python:

x = 5  
y = "Hello, World!"

Python no tiene ningún comando para declarar una variable.

Aprenderá más sobre las variables en el capítulo [Variables de Python .](https://www.w3schools.com/python/python_variables.asp)

## **Comentarios**

Python tiene la capacidad de comentar con el propósito de la documentación en el código.

Los comentarios comienzan con un # y Python representará el resto de la línea como un comentario:

### **Ejemplo**

Comentarios en Python:

#This is a comment.  
print("Hello, World!")

Principio del formulario

## **Po**

# Comentarios de Python

Los comentarios se pueden usar para explicar el código de Python.

Los comentarios se pueden utilizar para hacer que el código sea más legible.

Los comentarios se pueden usar para evitar la ejecución al probar el código.

## **Crear un comentario**

Los comentarios comienzan con un #, y Python los ignorará:

### **Ejemplo**

#This is a comment  
print("Hello, World!")

Los comentarios se pueden colocar al final de una línea y Python ignorará el resto de la línea:

### **Ejemplo**

print("Hello, World!") #This is a comment

Un comentario no tiene que ser texto que explique el código, también se puede usar para evitar que Python ejecute código:

### **Ejemplo**

#print("Hello, World!")  
print("Cheers, Mate!")

## **Comentarios de varias líneas**

Python realmente no tiene una sintaxis para comentarios de varias líneas.

Para agregar un comentario de varias líneas, puede insertar un #para cada línea:

### **Ejemplo**

#This is a comment  
#written in  
#more than just one line  
print("Hello, World!")

O, no del todo como se pretendía, puede usar una cadena de varias líneas.

Dado que Python ignorará los literales de cadena que no están asignados a una variable, puede agregar una cadena de varias líneas (comillas triples) en su código y colocar su comentario dentro de ella:

### **Ejemplo**

"""  
This is a comment  
written in  
more than just one line  
"""  
print("Hello, World!")

Siempre que la cadena no esté asignada a una variable, Python leerá el código, pero luego lo ignorará y habrá realizado un comentario de varias líneas.

Principio del formulario

Final del formulario

# Python Variables

## **Variables**

Las variables son contenedores para almacenar valores de datos.

## **Creación de variables**

Python no tiene ningún comando para declarar una variable.

Una variable se crea en el momento en que le asigna un valor por primera vez.

### **Ejemplo**

x = 5  
y = "John"  
print(x)  
print(y)

No es necesario declarar las variables con ningún tipo en particular , e incluso pueden cambiar de tipo después de que se hayan establecido.

### **Ejemplo**

x = 4       # x is of type int  
x = "Sally" # x is now of type str  
print(x)

## **Fundición**

Si desea especificar el tipo de datos de una variable, puede hacerlo con la conversión.

### **Ejemplo**

x = str(3)    # x will be '3'  
y = int(3)    # y will be 3  
z = float(3)  # z will be 3.0

## **obtener el tipo**

Puede obtener el tipo de datos de una variable con la type()función.

### **Ejemplo**

x = 5  
y = "John"  
print(type(x))  
print(type(y))

Aprenderá más sobre [los tipos de datos](https://www.w3schools.com/python/python_datatypes.asp) y [la conversión](https://www.w3schools.com/python/python_casting.asp) más adelante en este tutorial.

## **¿Comillas simples o dobles?**

Las variables de cadena se pueden declarar mediante comillas simples o dobles:

### **Ejemplo**

x = "John"  
# is the same as  
x = 'John'

## **Distingue mayúsculas y minúsculas**

Los nombres de las variables distinguen entre mayúsculas y minúsculas.

### **Ejemplo**

Esto creará dos variables:

a = 4  
A = "Sally"  
#A will not overwrite a

# Python - Nombres de variables

## **Nombres de variables**

Una variable puede tener un nombre corto (como x e y) o un nombre más descriptivo (edad, nombre del coche, volumen\_total). Reglas para las variables de Python:

* Un nombre de variable debe comenzar con una letra o el carácter de subrayado
* Un nombre de variable no puede comenzar con un número
* Un nombre de variable solo puede contener caracteres alfanuméricos y guiones bajos (Az, 0-9 y \_)
* Los nombres de las variables distinguen entre mayúsculas y minúsculas (edad, Edad y EDAD son tres variables diferentes)

### **Ejemplo**

Nombres de variables legales:

myvar = "John"  
my\_var = "John"  
\_my\_var = "John"  
myVar = "John"  
MYVAR = "John"  
myvar2 = "John"

### **Ejemplo**

Nombres de variables ilegales:

2myvar = "John"  
my-var = "John"  
my var = "John"

Recuerde que los nombres de las variables distinguen entre mayúsculas y minúsculas

## **Multi Words Variable Names**

Los nombres de variables con más de una palabra pueden ser difíciles de leer.

Hay varias técnicas que puede utilizar para hacerlos más legibles:

## **Camel Case**

Cada palabra, excepto la primera, comienza con una letra mayúscula:

myVariableName = "John"

## **Pascal Case**

Cada palabra comienza con una letra mayúscula:

MyVariableName = "John"

## **Snake Case**

Cada palabra está separada por un carácter de subrayado:

my\_variable\_name = "John"

# Python Variables - Assign Multiple Values

## **Muchos valores para múltiples variables**

Python le permite asignar valores a múltiples variables en una línea:

### **Ejemplo**

x, y, z = "Orange", "Banana", "Cherry"  
print(x)  
print(y)  
print(z)

**Nota:** asegúrese de que la cantidad de variables coincida con la cantidad de valores, de lo contrario obtendrá un error.

## **Un valor para múltiples variables**

Y puede asignar el mismo valor a múltiples variables en una línea:

### **Ejemplo**

x = y = z = "Orange"  
print(x)  
print(y)  
print(z)

## **Desempaquetar una colección**

Si tiene una colección de valores en una lista, tupla, etc. Python le permite extraer los valores en variables. Esto se llama desempacar .

### **Ejemplo**

Descomprimir una lista:

fruits = ["apple", "banana", "cherry"]  
x, y, z = fruits  
print(x)  
print(y)  
print(z)

# Python - Output Variables

## **Variables de salida**

La función de Python print()se usa a menudo para generar variables.

### **Ejemplo**

x = "Python is awesome"  
print(x)

En la print()función, genera múltiples variables, separadas por una coma:

### **Ejemplo**

x = "Python"  
y = "is"  
z = "awesome"  
print(x, y, z)

También puede usar el +operador para generar múltiples variables:

### **Ejemplo**

x = "Python "  
y = "is "  
z = "awesome"  
print(x + y + z)

Observe el carácter de espacio después de "Python "y "is ", sin ellos el resultado sería "Pythonisawesome".

Para números, el +carácter funciona como un operador matemático:

### **Ejemplo**

x = 5  
y = 10  
print(x + y)

En la print()función, cuando intenta combinar una cadena y un número con el + operador, Python le dará un error:

### **Ejemplo**

x = 5  
y = "John"  
print(x + y)

La mejor manera de generar múltiples variables en la print()función es separarlas con comas, que incluso admiten diferentes tipos de datos:

### **Ejemplo**

x = 5  
y = "John"  
print(x, y)

# Python - Global Variables

## **Variables globales**

Las variables que se crean fuera de una función (como en todos los ejemplos anteriores) se conocen como variables globales.

Las variables globales pueden ser utilizadas por todos, tanto dentro como fuera de las funciones.

### **Ejemplo**

Crear una variable fuera de una función y usarla dentro de la función

x = "awesome"  
  
def myfunc():  
  print("Python is " + x)  
  
myfunc()

Si crea una variable con el mismo nombre dentro de una función, esta variable será local y solo se puede usar dentro de la función. La variable global con el mismo nombre quedará como estaba, global y con el valor original.

### **Ejemplo**

Crear una variable dentro de una función, con el mismo nombre que la variable global

x = "awesome"  
  
def myfunc():  
  x = "fantastic"  
  print("Python is " + x)  
  
myfunc()  
  
print("Python is " + x)

## **La palabra clave mundial**

Normalmente, cuando crea una variable dentro de una función, esa variable es local y solo se puede usar dentro de esa función.

Para crear una variable global dentro de una función, puede usar la globalpalabra clave.

### **Ejemplo**

Si usa la globalpalabra clave, la variable pertenece al ámbito global:

def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

Además, use la globalpalabra clave si desea cambiar una variable global dentro de una función.

### **Ejemplo**

Para cambiar el valor de una variable global dentro de una función, consulte la variable usando la globalpalabra clave:

x = "awesome"  
  
def myfunc():  
  global x  
  x = "fantastic"  
  
myfunc()  
  
print("Python is " + x)

# Python Data Types

## **Tipos de datos integrados**

En programación, el tipo de datos es un concepto importante.

Las variables pueden almacenar datos de diferentes tipos, y diferentes tipos pueden hacer cosas diferentes.

Python tiene los siguientes tipos de datos integrados de forma predeterminada, en estas categorías:

|  |  |
| --- | --- |
| Tipo de texto: | str |
| Tipos numéricos: | int, float, complex |
| Tipos de secuencia: | list, tuple, range |
| Tipo de mapeo: | dict |
| Establecer tipos: | set,frozenset |
| Tipo booleano: | bool |
| Tipos binarios: | bytes, bytearray, memoryview |

## **Obtener el tipo de datos**

Puede obtener el tipo de datos de cualquier objeto utilizando la type()función:

### **Ejemplo**

Imprime el tipo de datos de la variable x:

x = 5  
print(type(x))

## **Configuración del tipo de datos**

En Python, el tipo de datos se establece cuando asigna un valor a una variable:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = "Hello World" | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str) |
| x = 20 | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int) |
| x = 20.5 | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float) |
| x = 1j | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex) |
| x = ["apple", "banana", "cherry"] | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list) |
| x = ("apple", "banana", "cherry") | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range) |
| x = {"name" : "John", "age" : 36} | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict) |
| x = {"apple", "banana", "cherry"} | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set) |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset) |
| x = True | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool) |
| x = b"Hello" | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview) |

ANUNCIO PUBLICITARIO

## **Configuración del tipo de datos específico**

Si desea especificar el tipo de datos, puede utilizar las siguientes funciones de construcción:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** | **Try it** |
| x = str("Hello World") | str | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_str2) |
| x = int(20) | int | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_int2) |
| x = float(20.5) | float | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_float2) |
| x = complex(1j) | complex | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_complex2) |
| x = list(("apple", "banana", "cherry")) | list | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_list2) |
| x = tuple(("apple", "banana", "cherry")) | tuple | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_tuple2) |
| x = range(6) | range | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_range2) |
| x = dict(name="John", age=36) | dict | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_dict2) |
| x = set(("apple", "banana", "cherry")) | set | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_set2) |
| x = frozenset(("apple", "banana", "cherry")) | frozenset | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_frozenset2) |
| x = bool(5) | bool | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bool2) |
| x = bytes(5) | bytes | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytes2) |
| x = bytearray(5) | bytearray | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_bytearray2) |
| x = memoryview(bytes(5)) | memoryview | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_type_memoryview2) |

Principio del formulario

# Números de Python

## **Números de Python**

Hay tres tipos numéricos en Python:

* int
* float
* complex

Las variables de tipo numérico se crean cuando les asignas un valor:

### **Ejemplo**

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex

Para verificar el tipo de cualquier objeto en Python, use la type()función:

### **Ejemplo**

print(type(x))  
print(type(y))  
print(type(z))

[Inténtalo tú mismo "](https://www.w3schools.com/python/trypython.asp?filename=demo_numbers)

## **Int**

Int, o entero, es un número entero, positivo o negativo, sin decimales, de longitud ilimitada.

### **Ejemplo**

enteros:

x = 1  
y = 35656222554887711  
z = -3255522  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Float**

Flotante, o "número de punto flotante" es un número, positivo o negativo, que contiene uno o más decimales.

### **Ejemplo**

flotadores:

x = 1.10  
y = 1.0  
z = -35.59  
  
print(type(x))  
print(type(y))  
print(type(z))

El flotador también puede ser un número científico con una "e" para indicar la potencia de 10.

### **Ejemplo**

flotadores:

x = 35e3  
y = 12E4  
z = -87.7e100  
  
print(type(x))  
print(type(y))  
print(type(z))

## **Complex**

Los números complejos se escriben con una "j" como parte imaginaria:

### **Ejemplo**

Complejo:

x = 3+5j  
y = 5j  
z = -5j  
  
print(type(x))  
print(type(y))  
print(type(z))

[Inténtalo tú mismo "](https://www.w3schools.com/python/trypython.asp?filename=demo_numbers_complex)

## **Type Conversion**

Puede convertir de un tipo a otro con los métodos int(), float()y :complex()

### **Ejemplo**

Convertir de un tipo a otro:

x = 1    # int  
y = 2.8  # float  
z = 1j   # complex  
  
#convert from int to float:  
a = float(x)  
  
#convert from float to int:  
b = int(y)  
  
#convert from int to complex:  
c = complex(x)  
  
print(a)  
print(b)  
print(c)  
  
print(type(a))  
print(type(b))  
print(type(c))

**Nota:** No puede convertir números complejos en otro tipo de número.

## **Random Number**

Python no tiene una random()función para hacer un número aleatorio, pero Python tiene un módulo integrado llamado randomque se puede usar para hacer números aleatorios:

### **Ejemplo**

Importe el módulo aleatorio y muestre un número aleatorio entre 1 y 9:

import random  
  
print(random.randrange(1, 10))

En nuestra [Referencia del módulo aleatorio](https://www.w3schools.com/python/module_random.asp) , aprenderá más sobre el módulo aleatorio.

# Python Casting

## **Especificar un tipo de variable**

Puede haber ocasiones en las que desee especificar un tipo en una variable. Esto se puede hacer con fundición. Python es un lenguaje orientado a objetos y, como tal, utiliza clases para definir tipos de datos, incluidos sus tipos primitivos.

Por lo tanto, la conversión en python se realiza mediante funciones de constructor:

* int() : construye un número entero a partir de un literal entero, un literal flotante (eliminando todos los decimales) o un literal de cadena (siempre que la cadena represente un número entero)
* float() : construye un número flotante a partir de un literal entero, un literal flotante o un literal de cadena (siempre que la cadena represente un flotante o un entero)
* str() : construye una cadena a partir de una amplia variedad de tipos de datos, incluidas cadenas, literales enteros y literales flotantes

### **Ejemplo**

enteros:

x = int(1)   # x will be 1  
y = int(2.8) # y will be 2  
z = int("3") # z will be 3

### **Ejemplo**

flotadores:

x = float(1)     # x will be 1.0  
y = float(2.8)   # y will be 2.8  
z = float("3")   # z will be 3.0  
w = float("4.2") # w will be 4.2

### **Ejemplo**

Instrumentos de cuerda:

x = str("s1") # x will be 's1'  
y = str(2)    # y will be '2'  
z = str(3.0)  # z will be '3.0'

# Python Strings

## **Strings**

Las cadenas en python están entre comillas simples o comillas dobles.

'hola' es lo mismo que 'hola' .

Puede mostrar un literal de cadena con la print()función:

### **Ejemplo**

print("Hello")  
print('Hello')

## **Assign String to a Variable**

La asignación de una cadena a una variable se realiza con el nombre de la variable seguido de un signo igual y la cadena:

### **Ejemplo**

a = "Hello"  
print(a)

## **Multiline Strings**

## Puede asignar una cadena de varias líneas a una variable usando tres comillas:

### **Ejemplo**

Puede utilizar tres comillas dobles:

a = """Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua."""  
print(a)

O tres comillas simples:

### **Ejemplo**

a = '''Lorem ipsum dolor sit amet,  
consectetur adipiscing elit,  
sed do eiusmod tempor incididunt  
ut labore et dolore magna aliqua.'''  
print(a)

**Nota:** en el resultado, los saltos de línea se insertan en la misma posición que en el código.

## **Strings are Arrays**

Como muchos otros lenguajes de programación populares, las cadenas en Python son matrices de bytes que representan caracteres Unicode.

Sin embargo, Python no tiene un tipo de datos de caracteres, un solo carácter es simplemente una cadena con una longitud de 1.

Se pueden usar corchetes para acceder a elementos de la cadena.

### **Ejemplo**

Obtenga el carácter en la posición 1 (recuerde que el primer carácter tiene la posición 0):

a = "Hello, World!"  
print(a[1])

## **Looping Through a String**

Dado que las cadenas son matrices, podemos recorrer los caracteres de una cadena, con un forbucle.

### **Ejemplo**

Repasa las letras de la palabra "banana":

for x in "banana":  
  print(x)

Obtenga más información sobre For Loops en nuestro capítulo de [Python For Loops .](https://www.w3schools.com/python/python_for_loops.asp)

## **String Length**

Para obtener la longitud de una cadena, use la len()función.

### **Ejemplo**

La len()función devuelve la longitud de una cadena:

a = "Hello, World!"  
print(len(a))

## **Check String**

Para verificar si una determinada frase o carácter está presente en una cadena, podemos usar la palabra clave in.

### **Ejemplo**

Compruebe si "gratis" está presente en el siguiente texto:

txt = "The best things in life are free!"  
print("free" in txt)

Úselo en una if declaración:

### **Ejemplo**

Imprima solo si "gratis" está presente:

txt = "The best things in life are free!"  
if "free" in txt:  
  print("Yes, 'free' is present.")

Obtenga más información sobre las sentencias If en nuestro capítulo [Python If...Else .](https://www.w3schools.com/python/python_conditions.asp)

## **Check if NOT**

Para verificar si una determinada frase o carácter NO está presente en una cadena, podemos usar la palabra clave not in.

### **Ejemplo**

Compruebe si "caro" NO está presente en el siguiente texto:

txt = "The best things in life are free!"  
print("expensive" not in txt)

Úselo en una ifdeclaración:

### **Ejemplo**

imprima solo si "caro" NO está presente:

txt = "The best things in life are free!"  
if "expensive" not in txt:  
  print("No, 'expensive' is NOT present.")

# Python - Slicing Strings

## **Slicing**

Puede devolver un rango de caracteres utilizando la sintaxis de división.

Especifique el índice inicial y el índice final, separados por dos puntos, para devolver una parte de la cadena.

### **Ejemplo**

Obtenga los caracteres de la posición 2 a la posición 5 (no incluidos):

b = "Hello, World!"  
print(b[2:5])

**Nota:** El primer carácter tiene el índice 0.

## **Slice From the Start**

Al omitir el índice de inicio, el rango comenzará en el primer carácter:

### **Ejemplo**

Consigue los personajes desde el inicio hasta la posición 5 (no incluido):

b = "Hello, World!"  
print(b[:5])

## **Slice To the End**

Al omitir el índice final , el rango irá hasta el final:

### **Ejemplo**

Consigue los personajes desde la posición 2 y hasta el final:

b = "Hello, World!"  
print(b[2:])

## **Negative Indexing**

Utilice índices negativos para iniciar el segmento desde el final de la cadena:

### **Ejemplo**

Consigue los personajes:

De: "o" en "Mundo!" (posición -5)

Para, pero no incluido: "d" en "¡Mundo!" (posición -2):

b = "Hello, World!"  
print(b[-5:-2])

# Python - Modify Strings

Python tiene un conjunto de métodos integrados que puede usar en cadenas.

## **Upper Case**

### **Ejemplo**

El upper()método devuelve la cadena en mayúsculas:

a = "Hello, World!"  
print(a.upper())

## **Lower Case**

### **Ejemplo**

El lower()método devuelve la cadena en minúsculas:

a = "Hello, World!"  
print(a.lower())

## **Remove Whitespace**

El espacio en blanco es el espacio antes y/o después del texto real, y muy a menudo desea eliminar este espacio.

### **Ejemplo**

El strip()método elimina cualquier espacio en blanco desde el principio o el final:

a = " Hello, World! "  
print(a.strip()) # returns "Hello, World!"

## **Replace String**

### **Ejemplo**

El replace()método reemplaza una cadena con otra cadena:

a = "Hello, World!"  
print(a.replace("H", "J"))

## **Split String**

El split()método devuelve una lista donde el texto entre el separador especificado se convierte en los elementos de la lista.

### **Ejemplo**

El split()método divide la cadena en subcadenas si encuentra instancias del separador:

a = "Hello, World!"  
print(a.split(",")) # returns ['Hello', ' World!']

# Python - String Concatenation

## **String Concatenation**

Para concatenar o combinar dos cadenas, puede usar el operador +.

### **Ejemplo**

Combinar variable acon variable ben variable c:

a = "Hello"  
b = "World"  
c = a + b  
print(c)

### **Ejemplo**

Para agregar un espacio entre ellos, agregue un " ":

a = "Hello"  
b = "World"  
c = a + " " + b  
print(c)

# Python - Format - Strings

## **Formato de cadena**

Como aprendimos en el capítulo Variables de Python, no podemos combinar cadenas y números como este:

### **Ejemplo**

age = 36  
txt = "My name is John, I am " + age  
print(txt)

¡Pero podemos combinar cadenas y números usando el format()método!

El format()método toma los argumentos pasados, les da formato y los coloca en la cadena donde {}están los marcadores de posición:

### **Ejemplo**

Use el format()método para insertar números en cadenas:

age = 36  
txt = "My name is John, and I am {}"  
print(txt.format(age))

El método format() toma un número ilimitado de argumentos y se colocan en los marcadores de posición respectivos:

### **Ejemplo**

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want {} pieces of item {} for {} dollars."  
print(myorder.format(quantity, itemno, price))

Puede usar números de índice {0}para asegurarse de que los argumentos se colocan en los marcadores de posición correctos:

### **Ejemplo**

quantity = 3  
itemno = 567  
price = 49.95  
myorder = "I want to pay {2} dollars for {0} pieces of item {1}."  
print(myorder.format(quantity, itemno, price))

# Python - Escape Characters

## **Escape Character**

Para insertar caracteres que no son válidos en una cadena, utilice un carácter de escape.

Un carácter de escape es una barra invertida \seguida del carácter que desea insertar.

Un ejemplo de un carácter ilegal es una comilla doble dentro de una cadena que está entre comillas dobles:

### **Ejemplo**

Obtendrá un error si usa comillas dobles dentro de una cadena que está entre comillas dobles:

txt = "We are the so-called "Vikings" from the north."

Para solucionar este problema, utilice el carácter de escape \":

### **Ejemplo**

El carácter de escape le permite usar comillas dobles cuando normalmente no se le permitiría:

txt = "We are the so-called \"Vikings\" from the north."

## **Escape Characters**

Otros caracteres de escape usados ​​en Python:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** | **Try it** |
| \' | Single Quote | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_escape2) |
| \\ | Backslash | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_backslash) |
| \n | New Line | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_newline) |
| \r | Carriage Return | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_r) |
| \t | Tab | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_t) |
| \b | Backspace | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_b) |
| \f | Form Feed |  |
| \ooo | Octal value | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_string_octal) |
| \xhh | Hex value |  |

# Python - String Methods

## **Métodos de cadena**

Python tiene un conjunto de métodos integrados que puede usar en cadenas.

**Nota:** Todos los métodos de cadena devuelven nuevos valores. No cambian la cadena original.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [capitalize()](https://www.w3schools.com/python/ref_string_capitalize.asp) | Converts the first character to upper case |
| [casefold()](https://www.w3schools.com/python/ref_string_casefold.asp) | Converts string into lower case |
| [center()](https://www.w3schools.com/python/ref_string_center.asp) | Returns a centered string |
| [count()](https://www.w3schools.com/python/ref_string_count.asp) | Returns the number of times a specified value occurs in a string |
| [encode()](https://www.w3schools.com/python/ref_string_encode.asp) | Returns an encoded version of the string |
| [endswith()](https://www.w3schools.com/python/ref_string_endswith.asp) | Returns true if the string ends with the specified value |
| [expandtabs()](https://www.w3schools.com/python/ref_string_expandtabs.asp) | Sets the tab size of the string |
| [find()](https://www.w3schools.com/python/ref_string_find.asp) | Searches the string for a specified value and returns the position of where it was found |
| [format()](https://www.w3schools.com/python/ref_string_format.asp) | Formats specified values in a string |
| format\_map() | Formats specified values in a string |
| [index()](https://www.w3schools.com/python/ref_string_index.asp) | Searches the string for a specified value and returns the position of where it was found |
| [isalnum()](https://www.w3schools.com/python/ref_string_isalnum.asp) | Returns True if all characters in the string are alphanumeric |
| [isalpha()](https://www.w3schools.com/python/ref_string_isalpha.asp) | Returns True if all characters in the string are in the alphabet |
| [isdecimal()](https://www.w3schools.com/python/ref_string_isdecimal.asp) | Returns True if all characters in the string are decimals |
| [isdigit()](https://www.w3schools.com/python/ref_string_isdigit.asp) | Returns True if all characters in the string are digits |
| [isidentifier()](https://www.w3schools.com/python/ref_string_isidentifier.asp) | Returns True if the string is an identifier |
| [islower()](https://www.w3schools.com/python/ref_string_islower.asp) | Returns True if all characters in the string are lower case |
| [isnumeric()](https://www.w3schools.com/python/ref_string_isnumeric.asp) | Returns True if all characters in the string are numeric |
| [isprintable()](https://www.w3schools.com/python/ref_string_isprintable.asp) | Returns True if all characters in the string are printable |
| [isspace()](https://www.w3schools.com/python/ref_string_isspace.asp) | Returns True if all characters in the string are whitespaces |
| [istitle()](https://www.w3schools.com/python/ref_string_istitle.asp) | Returns True if the string follows the rules of a title |
| [isupper()](https://www.w3schools.com/python/ref_string_isupper.asp) | Returns True if all characters in the string are upper case |
| [join()](https://www.w3schools.com/python/ref_string_join.asp) | Joins the elements of an iterable to the end of the string |
| [ljust()](https://www.w3schools.com/python/ref_string_ljust.asp) | Returns a left justified version of the string |
| [lower()](https://www.w3schools.com/python/ref_string_lower.asp) | Converts a string into lower case |
| [lstrip()](https://www.w3schools.com/python/ref_string_lstrip.asp) | Returns a left trim version of the string |
| [maketrans()](https://www.w3schools.com/python/ref_string_maketrans.asp) | Returns a translation table to be used in translations |
| [partition()](https://www.w3schools.com/python/ref_string_partition.asp) | Returns a tuple where the string is parted into three parts |
| [replace()](https://www.w3schools.com/python/ref_string_replace.asp) | Returns a string where a specified value is replaced with a specified value |
| [rfind()](https://www.w3schools.com/python/ref_string_rfind.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rindex()](https://www.w3schools.com/python/ref_string_rindex.asp) | Searches the string for a specified value and returns the last position of where it was found |
| [rjust()](https://www.w3schools.com/python/ref_string_rjust.asp) | Returns a right justified version of the string |
| [rpartition()](https://www.w3schools.com/python/ref_string_rpartition.asp) | Returns a tuple where the string is parted into three parts |
| [rsplit()](https://www.w3schools.com/python/ref_string_rsplit.asp) | Splits the string at the specified separator, and returns a list |
| [rstrip()](https://www.w3schools.com/python/ref_string_rstrip.asp) | Returns a right trim version of the string |
| [split()](https://www.w3schools.com/python/ref_string_split.asp) | Splits the string at the specified separator, and returns a list |
| [splitlines()](https://www.w3schools.com/python/ref_string_splitlines.asp) | Splits the string at line breaks and returns a list |
| [startswith()](https://www.w3schools.com/python/ref_string_startswith.asp) | Returns true if the string starts with the specified value |
| [strip()](https://www.w3schools.com/python/ref_string_strip.asp) | Returns a trimmed version of the string |
| [swapcase()](https://www.w3schools.com/python/ref_string_swapcase.asp) | Swaps cases, lower case becomes upper case and vice versa |
| [title()](https://www.w3schools.com/python/ref_string_title.asp) | Converts the first character of each word to upper case |
| [translate()](https://www.w3schools.com/python/ref_string_translate.asp) | Returns a translated string |
| [upper()](https://www.w3schools.com/python/ref_string_upper.asp) | Converts a string into upper case |
| [zfill()](https://www.w3schools.com/python/ref_string_zfill.asp) | Fills the string with a specified number of 0 values at the beginning |

# Python Booleans

Los booleanos representan uno de dos valores: Trueo False.

## **Valores booleanos**

En programación, a menudo necesitas saber si una expresión es Trueo False.

Puede evaluar cualquier expresión en Python y obtener una de dos respuestas, Trueo False.

Cuando compara dos valores, la expresión se evalúa y Python devuelve la respuesta booleana:

### **Ejemplo**

print(10 > 9)  
print(10 == 9)  
print(10 < 9)

Cuando ejecuta una condición en una declaración if, Python devuelve Trueo False:

### **Ejemplo**

Imprime un mensaje basado en si la condición es Trueo False:

a = 200  
b = 33  
  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")

## **Evaluar valores y variables**

La bool()función te permite evaluar cualquier valor, y darte Trueo False a cambio,

### **Ejemplo**

Evaluar una cadena y un número:

print(bool("Hello"))  
print(bool(15))

### **Ejemplo**

Evalúa dos variables:

x = "Hello"  
y = 15  
  
print(bool(x))  
print(bool(y))

## **La mayoría de los valores son verdaderos**

Casi cualquier valor se evalúa Truesi tiene algún tipo de contenido.

Cualquier cadena es True, excepto las cadenas vacías.

Cualquier número es True, excepto 0.

Cualquier lista, tupla, conjunto y diccionario son True, excepto los vacíos.

### **Ejemplo**

Lo siguiente devolverá True:

bool("abc")  
bool(123)  
bool(["apple", "cherry", "banana"])

## **Algunos valores son falsos**

De hecho, no hay muchos valores que se evalúen como False, excepto valores vacíos, como (), [], {}, "", el número 0y el valor None. Y, por supuesto, el valor Falsese evalúa como False.

### **Ejemplo**

Lo siguiente devolverá Falso:

bool(False)  
bool(None)  
bool(0)  
bool("")  
bool(())  
bool([])  
bool({})

Un valor más, u objeto en este caso, se evalúa como False, y eso es si tiene un objeto que se crea a partir de una clase con una \_\_len\_\_función que devuelve 0o False:

### **Ejemplo**

class myclass():  
  def \_\_len\_\_(self):  
    return 0  
  
myobj = myclass()  
print(bool(myobj))

## **Las funciones pueden devolver un valor booleano**

Puede crear funciones que devuelvan un valor booleano:

### **Ejemplo**

Imprime la respuesta de una función:

def myFunction() :  
  return True  
  
print(myFunction())

Puede ejecutar código basado en la respuesta booleana de una función:

### **Ejemplo**

Imprimir "¡SÍ!" si la función devuelve True, de lo contrario imprime "¡NO!":

def myFunction() :  
  return True  
  
if myFunction():  
  print("YES!")  
else:  
  print("NO!")

Python también tiene muchas funciones integradas que devuelven un valor booleano, como la isinstance() función, que se puede usar para determinar si un objeto es de cierto tipo de datos:

### **Ejemplo.** Compruebe si un objeto es un número entero o no:

x = 200  
print(isinstance(x, int))

# Python Operators

## **Operadores de Python**

Los operadores se utilizan para realizar operaciones en variables y valores.

En el siguiente ejemplo, usamos el +operador para sumar dos valores:

### **Ejemplo**

print(10 + 5)

Python divide a los operadores en los siguientes grupos:

* Operadores aritméticos
* Operadores de Asignación
* Operadores de comparación
* Operadores logicos
* Operadores de identidad
* Operadores de membresía
* Operadores bit a bit

**Python Arithmetic Operators**

Los operadores aritméticos se utilizan con valores numéricos para realizar operaciones matemáticas comunes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| + | Addition | x + y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_add) |
| - | Subtraction | x - y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_sub) |
| \* | Multiplication | x \* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mult) |
| / | Division | x / y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_div) |
| % | Modulus | x % y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_mod) |
| \*\* | Exponentiation | x \*\* y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_exp) |
| // | Floor division | x // y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_floordiv) |

## **Python Assignment Operators**

Los operadores de asignación se utilizan para asignar valores a las variables:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Example** | **Same As** | **Try it** |
| = | x = 5 | x = 5 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass1) |
| += | x += 3 | x = x + 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass2) |
| -= | x -= 3 | x = x - 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass3) |
| \*= | x \*= 3 | x = x \* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass4) |
| /= | x /= 3 | x = x / 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass5) |
| %= | x %= 3 | x = x % 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass6) |
| //= | x //= 3 | x = x // 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass7) |
| \*\*= | x \*\*= 3 | x = x \*\* 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass8) |
| &= | x &= 3 | x = x & 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass9) |
| |= | x |= 3 | x = x | 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass10) |
| ^= | x ^= 3 | x = x ^ 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass11) |
| >>= | x >>= 3 | x = x >> 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass12) |
| <<= | x <<= 3 | x = x << 3 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_ass13) |

## **Python Comparison Operators**

Los operadores de comparación se utilizan para comparar dos valores:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Name** | **Example** | **Try it** |
| == | Equal | x == y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare1) |
| != | Not equal | x != y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare2) |
| > | Greater than | x > y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare4) |
| < | Less than | x < y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare5) |
| >= | Greater than or equal to | x >= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare6) |
| <= | Less than or equal to | x <= y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_compare7) |

## **Python Logical Operators**

Los operadores lógicos se utilizan para combinar sentencias condicionales:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| and | Returns True if both statements are true | x < 5 and  x < 10 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical1) |
| or | Returns True if one of the statements is true | x < 5 or x < 4 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical2) |
| not | Reverse the result, returns False if the result is true | not(x < 5 and x < 10) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_logical3) |

## **Python Identity Operators**

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si en realidad son el mismo objeto, con la misma ubicación de memoria:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| is | Returns True if both variables are the same object | x is y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity1) |
| is not | Returns True if both variables are not the same object | x is not y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_identity2) |

## **Python Membership Operators**

Los operadores de membresía se utilizan para probar si una secuencia se presenta en un objeto:

|  |  |  |  |
| --- | --- | --- | --- |
| **Operator** | **Description** | **Example** | **Try it** |
| in | Returns True if a sequence with the specified value is present in the object | x in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership1) |
| not in | Returns True if a sequence with the specified value is not present in the object | x not in y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_membership2) |

## **Python Bitwise Operators**

Los operadores bit a bit se utilizan para comparar números (binarios):

|  |  |  |
| --- | --- | --- |
| **Operator** | **Name** | **Description** |
| & | AND | Sets each bit to 1 if both bits are 1 |
| | | OR | Sets each bit to 1 if one of two bits is 1 |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 |
| ~ | NOT | Inverts all the bits |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off |

# Python Lists

mylist = ["apple", "banana", "cherry"]

## **List** Las listas se utilizan para almacenar varios elementos en una sola variable.

Las listas son uno de los 4 tipos de datos incorporados en Python que se utilizan para almacenar colecciones de datos, los otros 3 son [Tuple](https://www.w3schools.com/python/python_tuples.asp) , [Set](https://www.w3schools.com/python/python_sets.asp) y [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp) , todos con diferentes calidades y usos.

Las listas se crean usando corchetes:

### **Ejemplo** Crear una lista:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## **List Items**

Los elementos de la lista están ordenados, se pueden cambiar y permiten valores duplicados.

Los elementos de la lista están indexados, el primer elemento tiene índice [0], el segundo elemento tiene índice, [1]etc.

## **Ordered**

Cuando decimos que las listas están ordenadas, significa que los elementos tienen un orden definido y ese orden no cambiará.

Si agrega nuevos elementos a una lista, los nuevos elementos se colocarán al final de la lista.

**Nota:** Hay algunos [métodos de lista](https://www.w3schools.com/python/python_lists_methods.asp) que cambiarán el orden, pero en general: el orden de los elementos no cambiará.

## **Changeable**

La lista se puede cambiar, lo que significa que podemos cambiar, agregar y eliminar elementos en una lista después de que se haya creado.

## **Allow Duplicates**

Dado que las listas están indexadas, las listas pueden tener elementos con el mismo valor:

### **Ejemplo** Las listas permiten valores duplicados:

thislist = ["apple", "banana", "cherry", "apple", "cherry"]  
print(thislist)

## **List Length**

Para determinar cuántos elementos tiene una lista, use la len()función:

### **Ejemplo** Imprime el número de artículos en la lista:

thislist = ["apple", "banana", "cherry"]  
print(len(thislist))

## **List Items - Data Types**

Los elementos de la lista pueden ser de cualquier tipo de datos:

### **Ejemplo** Tipos de datos de cadena, int y booleanos:

list1 = ["apple", "banana", "cherry"]  
list2 = [1, 5, 7, 9, 3]  
list3 = [True, False, False]

Una lista puede contener diferentes tipos de datos:

### **Ejemplo** Una lista con cadenas, enteros y valores booleanos:

list1 = ["abc", 34, True, 40, "male"]

## **type()**

Desde la perspectiva de Python, las listas se definen como objetos con el tipo de datos 'lista':

<class 'list'>

### **Ejemplo** ¿Cuál es el tipo de datos de una lista?

mylist = ["apple", "banana", "cherry"]  
print(type(mylist))

## **The list() Constructor**

También es posible usar el constructor list() al crear una nueva lista.

### **Ejemplo** Usando el list()constructor para hacer una Lista:

thislist = list(("apple", "banana", "cherry")) # note the double round-brackets  
print(thislist)

## **Python Collections (Arrays)**

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* **La lista** es una colección ordenada y modificable. Permite miembros duplicados.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) es una colección ordenada e inmutable. Permite miembros duplicados.
* [**Conjunto**](https://www.w3schools.com/python/python_sets.asp) es una colección desordenada, inmutable\* y no indexada. No hay miembros duplicados.
* [**El diccionario**](https://www.w3schools.com/python/python_dictionaries.asp) es una colección ordenada\*\* y modificable. No hay miembros duplicados.

\*Los elementos establecidos no se pueden cambiar, pero puede eliminar y/o agregar elementos cuando lo desee.

\*\*A partir de la versión 3.7 de Python, los diccionarios se ordenan . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado y podría significar un aumento en la eficiencia o la seguridad.

## **Access Items**

Los elementos de la lista están indexados y puede acceder a ellos consultando el número de índice:

### **Ejemplo** Imprime el segundo elemento de la lista:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

**Nota:** El primer elemento tiene índice 0.

### **Negative Indexing**

La indexación negativa significa comenzar desde el final.

-1se refiere al último elemento, -2se refiere al penúltimo elemento, etc.

### **Ejemplo** Imprime el último elemento de la lista:

thislist = ["apple", "banana", "cherry"]  
print(thislist[-1])

### **Range of Indexes**

Puede especificar un rango de índices especificando dónde comenzar y dónde terminar el rango.

Al especificar un rango, el valor devuelto será una nueva lista con los elementos especificados.

### **Ejemplo** Devuelve el tercer, cuarto y quinto elemento:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:5])

**Nota:** La búsqueda comenzará en el índice 2 (incluido) y terminará en el índice 5 (no incluido).

Recuerde que el primer elemento tiene índice 0.

Al omitir el valor inicial, el rango comenzará en el primer elemento:

### **Ejemplo** Este ejemplo devuelve los elementos desde el principio hasta, pero SIN incluir, "kiwi":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[:4])

Al omitir el valor final, el rango pasará al final de la lista:

### **Ejemplo** Este ejemplo devuelve los elementos desde "cereza" hasta el final:

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[2:])

### **Range of Negative Indexes**

Especifique índices negativos si desea iniciar la búsqueda desde el final de la lista:

### **Ejemplo** Este ejemplo devuelve los elementos de "naranja" (-4) a, pero NO incluye "mango" (-1):

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "melon", "mango"]  
print(thislist[-4:-1])

## **Check if Item Exists**

Para determinar si un elemento específico está presente en una lista, use la inpalabra clave:

### **Ejemplo** Compruebe si "apple" está presente en la lista:

thislist = ["apple", "banana", "cherry"]  
if "apple" in thislist:  
  print("Yes, 'apple' is in the fruits list")

## **Cambiar el valor del artículo**

Para cambiar el valor de un elemento específico, consulte el número de índice:

### **Ejemplo** Cambia el segundo elemento:

thislist = ["apple", "banana", "cherry"]  
thislist[1] = "blackcurrant"  
print(thislist)

## **Cambiar un rango de valores de elementos**

Para cambiar el valor de los elementos dentro de un rango específico, defina una lista con los nuevos valores y consulte el rango de números de índice donde desea insertar los nuevos valores:

### **Ejemplo** Cambie los valores "plátano" y "cereza" por los valores "grosella negra" y "sandía":

thislist = ["apple", "banana", "cherry", "orange", "kiwi", "mango"]  
thislist[1:3] = ["blackcurrant", "watermelon"]  
print(thislist)

Si inserta más elementos de los que reemplaza, los nuevos elementos se insertarán donde especificó, y los elementos restantes se moverán en consecuencia:

### **Ejemplo** Cambie el segundo valor reemplazándolo con dos nuevos valores:

thislist = ["apple", "banana", "cherry"]  
thislist[1:2] = ["blackcurrant", "watermelon"]  
print(thislist)

**Nota:** La longitud de la lista cambiará cuando la cantidad de elementos insertados no coincida con la cantidad de elementos reemplazados.

Si inserta menos elementos de los que reemplaza, los nuevos elementos se insertarán donde especificó, y los elementos restantes se moverán en consecuencia:

### **Ejemplo** Cambie el segundo y tercer valor reemplazándolo con un valor:

thislist = ["apple", "banana", "cherry"]  
thislist[1:3] = ["watermelon"]  
print(thislist)

## **Insertar elementos**

Para insertar un nuevo elemento de lista, sin reemplazar ninguno de los valores existentes, podemos usar el insert() método.

El insert() método inserta un elemento en el índice especificado:

### **Ejemplo** Inserte "sandía" como tercer elemento:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(2, "watermelon")  
print(thislist)

**Nota:** Como resultado del ejemplo anterior, la lista ahora contendrá 4 elementos.

# Add List Items

## **Agregar elementos**

Para agregar un elemento al final de la lista, use el método append() :

### **Ejemplo** Usando el append() método para agregar un elemento:

thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)

## **Insertar elementos**

Para insertar un elemento de lista en un índice específico, utilice el insert() método.

El insert() método inserta un elemento en el índice especificado:

### **Ejemplo** Insertar un elemento como segunda posición:

thislist = ["apple", "banana", "cherry"]  
thislist.insert(1, "orange")  
print(thislist)

**Nota:** Como resultado de los ejemplos anteriores, las listas ahora contendrán 4 elementos.

## **Ampliar lista .**Para agregar elementos de otra lista a la lista actual, use el extend()método.

### **Ejemplo** Agregue los elementos de tropicala thislist:

thislist = ["apple", "banana", "cherry"]  
tropical = ["mango", "pineapple", "papaya"]  
thislist.extend(tropical)  
print(thislist)

Los elementos se añadirán al final de la lista.

## **Agregar cualquier iterable**

El extend()método no tiene que agregar listas , puede agregar cualquier objeto iterable (tuplas, conjuntos, diccionarios, etc.).

### **Ejemplo** Agregue elementos de una tupla a una lista:

thislist = ["apple", "banana", "cherry"]  
thistuple = ("kiwi", "orange")  
thislist.extend(thistuple)  
print(thislist)

## **Eliminar elemento especificado**

El remove()método elimina el elemento especificado.

### **Ejemplo** Eliminar "plátano":

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

## **Eliminar índice especificado**

El pop()método elimina el índice especificado.

### **Ejemplo** Eliminar el segundo elemento:

thislist = ["apple", "banana", "cherry"]  
thislist.pop(1)  
print(thislist)

Si no especifica el índice, el pop()método elimina el último elemento.

### **Ejemplo** Eliminar el último elemento:

thislist = ["apple", "banana", "cherry"]  
thislist.pop()  
print(thislist)

La delpalabra clave también elimina el índice especificado:

### **Ejemplo** Eliminar el primer elemento:

thislist = ["apple", "banana", "cherry"]  
del thislist[0]  
print(thislist)

La delpalabra clave también puede eliminar la lista por completo.

### **Ejemplo** Eliminar toda la lista:

thislist = ["apple", "banana", "cherry"]  
del thislist

## **Borrar la lista**

El clear()método vacía la lista.

La lista aún permanece, pero no tiene contenido.

### **Ejemplo** Borrar el contenido de la lista:

thislist = ["apple", "banana", "cherry"]  
thislist.clear()  
print(thislist)

## **Recorrer una lista**

Puede recorrer los elementos de la lista usando un for bucle:

### **Ejemplo** Imprima todos los elementos de la lista, uno por uno:

thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
  print(x)

Obtenga más información sobre forlos bucles en nuestro capítulo [Python For Loops .](https://www.w3schools.com/python/python_for_loops.asp)

## **Bucle a través de los números de índice**

También puede recorrer los elementos de la lista consultando su número de índice.

Utilice las funciones range()y len()para crear un iterable adecuado.

### **Ejemplo** Imprima todos los elementos haciendo referencia a su número de índice:

thislist = ["apple", "banana", "cherry"]  
for i in range(len(thislist)):  
  print(thislist[i])

El iterable creado en el ejemplo anterior es [0, 1, 2].

## **Usar un ciclo while**

Puede recorrer los elementos de la lista mediante un whilebucle.

Use la len()función para determinar la longitud de la lista, luego comience en 0 y recorra los elementos de la lista consultando sus índices.

Recuerde aumentar el índice en 1 después de cada iteración.

### **Ejemplo** Imprima todos los artículos, usando un whilebucle para pasar por todos los números de índice

thislist = ["apple", "banana", "cherry"]  
i = 0  
while i < len(thislist):  
  print(thislist[i])  
  i = i + 1

Obtenga más información sobre whilelos bucles en nuestro Capítulo de [bucles while de Python .](https://www.w3schools.com/python/python_while_loops.asp)

## **Bucles utilizando la comprensión de lista**

List Comprehension ofrece la sintaxis más corta para recorrer listas:

### **Ejemplo** Un for bucle de mano corto que imprimirá todos los elementos en una lista:

thislist = ["apple", "banana", "cherry"]  
[print(x) for x in thislist]

## **Lista de comprensión**

La comprensión de listas ofrece una sintaxis más corta cuando desea crear una nueva lista basada en los valores de una lista existente.

Ejemplo: Basado en una lista de frutas, desea una nueva lista que contenga solo las frutas con la letra "a" en el nombre.

Sin comprensión de lista, tendrá que escribir una fordeclaración con una prueba condicional dentro:

### **Ejemplo**

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = []  
  
for x in fruits:  
  if "a" in x:  
    newlist.append(x)  
  
print(newlist)

Con la comprensión de listas puedes hacer todo eso con solo una línea de código:

### **Ejemplo**

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
  
newlist = [x for x in fruits if "a" in x]  
  
print(newlist)

## **la sintaxis**

newlist = [expression for item in iterable if condition == True]

El valor de retorno es una nueva lista, dejando la lista anterior sin cambios.

### **Condición**

La condición es como un filtro que solo acepta los elementos que se valoran en True.

### **Ejemplo** Solo acepte artículos que no sean "apple":

newlist = [x for x in fruits if x != "apple"]

La condición if x != "apple"  volverá Truepara todos los elementos que no sean "manzana", haciendo que la nueva lista contenga todas las frutas excepto "manzana".

La condición es opcional y se puede omitir:

### **Ejemplo** Sin if declaración:

newlist = [x for x in fruits]

### **Iterable**

El iterable puede ser cualquier objeto iterable, como una lista, una tupla, un conjunto, etc.

### **Ejemplo** Puede usar la range() función para crear un iterable:

newlist = [x for x in range(10)]

Mismo ejemplo, pero con una condición:

### **Ejemplo** Aceptar solo números inferiores a 5:

newlist = [x for x in range(10) if x < 5]

### **Expresión.** La expresión es el elemento actual en la iteración, pero también es el resultado, que puede manipular antes de que termine como un elemento de lista en la nueva lista:

### **Ejemplo.** Establezca los valores en la nueva lista en mayúsculas:

newlist = [x.upper() for x in fruits]

Puedes establecer el resultado como quieras:

### **Ejemplo.** Establezca todos los valores en la nueva lista en 'hola':

newlist = ['hello' for x in fruits]

La expresión también puede contener condiciones, no como un filtro, sino como una forma de manipular el resultado:

### **Ejemplo.** Devuelve "naranja" en lugar de "plátano":

newlist = [x if x != "banana" else "orange" for x in fruits]

La expresión del ejemplo anterior dice:

"Devolver el artículo si no es plátano, si es plátano devolver naranja".

# Python - Ordenar listas

## **Ordenar lista alfanuméricamente**

Los objetos de lista tienen un sort()método que ordenará la lista de forma alfanumérica, ascendente, de forma predeterminada:

### **Ejemplo.** Ordena la lista alfabéticamente:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort()  
print(thislist)

### **Ejemplo.** Ordene la lista numéricamente:

thislist = [100, 50, 65, 82, 23]  
thislist.sort()  
print(thislist)

## **Orden descendiente**

Para ordenar de forma descendente, use el argumento de palabra clave reverse = True:

### **Ejemplo**

Ordenar la lista de forma descendente:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort(reverse = True)  
print(thislist)

### **Ejemplo**

Ordenar la lista de forma descendente:

thislist = [100, 50, 65, 82, 23]  
thislist.sort(reverse = True)  
print(thislist)

## **Personalizar la función de clasificación**

También puede personalizar su propia función utilizando el argumento de palabra clave .key = function

La función devolverá un número que se usará para ordenar la lista (primero el número más bajo):

### **Ejemplo**

Ordene la lista según lo cerca que esté el número de 50:

def myfunc(n):  
  return abs(n - 50)  
  
thislist = [100, 50, 65, 82, 23]  
thislist.sort(key = myfunc)  
print(thislist)

## **Clasificación insensible a mayúsculas y minúsculas**

De forma predeterminada, el sort()método distingue entre mayúsculas y minúsculas, lo que da como resultado que todas las letras mayúsculas se clasifiquen antes que las minúsculas:

### **Ejemplo.** La clasificación entre mayúsculas y minúsculas puede dar un resultado inesperado:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.sort()  
print(thislist)

Afortunadamente, podemos usar funciones integradas como funciones clave al ordenar una lista.

Entonces, si desea una función de clasificación que no distinga entre mayúsculas y minúsculas, use str.lower como una función clave:

### **Ejemplo.** Realice una ordenación de la lista que no distinga entre mayúsculas y minúsculas:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.sort(key = str.lower)  
print(thislist)

## **Orden inverso**

¿Qué sucede si desea invertir el orden de una lista, independientemente del alfabeto?

El reverse()método invierte el orden de clasificación actual de los elementos.

### **Ejemplo.** Invierta el orden de los elementos de la lista:

thislist = ["banana", "Orange", "Kiwi", "cherry"]  
thislist.reverse()  
print(thislist)

# Python - Copy Lists

No puede copiar una lista simplemente escribiendo list2 = list1, porque: list2solo será una referencia a list1, y los cambios realizados en list1automáticamente también se realizarán en list2.

Hay formas de hacer una copia, una forma es usar el método List incorporado copy().

### **Ejemplo .**Haz una copia de una lista con el copy()método:

thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)

Otra forma de hacer una copia es usar el método incorporado list().

### **Ejemplo.** Haz una copia de una lista con el list()método:

thislist = ["apple", "banana", "cherry"]  
mylist = list(thislist)  
print(mylist)

# Python - Unirse a listas

Hay varias formas de unir o concatenar dos o más listas en Python.

Una de las formas más fáciles es usando el + operador.

### **Ejemplo.** Únete a dos listas:

list1 = ["a", "b", "c"]  
list2 = [1, 2, 3]  
  
list3 = list1 + list2  
print(list3)

Otra forma de unir dos listas es agregar todos los elementos de list2 a list1, uno por uno:

### **Ejemplo** Agregue list2 en list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
for x in list2:  
  list1.append(x)  
  
print(list1)

O puede usar el extend() método, cuyo propósito es agregar elementos de una lista a otra lista:

### **Ejemplo** Use el extend() método para agregar list2 al final de list1:

list1 = ["a", "b" , "c"]  
list2 = [1, 2, 3]  
  
list1.extend(list2)  
print(list1)

# Python - List Methods

## **List Methods**

Python has a set of built-in methods that you can use on lists.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Añade un elemento al final de la lista. |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Elimina todos los elementos de la lista. |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Devuelve una copia de la lista. |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Devuelve el número de elementos con el valor especificado |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Agregue los elementos de una lista (o cualquier iterable), al final de la lista actual |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Devuelve el índice del primer elemento con el valor especificado |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Agrega un elemento en la posición especificada |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Elimina el elemento en la posición especificada |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Elimina el elemento con el valor especificado |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Invierte el orden de la lista |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Ordena la lista |

# Tuplas de Python

mytuple = ("apple", "banana", "cherry")

## **tupla**

Las tuplas se utilizan para almacenar varios elementos en una sola variable.

Tuple es uno de los 4 tipos de datos incorporados en Python que se utilizan para almacenar colecciones de datos, los otros 3 son [List](https://www.w3schools.com/python/python_lists.asp) , [Set](https://www.w3schools.com/python/python_sets.asp) y [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp) , todos con diferentes calidades y usos.

Una tupla es una colección ordenada e **inmutable** .

Las tuplas se escriben con corchetes.

### **Ejemplo** Crear una tupla:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## **Elementos de tupla**

Los elementos de tupla están ordenados, no se pueden modificar y permiten valores duplicados.

Los elementos de tupla están indexados, el primer elemento tiene índice [0], el segundo elemento tiene índice, [1]etc.

## **Ordenado**

Cuando decimos que las tuplas están ordenadas, significa que los elementos tienen un orden definido y ese orden no cambiará.

## **inmutable**

Las tuplas no se pueden cambiar, lo que significa que no podemos cambiar, agregar o eliminar elementos después de que se haya creado la tupla.

## **Permitir duplicados**

Dado que las tuplas están indexadas, pueden tener elementos con el mismo valor:

### **Ejemplo** Las tuplas permiten valores duplicados:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## **Longitud de la tupla**

Para determinar cuántos elementos tiene una tupla, use la len()función:

### **Ejemplon** Imprime el número de elementos en la tupla:

thistuple = ("apple", "banana", "cherry")  
print(len(thistuple))

## **Crear tupla con un elemento**

Para crear una tupla con un solo elemento, debe agregar una coma después del elemento; de lo contrario, Python no lo reconocerá como una tupla.

### **Ejemplo** Tupla de un elemento, recuerda la coma:

thistuple = ("apple",)  
print(type(thistuple))  
  
#NOT a tuple  
thistuple = ("apple")  
print(type(thistuple))

## **Elementos de tupla: tipos de datos**

Los elementos de tupla pueden ser de cualquier tipo de datos:

### **Ejemplo.** Tipos de datos de cadena, int y booleanos:

tuple1 = ("apple", "banana", "cherry")  
tuple2 = (1, 5, 7, 9, 3)  
tuple3 = (True, False, False)

Una tupla puede contener diferentes tipos de datos:

### **Ejemplo.** Una tupla con cadenas, enteros y valores booleanos:

tuple1 = ("abc", 34, True, 40, "male")

## **tipo()**

Desde la perspectiva de Python, las tuplas se definen como objetos con el tipo de datos 'tupla':

<class 'tuple'>

### **Ejemplo** ¿Cuál es el tipo de datos de una tupla?

mytuple = ("apple", "banana", "cherry")  
print(type(mytuple))

## **El constructor tupla()**

También es posible usar el constructor tuple() para hacer una tupla.

### **Ejemplo** Usando el método tuple() para hacer una tupla:

thistuple = tuple(("apple", "banana", "cherry")) # note the double round-brackets  
print(thistuple)

## **Colecciones de Python (matrices)**

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* [**La lista**](https://www.w3schools.com/python/python_lists.asp) es una colección ordenada y modificable. Permite miembros duplicados.
* **Tuple** es una colección ordenada e inmutable. Permite miembros duplicados.
* [**Conjunto**](https://www.w3schools.com/python/python_sets.asp) es una colección desordenada, inmutable\* y no indexada. No hay miembros duplicados.
* [**El diccionario**](https://www.w3schools.com/python/python_dictionaries.asp) es una colección ordenada\*\* y modificable. No hay miembros duplicados.

\*Los elementos establecidos no se pueden cambiar, pero puede eliminar y/o agregar elementos cuando lo desee.

\*\*A partir de la versión 3.7 de Python, los diccionarios se ordenan . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado y podría significar un aumento en la eficiencia o la seguridad.

## **Acceder a elementos de tupla**

Puede acceder a los elementos de la tupla consultando el número de índice, entre corchetes:

### **Ejemplo** Imprime el segundo elemento de la tupla:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])

**Nota:** El primer elemento tiene índice 0.

## **Indexación negativa**

La indexación negativa significa comenzar desde el final.

-1se refiere al último elemento, -2se refiere al penúltimo elemento, etc.

### **Ejemplo** Imprime el último elemento de la tupla:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[-1])

## **Gama de índices**

Puede especificar un rango de índices especificando dónde comenzar y dónde terminar el rango.

Al especificar un rango, el valor devuelto será una nueva tupla con los elementos especificados.

### **Ejemplo** Devuelve el tercer, cuarto y quinto elemento:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:5])

**Nota:** La búsqueda comenzará en el índice 2 (incluido) y terminará en el índice 5 (no incluido).

Recuerde que el primer elemento tiene índice 0.

Al omitir el valor inicial, el rango comenzará en el primer elemento:

### **Ejemplo** Este ejemplo devuelve los elementos desde el principio hasta, pero NO incluidos, "kiwi":

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[:4])

Al omitir el valor final, el rango pasará al final de la lista:

### **Ejemplo** Este ejemplo devuelve los elementos de "cherry" y hasta el final:

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[2:])

## **Rango de índices negativos**

Especifique índices negativos si desea iniciar la búsqueda desde el final de la tupla:

### **Ejemplo** Este ejemplo devuelve los elementos del índice -4 (incluido) al índice -1 (excluido)

thistuple = ("apple", "banana", "cherry", "orange", "kiwi", "melon", "mango")  
print(thistuple[-4:-1])

## **Comprobar si el artículo existe**

Para determinar si un elemento específico está presente en una tupla, use la inpalabra clave:

### **Ejemplo** Compruebe si "apple" está presente en la tupla:

thistuple = ("apple", "banana", "cherry")  
if "apple" in thistuple:  
  print("Yes, 'apple' is in the fruits tuple")

# Python - Actualizar tuplas

Las tuplas no se pueden cambiar, lo que significa que no puede cambiar, agregar o eliminar elementos una vez que se crea la tupla.

Pero hay algunas soluciones.

## **Cambiar valores de tupla**

Una vez que se crea una tupla, no puede cambiar sus valores. Las tuplas son **inmutables** , o **inmutables** como también se le llama.

Pero hay una solución. Puede convertir la tupla en una lista, cambiar la lista y volver a convertir la lista en una tupla.

### **Ejemplo** Convierta la tupla en una lista para poder cambiarla:

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

## **Agregar elementos**

Dado que las tuplas son inmutables, no tienen un append()método incorporado, pero hay otras formas de agregar elementos a una tupla.

1. **Convertir en una lista** : al igual que la solución alternativa para cambiar una tupla, puede convertirla en una lista, agregar sus elementos y volver a convertirla en una tupla.

### **Ejemplo** Convierta la tupla en una lista, agregue "naranja" y vuelva a convertirla en una tupla:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.append("orange")  
thistuple = tuple(y)

2. **Agregue una tupla a una tupla** . Puede agregar tuplas a tuplas, por lo que si desea agregar un elemento (o muchos), cree una nueva tupla con los elementos y agréguela a la tupla existente:

### **Ejemplo**Cree una nueva tupla con el valor "naranja" y agregue esa tupla:

thistuple = ("apple", "banana", "cherry")  
y = ("orange",)  
thistuple += y  
  
print(thistuple)

**Nota:** Al crear una tupla con un solo elemento, recuerde incluir una coma después del elemento; de lo contrario, no se identificará como una tupla.

## **Eliminar elementos**

**Nota:** No puede eliminar elementos en una tupla.

Las tuplas no se pueden **modificar** , por lo que no puede eliminar elementos, pero puede usar la misma solución alternativa que usamos para cambiar y agregar elementos de tupla:

### **Ejemplo** Convierta la tupla en una lista, elimine "manzana" y vuelva a convertirla en una tupla:

thistuple = ("apple", "banana", "cherry")  
y = list(thistuple)  
y.remove("apple")  
thistuple = tuple(y)

O puede eliminar la tupla por completo:

### **Ejemplo** La del palabra clave puede eliminar la tupla por completo:

thistuple = ("apple", "banana", "cherry")  
del thistuple  
print(thistuple) #this will raise an error because the tuple no longer exists

# Python - Desempaquetando una tupla

Cuando creamos una tupla, normalmente le asignamos valores. Esto se llama "empaquetar" una tupla:

### **Ejemplo** Empaquetando una tupla:

fruits = ("apple", "banana", "cherry")

Pero, en Python, también podemos volver a extraer los valores en variables. Esto se llama "desempacar":

### **Ejemplo** Desempaquetando una tupla:

fruits = ("apple", "banana", "cherry")  
  
(green, yellow, red) = fruits  
  
print(green)  
print(yellow)  
print(red)

**Nota:** El número de variables debe coincidir con el número de valores en la tupla, de lo contrario, debe usar un asterisco para recopilar los valores restantes como una lista.

## **Usando asterisco\***

Si el número de variables es menor que el número de valores, puede agregar un \*al nombre de la variable y los valores se asignarán a la variable como una lista:

### **Ejemplo** Asigne el resto de los valores como una lista llamada "rojo":

fruits = ("apple", "banana", "cherry", "strawberry", "raspberry")  
  
(green, yellow, \*red) = fruits  
  
print(green)  
print(yellow)  
print(red)

Si el asterisco se agrega a otro nombre de variable que no sea el último, Python asignará valores a la variable hasta que la cantidad de valores restantes coincida con la cantidad de variables restantes.

### **Ejemplo** Agrega una lista de valores a la variable "trópico":

fruits = ("apple", "mango", "papaya", "pineapple", "cherry")  
  
(green, \*tropic, red) = fruits  
  
print(green)  
print(tropic)  
print(red)

## **Bucle a través de una tupla**

Puede recorrer los elementos de la tupla mediante un forbucle.

### **Ejemplo** Iterar a través de los elementos e imprimir los valores:

thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
  print(x)

Obtenga más información sobre forlos bucles en nuestro capítulo [Python For Loops .](https://www.w3schools.com/python/python_for_loops.asp)

## **Bucle a través de los números de índice**

También puede recorrer los elementos de la tupla haciendo referencia a su número de índice.

Utilice las funciones range()y len()para crear un iterable adecuado.

### **Ejemplo** Imprima todos los elementos haciendo referencia a su número de índice:

thistuple = ("apple", "banana", "cherry")  
for i in range(len(thistuple)):  
  print(thistuple[i])

**Usar un ciclo while**

Puede recorrer los elementos de la lista mediante un whilebucle.

Use la len()función para determinar la longitud de la tupla, luego comience en 0 y recorra los elementos de la tupla haciendo referencia a sus índices.

Recuerde aumentar el índice en 1 después de cada iteración.

### **Ejemplo** Imprima todos los elementos, utilizando un whilebucle para pasar por todos los números de índice:

thistuple = ("apple", "banana", "cherry")  
i = 0  
while i < len(thistuple):  
  print(thistuple[i])  
  i = i + 1

## **Une dos tuplas**

Para unir dos o más tuplas puedes usar el + operador:

### **Ejemplo** Une dos tuplas:

tuple1 = ("a", "b" , "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

## **multiplicar tuplas**

Si desea multiplicar el contenido de una tupla un número determinado de veces, puede utilizar el \* operador:

### **Ejemplo** Multiplica la tupla de frutas por 2:

fruits = ("apple", "banana", "cherry")  
mytuple = fruits \* 2  
  
print(mytuple)

## **Métodos de tupla**

Python tiene dos métodos integrados que puede usar en tuplas.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Devuelve el número de veces que aparece un valor especificado en una tupla |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Busca en la tupla un valor específico y devuelve la posición donde se encontró |

# Sets de Python

myset = {"apple", "banana", "cherry"}

## **Colocar**

Los conjuntos se utilizan para almacenar varios elementos en una sola variable.

Set es uno de los 4 tipos de datos incorporados en Python que se utilizan para almacenar colecciones de datos, los otros 3 son [List](https://www.w3schools.com/python/python_lists.asp) , [Tuple](https://www.w3schools.com/python/python_tuples.asp) y [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp) , todos con diferentes calidades y usos.

Un conjunto es una colección desordenada , inmutable\* y no indexada .

**\* Nota:** Los elementos establecidos no se pueden modificar, pero puede eliminar elementos y agregar elementos nuevos.

Los conjuntos se escriben con corchetes.

### **Ejemplo** Crear un conjunto:

thisset = {"apple", "banana", "cherry"}  
print(thisset)

**Nota:** Los conjuntos no están ordenados, por lo que no puede estar seguro en qué orden aparecerán los elementos.

## **Establecer elementos**

Los elementos establecidos no están ordenados, no se pueden modificar y no permiten valores duplicados.

## **desordenado**

Desordenado significa que los elementos de un conjunto no tienen un orden definido.

Los elementos establecidos pueden aparecer en un orden diferente cada vez que los usa y no se puede hacer referencia a ellos por índice o clave.

## **inmutable**

Los elementos del conjunto no se pueden cambiar, lo que significa que no podemos cambiar los elementos después de que se haya creado el conjunto.

Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede eliminar elementos y agregar elementos nuevos.

## **No se permiten duplicados**

Los conjuntos no pueden tener dos artículos con el mismo valor.

### **Ejemplo** Los valores duplicados serán ignorados:

thisset = {"apple", "banana", "cherry", "apple"}  
  
print(thisset)

## **Obtener la longitud de un conjunto**

Para determinar cuántos artículos tiene un conjunto, use la len() función.

### **Ejemplo** Obtener el número de artículos en un conjunto:

thisset = {"apple", "banana", "cherry"}  
  
print(len(thisset))

## **Establecer elementos: tipos de datos**

Los elementos establecidos pueden ser de cualquier tipo de datos:

### **Ejemplo** Tipos de datos de cadena, int y booleanos:

set1 = {"apple", "banana", "cherry"}  
set2 = {1, 5, 7, 9, 3}  
set3 = {True, False, False}

Un conjunto puede contener diferentes tipos de datos:

### **Ejemplo** Un conjunto con cadenas, enteros y valores booleanos:

set1 = {"abc", 34, True, 40, "male"}

**tipo().** Desde la perspectiva de Python, los conjuntos se definen como objetos con el tipo de datos 'conjunto':

<class 'set'>

### **Ejemplo** ¿Cuál es el tipo de datos de un conjunto?

myset = {"apple", "banana", "cherry"}  
print(type(myset))

## **El constructor set()**

También es posible usar el constructor set() para hacer un conjunto.

### **Ejemplo** Usando el constructor set() para hacer un conjunto:

thisset = set(("apple", "banana", "cherry")) # note the double round-brackets  
print(thisset)

## **Colecciones de Python (matrices)**

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* [**La lista**](https://www.w3schools.com/python/python_lists.asp) es una colección ordenada y modificable. Permite miembros duplicados.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) es una colección ordenada e inmutable. Permite miembros duplicados.
* **Conjunto** es una colección desordenada, inmutable\* y no indexada. No hay miembros duplicados.
* [**El diccionario**](https://www.w3schools.com/python/python_dictionaries.asp) es una colección ordenada\*\* y modificable. No hay miembros duplicados.

\*Los elementos establecidos no se pueden cambiar, pero puede eliminar elementos y agregar nuevos elementos.

\*\*A partir de la versión 3.7 de Python, los diccionarios se ordenan . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado y podría significar un aumento en la eficiencia o la seguridad.

## **Elementos de acceso**

No puede acceder a los elementos de un conjunto haciendo referencia a un índice o una clave.

Pero puede recorrer los elementos del conjunto usando un for bucle, o preguntar si un valor específico está presente en un conjunto, usando la inpalabra clave.

### **Ejemplo** Recorra el conjunto e imprima los valores:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

### **Ejemplo** Compruebe si "banana" está presente en el conjunto:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" in thisset)

## **Cambiar artículos**

Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos.

# Python -Agregar elementos

Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos.

Para agregar un elemento a un conjunto, utilice el add() método.

### **Ejemplo** Agregue un artículo a un conjunto, usando el add() método:

thisset = {"apple", "banana", "cherry"}  
  
thisset.add("orange")  
  
print(thisset)

## **Agregar conjuntos** Para agregar elementos de otro conjunto al conjunto actual, utilice el update() método.

### **Ejemplo** Añadir elementos de tropicalen thisset:

thisset = {"apple", "banana", "cherry"}  
tropical = {"pineapple", "mango", "papaya"}  
  
thisset.update(tropical)  
  
print(thisset)

## **Agregar cualquier iterable**

El objeto en el update()método no tiene que ser un conjunto, puede ser cualquier objeto iterable (tuplas, listas, diccionarios, etc.).

### **Ejemplo** Agregue elementos de una lista a un conjunto:

thisset = {"apple", "banana", "cherry"}  
mylist = ["kiwi", "orange"]  
  
thisset.update(mylist)  
  
print(thisset)

**Remover el artículo**

Para eliminar un elemento de un conjunto, utilice el método remove()o el .discard()

### **Ejemplo** Elimina "banana" usando el remove() método:

thisset = {"apple", "banana", "cherry"}  
  
thisset.remove("banana")  
  
print(thisset)

**Nota:** Si el elemento a eliminar no existe, remove()generará un error.

### **Ejemplo** Elimina "banana" usando el discard() método:

thisset = {"apple", "banana", "cherry"}  
  
thisset.discard("banana")  
  
print(thisset)

**Nota:** Si el elemento a eliminar no existe, **NO**discard() generará un error.

También puede usar el pop()método para eliminar un elemento, pero este método eliminará el último elemento. Recuerde que los conjuntos están desordenados, por lo que no sabrá qué elemento se elimina.

El valor de retorno del pop()método es el elemento eliminado.

### **Ejemplo** Elimine el último elemento utilizando el pop() método:

thisset = {"apple", "banana", "cherry"}  
  
x = thisset.pop()  
  
print(x)  
  
print(thisset)

**Nota:** los conjuntos no están ordenados , por lo que al usar el pop()método, no sabe qué elemento se elimina.

### **Ejemplo** El clear() método vacía el conjunto:

thisset = {"apple", "banana", "cherry"}  
  
thisset.clear()  
  
print(thisset)

### **Ejemplo** La delpalabra clave eliminará el conjunto por completo:

thisset = {"apple", "banana", "cherry"}  
  
del thisset  
  
print(thisset)

# Python - Sets de bucles

## **Elementos de bucle**

Puede recorrer los elementos establecidos usando un for bucle:

### **Ejemplo** Recorra el conjunto e imprima los valores:

thisset = {"apple", "banana", "cherry"}  
  
for x in thisset:  
  print(x)

# Python - Unir Sets

Hay varias formas de unir dos o más conjuntos en Python.

Puede usar el union()método que devuelve un nuevo conjunto que contiene todos los elementos de ambos conjuntos, o el update()método que inserta todos los elementos de un conjunto en otro:

### **Ejemplo** El union()método devuelve un nuevo conjunto con todos los elementos de ambos conjuntos:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set3 = set1.union(set2)  
print(set3)

### **Ejemplo** El update()método inserta los elementos en set2 en set1:

set1 = {"a", "b" , "c"}  
set2 = {1, 2, 3}  
  
set1.update(set2)  
print(set1)

**Nota:** Ambos union()y update() excluirán cualquier elemento duplicado.

## **Guarde SOLO los duplicados**

El intersection\_update()método mantendrá solo los elementos que están presentes en ambos conjuntos.

### **Ejemplo** Mantenga los elementos que existen tanto en set xcomo en set y:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.intersection\_update(y)  
  
print(x)

El intersection()método devolverá un nuevo conjunto, que solo contiene los elementos que están presentes en ambos conjuntos.

### **Ejemplo** Devuelve un conjunto que contiene los elementos que existen tanto en conjunto xcomo en conjunto y:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.intersection(y)  
  
print(z)

## **Conservar todo, pero NO los uplicados**

El symmetric\_difference\_update()método mantendrá solo los elementos que NO están presentes en ambos conjuntos.

### **Ejemplo** Quédese con los elementos que no están presentes en ambos conjuntos:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
x.symmetric\_difference\_update(y)  
  
print(x)

El symmetric\_difference()método devolverá un nuevo conjunto, que contiene solo los elementos que NO están presentes en ambos conjuntos.

### **Ejemplo**

Devuelve un conjunto que contiene todos los elementos de ambos conjuntos, excepto los elementos que están presentes en ambos:

x = {"apple", "banana", "cherry"}  
y = {"google", "microsoft", "apple"}  
  
z = x.symmetric\_difference(y)  
  
print(z)

Python - Establecer métodos

**Establecer métodos**

Python tiene un conjunto de métodos integrados que puede usar en conjuntos.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [add()](https://www.w3schools.com/python/ref_set_add.asp) | Añade un elemento al conjunto. |
| [clear()](https://www.w3schools.com/python/ref_set_clear.asp) | Elimina todos los elementos del conjunto. |
| [copy()](https://www.w3schools.com/python/ref_set_copy.asp) | Devuelve una copia del conjunto. |
| [difference()](https://www.w3schools.com/python/ref_set_difference.asp) | Devuelve un conjunto que contiene la diferencia entre dos o más conjuntos |
| [difference\_update()](https://www.w3schools.com/python/ref_set_difference_update.asp) | Elimina los elementos de este conjunto que también están incluidos en otro conjunto especificado |
| [discard()](https://www.w3schools.com/python/ref_set_discard.asp) | Eliminar el elemento especificado |
| [intersection()](https://www.w3schools.com/python/ref_set_intersection.asp) | Devuelve un conjunto, que es la intersección de otros dos conjuntos. |
| [intersection\_update()](https://www.w3schools.com/python/ref_set_intersection_update.asp) | Elimina los elementos de este conjunto que no están presentes en otros conjuntos especificados |
| [isdisjoint()](https://www.w3schools.com/python/ref_set_isdisjoint.asp) | Devuelve si dos conjuntos tienen una intersección o no |
| [issubset()](https://www.w3schools.com/python/ref_set_issubset.asp) | Devuelve si otro conjunto contiene este conjunto o no |
| [issuperset()](https://www.w3schools.com/python/ref_set_issuperset.asp) | Devuelve si este conjunto contiene otro conjunto o no |
| [pop()](https://www.w3schools.com/python/ref_set_pop.asp) | Elimina un elemento del conjunto. |
| [remove()](https://www.w3schools.com/python/ref_set_remove.asp) | Elimina el elemento especificado |
| [symmetric\_difference()](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | Devuelve un conjunto con las diferencias simétricas de dos conjuntos. |
| [symmetric\_difference\_update()](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | inserta las diferencias simétricas de este conjunto y otro |
| [union()](https://www.w3schools.com/python/ref_set_union.asp) | Devuelve un conjunto que contiene la unión de conjuntos |
| [update()](https://www.w3schools.com/python/ref_set_update.asp) | Actualizar el conjunto con la unión de este conjunto y otros |

# Diccionarios de Python

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}

## **Diccionario**

Los diccionarios se utilizan para almacenar valores de datos en pares clave:valor.

Un diccionario es una colección ordenada\*, modificable y que no admite duplicados.

A partir de la versión 3.7 de Python, los diccionarios están ordenados . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Los diccionarios se escriben con corchetes y tienen claves y valores:

### **Ejemplo**

Crear e imprimir un diccionario:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict)

## **Elementos del diccionario**

Los elementos del diccionario están ordenados, se pueden modificar y no permiten duplicados.

Los elementos del diccionario se presentan en pares clave:valor y se puede hacer referencia a ellos mediante el nombre de la clave.

### **Ejemplo** Imprime el valor de "marca" del diccionario:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(thisdict["brand"])

## **¿Ordenado o Desordenado?**

A partir de la versión 3.7 de Python, los diccionarios están ordenados . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Cuando decimos que los diccionarios están ordenados, significa que los elementos tienen un orden definido y ese orden no cambiará.

Desordenado significa que los elementos no tienen un orden definido, no puede hacer referencia a un elemento mediante el uso de un índice.

## **Cambiable**

Los diccionarios se pueden cambiar, lo que significa que podemos cambiar, agregar o eliminar elementos después de que se haya creado el diccionario.

## **No se permiten duplicados**

Los diccionarios no pueden tener dos elementos con la misma clave:

### **Ejemplo** Los valores duplicados sobrescribirán los valores existentes:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964,  
  "year": 2020  
}  
print(thisdict)

## **Longitud del diccionario**

Para determinar cuántos elementos tiene un diccionario, use la len()función:

### **Ejemplo** Imprime el número de elementos en el diccionario:

print(len(thisdict))

**Elementos del diccionario - Tipos de datos**

Los valores en los elementos del diccionario pueden ser de cualquier tipo de datos:

### **Ejemplo** Tipos de datos de cadena, int, booleanos y de lista:

thisdict = {  
  "brand": "Ford",  
  "electric": False,  
  "year": 1964,  
  "colors": ["red", "white", "blue"]  
}

**tipo()** Desde la perspectiva de Python, los diccionarios se definen como objetos con el tipo de datos 'dict':

<class 'dict'>

### **Ejemplo** Imprime el tipo de datos de un diccionario:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
print(type(thisdict))

**Colecciones de Python (matrices)**

Hay cuatro tipos de datos de recopilación en el lenguaje de programación Python:

* [**La lista**](https://www.w3schools.com/python/python_lists.asp) es una colección ordenada y modificable. Permite miembros duplicados.
* [**Tuple**](https://www.w3schools.com/python/python_tuples.asp) es una colección ordenada e inmutable. Permite miembros duplicados.
* [**Conjunto**](https://www.w3schools.com/python/python_sets.asp) es una colección desordenada, inmutable\* y no indexada. No hay miembros duplicados.
* **El diccionario** es una colección ordenada\*\* y modificable. No hay miembros duplicados.

\*Los elementos establecidos no se pueden cambiar, pero puede eliminar y/o agregar elementos cuando lo desee.

\*\*A partir de la versión 3.7 de Python, los diccionarios se ordenan . En Python 3.6 y versiones anteriores, los diccionarios están desordenados .

Al elegir un tipo de colección, es útil comprender las propiedades de ese tipo. Elegir el tipo correcto para un conjunto de datos en particular podría significar la retención del significado y podría significar un aumento en la eficiencia o la seguridad.

# Python - Acceder a elementos del diccionario

## **Acceso a elementos**

Puede acceder a los elementos de un diccionario haciendo referencia a su nombre clave, entre corchetes:

### **Ejemplo** Obtenga el valor de la clave "modelo":

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
x = thisdict["model"]

También hay un método llamado get()que le dará el mismo resultado:

### **Ejemplo** Obtenga el valor de la clave "modelo":

x = thisdict.get("model")

## **Obtener claves**

El keys()método devolverá una lista de todas las claves del diccionario.

### **Ejemplo** Obtenga una lista de las claves:

x = thisdict.keys()

La lista de claves es una vista del diccionario, lo que significa que cualquier cambio realizado en el diccionario se reflejará en la lista de claves.

### **Ejemplo** Agregue un nuevo elemento al diccionario original y vea que la lista de claves también se actualice:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.keys()  
  
print(x) #before the change  
  
car["color"] = "white"  
  
print(x) #after the change

## **Obtener valores**

El values()método devolverá una lista de todos los valores en el diccionario.

### **Ejemplo** Obtenga una lista de los valores:

x = thisdict.values()

La lista de valores es una vista del diccionario, lo que significa que cualquier cambio realizado en el diccionario se reflejará en la lista de valores.

### **Ejemplo** Realice un cambio en el diccionario original y vea que la lista de valores también se actualice:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### **Ejemplo** Agregue un nuevo elemento al diccionario original y vea que la lista de valores también se actualice:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.values()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Obtener artículos**

El items()método devolverá cada elemento en un diccionario, como tuplas en una lista.

### **Ejemplo** Obtener una lista de los pares clave:valor

x = thisdict.items()

La lista devuelta es una vista de los elementos del diccionario, lo que significa que cualquier cambio realizado en el diccionario se reflejará en la lista de elementos.

### **Ejemplo** Realice un cambio en el diccionario original y vea que la lista de elementos también se actualice:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["year"] = 2020  
  
print(x) #after the change

### **Ejemplo** Agregue un nuevo elemento al diccionario original y vea que la lista de elementos también se actualice:

car = {  
"brand": "Ford",  
"model": "Mustang",  
"year": 1964  
}  
  
x = car.items()  
  
print(x) #before the change  
  
car["color"] = "red"  
  
print(x) #after the change

## **Comprobar si existe la clave**

Para determinar si una clave específica está presente en un diccionario, use la inpalabra clave:

### **Ejemplo**Compruebe si "modelo" está presente en el diccionario:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
if "model" in thisdict:  
  print("Yes, 'model' is one of the keys in the thisdict dictionary")

# Python Cambiar valores

Puede cambiar el valor de un elemento específico haciendo referencia a su nombre clave:

### **Ejemplo** Cambia el "año" a 2018:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["year"] = 2018

**Actualizar diccionario**

El update()método actualizará el diccionario con los elementos del argumento dado.

El argumento debe ser un diccionario o un objeto iterable con pares clave:valor.

### **Ejemplo** Actualice el "año" del automóvil utilizando el update() método:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"year": 2020})

# Python - Agregar elementos del diccionario

La adición de un elemento al diccionario se realiza utilizando una nueva clave de índice y asignándole un valor:

### **Ejemplo**

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict["color"] = "red"  
print(thisdict)

## **Actualizar diccionario** El update() método actualizará el diccionario con los elementos de un argumento dado. Si el artículo no existe, se agregará.

El argumento debe ser un diccionario o un objeto iterable con pares clave:valor.

### **Ejemplo** Agregue un elemento de color al diccionario usando el update() método:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.update({"color": "red"})

# Eliminar elementos del diccionario

Existen varios métodos para eliminar elementos de un diccionario:

### **Ejemplo** El pop()método elimina el elemento con el nombre de clave especificado:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.pop("model")  
print(thisdict)

### **Ejemplo** El popitem()método elimina el último elemento insertado (en versiones anteriores a la 3.7, en su lugar, se elimina un elemento aleatorio):

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.popitem()  
print(thisdict)

### **Ejemplo** La delpalabra clave elimina el elemento con el nombre de clave especificado:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict["model"]  
print(thisdict)

### **Ejemplo**La delpalabra clave también puede eliminar el diccionario por completo:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
del thisdict  
print(thisdict) #this will cause an error because "thisdict" no longer exists.

### **Ejemplo** El clear()método vacía el diccionario:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
thisdict.clear()  
print(thisdict)

**Bucle a través de un diccionario**

Puede recorrer un diccionario utilizando un forbucle.

Al recorrer un diccionario, el valor devuelto son las claves del diccionario, pero también hay métodos para devolver los valores .

### **Ejemplo** Imprima todos los nombres clave en el diccionario, uno por uno:

for x in thisdict:  
  print(x)

### **Ejemplo** Imprime todos los valores en el diccionario, uno por uno:

for x in thisdict:  
  print(thisdict[x])

### **Ejemplo** También puede usar el values()método para devolver valores de un diccionario:

for x in thisdict.values():  
  print(x)

### **Ejemplo** Puede usar el keys()método para devolver las claves de un diccionario:

for x in thisdict.keys():  
  print(x)

### **Ejemplo** Recorra las claves y los valores usando el items()método:

for x, y in thisdict.items():  
  print(x, y)

# Copiar diccionarios No puede copiar un diccionario simplemente escribiendo dict2 = dict1, porque: dict2solo será una referencia a dict1, y los cambios realizados en dict1automáticamente también se realizarán en dict2.

Hay formas de hacer una copia, una forma es usar el método Dictionary incorporado copy().

### **Ejemplo** Haz una copia de un diccionario con el copy()método:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = thisdict.copy()  
print(mydict)

Otra forma de hacer una copia es usar la función incorporada dict().

### **Ejemplo** Haz una copia de un diccionario con la dict() función:

thisdict = {  
  "brand": "Ford",  
  "model": "Mustang",  
  "year": 1964  
}  
mydict = dict(thisdict)  
print(mydict)

# Python - Diccionarios anidados

## Un diccionario puede contener diccionarios, esto se llama diccionarios anidados.

### **Ejemplo** Cree un diccionario que contenga tres diccionarios:

myfamily = {  
  "child1" : {  
    "name" : "Emil",  
    "year" : 2004  
  },  
  "child2" : {  
    "name" : "Tobias",  
    "year" : 2007  
  },  
  "child3" : {  
    "name" : "Linus",  
    "year" : 2011  
  }  
}

O, si desea agregar tres diccionarios en un nuevo diccionario:

### **Ejemplo** Cree tres diccionarios, luego cree un diccionario que contenga los otros tres diccionarios:

child1 = {  
  "name" : "Emil",  
  "year" : 2004  
}  
child2 = {  
  "name" : "Tobias",  
  "year" : 2007  
}  
child3 = {  
  "name" : "Linus",  
  "year" : 2011  
}  
  
myfamily = {  
  "child1" : child1,  
  "child2" : child2,  
  "child3" : child3  
}

# Python si... si no

**Condiciones de Python y sentencias If**

Python admite las condiciones lógicas habituales de las matemáticas:

* Es igual a : a == b
* No es igual a: a != b
* Menos que: a < b
* Menor o igual que: a <= b
* Mayor que: a > b
* Mayor o igual que: a >= b

Estas condiciones se pueden usar de varias maneras, más comúnmente en "sentencias if" y bucles.

Una "sentencia if" se escribe utilizando la palabra clave if .

### **Ejemplo**

Si declaración:

a = 33  
b = 200  
if b > a:  
  print("b is greater than a")

En este ejemplo, usamos dos variables, a y b , que se usan como parte de la declaración if para probar si b es mayor que a . Como a es 33 y b es 200 , sabemos que 200 es mayor que 33, por lo que imprimimos en pantalla que "b es mayor que a".

## **Indentation.** Python se basa en la sangría (espacio en blanco al comienzo de una línea) para definir el alcance en el código. Otros lenguajes de programación a menudo usan corchetes para este propósito.

### **Ejemplo**

Si declaración, sin sangría (generará un error):

a = 33  
b = 200  
if b > a:  
print("b is greater than a") # you will get an error

## **Elif** La palabra clave elif es la forma de Python de decir "si las condiciones anteriores no fueron ciertas, intente esta condición".

### **Ejemplo**

a = 33  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
En este ejemplo , a es igual a b , por lo que la primera condición no es verdadera, pero la condición elif es verdadera, por lo que imprimimos en la pantalla que "a y b son iguales".

## **Else** La palabra clave else captura cualquier cosa que no esté capturada por las condiciones anteriores.

### **Ejemplo**

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
elif a == b:  
  print("a and b are equal")  
else:  
  print("a is greater than b")  
En este ejemplo , a es mayor que b , por lo que la primera condición no es verdadera, tampoco la condición elif es verdadera, así que vamos a la condición else e imprimimos en la pantalla que "a es mayor que b".

También puede tener un elsesin el elif:

### **Ejemplo**

a = 200  
b = 33  
if b > a:  
  print("b is greater than a")  
else:  
  print("b is not greater than a")  
**Short Hand If** Si solo tiene una declaración para ejecutar, puede colocarla en la misma línea que la declaración if.

### **Ejemplo** Una línea si declaración:

if a > b: print("a is greater than b")

## **Short Hand If ... Else** Si solo tiene una declaración para ejecutar, una para if y otra para else, puede ponerlas todas en la misma línea:

### **Ejemplo** Una línea si otra declaración:

a = 2  
b = 330  
print("A") if a > b else print("B")

Esta técnica se conoce como **Operadores Ternarios** o **Expresiones Condicionales** .

También puede tener varias declaraciones else en la misma línea:

### **Ejemplo** Declaración if else de una línea, con 3 condiciones:

a = 330  
b = 330  
print("A") if a > b else print("=") if a == b else print("B")

## **And** La palabra clave and es un operador lógico y se usa para combinar sentencias condicionales:

### **Ejemplo** Prueba si aes mayor que b, Y si c es mayor que a:

a = 200  
b = 33  
c = 500  
if a > b and c > a:  
  print("Both conditions are True")  
**Or** La or palabra clave es un operador lógico y se usa para combinar declaraciones condicionales:

### **Ejemplo** Prueba si aes mayor que b, O si a es mayor que c:

a = 200  
b = 33  
c = 500  
if a > b or a > c:  
  print("At least one of the conditions is True")

**Nested If**

Puede tener ifdeclaraciones dentro ifde declaraciones, esto se llama declaraciones anidadas if .

### **Ejemplo** x = 41 if x > 10:   print("Above ten,")   if x > 20:     print("and also above 20!")   else:     print("but not above 20.")

## **The pass Statement**

ifLas declaraciones no pueden estar vacías, pero si por alguna razón tiene una ifdeclaración sin contenido, passintrodúzcala para evitar errores.

### **Ejemplo**

a = 33  
b = 200  
  
if b > a:  
  pass

# Ciclos while de Python

## **Bucles de Python**

Python tiene dos comandos de bucle primitivos:

* while los bucles
* for bucles

## **The while Loop**

Con el bucle while podemos ejecutar un conjunto de declaraciones siempre que una condición sea verdadera.

### **Ejemplo** Imprima i siempre que i sea menor que 6:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
**Nota:** recuerda incrementar i, o de lo contrario el ciclo continuará para siempre.

El ciclo while requiere que las variables relevantes estén listas, en este ejemplo necesitamos definir una variable de indexación, i , que establecemos en 1.

## **The break Statement**

Con la instrucción break podemos detener el bucle incluso si la condición while es verdadera:

### **Ejemplo** Salga del bucle cuando tenga 3 años:

i = 1  
while i < 6:  
  print(i)  
  if i == 3:  
    break  
  i += 1

## **The continue Statement** Con la instrucción continuar podemos detener la iteración actual y continuar con la siguiente:

### **Ejemplo** Continúe con la siguiente iteración si i es 3:

i = 0  
while i < 6:  
  i += 1  
  if i == 3:  
    continue  
  print(i)  
**The else Statement** Con la declaración else podemos ejecutar un bloque de código una vez cuando la condición ya no sea verdadera:

### **Ejemplo** Imprima un mensaje una vez que la condición sea falsa:

i = 1  
while i < 6:  
  print(i)  
  i += 1  
else:  
  print("i is no longer less than 6")

## **Python For Loops** Un bucle for se usa para iterar sobre una secuencia (es decir, una lista, una tupla, un diccionario, un conjunto o una cadena).

Esto se parece menos a la palabra clave for en otros lenguajes de programación y funciona más como un método iterador como se encuentra en otros lenguajes de programación orientados a objetos.

Con el bucle for podemos ejecutar un conjunto de sentencias, una vez por cada elemento de una lista, tupla, conjunto, etc.

### **Ejemplo** Imprime cada fruta en una lista de frutas:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)

El ciclo for no requiere una variable de indexación para establecer de antemano.

## **Bucle a través de una cadena**

Incluso las cadenas son objetos iterables, contienen una secuencia de caracteres:

### **Ejemplo** Repasa las letras de la palabra "banana":

for x in "banana":  
  print(x)

## **La declaración de ruptura**

Con la instrucción break podemos detener el bucle antes de que haya recorrido todos los elementos:

### **Ejemplo** Salga del bucle cuando xsea "banana":

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  print(x)  
  if x == "banana":  
    break

### **Ejemplo** Salga del ciclo cuando xsea "banana", pero esta vez el descanso viene antes de la impresión:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    break  
  print(x)

## **La declaración de continuación**

Con la instrucción continuar podemos detener la iteración actual del ciclo y continuar con la siguiente:

### **Ejemplo** No imprimir plátano:

fruits = ["apple", "banana", "cherry"]  
for x in fruits:  
  if x == "banana":  
    continue  
  print(x)

## **La función rango()**

Para recorrer un conjunto de código un número específico de veces, podemos usar la función range() ,

La función range() devuelve una secuencia de números, comenzando desde 0 de forma predeterminada, se incrementa en 1 (de forma predeterminada) y termina en un número específico.

### **Ejemplo** Usando la función range():

for x in range(6):  
  print(x)

Tenga en cuenta que range(6) no son los valores de 0 a 6, sino los valores de 0 a 5.

La función range() por defecto es 0 como valor inicial, sin embargo, es posible especificar el valor inicial agregando un parámetro: range(2, 6) , lo que significa valores del 2 al 6 (pero sin incluir el 6):

### **Ejemplo** Usando el parámetro de inicio:

for x in range(2, 6):  
  print(x)

La función range() por defecto incrementa la secuencia en 1, sin embargo, es posible especificar el valor del incremento agregando un tercer parámetro: range(2, 30, **3** ) :

### **Ejemplo** Incremente la secuencia con 3 (el valor predeterminado es 1):

for x in range(2, 30, 3):  
  print(x)

## **Más en For Loop**

La else palabra clave en un for ciclo especifica un bloque de código que se ejecutará cuando finalice el ciclo:

### **Ejemplo** Imprime todos los números del 0 al 5 e imprime un mensaje cuando el ciclo haya terminado:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

**Nota:** El elsebloque NO se ejecutará si el bucle se detiene mediante una breakinstrucción.

### **Ejemplo** Rompa el bucle cuando xsea 3 y vea qué sucede con el elsebloque:

for x in range(6):  
  if x == 3: break  
  print(x)  
else:  
  print("Finally finished!")

## **Bucles anidados**

Un bucle anidado es un bucle dentro de un bucle.

El "bucle interno" se ejecutará una vez por cada iteración del "bucle externo":

### **Ejemplo** Escriba cada adjetivo para cada fruta:

adj = ["red", "big", "tasty"]  
fruits = ["apple", "banana", "cherry"]  
  
for x in adj:  
  for y in fruits:  
    print(x, y)

## **La declaración de pase** forlos bucles no pueden estar vacíos, pero si por alguna razón tiene un forbucle sin contenido, ingrese la passdeclaración para evitar recibir un error.

### **Ejemplo**

### for x in [0, 1, 2]:   pass

# Python Functions

Una función es un bloque de código que solo se ejecuta cuando se le llama.

Puede pasar datos, conocidos como parámetros, a una función.

Una función puede devolver datos como resultado.

## **Crear una función**

En Python, una función se define usando la palabra clave def :

### **Ejemplo**

def my\_function():  
  print("Hello from a function")

## **Llamar a una función**

Para llamar a una función, use el nombre de la función seguido de paréntesis:

### **Ejemplo**

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## **Argumentos**

La información se puede pasar a funciones como argumentos.

Los argumentos se especifican después del nombre de la función, entre paréntesis. Puede agregar tantos argumentos como desee, simplemente sepárelos con una coma.

El siguiente ejemplo tiene una función con un argumento (fname). Cuando se llama a la función, pasamos un nombre, que se usa dentro de la función para imprimir el nombre completo:

### **Ejemplo**

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

Los argumentos a menudo se reducen a args en la documentación de Python.

## **¿Parámetros o argumentos?**

Los términos parámetro y argumento se pueden usar para lo mismo: información que se pasa a una función.

Desde la perspectiva de una función:

Un parámetro es la variable que aparece entre paréntesis en la definición de la función.

Un argumento es el valor que se envía a la función cuando se llama.

## **Número de argumentos**

Por defecto, una función debe llamarse con el número correcto de argumentos. Lo que significa que si su función espera 2 argumentos, debe llamar a la función con 2 argumentos, ni más ni menos.

### **Ejemplo** Esta función espera 2 argumentos y obtiene 2 argumentos:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil", "Refsnes")

Si intenta llamar a la función con 1 o 3 argumentos, obtendrá un error:

### **Ejemplo** Esta función espera 2 argumentos, pero solo obtiene 1:

def my\_function(fname, lname):  
  print(fname + " " + lname)  
  
my\_function("Emil")

## **Argumentos arbitrarios, \*args**

Si no sabe cuántos argumentos se pasarán a su función, agregue un \*antes del nombre del parámetro en la definición de la función.

De esta manera, la función recibirá una tupla de argumentos y podrá acceder a los elementos en consecuencia:

### **Ejemplo** Si se desconoce el número de argumentos, agregue un \*antes del nombre del parámetro:

def my\_function(\*kids):  
  print("The youngest child is " + kids[2])  
  
my\_function("Emil", "Tobias", "Linus")

Los argumentos arbitrarios a menudo se abrevian a \*args en la documentación de Python.

## **Argumentos de palabras clave**

También puede enviar argumentos con la sintaxis clave = valor .

De esta manera el orden de los argumentos no importa.

### **Ejemplo**

def my\_function(child3, child2, child1):  
  print("The youngest child is " + child3)  
  
my\_function(child1 = "Emil", child2 = "Tobias", child3 = "Linus")

La frase Argumentos de palabras clave a menudo se abrevia como kwargs en la documentación de Python.

## **Argumentos de palabras clave arbitrarias, \*\*kwargs**

Si no sabe cuántos argumentos de palabras clave se pasarán a su función, agregue dos asteriscos: \*\*antes del nombre del parámetro en la definición de la función.

De esta forma, la función recibirá un diccionario de argumentos y podrá acceder a los elementos en consecuencia:

### **Ejemplo** Si se desconoce el número de argumentos de palabras clave, agregue un doble \*\*antes del nombre del parámetro:

def my\_function(\*\*kid):  
  print("His last name is " + kid["lname"])  
  
my\_function(fname = "Tobias", lname = "Refsnes")

Los argumentos Kword arbitrarios a menudo se reducen a \*\*kwargs en la documentación de Python.

## **Valor de parámetro predeterminado**

El siguiente ejemplo muestra cómo utilizar un valor de parámetro predeterminado.

Si llamamos a la función sin argumento, usa el valor predeterminado:

### **Ejemplo**

def my\_function(**country = "Norway"**):  
  print("I am from " + country)  
  
my\_function("Sweden")  
my\_function("India")  
my\_function()  
my\_function("Brazil")

## **Pasar una lista como argumento**

Puede enviar cualquier tipo de datos de argumento a una función (cadena, número, lista, diccionario, etc.), y se tratará como el mismo tipo de datos dentro de la función.

Por ejemplo, si envía una Lista como argumento, seguirá siendo una Lista cuando llegue a la función:

### **Ejemplo**

def my\_function(food):  
  for x in food:  
    print(x)  
  
fruits = ["apple", "banana", "cherry"]  
  
my\_function(fruits)

**Valores devueltos**

Para permitir que una función devuelva un valor, use la return declaración:

### **Ejemplo**

def my\_function(x):  
  **return 5 \* x**  
print(my\_function(3))  
print(my\_function(5))  
print(my\_function(9))

## **La declaración de pase**

functionlas definiciones no pueden estar vacías, pero si por algún motivo tiene una functiondefinición sin contenido, introdúzcala en la passdeclaración para evitar que se produzca un error.

### **Ejemplo**

def myfunction():  
  pass

## **recursividad**

Python también acepta la función recursiva, lo que significa que una función definida puede llamarse a sí misma.

La recursividad es un concepto matemático y de programación común. Significa que una función se llama a sí misma. Esto tiene la ventaja de que puede recorrer los datos para llegar a un resultado.

El desarrollador debe tener mucho cuidado con la recursividad, ya que puede ser bastante fácil escribir una función que nunca termina, o una que usa cantidades excesivas de memoria o potencia del procesador. Sin embargo, cuando se escribe correctamente, la recursión puede ser un enfoque de programación muy eficiente y matemáticamente elegante.

En este ejemplo, tri\_recursion() es una función que hemos definido para llamarse a sí misma ("recursión"). Usamos la variable k como los datos, que decrementa ( -1 ) cada vez que recursamos. La recursión termina cuando la condición no es mayor que 0 (es decir, cuando es 0).

Para un nuevo desarrollador, puede tomar algún tiempo averiguar cómo funciona exactamente esto, la mejor manera de averiguarlo es probándolo y modificándolo.

### **Ejemplo** Ejemplo de recursividad

def tri\_recursion(k):  
  if(k > 0):  
    result = k + tri\_recursion(k - 1)  
    print(result)  
  else:  
    result = 0  
  return result  
  
print("\n\nRecursion Example Results")  
tri\_recursion(6)

# Python Lambda

Una función lambda es una pequeña función anónima.

Una función lambda puede tomar cualquier cantidad de argumentos, pero solo puede tener una expresión.

## **Sintaxis** lambda arguments : expression

Se ejecuta la expresión y se devuelve el resultado:

### **Ejemplo** Agregue 10 al argumento ay devuelva el resultado:

x = lambda a : a + 10  
print(x(5))

Las funciones Lambda pueden tomar cualquier número de argumentos:

### **Ejemplo** Multiplique argumento a con argumento b y devuelva el resultado:

x = lambda a, b : a \* b  
print(x(5, 6))

### **Ejemplo** Resumir argumento a, b y c y devolver el resultado:

x = lambda a, b, c : a + b + c  
print(x(5, 6, 2))

## **¿Por qué utilizar las funciones de Lambda?**

El poder de lambda se muestra mejor cuando los usa como una función anónima dentro de otra función.

Digamos que tiene una definición de función que toma un argumento, y ese argumento se multiplicará con un número desconocido:

def myfunc(n):  
  return lambda a : a \* n

Use esa definición de función para hacer una función que siempre duplique el número que envía:

### **Ejemplo**

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
  
print(mydoubler(11))

O use la misma definición de función para hacer una función que siempre triplique el número que envía:

### **Ejemplo**

def myfunc(n):  
  return lambda a : a \* n  
  
mytripler = myfunc(3)  
  
print(mytripler(11))

O use la misma definición de función para hacer ambas funciones, en el mismo programa:

### **Ejemplo**

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
mytripler = myfunc(3)  
  
print(mydoubler(11))  
print(mytripler(11))

# Matrices de Python

**Nota:** Python no tiene soporte integrado para arreglos, pero en su lugar se pueden usar [listas de Python .](https://www.w3schools.com/python/python_lists.asp)

## **arreglos**

**Nota:** esta página le muestra cómo usar LISTS como ARRAYS, sin embargo, para trabajar con matrices en Python, deberá importar una biblioteca, como la [biblioteca NumPy](https://www.w3schools.com/python/numpy/default.asp) .

Las matrices se utilizan para almacenar múltiples valores en una sola variable:

### **Ejemplo** Cree una matriz que contenga nombres de automóviles:

cars = ["Ford", "Volvo", "BMW"]

## **¿Qué es una matriz?**

Una matriz es una variable especial que puede contener más de un valor a la vez.

Si tiene una lista de elementos (una lista de nombres de automóviles, por ejemplo), almacenar los automóviles en variables individuales podría tener este aspecto:

car1 = "Ford"  
car2 = "Volvo"  
car3 = "BMW"

Sin embargo, ¿qué sucede si desea recorrer los autos y encontrar uno específico? ¿Y si no tuvieras 3 coches, sino 300?

¡La solución es una matriz!

Una matriz puede contener muchos valores bajo un solo nombre y puede acceder a los valores haciendo referencia a un número de índice.

## **Acceder a los Elementos de un Array**

Se hace referencia a un elemento de matriz haciendo referencia al número de índice .

### **Ejemplo** Obtenga el valor del primer elemento de la matriz:

x = cars[0]

### **Ejemplo** Modifique el valor del primer elemento de la matriz:

cars[0] = "Toyota"

## **La longitud de una matriz**

Utilice el len() método para devolver la longitud de una matriz (la cantidad de elementos en una matriz).

### **Ejemplo** Devuelve el número de elementos en la cars matriz:

x = len(cars)

**Nota:** La longitud de una matriz siempre es uno más que el índice de matriz más alto.

## **Elementos de matriz en bucle**

Puede usar el for in ciclo para recorrer todos los elementos de una matriz.

### **Ejemplo** Imprima cada elemento en la cars matriz:

for x in cars:  
  print(x)

## **Adición de elementos de matriz**

Puede usar el append() método para agregar un elemento a una matriz.

### **Ejemplo** Agregue un elemento más a la cars matriz:

cars.append("Honda")

## **Eliminación de elementos de matriz**

Puede usar el pop() método para eliminar un elemento de la matriz.

### **Ejemplo** Eliminar el segundo elemento de la cars matriz:

cars.pop(1)

También puede usar el remove() método para eliminar un elemento de la matriz.

### **Ejemplo** Eliminar el elemento que tiene el valor "Volvo":

cars.remove("Volvo")

**Nota:** El remove() método de la lista solo elimina la primera aparición del valor especificado.

## **Métodos de matriz**

Python tiene un conjunto de métodos integrados que puede usar en listas/matrices.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Añade un elemento al final de la lista. |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Elimina todos los elementos de la lista. |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Devuelve una copia de la lista. |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Devuelve el número de elementos con el valor especificado |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Agregue los elementos de una lista (o cualquier iterable), al final de la lista actual |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Devuelve el índice del primer elemento con el valor especificado |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Agrega un elemento en la posición especificada |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Elimina el elemento en la posición especificada |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Elimina el primer elemento con el valor especificado |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Invierte el orden de la lista |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Ordena la lista |

**Nota:** Python no tiene soporte integrado para arreglos, pero en su lugar se pueden usar listas de Python.

## **Clases/Objetos de Python**

Python es un lenguaje de programación orientado a objetos.

Casi todo en Python es un objeto, con sus propiedades y métodos.

Una clase es como un constructor de objetos o un "modelo" para crear objetos.

## **Crear una clase**

Para crear una clase, utilice la palabra clave class:

### **Ejemplo** Cree una clase llamada MyClass, con una propiedad llamada x:

class MyClass:  
  x = 5

## **Crear objeto**

Ahora podemos usar la clase llamada MyClass para crear objetos:

### **Ejemplo** Cree un objeto llamado p1 e imprima el valor de x:

p1 = MyClass()  
print(p1.x)

## **La función \_\_init\_\_()**

Los ejemplos anteriores son clases y objetos en su forma más simple y no son realmente útiles en aplicaciones de la vida real.

Para comprender el significado de las clases, debemos comprender la función \_\_init\_\_() incorporada.

Todas las clases tienen una función llamada \_\_init\_\_(), que siempre se ejecuta cuando se inicia la clase.

Use la función \_\_init\_\_() para asignar valores a las propiedades del objeto u otras operaciones que sean necesarias cuando se crea el objeto:

### **Ejemplo** Cree una clase llamada Persona, use la función \_\_init\_\_() para asignar valores para el nombre y la edad:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1.name)  
print(p1.age)

**Nota:** La \_\_init\_\_()función se llama automáticamente cada vez que la clase se usa para crear un nuevo objeto.

## **Métodos de objetos**

Los objetos también pueden contener métodos. Los métodos en los objetos son funciones que pertenecen al objeto.

Vamos a crear un método en la clase Person:

### **Ejemplo** Inserte una función que imprima un saludo y ejecútela en el objeto p1:

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def myfunc(self):  
    print("Hello my name is " + self.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

**Nota:** El self parámetro es una referencia a la instancia actual de la clase y se utiliza para acceder a las variables que pertenecen a la clase.

## **El auto parámetro**

El selfparámetro es una referencia a la instancia actual de la clase y se utiliza para acceder a las variables que pertenecen a la clase.

No tiene que ser nombrado self, puedes llamarlo como quieras, pero tiene que ser el primer parámetro de cualquier función en la clase:

### **Ejemplo** Usa las palabras mysillyobject y abc en lugar de self :

class Person:  
  def \_\_init\_\_(mysillyobject, name, age):  
    mysillyobject.name = name  
    mysillyobject.age = age  
  
  def myfunc(abc):  
    print("Hello my name is " + abc.name)  
  
p1 = Person("John", 36)  
p1.myfunc()

## **Modificar propiedades de objetos**

Puede modificar propiedades en objetos como este:

### **Ejemplo** Establezca la edad de p1 en 40:

p1.age = 40

## **Eliminar propiedades de objeto**

Puede eliminar propiedades en objetos usando la delpalabra clave:

### **Ejemplo** Elimine la propiedad de edad del objeto p1:

del p1.age

## **Eliminar objetos**

Puede eliminar objetos utilizando la delpalabra clave:

### **Ejemplo** Eliminar el objeto p1:

del p1

## **La declaración de pase**

Class las definiciones no pueden estar vacías, pero si por algún motivo tiene una class definición sin contenido, introdúzcala en la pass declaración para evitar que se produzca un error.

### **Ejemplo**

class Person:  
  pass

# Herencia de Python

La herencia nos permite definir una clase que hereda todos los métodos y propiedades de otra clase.

**La clase principal** es la clase de la que se hereda, también llamada clase base.

**La clase secundaria** es la clase que hereda de otra clase, también llamada clase derivada.

## **Crear una clase principal**

Cualquier clase puede ser una clase principal, por lo que la sintaxis es la misma que para crear cualquier otra clase:

### **Ejemplo** Cree una clase llamada Person, con propiedades y un firstname método :lastname printname

class Person:  
  def \_\_init\_\_(self, fname, lname):  
    self.firstname = fname  
    self.lastname = lname  
  
  def printname(self):  
    print(self.firstname, self.lastname)  
  
#Use the Person class to create an object, and then execute the printname method:  
  
x = Person("John", "Doe")  
x.printname()

## **Crear una clase secundaria**

Para crear una clase que herede la funcionalidad de otra clase, envíe la clase principal como parámetro al crear la clase secundaria:

### **Ejemplo** Cree una clase llamada Student, que heredará las propiedades y métodos de la Personclase:

class Student(Person):  
  pass

**Nota:** utilice la pass palabra clave cuando no desee agregar otras propiedades o métodos a la clase.

Ahora la clase Estudiante tiene las mismas propiedades y métodos que la clase Persona.

### **Ejemplo** Use la Studentclase para crear un objeto y luego ejecute el printnamemétodo:

x = Student("Mike", "Olsen")  
x.printname()

## **Agregue la función \_\_init\_\_()**

Hasta ahora hemos creado una clase secundaria que hereda las propiedades y métodos de su padre.

Queremos agregar la \_\_init\_\_()función a la clase secundaria (en lugar de la passpalabra clave).

**Nota:** La \_\_init\_\_()función se llama automáticamente cada vez que la clase se usa para crear un nuevo objeto.

### **Ejemplo** Agregue la \_\_init\_\_()función a la Studentclase:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    #add properties etc.

Cuando agrega la \_\_init\_\_()función, la clase secundaria ya no heredará la \_\_init\_\_()función principal.

**Nota:** La \_\_init\_\_() función del hijo **anula** la herencia de la \_\_init\_\_()función del padre.

Para mantener la herencia de la \_\_init\_\_() función principal, agregue una llamada a la \_\_init\_\_()función principal:

### **Ejemplo**

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    Person.\_\_init\_\_(self, fname, lname)

Ahora hemos agregado con éxito la función \_\_init\_\_() y conservamos la herencia de la clase principal, y estamos listos para agregar funcionalidad en la \_\_init\_\_()función.

## **Usa la función super()**

Python también tiene una super()función que hará que la clase secundaria herede todos los métodos y propiedades de su padre:

### **Ejemplo**

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)

Al usar la super()función, no tiene que usar el nombre del elemento principal, automáticamente heredará los métodos y propiedades de su elemento principal.

## **Agregar Propiedades**

### **Ejemplo** Agregue una propiedad llamada graduationyeara la Studentclase:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = 2019

En el ejemplo a continuación, el año 2019debe ser una variable y pasarse a la Studentclase al crear objetos de estudiante. Para hacerlo, agregue otro parámetro en la función \_\_init\_\_():

### **Ejemplo** Agregue un yearparámetro y pase el año correcto al crear objetos:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname, year):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = year  
  
x = Student("Mike", "Olsen", 2019)

## **Añadir métodos**

### **Ejemplo** Agregue un método llamado welcomea la Studentclase:

class Student(Person):  
  def \_\_init\_\_(self, fname, lname, year):  
    super().\_\_init\_\_(fname, lname)  
    self.graduationyear = year  
  
  def welcome(self):  
    print("Welcome", self.firstname, self.lastname, "to the class of", self.graduationyear)

Si agrega un método en la clase secundaria con el mismo nombre que una función en la clase principal, se anulará la herencia del método principal.