# Sintaxis de Python

## 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 en 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, en Python es muy importante.

Python utiliza sangría para indicar un bloque de código.

Ejemplo

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

Python te dará un error si omites 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!")

Debes utilizar 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 se les asigna 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 capacidad para comentar con el propósito de documentación dentro del 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!")

Los comentarios se pueden utilizar para explicar el código Python.

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

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

## Creando 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 por qué ser un texto que explique el código, también se puede utilizar para evitar que Python ejecute el código:

Ejemplo

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

## Comentarios multilineales

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

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

Ejemplo

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

O, aunque no exactamente como estaba previsto, puedes utilizar una cadena de varias líneas.

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

Ejemplo

"""  
This is a comment  
written in

“””

# Variables de Python

## Variables​​​​

Las variables son contenedores para almacenar valores de datos.

## Creando variables

Python no tiene ningún comando para declarar una variable.

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

Ejemplo

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

Las variables no necesitan declararse con ningún *tipo* particular, e incluso pueden cambiar de tipo después de haber sido configuradas.

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, esto se puede hacer mediante 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 utilizando 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\_vehículo, volumen\_total). Reglas para las variables de Python:

* Un nombre de variable debe comenzar con una letra o el carácter de guión bajo.
* 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)
* Un nombre de variable no puede ser ninguna de las [palabras clave de Python](https://www.w3schools.com/python/python_ref_keywords.asp) .

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.

## Nombres de variables de varias palabras

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

Hay varias técnicas que puedes utilizar para hacerlos más legibles:

Estuche de camello

Cada palabra, excepto la primera, comienza con mayúscula:

myVariableName = "John"

## Caso Pascal

Cada palabra comienza con una letra mayúscula:

MyVariableName = "John"

## Caso de la serpiente

Cada palabra está separada por un guión bajo:

my\_variable\_name = "John"

## Variables de Python: asignación de múltiples valores

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 puedes 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 tienes una colección de valores en una lista, tupla, etc., Python te permite extraer los valores en variables. Esto se llama *desempaquetar* .

Ejemplo

Descomprimir una lista:

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

Obtenga más información sobre cómo desempaquetar tuplas en nuestro capítulo [Desempaquetar tuplas](https://www.w3schools.com/python/python_tuples_unpack.asp)

 .

# Python - Variables de salida

## Variables de salida

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

Ejemplo

x = "Python is awesome"  
En la print()función, se generan múltiples variables, separadas por una coma:

Ejemplo

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

También puedes utilizar el +operador para generar múltiples variables:

Ejemplo

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

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

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

Ejemplo

x = 5  
y = 10  
print(x + y)

En la print()función, cuando intentas combinar una cadena y un número con el + operador, Python te 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 - Variables globales

## Variables globales

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

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

Ejemplo

Crea una variable fuera de una función y úsala dentro de la función

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

Si creas una variable con el mismo nombre dentro de una función, esta variable será local y solo podrá usarse dentro de la función. La variable global con el mismo nombre permanecerá como estaba, global y con el valor original.

Ejemplo

Crea 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 global

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

Para crear una variable global dentro de una función, puede utilizar la palabra clave “global”.

Ejemplo

Si utiliza la palabra clave global, la variable pertenece al ámbito global:

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

Además, utilice la palabra clave global si desea cambiar una variable global dentro de una función.

Ejemplo

Para cambiar el valor de una variable global dentro de una función, haga referencia a la variable utilizando la palabra clave global clave:

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

# Tipos de datos de Python

## 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 |
| Tipos de conjuntos: | set,frozenset |
| Tipo booleano: | bool |
| Tipos binarios: | bytes, bytearray, memoryview |
| Ninguno Tipo: | NoneType |

Obteniendo el tipo de datos

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

Ejemplo

Imprima 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 se asigna un valor a una variable:

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** |  |
| x = "Hello World" | str |  |
| x = 20 | int |  |
| x = 20.5 | float |  |
| x = 1j | complex |  |
| x = ["apple", "banana", "cherry"] | list |  |
| x = ("apple", "banana", "cherry") | tuple |  |
| x = range(6) | range |  |
| x = {"name" : "John", "age" : 36} | dict |  |
| x = {"apple", "banana", "cherry"} | set |  |
| x = frozenset({"apple", "banana", "cherry"}) | frozenset |  |
| x = True | bool |  |
| x = b"Hello" | bytes |  |
| x = bytearray(5) | bytearray |  |
| x = memoryview(bytes(5)) | memoryview |  |
| x = None | NoneType |  |

## Configuración del tipo de datos específico

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

|  |  |  |
| --- | --- | --- |
| **Example** | **Data Type** |  |
| x = str("Hello World") | str |  |
| x = int(20) | int |  |
| x = float(20.5) | float |  |
| x = complex(1j) | complex |  |
| x = list(("apple", "banana", "cherry")) | list |  |
| x = tuple(("apple", "banana", "cherry")) | tuple |  |
| x = range(6) | range |  |
| x = dict(name="John", age=36) | dict |  |
| x = set(("apple", "banana", "cherry")) | set |  |
| x = frozenset(("apple", "banana", "cherry")) | frozenset |  |
| x = bool(5) | bool |  |
| x = bytes(5) | bytes |  |
| x = bytearray(5) | bytearray |  |
| x = memoryview(bytes(5)) | memoryview |  |

# Números de Python

## Números de Python

Hay tres tipos numéricos en Python:

* int
* float
* complex

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

Ejemplo

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

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

Ejemplo

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

### Int

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

Ejemplo

Números enteros:

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

### Float

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

Ejemplo

Floats:

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

Los flotantes también pueden ser números científicos con una "e" para indicar la potencia de 10.

Ejemplo

Floats:

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

### Complejo

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))

## Conversión de tipos

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 se pueden convertir números complejos en otro tipo de número.

## Número aleatorio

Python no tiene una random()función para crear un número aleatorio, pero Python tiene un módulo incorporado llamado random que se puede usar para crear números aleatorios:

Ejemplo

Importa el módulo aleatorio y muestra un número aleatorio entre 1 y 9:

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

## Conversión de Python

### Especificar un tipo de variable

Puede haber ocasiones en las que desees especificar un tipo en una variable. Esto se puede hacer mediante conversión. 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 constructoras:

* 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

Números 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'

# Strings de Python

## Instrumentos de String

Las Strings en Python están rodeadas por 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')

### Citas dentro de citas

Puedes utilizar comillas dentro de una cadena, siempre que no coincidan con las comillas que rodean la cadena:

Ejemplo

print("It's alright")  
print("He is called 'Johnny'")  
print('He is called "Johnny"')

### Asignar una String a una 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)

### Strings multilineales

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

Ejemplo

Puedes 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.

### Las strings son matrices

Al igual que 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 carácter, un solo carácter es simplemente una cadena con una longitud de 1.

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

Ejemplo

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

a = "Hello, World!"  
print(a[1])

## Hacer un bucle a través de una String

Dado que las cadenas son matrices, podemos recorrer los caracteres de una cadena mediante un bucle for.

Ejemplo

Recorre las letras de la palabra "banana":

for x in "banana":  
  print(x)

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

## Longitud de la String

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

Ejemplo

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

a = "Hello, World!"  
print(len(a))

## String de verificación

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

Ejemplo

Comprueba si "free" está presente en el siguiente texto:

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

Úselo en una declaración if :

Ejemplo

Imprimir sólo si está presente "gratis":

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

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

### Marcar si NO

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

Ejemplo

Comprueba 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 declaración if :

Ejemplo

imprimir 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 - Segmentación de Strings

## Rebanar

Puedes devolver un rango de caracteres utilizando la sintaxis de segmentación.

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

Ejemplo

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

b = "Hello, World!"  
print(b[2:5])

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

## Cortar desde el principio

Si omitimos el índice de inicio, el rango comenzará en el primer carácter:

Ejemplo

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

b = "Hello, World!"  
print(b[:5])

## Cortar hasta el final

Si omitimos 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:])

## Indexación negativa

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

Ejemplo

Consigue los personajes:

De: "o" en "¡Mundo!" (posición -5)

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

b = "Hello, World!"  
print(b[-5:-2])

# Python - Modificar Strings

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

## Mayúsculas

Ejemplo

El upper()método devuelve la cadena en mayúsculas:

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

## Minúscula

Ejemplo

El lower()método devuelve la cadena en minúsculas:

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

## Eliminar espacios en blanco

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

Ejemplo

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

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

## Reemplazar Strings

Ejemplo

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

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

## Strings dividida

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!']

Obtenga más información sobre listas en nuestro capítulo [Listas de Python](https://www.w3schools.com/python/python_lists.asp) .

## Métodos de cadena

Obtenga más información sobre los métodos de cadena con nuestra [Referencia de métodos de cadena](https://www.w3schools.com/python/python_ref_string.asp)

# Python - Concatenación de cadenas

## Concatenación de cadenas

Para concatenar o combinar dos cadenas puedes utilizar el operador +.

Ejemplo

Fusionar variable a con 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 - Formato - Cadenas

# Formato de cadena

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

Ejemplo

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

¡Pero podemos combinar cadenas y números utilizando f-strings o el format()método !

## Cuerdas F

F-String se introdujo en Python 3.6 y ahora es la forma preferida de formatear cadenas.

Para especificar una cadena como una f-string, simplemente coloque un f delante del literal de cadena y agregue llaves {}como marcadores de posición para variables y otras operaciones.

Ejemplo

Crear una cadena f:

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

## Marcadores de posición y modificadores

Un marcador de posición puede contener variables, operaciones, funciones y modificadores para formatear el valor.

Ejemplo

Añade un marcador de posición para la pricevariable:

price = 59  
txt = f"The price is {price} dollars"  
print(txt)

Un marcador de posición puede incluir un *modificador* para formatear el valor.

Se incluye un modificador agregando dos puntos :seguidos de un tipo de formato legal, como .2f que significa un número de punto fijo con 2 decimales:

Ejemplo

Mostrar el precio con 2 decimales:

price = 59  
txt = f"The price is {price:.2f} dollars"  
print(txt)

Un marcador de posición puede contener código Python, como operaciones matemáticas:

Ejemplo

Realizar una operación matemática en el marcador de posición y devolver el resultado:

txt = f"The price is {20 \* 59} dollars"  
print(txt)

# Python - Caracteres de escape

## Caracteres de escape

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 rodeada por comillas dobles:

Ejemplo

Recibirá un error si utiliza comillas dobles dentro de una cadena que está rodeada por 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 utilizar comillas dobles cuando normalmente no estaría permitido:

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

## Caracteres de escape

Otros caracteres de escape utilizados en Python:

|  |  |  |
| --- | --- | --- |
| **Code** | **Result** |  |
| \' | Single Quote |  |
| \\ | Backslash |  |
| \n | New Line |  |
| \r | Carriage Return |  |
| \t | Tab |  |
| \b | Backspace |  |
| \f | Form Feed |  |
| \ooo | Octal value |  |
| \xhh | Hex value |  |

# Python - Métodos de Strings

## Métodos de Strings

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

**Nota:** todos los métodos de cadena devuelven nuevos valores. No modifican 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 |
| [isascii()](https://www.w3schools.com/python/ref_string_isascii.asp) | Returns True if all characters in the string are ascii characters |
| [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 |

# Valores booleanos de Python

Los valores booleanos representan uno de dos valores: True o False.

## Valores booleanos

En programación, a menudo es necesario saber si una expresión es True o False.

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

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

Ejemplo

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

Cuando se ejecuta una condición en una declaración if, Python devuelve True o False:

Ejemplo

Imprima un mensaje en función de si la condición es True o 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 True o False a cambio,

Ejemplo

Evaluar una cadena y un número:

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

Ejemplo

Evaluar dos variables:

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

## La mayoría de los valores son verdaderos

Casi cualquier valor se evalúa True si 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 Verdadero:

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

## Algunos valores son falsos

De hecho, no hay muchos valores que 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 tienes un objeto que está hecho 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

Imprima la respuesta de una función:

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

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

Ejemplo

Imprima "¡SÍ!" si la función devuelve Verdadero, de lo contrario imprima "¡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 utilizar para determinar si un objeto es de un determinado tipo de datos:

Ejemplo

Comprueba si un objeto es un entero o no:

x = 200  
print(isinstance(x, int))

# Operadores de Python

## Operadores de Python

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

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

Ejemplo

print(10 + 5)

Python divide los operadores en los siguientes grupos:

* Operadores aritméticos
* Operadores de asignación
* Operadores de comparación
* Operadores lógicos
* Operadores de identidad
* Operadores de membresía
* Operadores bit a bit

## Operadores aritméticos de Python

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) |

## Operadores de asignación de Python

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) |
| := | print(x := 3) | x = 3 print(x) | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_walrus) |

## Operadores de comparación de Python

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) |

## Operadores lógicos de Python

Los operadores lógicos se utilizan para combinar declaraciones 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) |

## Operadores de identidad de Python

Los operadores de identidad se utilizan para comparar los objetos, no si son iguales, sino si son realmente 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) |

## Operadores de membresía de Python

Los operadores de pertenencia se utilizan para probar si una secuencia está presente 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) |

## Operadores bit a bit de Python

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Operator** | **Name** | **Description** | **Example** | **Try it** |
| & | AND | Sets each bit to 1 if both bits are 1 | x & y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_and) |
| | | OR | Sets each bit to 1 if one of two bits is 1 | x | y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_or) |
| ^ | XOR | Sets each bit to 1 if only one of two bits is 1 | x ^ y | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_xor) |
| ~ | NOT | Inverts all the bits | ~x | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_not) |
| << | Zero fill left shift | Shift left by pushing zeros in from the right and let the leftmost bits fall off | x << 2 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_left_shift) |
| >> | Signed right shift | Shift right by pushing copies of the leftmost bit in from the left, and let the rightmost bits fall off | x >> 2 | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_oper_right_shift) |

## Precedencia del operador

La precedencia del operador describe el orden en que se realizan las operaciones.

Ejemplo

Los paréntesis tienen la mayor precedencia, lo que significa que las expresiones dentro de paréntesis deben evaluarse primero:

print((6 + 3) - (6 + 3))

Ejemplo

La multiplicación \*tiene mayor precedencia que la suma +y, por lo tanto, las multiplicaciones se evalúan antes que las sumas:

print(100 + 5 \* 3)

El orden de precedencia se describe en la siguiente tabla, comenzando con la precedencia más alta en la parte superior:

|  |  |  |
| --- | --- | --- |
| **Operator** | **Description** | **Try it** |
| () | Parentheses | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_parentheses) |
| \*\* | Exponentiation | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_exponent) |
| +x  -x  ~x | Unary plus, unary minus, and bitwise NOT | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_not) |
| \*  /  //  % | Multiplication, division, floor division, and modulus | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_multiplication) |
| +  - | Addition and subtraction | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_subtraction) |
| <<  >> | Bitwise left and right shifts | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_shift) |
| & | Bitwise AND | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_and) |
| ^ | Bitwise XOR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_xor) |
| | | Bitwise OR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_bitwise_or) |
| ==  !=  >  >=  <  <=  is  is not  in  not in | Comparisons, identity, and membership operators | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_like) |
| not | Logical NOT | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_not) |
| and | AND | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_and) |
| or | OR | [Try it »](https://www.w3schools.com/python/trypython.asp?filename=demo_precedence_or) |

Si dos operadores tienen la misma precedencia, la expresión se evalúa de izquierda a derecha.

Ejemplo

La suma +y la resta -tienen la misma precedencia, por lo tanto, evaluamos la expresión de izquierda a derecha:

print(5 + 4 - 7 + 3)

# Listas de Python

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

## Lista

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

Las listas son uno de los cuatro tipos de datos integrados en Python que se utilizan para almacenar colecciones de datos; los otros tres 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 cualidades y usos.

Las listas se crean utilizando corchetes:

Ejemplo

Crear una lista:

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

## Elementos de la lista

Los elementos de la lista están ordenados, son modificables y permiten valores duplicados.

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

### Ordenado

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.

### Cambiable

La lista es modificable, lo que significa que podemos cambiar, agregar y eliminar elementos en una lista después de haberla creado.

### Permitir duplicados

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)

### Longitud de la lista

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

Ejemplo

Imprima el número de elementos en la lista:

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

## Elementos de lista: tipos de datos

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, números enteros y valores booleanos:

list1 = ["abc", 34, True, 40, "male"]

### tipo()

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))

## El constructor list()

También es posible utilizar 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)

## Colecciones de Python (matrices)

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

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

\* *Los elementos* del conjunto no se pueden modificar, pero puedes eliminarlos y/o agregarlos cuando lo desees.

\*\*A partir de la versión 3.7 de Python, los diccionarios están *ordenados* . 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 un aumento en la eficiencia o la seguridad.

# Python - Acceso a elementos de la lista

## Acceder a elementos

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

Ejemplo

Imprima el segundo elemento de la lista:

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

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

Indexación negativa

La indexación negativa significa empezar desde el final.

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

Ejemplo

Imprima el último elemento de la lista:

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

## Rango de índices

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

Al especificar un rango, el valor de retorno 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 finalizará en el índice 5 (no incluido).

Recuerde que el primer elemento tiene índice 0.

Si omitimos el valor inicial, el rango comenzará en el primer elemento:

Ejemplo

Este ejemplo devuelve los elementos desde el principio hasta "kiwi", pero SIN incluirlo:

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

Al omitir el valor final, el rango continuará hasta el final de la lista:

Ejemplo

Este ejemplo devuelve los elementos desde "cherry" hasta el final:

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

## Rango de índices negativos

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

Ejemplo

Este ejemplo devuelve los elementos desde "naranja" (-4) hasta "mango" (-1), pero SIN incluirlo:

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

## Comprobar si el artículo existe

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

Ejemplo

Comprueba si "manzana" está presente en la lista:

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

# Python - Cambiar elementos de la lista

## Cambiar el valor del artículo

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

Ejemplo

Cambiar 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 haga referencia al rango de números de índice donde desea insertar los nuevos valores:

Ejemplo

Cambie los valores "banana" 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 reemplazándolos con *un* valor:

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

## Insertar elementos

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

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

Ejemplo

Insertar "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.

# Python - Agregar elementos a una lista

## Añadir elementos

Para agregar un elemento al final de la lista, utilice 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 especificado, 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, utilice el extend()método.

Ejemplo

Añade los elementos de tropicala thislist:

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

Los elementos se agregarán al *final* de la lista.

## Agregar cualquier iterable

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

Ejemplo

Agregar elementos de una tupla a una lista:

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

# Eliminar el elemento especificado

## El remove()método elimina el elemento especificado.

Ejemplo

Eliminar "banana":

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

Si hay más de un elemento con el valor especificado, el remove()método elimina la primera aparición:

Ejemplo

Eliminar la primera aparición de "banana":

thislist = ["apple", "banana", "cherry", "banana", "kiwi"]  
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 del palabra clave también elimina el índice especificado:

Ejemplo

Eliminar el primer elemento:

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

La del palabra clave también puede eliminar la lista por completo.

Ejemplo

Eliminar la lista completa:

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

## Limpiar la lista

El clear()método vacía la lista.

La lista aún permanece, pero no tiene contenido.

Ejemplo

Limpiar el contenido de la lista:

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

# Python - Listas de bucles

## Recorrer una lista en bucle

Puedes recorrer los elementos de la lista utilizando 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 [Bucles For de Python .](https://www.w3schools.com/python/python_for_loops.asp)

## Recorrer 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 artículos consultando 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].

## Usando un bucle While

Puedes recorrer los elementos de la lista utilizando un whilebucle.

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

Recuerde aumentar el índice en 1 después de cada iteración.

Ejemplo

Imprimir todos los elementos, utilizando un whilebucle para recorrer 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 [Bucles While de Python .](https://www.w3schools.com/python/python_while_loops.asp)

## Bucles mediante comprensión de listas

La comprensión de listas ofrece la sintaxis más corta para recorrer listas:

Ejemplo

Un forbucle manual corto que imprimirá todos los elementos de una lista:

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

Obtenga más información sobre la comprensión de listas en el siguiente capítulo: [Comprensión de listas](https://www.w3schools.com/python/python_lists_comprehension.asp) .

Ejercicio?

¿Cuál es la sintaxis correcta para recorrer los elementos de una lista?

Principio del formulario

for x in ['apple', 'banana', 'cherry']:  
  print(x)

for x in ['apple', 'banana', 'cherry']  
  print(x)

foreach x in ['apple', 'banana', 'cherry']  
  print(x)

Final del formulario

## Python - Listas de bucles

Recorrer una lista en bucle

Puedes recorrer los elementos de la lista utilizando 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 [Bucles For de Python .](https://www.w3schools.com/python/python_for_loops.asp)

## Recorrer 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 artículos consultando 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].

## Usando un bucle While

Puedes recorrer los elementos de la lista utilizando un whilebucle.

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

Recuerde aumentar el índice en 1 después de cada iteración.

Ejemplo

Imprimir todos los elementos, utilizando un whilebucle para recorrer 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 [Bucles While de Python .](https://www.w3schools.com/python/python_while_loops.asp)

# Python - Comprensión de listas

## Comprensión de listas

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, quieres una nueva lista, que contenga solo las frutas con la letra "a" en el nombre.

Sin comprensión de listas, tendrás que escribir una for declaración 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 sólo acepta los artículos que valen True.

Ejemplo

Acepta únicamente artículos que no sean “manzana”:

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

La condición if x != "apple"  regresará Truepara todos los elementos excepto "manzana", lo que hará que la nueva lista contenga todas las frutas excepto "manzana".

La *condición* es opcional y se puede omitir:

Ejemplo

Sin ninguna ifdeclaración:

newlist = [x for x in fruits]

## Iterable

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

Ejemplo

Puede utilizar la range()función para crear un iterable:

newlist = [x for x in range(10)]

Mismo ejemplo, pero con una condición:

Ejemplo

Acepte únicamente 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 puedes manipular antes de que termine como un elemento de lista en la nueva lista:

Ejemplo

Establezca los valores de la nueva lista en mayúsculas:

newlist = [x.upper() for x in fruits]

Puedes configurar el resultado como desees:

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 "banana":

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

La *expresión* del ejemplo anterior dice:

*"Devuelve el artículo si no es plátano, si es plátano devuelve naranja".*

## Bucles mediante comprensión de listas

La comprensión de listas ofrece la sintaxis más corta para recorrer listas:

Ejemplo

Un forbucle manual corto que imprimirá todos los elementos de una lista:

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

Obtenga más información sobre la comprensión de listas en el siguiente capítulo: [Comprensión de listas](https://www.w3schools.com/python/python_lists_comprehension.asp) .

# Python - Ordenar listas

## Ordenar lista alfanuméricamente

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

Ejemplo

Ordenar la lista alfabéticamente:

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

Ejemplo

Ordenar la lista numéricamente:

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

## Ordena

## r descendente

Para ordenar descendentemente, utilice el argumento de palabra clave reverse = True:

Ejemplo

Ordenar la lista descendentemente:

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

Ejemplo

Ordenar la lista descendentemente:

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

## Personalizar la función de clasificación

También puedes personalizar tu propia función utilizando la palabra clave argumento .key =*function*

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

Ejemplo

Ordene la lista según qué tan cerca esté el número de 50:

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

## Ordenación sin distinción entre 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 ordenen antes que las minúsculas:

Ejemplo

La clasificación que distingue entre mayúsculas y minúsculas puede dar un resultado inesperado:

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

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

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

Ejemplo

Realizar una ordenación de la lista sin distinguir entre mayúsculas y minúsculas:

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

## Orden inverso

¿Qué pasa si quieres invertir el orden de una lista, independientemente del alfabeto?

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

Ejemplo

Invertir el orden de los elementos de la lista:

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

Final del formulario

# Python - Copiar listas

## Copiar una lista

No puedes copiar una lista simplemente escribiendo list2 = list1, porque: list2solo será una *referencia* a list1, y los cambios realizados en list1se realizarán automáticamente también en list2.

## Utilice el método copy()

Puede utilizar el método de lista incorporado copy()para copiar una lista.

Ejemplo

Haz una copia de una lista con el copy()método:

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

Utilice el método list()

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

Ejemplo

Haz una copia de una lista con el list()método:

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

## Utilice el operador slice

También puedes hacer una copia de una lista utilizando el :operador (segmento).

Ejemplo

Hacer una copia de una lista con el :operador:

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

# Python - Unir listas

## Unir dos listas

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

Una de las formas más sencillas es utilizando 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 agregando todos los elementos de la lista2 a la lista1, uno por uno:

Ejemplo

Añadir lista2 a lista1:

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

O puedes utilizar el extend() método, donde el propósito es agregar elementos de una lista a otra lista:

Ejemplo

Utilice el extend()método para agregar la lista2 al final de la lista1:

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

# Python - Métodos de lista

## Métodos de lista

Python tiene un conjunto de métodos integrados que puedes usar en listas.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

# Tupla

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

Tupla es uno de los cuatro tipos de datos integrados en Python que se utilizan para almacenar colecciones de datos; los otros tres son [Lista](https://www.w3schools.com/python/python_lists.asp) , [Conjunto](https://www.w3schools.com/python/python_sets.asp) y [Diccionario](https://www.w3schools.com/python/python_dictionaries.asp) , todos con diferentes cualidades y usos.

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

Las tuplas se escriben entre paréntesis.

Ejemplo

Crear una tupla:

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

## Elementos de tupla

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

Los elementos de una 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 son inmutables, lo que significa que no podemos cambiar, agregar o eliminar elementos una vez creada 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 tupla

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

Ejemplo

Imprima el número de elementos en la tupla:

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

## Crear una tupla con un elemento

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

Ejemplo

Una tupla de elementos 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 la 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, números 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 tuple ()

También es posible utilizar el

Ejemplo

Usando el método tuple() para crear una tupla:

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

## Colecciones de Python (matrices)

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

* [**Lista**](https://www.w3schools.com/python/python_lists.asp)
* **Tupla**
* [**Colocar**](https://www.w3schools.com/python/python_sets.asp)
* [**El diccionario**](https://www.w3schools.com/python/python_dictionaries.asp) es una colección ordenada\*\* y modificable. No contiene miembros duplicados.

\*Colocar

\*\*A partir de la versión 3.7 de Python, los diccionarios están *ordenados* . 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 un aumento en la eficiencia o la seguridad.

# Python: acceso a elementos de tuplas

## Acceder a elementos de tupla

Puede acceder a los elementos de la tupla haciendo referencia al número de índice, dentro de corchetes:

Ejemplo

Imprima 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 empezar desde el final.

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

Ejemplo

Imprima el último elemento de la tupla:

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

## Rango de índices

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

Al especificar un rango, el valor de retorno 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 finalizará en el índice 5 (no incluido).

Recuerde que el primer elemento tiene índice 0.

Si omitimos el valor inicial, el rango comenzará en el primer elemento:

Ejemplo

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

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

Al omitir el valor final, el rango continuará hasta el final de la tupla:

Ejemplo

Este ejemplo devuelve los elementos desde "cherry" 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, utilice la in palabra clave:

Ejemplo

Comprueba si "manzana" 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 son inmutables, lo que significa que no se pueden cambiar, agregar ni eliminar elementos una vez creada la tupla.

Pero hay algunas soluciones alternativas.

## Cambiar valores de tupla

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

Pero existe una solución alternativa: puedes convertir la tupla en una lista, modificar la lista y volver a convertir la lista en una tupla.

Ejemplo

Convierte la tupla en una lista para poder modificarla:

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

## Agregar artículos

Como las tuplas son inmutables, no tienen un método incorporado append(), 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 convertirla nuevamente en una tupla.

Ejemplo

Convierte la tupla en una lista, agrega "naranja" y conviértela nuevamente en una tupla:

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

2. **Agregar una tupla a otra tupla**. Se permite 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

Crea una nueva tupla con el valor "naranja" y agrega 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 se pueden eliminar elementos de una tupla.

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

Ejemplo

Convierte la tupla en una lista, elimina "manzana" y conviértela nuevamente en una tupla:

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

O puedes 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 - Descomprimir tuplas

## 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 extraer los valores y convertirlos en variables. Esto se llama "desempaquetar":

Ejemplo

Desempaquetando una tupla:

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

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

## Usando Asterisk\*

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

Asigna 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 se agrega el asterisco 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

Añade una lista de valores a la variable "trópico":

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

# Python - Tuplas de bucleRecorrer una tupla en bucle

Puedes recorrer los elementos de la tupla utilizando un for bucle.

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 for los bucles en nuestro capítulo [Bucles For de Python .](https://www.w3schools.com/python/python_for_loops.asp)

## Recorrer los números de índice

También puedes 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 artículos consultando su número de índice:

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

## Usando un bucle While

Puedes recorrer los elementos de la tupla utilizando un while bucle.

Utilice 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

Imprimir todos los elementos, utilizando un while bucle para recorrer todos los números de índice:

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

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

# Python - Unir tuplas

## Unir dos tuplas

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

Ejemplo

Unir 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)

# Python - Métodos de tupla

## Métodos de tupla

Python tiene dos métodos integrados que puedes usar en tuplas.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of where it was found |

# Conjuntos de Python

myset = {"apple", "banana", "cherry"}

## Colocar

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

Conjunto es uno de los cuatro tipos de datos integrados en Python que se utilizan para almacenar colecciones de datos; los otros tres son [Lista](https://www.w3schools.com/python/python_lists.asp) , [Tupla](https://www.w3schools.com/python/python_tuples.asp) y [Diccionario](https://www.w3schools.com/python/python_dictionaries.asp) , todos con diferentes cualidades y usos.

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

**\* Nota:***Los elementos* del conjunto no se pueden modificar, pero puedes eliminar elementos y agregar elementos nuevos.

Los conjuntos se escriben con llaves.

Ejemplo

Crear un conjunto:

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

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

## Elementos del conjunto

Los elementos del conjunto 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 del conjunto pueden aparecer en un orden diferente cada vez que los uses y no es posible hacer referencia a ellos mediante índice o clave.

## Inmutable

Los elementos del conjunto son inmutables, lo que significa que no podemos cambiarlos una vez creado el conjunto.

Una vez creado un conjunto, no puedes cambiar sus elementos, pero puedes eliminar elementos y agregar elementos nuevos.

## Duplicados no permitidos

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

Ejemplo

Los valores duplicados serán ignorados:

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

**Nota:** Los valores True y 1se consideran el mismo valor en los conjuntos y se tratan como duplicados:

Ejemplo

True y 1se considera el mismo valor:

thisset = {"apple", "banana", "cherry", True, 1, 2}  
  
print(thisset)

**Nota:** Los valores False y 0se consideran el mismo valor en los conjuntos y se tratan como duplicados:

Ejemplo

False y 0se considera el mismo valor:

thisset = {"apple", "banana", "cherry", False, True, 0}  
  
print(thisset)

## Obtener la longitud de un conjunto

Para determinar cuántos elementos tiene un conjunto, utilice la len() función.

Ejemplo

Obtener el número de elementos en un conjunto:

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

## Elementos del conjunto: tipos de datos

Los elementos del conjunto 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, números 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 utilizar el constructor set() para crear un conjunto.

Ejemplo

Usando el constructor set() para crear un conjunto:

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

## Colecciones de Python (matrices)

Hay cuatro tipos de datos de colección 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.
* [**Tupla**](https://www.w3schools.com/python/python_tuples.asp) es una colección ordenada e inmutable. Permite miembros duplicados.
* **Un conjunto** es una colección desordenada, inmutable\* y no indexada. No contiene miembros duplicados.
* [**El diccionario**](https://www.w3schools.com/python/python_dictionaries.asp) es una colección ordenada\*\* y modificable. No contiene miembros duplicados.

\* *Los elementos* del conjunto no se pueden modificar, pero puedes eliminar elementos y agregar elementos nuevos.

\*\*A partir de la versión 3.7 de Python, los diccionarios están *ordenados*. 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 un aumento en la eficiencia o la seguridad.

# Python - Acceso a elementos de un conjunto

## Acceder a elementos

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

Pero puedes recorrer los elementos del conjunto usando un for bucle, o preguntar si un valor especificado está presente en un conjunto, usando la in palabra clave.

Ejemplo

Recorre el conjunto e imprime los valores:

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

Ejemplo

Comprueba si "banana" está presente en el conjunto:

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

Ejemplo

Comprueba si "banana" NO está presente en el conjunto:

thisset = {"apple", "banana", "cherry"}  
  
print("banana" not in thisset)

## Cambiar elementos

Una vez creado un conjunto, no puedes cambiar sus elementos, pero puedes agregar elementos nuevos.

# Python - Agregar elementos a un conjunto

## Agregar artículos

Una vez creado un conjunto, no puedes cambiar sus elementos, pero puedes agregar elementos nuevos.

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

Ejemplo

Agregue un elemento a un conjunto, utilizando 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

Añadir elementos de una lista a un conjunto:

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

# Python - Eliminar elementos del conjunto

## Eliminar artículo

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

Ejemplo

Eliminar "banana" utilizando el remove() método:

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

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

Ejemplo

Eliminar "banana" utilizando el discard() método:

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

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

También puedes usar el pop () método para eliminar un elemento, pero este método eliminará un elemento aleatorio, por lo que no puedes estar seguro de qué elemento se elimina.

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

Ejemplo

Eliminar un elemento aleatorio 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 utilizar el pop () método no se sabe qué elemento se elimina.

Ejemplo

El clear() método vacía el conjunto:

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

Ejemplo

La del palabra clave eliminará el conjunto por completo:

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

# Python - Conjuntos de bucles

## Elementos de bucle

Puedes recorrer los elementos del conjunto mediante un for bucle:

Ejemplo

Recorre el conjunto e imprime los valores:

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

# Python - Unir conjuntos

## Unir conjuntos

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

Los métodos union()y update()unen todos los elementos de ambos conjuntos.

El intersection()método conserva SÓLO los duplicados.

El difference()método conserva los elementos del primer conjunto que no están en los otros conjuntos.

El symmetric\_difference()método conserva todos los elementos EXCEPTO los duplicados.

## Unión

El union()método devuelve un nuevo conjunto con todos los elementos de ambos conjuntos.

Ejemplo

Unir el conjunto 1 y el conjunto 2 en un nuevo conjunto:

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

Puedes utilizar el |operador en lugar del union()método y obtendrás el mismo resultado.

Ejemplo

Se utiliza |para unir dos conjuntos:

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

## Unir varios conjuntos

Todos los métodos y operadores de unión se pueden utilizar para unir varios conjuntos.

Al utilizar un método, simplemente agregue más conjuntos entre paréntesis, separados por comas:

Ejemplo

Unir varios conjuntos con el union()método:

set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set3 = {"John", "Elena"}  
set4 = {"apple", "bananas", "cherry"}  
  
myset = set1.union(set2, set3, set4)  
print(myset)

Al utilizar el |operador, separe los conjuntos con más |operadores:

Ejemplo

Se utiliza |para unir dos conjuntos:

set1 = {"a", "b", "c"}  
set2 = {1, 2, 3}  
set3 = {"John", "Elena"}  
set4 = {"apple", "bananas", "cherry"}  
  
myset = set1 | set2 | set3 |set4  
print(myset)

## Unir un conjunto y una tupla

El union()método le permite unir un conjunto con otros tipos de datos, como listas o tuplas.

El resultado será un conjunto.

Ejemplo

Unir un conjunto con una tupla:

x = {"a", "b", "c"}  
y = (1, 2, 3)  
  
z = x.union(y)  
print(z)

**Nota:** El  | operador solo permite unir conjuntos con conjuntos, y no con otros tipos de datos como puede hacerlo con el  union()método.

## Actualizar

El update()método inserta todos los elementos de un conjunto en otro.

Cambia update()el conjunto original y no devuelve un conjunto nuevo.

Ejemplo

El update()método inserta los elementos del conjunto 2 en el conjunto 1:

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

**Nota:** Tanto union()como update() excluirán cualquier elemento duplicado.

## Intersección

Conservar SÓLO los duplicados

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

Ejemplo

Unir set1 y set2, pero conservar sólo los duplicados:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1.intersection(set2)  
print(set3)

Puedes utilizar el &operador en lugar del intersection()método y obtendrás el mismo resultado.

Ejemplo

Se utiliza &para unir dos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1 & set2  
print(set3)

**Nota:** El & operador solo permite unir conjuntos con conjuntos, y no con otros tipos de datos como puede hacerlo con el intersection()método.

El intersection\_update()método también conservará SÓLO los duplicados, pero cambiará el conjunto original en lugar de devolver un nuevo conjunto.

Ejemplo

Conservar los elementos que existen tanto en set1, como en set2:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set1.intersection\_update(set2)  
  
print(set1)

Los valores Truey 1 se consideran el mismo valor. Lo mismo ocurre con Falsey 0.

Ejemplo

Unir conjuntos que contienen los valores True, False, 1, y 0, y ver qué se considera duplicado:

set1 = {"apple", 1,  "banana", 0, "cherry"}  
set2 = {False, "google", 1, "apple", 2, True}  
  
set3 = set1.intersection(set2)  
  
print(set3)

## Diferencia

El difference()método devolverá un nuevo conjunto que contendrá solo los elementos del primer conjunto que no están presentes en el otro conjunto.

Ejemplo

Conservar todos los elementos del conjunto 1 que no estén en el conjunto 2:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1.difference(set2)  
  
print(set3)

Puedes utilizar el -operador en lugar del difference()método y obtendrás el mismo resultado.

Ejemplo

Se utiliza -para unir dos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1 - set2  
print(set3)

**Nota:** El - operador solo permite unir conjuntos con conjuntos, y no con otros tipos de datos como puede hacerlo con el difference()método.

El difference\_update()método también conservará los elementos del primer conjunto que no estén en el otro conjunto, pero cambiará el conjunto original en lugar de devolver un nuevo conjunto.

Ejemplo

Utilice el difference\_update()método para conservar los elementos que no están presentes en ambos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set1.difference\_update(set2)  
  
print(set1)

## Diferencias simétricas

El symmetric\_difference()método conservará únicamente los elementos que NO estén presentes en ambos conjuntos.

Ejemplo

Conservar los elementos que no están presentes en ambos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1.symmetric\_difference(set2)  
  
print(set3)

Puedes utilizar el ^operador en lugar del symmetric\_difference()método y obtendrás el mismo resultado.

Ejemplo

Se utiliza ^para unir dos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set3 = set1 ^ set2  
print(set3)

**Nota:** El ^ operador solo permite unir conjuntos con conjuntos, y no con otros tipos de datos como puede hacerlo con el symmetric\_difference()método.

El symmetric\_difference\_update()método también conservará todos los datos excepto los duplicados, pero cambiará el conjunto original en lugar de devolver un nuevo conjunto.

Ejemplo

Utilice el symmetric\_difference\_update()método para conservar los elementos que no están presentes en ambos conjuntos:

set1 = {"apple", "banana", "cherry"}  
set2 = {"google", "microsoft", "apple"}  
  
set1.symmetric\_difference\_update(set2)  
  
print(set1)

# Python - Métodos de conjunto

## Métodos de configuración

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

|  |  |  |
| --- | --- | --- |
| **Method** | **Shortcut** | **Description** |
| [add()](https://www.w3schools.com/python/ref_set_add.asp) |  | Adds an element to the set |
| [clear()](https://www.w3schools.com/python/ref_set_clear.asp) |  | Removes all the elements from the set |
| [copy()](https://www.w3schools.com/python/ref_set_copy.asp) |  | Returns a copy of the set |
| [difference()](https://www.w3schools.com/python/ref_set_difference.asp) | [-](https://www.w3schools.com/python/ref_set_difference.asp) | Returns a set containing the difference between two or more sets |
| [difference\_update()](https://www.w3schools.com/python/ref_set_difference_update.asp) | [-=](https://www.w3schools.com/python/ref_set_difference_update.asp) | Removes the items in this set that are also included in another, specified set |
| [discard()](https://www.w3schools.com/python/ref_set_discard.asp) |  | Remove the specified item |
| [intersection()](https://www.w3schools.com/python/ref_set_intersection.asp) | [&](https://www.w3schools.com/python/ref_set_intersection.asp) | Returns a set, that is the intersection of two other sets |
| [intersection\_update()](https://www.w3schools.com/python/ref_set_intersection_update.asp) | [&=](https://www.w3schools.com/python/ref_set_intersection_update.asp) | Removes the items in this set that are not present in other, specified set(s) |
| [isdisjoint()](https://www.w3schools.com/python/ref_set_isdisjoint.asp) |  | Returns whether two sets have a intersection or not |
| [issubset()](https://www.w3schools.com/python/ref_set_issubset.asp) | [<=](https://www.w3schools.com/python/ref_set_issubset.asp) | Returns whether another set contains this set or not |
|  | [<](https://www.w3schools.com/python/ref_set_issubset.asp) | Returns whether all items in this set is present in other, specified set(s) |
| [issuperset()](https://www.w3schools.com/python/ref_set_issuperset.asp) | [>=](https://www.w3schools.com/python/ref_set_issuperset.asp) | Returns whether this set contains another set or not |
|  | [>](https://www.w3schools.com/python/ref_set_issuperset.asp) | Returns whether all items in other, specified set(s) is present in this set |
| [pop()](https://www.w3schools.com/python/ref_set_pop.asp) |  | Removes an element from the set |
| [remove()](https://www.w3schools.com/python/ref_set_remove.asp) |  | Removes the specified element |
| [symmetric\_difference()](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | [^](https://www.w3schools.com/python/ref_set_symmetric_difference.asp) | Returns a set with the symmetric differences of two sets |
| [symmetric\_difference\_update()](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | [^=](https://www.w3schools.com/python/ref_set_symmetric_difference_update.asp) | Inserts the symmetric differences from this set and another |
| [union()](https://www.w3schools.com/python/ref_set_union.asp) | [|](https://www.w3schools.com/python/ref_set_union.asp) | Return a set containing the union of sets |
| [update()](https://www.w3schools.com/python/ref_set_update.asp) | [|=](https://www.w3schools.com/python/ref_set_update.asp) | Update the set with the union of this set and others |

# 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 permite 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 entre llaves 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, son modificables y no permiten duplicados.

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

Ejemplo

Imprima el valor "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 se puede hacer referencia a un elemento mediante un índice.

## Cambiable

Los diccionarios son modificables, lo que significa que podemos cambiar, agregar o eliminar elementos después de haber creado el diccionario.

## Duplicados no permitidos

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, utilice la len()función:

Ejemplo

Imprima el número de elementos del diccionario:

print(len(thisdict))

## Elementos del diccionario: tipos de datos

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

Ejemplo

Tipos de datos de cadena, int, booleano y 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

Imprima el tipo de datos de un diccionario:

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

El constructor dict()

También es posible utilizar el constructor dict() para crear un diccionario.

Ejemplo

Usando el método dict() para crear un diccionario:

thisdict = dict(name = "John", age = 36, country = "Norway")  
print(thisdict)

## Colecciones de Python (matrices)

Hay cuatro tipos de datos de colección 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.
* [**Tupla**](https://www.w3schools.com/python/python_tuples.asp) es una colección ordenada e inmutable. Permite miembros duplicados.
* [**Un conjunto**](https://www.w3schools.com/python/python_sets.asp) es una colección desordenada, inmutable\* y no indexada. No contiene miembros duplicados.
* **El diccionario** es una colección ordenada\*\* y modificable. No contiene miembros duplicados.

\* *Los elementos* del conjunto no se pueden modificar, pero puedes eliminarlos y/o agregarlos cuando lo desees.

\*\*A partir de la versión 3.7 de Python, los diccionarios están *ordenados*. 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 un aumento en la eficiencia o la seguridad.

Python: acceso a elementos del diccionario

## Acceder a los elementos

Puede acceder a los elementos de un diccionario consultando su nombre clave, dentro de 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 observe que la lista de claves también se actualiza:

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 del 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 verá que la lista de valores también se actualiza:

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 observe que la lista de valores también se actualiza:

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 de un diccionario, como tuplas en una lista.

Ejemplo

Obtenga 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 verá que la lista de elementos también se actualiza:

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 observe que la lista de elementos también se actualiza:

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, utilice la in palabra clave:

Ejemplo

Comprueba 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 elementos del diccionario

## Cambiar valores

Puede cambiar el valor de un elemento específico consultando su nombre clave:

Ejemplo

Cambie 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

## Agregar elementos

Para agregar un elemento al diccionario se utiliza una nueva clave de índice y se le asigna 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 determinado. Si el elemento 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 utilizando el update() método:

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

# Python - Eliminar elementos del diccionario

## Eliminación de elementos

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 pop item()método elimina el último elemento insertado (en versiones anteriores a 3.7, se elimina un elemento aleatorio):

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

Ejemplo

La del palabra clave elimina el elemento con el nombre de clave especificado:

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

Ejemplo

La del palabra 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)

# Python - Diccionarios de bucles

## Recorrer un diccionario en bucle

Puedes recorrer un diccionario utilizando un for bucle.

Al recorrer un diccionario, el valor de retorno son las *claves* del diccionario, pero también existen métodos para devolver los *valores.*

Ejemplo

Imprima todos los nombres de las claves del diccionario, uno por uno:

for x in thisdict:  
  print(x)

Ejemplo

Imprima todos *los valores* del diccionario, uno por uno:

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

También puedes utilizar el values()método para devolver valores de un diccionario:

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

Ejemplo

Puede utilizar el keys()método para devolver las claves de un diccionario:

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

Ejemplo

Recorrer *las claves* y *los valores* mediante el items()método:

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

# Python - Copiar diccionarios

## Copiar un diccionario

No puedes copiar un diccionario simplemente escribiendo dict2 = dict1, porque: dict2 solo será una *referencia* a dict1, y los cambios realizados en dict1se realizarán automáticamente también en dict2.

Hay formas de hacer una copia, una forma es utilizar el método Diccionario 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 utilizar la función incorporada dict().

Ejemplo

Hacer una copia de un diccionario con la dict() función:

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

# Python - Diccionarios anidados

## Diccionarios anidados

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

Ejemplo

Crea un diccionario que contenga tres diccionarios:

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

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

Ejemplo

Crea tres diccionarios y luego crea un diccionario que contendrá 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  
}

## Acceder a elementos en diccionarios anidados

Para acceder a los elementos de un diccionario anidado, utilice el nombre de los diccionarios, comenzando con el diccionario externo:

Ejemplo

Imprima el nombre del niño 2:

print(myfamily["child2"]["name"])

## Recorrer diccionarios anidados

Puedes recorrer un diccionario utilizando el items()método siguiente:

Ejemplo

Recorrer las claves y valores de todos los diccionarios anidados:

for x, obj in myfamily.items():  
  print(x)  
  
  for y in obj:  
    print(y + ':', obj[y])

# Métodos de diccionario

Python tiene un conjunto de métodos integrados que puedes usar en diccionarios.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [clear()](https://www.w3schools.com/python/ref_dictionary_clear.asp) | Removes all the elements from the dictionary |
| [copy()](https://www.w3schools.com/python/ref_dictionary_copy.asp) | Returns a copy of the dictionary |
| [fromkeys()](https://www.w3schools.com/python/ref_dictionary_fromkeys.asp) | Returns a dictionary with the specified keys and value |
| [get()](https://www.w3schools.com/python/ref_dictionary_get.asp) | Returns the value of the specified key |
| [items()](https://www.w3schools.com/python/ref_dictionary_items.asp) | Returns a list containing a tuple for each key value pair |
| [keys()](https://www.w3schools.com/python/ref_dictionary_keys.asp) | Returns a list containing the dictionary's keys |
| [pop()](https://www.w3schools.com/python/ref_dictionary_pop.asp) | Removes the element with the specified key |
| [popitem()](https://www.w3schools.com/python/ref_dictionary_popitem.asp) | Removes the last inserted key-value pair |
| [setdefault()](https://www.w3schools.com/python/ref_dictionary_setdefault.asp) | Returns the value of the specified key. If the key does not exist: insert the key, with the specified value |
| [update()](https://www.w3schools.com/python/ref_dictionary_update.asp) | Updates the dictionary with the specified key-value pairs |
| [values()](https://www.w3schools.com/python/ref_dictionary_values.asp) | Returns a list of all the values in the dictionary |

# Python Si... De lo contrario

## Condiciones y declaraciones If de Python

Python admite las condiciones lógicas habituales de las matemáticas:

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

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

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

Ejemplo

Declaración if:

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

En este ejemplo, utilizamos dos variables, a y b, que se utilizan como parte de la declaración if para comprobar 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 la pantalla que "b es mayor que a".

## Sangría

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 suelen utilizar llaves para este propósito.

Ejemplo

Declaración if, 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 que tiene Python de decir "si las condiciones anteriores no eran verdaderas, entonces pruebe 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".

## Demás

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 y la condición elif tampoco es verdadera, por lo que vamos a la condición else e imprimimos en la pantalla que "a es mayor que b".

También puedes tener uno elsesin elif:

Ejemplo

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

## Forma corta

Si solo tiene una declaración para ejecutar, puede colocarla en la misma línea que la declaración if.

Ejemplo

Declaración if de una línea:

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

## Abreviatura Si... De lo contrario

Si solo tiene una declaración para ejecutar, una para if y otra para else, puede colocarlas todas en la misma línea:

Ejemplo

Declaración if else de una línea:

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

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

También puedes tener varias declaraciones else en la misma línea:

Ejemplo

Una 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")

## Y

La palabra clave and es un operador lógico y se utiliza para combinar declaraciones condicionales:

Ejemplo

Comprueba 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")

## O

La or palabra clave es un operador lógico y se utiliza 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")

## No

La not palabra clave es un operador lógico y se utiliza para invertir el resultado de la declaración condicional:

Ejemplo

Comprueba si a NO es mayor que b:

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

## Anidado si

Puedes tener if declaraciones dentro de if 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.")

## La Declaración de Pase

If Las declaraciones no pueden estar vacías, pero si por alguna razón tiene una if declaración sin contenido, coloque la pass declaración para evitar obtener un error.

Ejemplo

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

# Bucles While de Python

## Bucles de Python

Python tiene dos comandos de bucle primitivos:

* bucles while
* bucles for

El bucle while

Con el bucle while podemos ejecutar un conjunto de sentencias 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, de lo contrario el bucle continuará para siempre.

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

## La declaración de ruptura

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

Ejemplo

Salir del bucle cuando i es 3:

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

## La Declaración Continua

Con la declaración continue 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)

## La declaración else

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")

# Bucles For de Python

## Bucles For de Python

Un bucle for se utiliza para iterar sobre una secuencia (que puede ser 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 los que se encuentran en otros lenguajes de programación orientados a objetos.

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

Ejemplo

Imprima cada fruta en una lista de frutas:

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

El bucle for no requiere que se establezca una variable de indexación de antemano.

## Hacer un bucle a través de una string

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

Ejemplo

Recorre las letras de la palabra "banana":

for x in "banana":  
  print(x)

## La declaración de ruptura

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

Ejemplo

Salir del bucle cuando xes "banana":

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

Ejemplo

Salir del bucle cuando xes "banana", pero esta vez el salto viene antes de la impresión:

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

## La Declaración Continua

Con la declaración continue podemos detener la iteración actual del bucle 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 range()

Para recorrer un conjunto de código una cantidad específica de veces, podemos usar la función range() ,

La función range() devuelve una secuencia de números, comenzando desde 0 de manera predeterminada, y se incrementa en 1 (de manera predeterminada), y finaliza en un número especificado.

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() tiene como valor inicial predeterminado 0, sin embargo es posible especificar el valor inicial agregando un parámetro: range(2, 6) , que significa valores de 2 a 6 (pero sin incluir 6):

Ejemplo

Usando el parámetro de inicio:

for x in range(2, 6):  
  print(x)

La función range() tiene como valor predeterminado incrementar la secuencia en 1, sin embargo es posible especificar el valor de incremento agregando un tercer parámetro: range(2, 30, **3** ) :

Ejemplo

Incrementa la secuencia con 3 (el valor predeterminado es 1):

for x in range(2, 30, 3):  
  print(x)

## De lo contrario en el bucle For

La elsepalabra clave en un forbucle especifica un bloque de código que se ejecutará cuando finalice el bucle:

Ejemplo

Imprime todos los números del 0 al 5 e imprime un mensaje cuando el bucle haya finalizado:

for x in range(6):  
  print(x)  
else:  
  print("Finally finished!")

**Nota:** El elsebloque NO se ejecutará si el bucle se detiene mediante una breakdeclaración.

Ejemplo

Rompa el bucle cuando xsea 3 y observe 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

Imprime 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

For Los bucles no pueden estar vacíos, pero si por alguna razón tienes un for bucle sin contenido, ponlo en la pass declaración para evitar obtener un error.

Ejemplo

for x in [0, 1, 2]:  
  pass

# Funciones de Python

Una función es un bloque de código que solo se ejecuta cuando se lo llama.

Puede pasar datos, conocidos como parámetros, a una función.

Una función puede devolver datos como resultado.

## Creando una función

En Python, una función se define utilizando la palabra clave def :

Ejemplo

def my\_function():  
  print("Hello from a function")

## Llamar a una función

Para llamar a una función, utilice 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, dentro de los paréntesis. Puedes agregar tantos argumentos como quieras, solo sepáralos con una coma.

El siguiente ejemplo tiene una función con un argumento (fname). Cuando se llama a la función, pasamos un nombre, que se utiliza 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"**)

En la documentación de Python, *los argumentos* suelen abreviarse como *args .*

¿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 dentro de los paréntesis en la definición de la función.

Un argumento es el valor que se envía a la función cuando se llama.

## Número de argumentos

De forma predeterminada, una función debe llamarse con la cantidad correcta de argumentos. Esto significa que, si su función espera 2 argumentos, debe llamarla 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 como *\*args* en la documentación de Python.

## Argumentos de palabras clave

También puedes 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 manera, 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 abrevian como *\*\*kwargs* en la documentación de Python.

## Valor del parámetro predeterminado

El siguiente ejemplo muestra cómo utilizar un valor de parámetro predeterminado.

Si llamamos a la función sin argumentos, utiliza 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 de retorno

Para permitir que una función devuelva un valor, utilice 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

Function Las definiciones no pueden estar vacías, pero si por alguna razón tienes una function definición sin contenido, ponla en la pass declaración para evitar obtener un error.

Ejemplo

def myfunction():  
  pass

## Argumentos exclusivamente posicionales

Puede especificar que una función puede tener SÓLO argumentos posicionales o SÓLO argumentos de palabras clave.

Para especificar que una función solo puede tener argumentos posicionales, agregue, / después de los argumentos:

Ejemplo

def my\_function(x, /):  
  print(x)  
  
my\_function(3)

Sin embargo, , /en realidad se permite usar argumentos de palabras clave incluso si la función espera argumentos posicionales:

Ejemplo

def my\_function(x):  
  print(x)  
  
my\_function(x = 3)

Pero al agregarlo, , /obtendrá un error si intenta enviar un argumento de palabra clave:

Ejemplo

def my\_function(x, /):  
  print(x)  
  
my\_function(x = 3)

## Argumentos basados ​​únicamente en palabras clave

Para especificar que una función solo puede tener argumentos de palabras clave, agregue \*, *antes de* los argumentos:

Ejemplo

def my\_function(\*, x):  
  print(x)  
  
my\_function(x = 3)

Sin él, \*, se permite utilizar argumentos posicionales incluso si la función espera argumentos de palabras clave:

Ejemplo

def my\_function(x):  
  print(x)  
  
my\_function(3)

Pero al agregarlo, \*, /obtendrá un error si intenta enviar un argumento posicional:

Ejemplo

def my\_function(\*, x):  
  print(x)  
  
my\_function(3)

Combine posicionamiento únicamente y palabras clave únicamente

Puede combinar los dos tipos de argumentos en la misma función.

Cualquier argumento *antes de* son /, solo posicionales, y cualquier argumento *después* de \*, son solo palabras clave.

Ejemplo

def my\_function(a, b, /, \*, c, d):  
  print(a + b + c + d)  
  
my\_function(5, 6, c = 7, d = 8)

## Recursión

Python también acepta la recursión de funciones, lo que significa que una función definida puede llamarse a sí misma.

La recursión es un concepto matemático y de programación común. Significa que una función se llama a sí misma. Esto tiene la ventaja de que permite recorrer los datos en bucle para llegar a un resultado.

El desarrollador debe tener mucho cuidado con la recursión, ya que puede resultar muy fácil escribir una función que nunca finalice o que utilice cantidades excesivas de memoria o potencia del procesador. Sin embargo, cuando se escribe correctamente, la recursión puede ser un método de programación muy eficiente y matemáticamente elegante.

En este ejemplo, tri\_recursion() es una función que hemos definido para que se llame a sí misma ("recurse"). Usamos la variable k como dato, que decrementa ( -1) cada vez que realizamos una recursión. La recursión finaliza cuando la condición no es mayor que 0 (es decir, cuando es 0).

Para un desarrollador nuevo puede llevar algún tiempo comprender exactamente cómo funciona esto; la mejor forma de descubrirlo es probándolo y modificándolo.

Ejemplo

Ejemplo de recursión

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)

# Lambda de Python

Una función lambda es una pequeña función anónima.

Una función lambda puede tomar cualquier número 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

Multiplica un argumento apor otro by devuelve el resultado:

x = lambda a, b : a \* b  
print(x(5, 6))

Ejemplo

Resume el argumento a, b, y cy devuelve el resultado:

x = lambda a, b, c : a + b + c  
print(x(5, 6, 2))

¿Por qué utilizar funciones Lambda?

El poder de lambda se muestra mejor cuando se utiliza como una función anónima dentro de otra función.

Digamos que tienes una definición de función que toma un argumento y ese argumento se multiplicará por un número desconocido:

def myfunc(n):  
  return lambda a : a \* n

Utilice esa definición de función para crear una función que siempre duplique el número que envíe:

Ejemplo

def myfunc(n):  
  return lambda a : a \* n  
  
mydoubler = myfunc(2)  
  
print(mydoubler(11))

O bien, utilice la misma definición de función para crear una función que siempre *triplique* el número que envíe:

Ejemplo

def myfunc(n):  
  return lambda a : a \* n  
  
mytripler = myfunc(3)  
  
print(mytripler(11))

O bien, utilice la misma definición de función para realizar 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 matrices, pero se pueden usar [listas de Python en su lugar.](https://www.w3schools.com/python/python_lists.asp)

## Matrices

**Nota:** Esta página le muestra cómo usar LISTAS como MATRICES, sin embargo, para trabajar con matrices en Python tendrá que 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

Crea una matriz que contenga los nombres de los 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 verse así:

car1 = "Ford"  
car2 = "Volvo"  
car3 = "BMW"

Sin embargo, ¿qué sucede si desea recorrer los autos y encontrar uno específico? ¿Y si no tiene 3 autos, sino 300?

¡La solución es una matriz!

Una matriz puede contener muchos valores bajo un solo nombre y se puede acceder a los valores haciendo referencia a un número de índice.

## Acceder a los elementos de una matriz

Para hacer referencia a un elemento de la matriz, haga referencia al *número de índice* .

Ejemplo

Obtener el valor del primer elemento de la matriz:

x = cars[0]

Ejemplo

Modificar 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)

[Pruébelo usted mismo »](https://www.w3schools.com/python/trypython.asp?filename=demo_array4)

**Nota:** La longitud de una matriz es siempre uno más que el índice de matriz más alto.

## Elementos de matriz en bucle

Puedes utilizar el for inbucle para recorrer todos los elementos de una matriz.

Ejemplo

Imprima cada elemento de la carsmatriz:

for x in cars:  
  print(x)

## Agregar elementos de matriz

Puede utilizar el append()método para agregar un elemento a una matriz.

Ejemplo

Añade un elemento más a la carsmatriz:

cars.append("Honda")

## Eliminación de elementos de matriz

Puede utilizar el pop()método para eliminar un elemento de la matriz.

Ejemplo

Eliminar el segundo elemento de la carsmatriz:

cars.pop(1)

También puedes utilizar el remove()método para eliminar un elemento de la matriz.

Ejemplo

Eliminar el elemento que tiene el valor "Volvo":

cars.remove("Volvo")

**Nota:** El método de la lista remove()solo elimina la primera aparición del valor especificado.

## Métodos de matriz

Python tiene un conjunto de métodos integrados que puedes usar en listas/matrices.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [append()](https://www.w3schools.com/python/ref_list_append.asp) | Adds an element at the end of the list |
| [clear()](https://www.w3schools.com/python/ref_list_clear.asp) | Removes all the elements from the list |
| [copy()](https://www.w3schools.com/python/ref_list_copy.asp) | Returns a copy of the list |
| [count()](https://www.w3schools.com/python/ref_list_count.asp) | Returns the number of elements with the specified value |
| [extend()](https://www.w3schools.com/python/ref_list_extend.asp) | Add the elements of a list (or any iterable), to the end of the current list |
| [index()](https://www.w3schools.com/python/ref_list_index.asp) | Returns the index of the first element with the specified value |
| [insert()](https://www.w3schools.com/python/ref_list_insert.asp) | Adds an element at the specified position |
| [pop()](https://www.w3schools.com/python/ref_list_pop.asp) | Removes the element at the specified position |
| [remove()](https://www.w3schools.com/python/ref_list_remove.asp) | Removes the first item with the specified value |
| [reverse()](https://www.w3schools.com/python/ref_list_reverse.asp) | Reverses the order of the list |
| [sort()](https://www.w3schools.com/python/ref_list_sort.asp) | Sorts the list |

**Nota:** Python no tiene soporte integrado para matrices, pero se pueden usar listas de Python en su lugar.

# Clases y objetos 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 "plano" para crear objetos.

## Crear una clase

Para crear una clase, utilice la palabra clave class:

Ejemplo

Crea 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

Crea un objeto llamado p1 e imprime 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 entender el significado de las clases debemos comprender la función incorporada \_\_init\_\_().

Todas las clases tienen una función llamada \_\_init\_\_(), que siempre se ejecuta cuando se inicia la clase.

Utilice la función \_\_init\_\_() para asignar valores a las propiedades del objeto u otras operaciones que sean necesarias realizar 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 se utiliza la clase para crear un nuevo objeto.

## La función \_\_str\_\_()

La función \_\_str\_\_() controla lo que se debe devolver cuando el objeto de clase se representa como una cadena.

Si la función \_\_str\_\_() no está configurada, se devuelve la representación de cadena del objeto:

Ejemplo

La representación de cadena de un objeto SIN la función \_\_str\_\_():

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
p1 = Person("John", 36)  
  
print(p1)

Ejemplo

La representación de cadena de un objeto CON la función \_\_str\_\_():

class Person:  
  def \_\_init\_\_(self, name, age):  
    self.name = name  
    self.age = age  
  
  def \_\_str\_\_(self):  
    return f"{self.name}({self.age})"  
  
p1 = Person("John", 36)  
  
print(p1)

## Métodos de objeto

Los objetos también pueden contener métodos. Los métodos de los objetos son funciones que pertenecen al objeto.

Creemos un método en la clase Persona:

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 parámetro self es una referencia a la instancia actual de la clase y se utiliza para acceder a las variables que pertenecen a la clase.

## El parámetro Self

El parámetro self es una referencia a la instancia actual de la clase y se utiliza para acceder a las variables que pertenecen a la clase.

No es necesario que tenga nombre self, puedes llamarlo como quieras, pero tiene que ser el primer parámetro de cualquier función de la clase:

Ejemplo

Utilice 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

Puedes modificar propiedades en objetos de esta manera:

Ejemplo

Establezca la edad de p1 en 40:

p1.age = 40

## Eliminar propiedades de objeto

Puede eliminar propiedades de objetos utilizando la del palabra clave:

Ejemplo

Eliminar la propiedad de edad del objeto p1:

del p1.age

## Eliminar objetos

Puede eliminar objetos utilizando la del palabra clave:

Ejemplo

Eliminar el objeto p1:

del p1

## La Declaración de Pase

Class las definiciones no pueden estar vacías, pero si por alguna razón tienes una class definición sin contenido, ponla en la pass declaración para evitar obtener un error.

Ejemplo

class Person:  
  pass