Contenido

[Phyton 1](#_Toc206273961)

[Características 1](#_Toc206273962)

[Usos comunes de Python 2](#_Toc206273963)

[Tipos de Datos 3](#_Toc206273964)

[Datos Simples 3](#_Toc206273965)

[Datos Compuestos 3](#_Toc206273966)

[Operadores aritméticos 4](#_Toc206273967)

[Operadores de comparación 8](#_Toc206273968)

[Condicionales 9](#_Toc206273969)

[Operadores logicos 10](#_Toc206273970)

[Métodos de cadenas 10](#_Toc206273971)

[Métodos de listas 11](#_Toc206273972)

# Phyton

## Características

Python es un lenguaje de programación versátil y popular, conocido por su sintaxis clara y fácil de aprender, lo que lo hace ideal para principiantes y profesionales. Es interpretado, lo que significa que se ejecuta línea por línea, y es de alto nivel, lo que facilita la lectura y escritura del código. Además, Python es multiparadigma, soportando programación orientada a objetos, funcional e imperativa.

Características clave de Python:

* **Sintaxis clara y legible:**

Python utiliza una sintaxis simple y concisa que se asemeja al lenguaje natural, lo que facilita la lectura y escritura del código.

* **Interpretado:**

Python es un lenguaje interpretado, lo que significa que el código se ejecuta línea por línea, facilitando la depuración y detección de errores.

* **Alto nivel:**

Python es un lenguaje de alto nivel, lo que significa que abstrae muchos detalles de bajo nivel de la máquina, permitiendo a los desarrolladores concentrarse en la lógica del programa.

* **Tipado dinámico:**

Python no requiere que se especifiquen los tipos de datos de las variables por adelantado, ya que el tipo se determina automáticamente en tiempo de ejecución.

* **Multiparadigma:**

Python soporta múltiples paradigmas de programación, incluyendo la programación orientada a objetos, funcional e imperativa.

* **Orientado a objetos:**

Python soporta la programación orientada a objetos, permitiendo la creación de clases y objetos para organizar y modularizar el código.

* **Extensible:**

Python puede ser extendido con código escrito en otros lenguajes como C y C++, lo que permite mejorar su rendimiento en áreas específicas.

* **Multiplataforma:**

Python puede ejecutarse en diferentes sistemas operativos como Windows, macOS y Linux.

* **Amplia biblioteca estándar:**

Python cuenta con una gran cantidad de módulos y paquetes predefinidos que facilitan el desarrollo de diversas tareas.

* **Comunidad activa y gran cantidad de recursos:**

Python tiene una comunidad de desarrolladores muy activa que contribuye con bibliotecas, documentación y soporte.

## Usos comunes de Python

* **Desarrollo web:**

Python es ampliamente utilizado para el desarrollo web, especialmente con frameworks como Django y Flask.

* **Análisis de datos y aprendizaje automático:**

Python es el lenguaje estándar para estas áreas, gracias a bibliotecas como Pandas, NumPy, Matplotlib, TensorFlow, Keras y Scikit-learn.

* **Automatización de tareas:**

Python se utiliza para automatizar tareas repetitivas y scripts de sistema.

* **Desarrollo de juegos:**

Python se puede utilizar para desarrollar juegos, especialmente con bibliotecas como Pygame.

* **Ciberseguridad:**

Python es utilizado para desarrollar herramientas de seguridad informática, análisis de vulnerabilidades y pruebas de penetración.

## Tipos de Datos

### Datos Simples

En Python, los tipos de datos se clasifican principalmente en numéricos, cadenas, booleanos, secuencias (listas, tuplas, rangos), diccionarios y conjuntos. Cada tipo tiene características y métodos específicos para manipular datos.

Tipos de datos principales:

* Numéricos:
  + int: Números enteros (ej: -3, 0, 10).
  + float: Números con decimales (ej: 3.14, -2.5).
  + complex: Números complejos (ej: 2 + 3j).
  + str: Cadenas de texto (ej: "Hola", 'Python').
  + bool: Valores booleanos, True o False.
* Secuencias:
  + list: Listas, colecciones ordenadas y mutables de elementos (ej: [1, "a", True]).
  + tuple: Tuplas, colecciones ordenadas e inmutables (ej: (1, "a", True)).
  + range: Secuencias de números.
  + dict: Diccionarios, colecciones de pares clave-valor (ej: {"nombre": "Juan", "edad": 30}).
* Conjuntos:
  + set: Conjuntos, colecciones no ordenadas de elementos únicos (ej: {1, 2, 3}).
  + frozenset: Conjuntos inmutables.
  + bytes: Secuencias de bytes inmutables.
  + bytearray: Secuencias de bytes mutables.
  + memoryview: Permite acceder a la memoria de otros objetos binarios.

### Datos Compuestos

En Python, los tipos de datos compuestos son estructuras que permiten almacenar múltiples valores bajo una misma variable. Estos tipos de datos son fundamentales para organizar y manipular datos de manera más compleja en tus programas. Python ofrece varios tipos de datos compuestos, incluyendo listas, tuplas, diccionarios y conjuntos.

Tipos de datos compuestos principales:

* [**Listas (list)**](https://www.google.com/search?sca_esv=972c76ef38736f5c&cs=0&sxsrf=AE3TifP2FCM8COkNw-KW_Oa1nNe_LgsiQA%3A1755302983911&q=Listas+%28list%29&sa=X&ved=2ahUKEwjLmLyWhY6PAxX_fjABHXf2KXkQxccNegQIDhAB&mstk=AUtExfDwqJgAW5mRgCRVeTCGAxNS9nKddaXE26_cINkPOj4TnJvGIk2uOt7VKCgS880O1_AuAfQ-zclulXfmYn8K5NHl_K3-U0F_P6Ldzfp_X3x9LPivhJt5w3OksMYe72dupIWd4aVw4DK-7QDR-UKk6V0Bz8v1xoawVX9ktxL1vlxh-nKxsIdDROQFvF5jKEFNRVhdVXH2Bpvwa2Yk72XRYgouEV5GfA5opebCqfcjw8jyj1pXQFfWSBsQHDBLLeFk-3YWMV6lBTFzv3YCPkhXTy2I&csui=3)**:**

Son colecciones ordenadas y mutables de elementos. Se crean con corchetes [] y sus elementos se acceden mediante índices.

* [**Tuplas (tuple)**](https://www.google.com/search?sca_esv=972c76ef38736f5c&cs=0&sxsrf=AE3TifP2FCM8COkNw-KW_Oa1nNe_LgsiQA%3A1755302983911&q=Tuplas+%28tuple%29&sa=X&ved=2ahUKEwjLmLyWhY6PAxX_fjABHXf2KXkQxccNegQIEhAB&mstk=AUtExfDwqJgAW5mRgCRVeTCGAxNS9nKddaXE26_cINkPOj4TnJvGIk2uOt7VKCgS880O1_AuAfQ-zclulXfmYn8K5NHl_K3-U0F_P6Ldzfp_X3x9LPivhJt5w3OksMYe72dupIWd4aVw4DK-7QDR-UKk6V0Bz8v1xoawVX9ktxL1vlxh-nKxsIdDROQFvF5jKEFNRVhdVXH2Bpvwa2Yk72XRYgouEV5GfA5opebCqfcjw8jyj1pXQFfWSBsQHDBLLeFk-3YWMV6lBTFzv3YCPkhXTy2I&csui=3)**:**

Similares a las listas, pero son inmutables, es decir, no se pueden modificar después de su creación. Se crean con paréntesis ().

* [**Diccionarios (dict)**](https://www.google.com/search?sca_esv=972c76ef38736f5c&cs=0&sxsrf=AE3TifP2FCM8COkNw-KW_Oa1nNe_LgsiQA%3A1755302983911&q=Diccionarios+%28dict%29&sa=X&ved=2ahUKEwjLmLyWhY6PAxX_fjABHXf2KXkQxccNegQIExAB&mstk=AUtExfDwqJgAW5mRgCRVeTCGAxNS9nKddaXE26_cINkPOj4TnJvGIk2uOt7VKCgS880O1_AuAfQ-zclulXfmYn8K5NHl_K3-U0F_P6Ldzfp_X3x9LPivhJt5w3OksMYe72dupIWd4aVw4DK-7QDR-UKk6V0Bz8v1xoawVX9ktxL1vlxh-nKxsIdDROQFvF5jKEFNRVhdVXH2Bpvwa2Yk72XRYgouEV5GfA5opebCqfcjw8jyj1pXQFfWSBsQHDBLLeFk-3YWMV6lBTFzv3YCPkhXTy2I&csui=3)**:**

Colecciones de pares clave-valor, donde cada clave es única y se utiliza para acceder al valor asociado. Se crean con llaves {} y se definen como clave: valor.

* [**Conjuntos (set)**](https://www.google.com/search?sca_esv=972c76ef38736f5c&cs=0&sxsrf=AE3TifP2FCM8COkNw-KW_Oa1nNe_LgsiQA%3A1755302983911&q=Conjuntos+%28set%29&sa=X&ved=2ahUKEwjLmLyWhY6PAxX_fjABHXf2KXkQxccNegQIERAB&mstk=AUtExfDwqJgAW5mRgCRVeTCGAxNS9nKddaXE26_cINkPOj4TnJvGIk2uOt7VKCgS880O1_AuAfQ-zclulXfmYn8K5NHl_K3-U0F_P6Ldzfp_X3x9LPivhJt5w3OksMYe72dupIWd4aVw4DK-7QDR-UKk6V0Bz8v1xoawVX9ktxL1vlxh-nKxsIdDROQFvF5jKEFNRVhdVXH2Bpvwa2Yk72XRYgouEV5GfA5opebCqfcjw8jyj1pXQFfWSBsQHDBLLeFk-3YWMV6lBTFzv3YCPkhXTy2I&csui=3)**:**

Colecciones no ordenadas de elementos únicos. Se crean con llaves {} (o la función set()) y son mutables, permitiendo agregar o eliminar elementos.

Características clave de los tipos de datos compuestos:

* **Permiten almacenar diferentes tipos de datos:**

Una lista, por ejemplo, puede contener números, cadenas, e incluso otras listas o diccionarios.

* **Facilitan la organización y manipulación de datos:**

Al agrupar datos relacionados, se puede acceder y procesar la información de manera más eficiente.

* **Ofrecen flexibilidad:**

Los tipos de datos compuestos se adaptan a diversas necesidades, permitiendo construir estructuras de datos complejas y personalizadas.

En resumen: Los tipos de datos compuestos son herramientas esenciales en Python para trabajar con colecciones de datos. Comprender sus características y cómo utilizarlos te permitirá escribir código más eficiente y organizado.

## Operadores aritméticos

Los operadores aritméticos o arithmetic operators son los más comunes que nos podemos encontrar, y nos **permiten realizar operaciones aritméticas** sencillas, como pueden ser la suma, resta o exponente. A continuación, condensamos en la siguiente tabla todos ellos con un ejemplo, donde x=10 y y=3.

| **Operador** | **Nombre** | **Ejemplo** |
| --- | --- | --- |
| + | Suma | x + y = 13 |
| - | Resta | x - y = 7 |
| \* | Multiplicación | x \* y = 30 |
| / | División | x/y = 3.333 |
| % | Módulo | x%y = 1 |
| \*\* | Exponente | x \*\* y = 1000 |
| // | Cociente | 3 |

x = 10; y = 3

print("Operadores aritméticos")

print("x+y =", x+y) #13

print("x-y =", x-y) #7

print("x\*y =", x\*y) #30

print("x/y =", x/y) #3.3333333333333335

print("x%y =", x%y) #1

print("x\*\*y =", x\*\*y) #1000

print("x//y =", x//y) #3

* **Operador +**

El operador + suma los números presentes a la izquierda y derecha del operador. Recalcamos lo de números porque no tendría sentido sumar dos cadenas de texto, o dos listas, pero en Python es posible hacer este tipo de cosas.

print(10 + 3) # 13

Es posible sumar también dos cadenas de texto, pero la suma no será aritmética, sino que se unirán ambas cadenas en una. También se pueden sumar dos listas, cuyo resultado es la unión de ambas.

print("2" + "2") # 22

print([1, 3] + [6, 7]) # [1, 3, 6, 7]

* **Operador –**

El operador - resta los números presentes a la izquierda y derecha del operador. A diferencia el operador + en este caso no podemos restar cadenas o listas.

print(10 - 3) #7

* **Operador \***

El operador \* multiplica los números presentes a la izquierda y derecha del operador.

print(10 \* 3) #30

Como también pasaba con el operador + podemos hacer cosas “raras” con \*. Explicar porque pasan estas cosas es un poquito más complejo, por lo que lo dejamos para otro capítulo, donde explicaremos como definir el comportamiento de determinados operadores para nuestras clases.

print("Hola" \* 3) #HolaHolaHola

* **Operador /**

El operador / divide los números presentes a la izquierda y derecha del operador. Un aspecto importante a tener en cuenta es que si realizamos una división cuyo resultado no es entero (es decimal) podríamos tener problemas. En Python 3 esto no supone un problema porque el mismo se encarga de convertir los números y el resultado que se muestra si es decimal.

print(10/3) #3.3333333333333335

print(1/2) #0.5

Sin embargo, en Python 2, esto hubiera tenido un resultado diferente. El primer ejemplo 10/3=3 y el segundo 1/2=0. El comportamiento realmente sería el de calcular el cociente y no la división.

* **Operador %**

El operador % realiza la operación módulo entre los números presentes a la izquierda y la derecha. Se trata de calcular el resto de la división entera entre ambos números. Es decir, si dividimos 10 entre 3, el cociente sería 3 y el resto 1. Ese resto es lo que calcula el módulo.

print(10%3) # 1

print(10%2) # 0

* **Operador \*\***

El operador \*\* realiza el exponente del número a la izquierda elevado al número de la derecha.

print(10\*\*3) #1000

print(2\*\*2) #4

Si ya has usado alguna vez Python, tal vez hayas oido hablar de la librería math. En esta librería también tenemos una función llamada pow() que es equivalente al operador \*\*.

import math

print(math.pow(10, 3)) #1000.0

* **Operador //**

Por último, el operador // calcula el cociente de la división entre los números que están a su izquierda y derecha.

print(10//3) #3

print(10//10) #1

Tal vez te hayas dado cuenta que el operador cociente // está muy relacionado con el operador módulo %. Volviendo a las lecciones del colegio sobre la división, recordaremos que el Dividendo D es igual al divisor d multiplicado por el cociente c y sumado al resto r, es decir D=d\*c+r. Se puede ver como en el siguiente ejemplo, 10//3 es el cociente y 10%3 es el resto. Al aplicar la fórmula, verificamos que efectivamente 10 era el dividendo.

D = 10 # Número que queremos dividir

d = 3 # Número entre el que queremos dividir

print(3 \* (10//3) + 10%3) # 10

* **Orden de aplicación**

En los ejemplos anteriores simplemente hemos aplicado un operador a dos números sin mezclarlos entre ellos. También es posible tener varios operadores en la misma línea de código, y en este caso es muy importante tener en cuenta las prioridades de cada operador y cual se aplica primero. Ante la duda siempre podemos usar paréntesis, ya que todo lo que está dentro de un paréntesis se evaluará conjuntamente, pero es importante saber las prioridades.

El orden de prioridad sería el siguiente para los operadores aritméticos, siendo el primero el de mayor prioridad:

() Paréntesis

\*\* Exponente

-x Negación

\* / // Multiplicación, División, Cociente, Módulo

+ - Suma, Resta

print(10\*(5+3)) # Con paréntesis se realiza primero la suma

# 80

print(10\*5+3) # Sin paréntesis se realiza primero la multiplicación

# 53

print(3\*3+2/5+5%4) # Primero se multiplica y divide, después se suma

#10.4

print(-2\*\*4) # Primero se hace la potencia, después se aplica el signo #-16

## Operadores de comparación

En Python, los operadores de comparación se utilizan para comparar valores y devolver un resultado booleano (True o False). Los seis operadores de comparación principales son: == (igualdad),!= (desigualdad), > (mayor que), < (menor que), >= (mayor o igual que) y <= (menor o igual que).

* **== (Igualdad):**

Comprueba si dos valores son iguales. Por ejemplo, 5 == 5 devuelve True, mientras que 5 == 6 devuelve False.

* **!= (Desigualdad):**

Comprueba si dos valores no son iguales. Por ejemplo, 5 != 6 devuelve True, mientras que 5 != 5 devuelve False.

* **(Mayor que):**

Comprueba si el valor de la izquierda es mayor que el de la derecha. Por ejemplo, 5 > 3 devuelve True, mientras que 3 > 5 devuelve False.

* **< (Menor que):**

Comprueba si el valor de la izquierda es menor que el de la derecha. Por ejemplo, 3 < 5 devuelve True, mientras que 5 < 3 devuelve False.

* **>= (Mayor o igual que):**

Comprueba si el valor de la izquierda es mayor o igual que el de la derecha. Por ejemplo, 5 >= 5 devuelve True, y 5 >= 3 devuelve True, pero 3 >= 5 devuelve False.

* **<= (Menor o igual que):**

Comprueba si el valor de la izquierda es menor o igual que el de la derecha. Por ejemplo, 3 <= 5 devuelve True, 5 <= 5 devuelve True, pero 5 <= 3 devuelve False.

Ejemplos:

Python

x = 10  
y = 5  
  
print(x == y) *# Output: False*  
print(x != y) *# Output: True*  
print(x > y) *# Output: True*  
print(x < y) *# Output: False*  
print(x >= y) *# Output: True*  
print(x <= y) # Output: False

## Condicionales

En **Python**, los condicionales sirven para que tu programa tome decisiones dependiendo de si una condición es **verdadera** (True) o **falsa** (False).  
Se basan principalmente en la sentencia if, junto con elif y else.

**Estructura básica**

if condicion:

# Bloque de código si la condición es True

elif otra\_condicion:

# Bloque de código si la primera es False y esta es True

else:

# Bloque de código si ninguna condición anterior se cumple

**Ejemplo simple**

edad = 18

if edad >= 18:

print("Eres mayor de edad")

else:

print("Eres menor de edad")

**Ejemplo con varias condiciones**

nota = 85

if nota >= 90:

print("Excelente")

elif nota >= 70:

print("Aprobado")

else:

print("Reprobado")

**Condicional en una sola línea (if ternario)**

edad = 20

mensaje = "Mayor de edad" if edad >= 18 else "Menor de edad"

print(mensaje)

## Operadores logicos

| **Operador** | **Significado** | **Ejemplo (x = 5)** |
| --- | --- | --- |
| and | Todas las condiciones son True | (x > 2 and x < 10) |
| or | Al menos una condición es True | (x < 2 or x > 4) |
| not | Niega la condición | not(x > 4) → False |

## Métodos de cadenas

En Python, los métodos de cadena son funciones que se aplican a objetos de tipo string (cadenas) para realizar diversas operaciones sobre ellos. Estos métodos no modifican la cadena original, sino que devuelven una nueva cadena con los cambios aplicados.

A continuación, se presentan algunos de los métodos de cadena más comunes:

Manipulación de mayúsculas y minúsculas:

* s.lower(): Convierte la cadena a minúsculas.
* s.upper(): Convierte la cadena a mayúsculas.
* s.capitalize(): Convierte el primer carácter de la cadena a mayúscula y el resto a minúsculas.
* s.title(): Convierte la primera letra de cada palabra en mayúscula.
* s.swapcase(): Intercambia mayúsculas por minúsculas y viceversa.

Eliminación de espacios en blanco:

* s.strip(): Elimina los espacios en blanco al principio y al final de la cadena.
* s.lstrip(): Elimina los espacios en blanco al principio de la cadena.
* s.rstrip(): Elimina los espacios en blanco al final de la cadena.

Búsqueda y reemplazo:

* s.find(sub): Busca la primera aparición de la subcadena sub en la cadena y devuelve su índice (o -1 si no se encuentra).
* s.rfind(sub): Busca la última aparición de la subcadena sub en la cadena y devuelve su índice (o -1 si no se encuentra).
* s.index(sub): Similar a find(), pero lanza una excepción ValueError si la subcadena no se encuentra.
* s.rindex(sub): Similar a rfind(), pero lanza una excepción ValueError si la subcadena no se encuentra.
* s.replace(old, new[, count]): Reemplaza todas las ocurrencias de la subcadena old por la subcadena new. El parámetro count (opcional) limita el número de reemplazos.

Verificación del contenido de la cadena:

* s.isalpha(): Devuelve True si la cadena solo contiene caracteres alfabéticos.
* s.isdigit(): Devuelve True si la cadena solo contiene dígitos.
* s.isalnum(): Devuelve True si la cadena solo contiene caracteres alfanuméricos.
* s.isspace(): Devuelve True si la cadena solo contiene espacios en blanco.
* s.islower(): Devuelve True si la cadena contiene solo caracteres en minúsculas.
* s.isupper(): Devuelve True si la cadena contiene solo caracteres en mayúsculas.
* s.istitle(): Devuelve True si la cadena tiene formato de título (primera letra de cada palabra en mayúscula).
* s.startswith(prefix): Devuelve True si la cadena comienza con el prefijo prefix.
* s.endswith(suffix): Devuelve True si la cadena termina con el sufijo suffix.

División y unión:

* s.split([sep[, maxsplit]]): Divide la cadena en una lista de subcadenas, utilizando el separador sep (por defecto, espacios en blanco). El parámetro maxsplit (opcional) limita el número de divisiones.
* s.join(iterable): Une los elementos de un iterable (por ejemplo, una lista) en una cadena, utilizando la cadena como separador.

Otros métodos:

* s.count(sub[, start[, end]]): Cuenta el número de ocurrencias de la subcadena sub en la cadena.
* s.encode(encoding='utf-8', errors='strict'): Codifica la cadena en una secuencia de bytes utilizando la codificación especificada.
* s.format(\*args, \*\*kwargs): Da formato a la cadena utilizando los argumentos especificados.
* s.zfill(width): Rellena la cadena con ceros a la izquierda hasta alcanzar la longitud especificada.

## Métodos de listas

En Python, las listas son estructuras de datos dinámicas y mutables que pueden almacenar elementos de diferentes tipos. Cuentan con varios métodos integrados para manipularlas, como agregar, eliminar, ordenar y buscar elementos.

A continuación, se describen algunos de los métodos más comunes para listas en Python:

[**Métodos para modificar la lista**](https://www.google.com/search?sca_esv=3dbb46daba6132fc&cs=0&sxsrf=AE3TifM1RYa13GmRu0kMTeiHXmzR4GcSVw%3A1755397816010&q=M%C3%A9todos+para+modificar+la+lista&sa=X&ved=2ahUKEwjopOC55pCPAxXqRDABHZfFFLsQxccNegQIDhAC&mstk=AUtExfABqbPCuLPhrarJAp2ef4CmdrUObqQAE8TU1wDLo76Ikukj1BjNdFAocHiL8gBtH5QEHeU-x9KYKuMF6GyOAL7CvXzfu9SqTr6uXM1-6_e1F3Ff721uibFkHBoH7FnYlvvk-rwfwobWHYa11FyFxicYVpKa2qtRWNWW-dFoKEv_6yk&csui=3)**:**

* append(x): Agrega un elemento x al final de la lista.
* insert(i, x): Inserta un elemento x en la posición i de la lista. Los elementos existentes sedesplazan hacia la derecha.
* extend(iterable): Extiende la lista agregando todos los elementos de un iterable (como otra lista) al final.
* remove(x): Elimina la primera ocurrencia de un elemento x en la lista. Si el elemento no existe, se lanza una excepción ValueError.
* pop([i]): Elimina el elemento en la posición i y lo devuelve. Si no se especifica i, elimina y devuelve el último elemento.
* clear(): Elimina todos los elementos de la lista, dejando la lista vacía. [].

[**Métodos para consultar la lista**](https://www.google.com/search?sca_esv=3dbb46daba6132fc&cs=0&sxsrf=AE3TifM1RYa13GmRu0kMTeiHXmzR4GcSVw%3A1755397816010&q=M%C3%A9todos+para+consultar+la+lista&sa=X&ved=2ahUKEwjopOC55pCPAxXqRDABHZfFFLsQxccNegQIOxAB&mstk=AUtExfABqbPCuLPhrarJAp2ef4CmdrUObqQAE8TU1wDLo76Ikukj1BjNdFAocHiL8gBtH5QEHeU-x9KYKuMF6GyOAL7CvXzfu9SqTr6uXM1-6_e1F3Ff721uibFkHBoH7FnYlvvk-rwfwobWHYa11FyFxicYVpKa2qtRWNWW-dFoKEv_6yk&csui=3)**:**

* index(x[, inicio[, fin]]): Devuelve el índice de la primera ocurrencia de un elemento x en la lista. Se puede especificar un rango de búsqueda con los parámetros inicio y fin. Lanza una excepción ValueError si el elemento no se encuentra.
* count(x): Devuelve el número de veces que un elemento x aparece en la lista.
* len(lista): Devuelve la longitud de la lista (número de elementos).

[**Métodos para ordenar y manipular el orden de la lista**](https://www.google.com/search?sca_esv=3dbb46daba6132fc&cs=0&sxsrf=AE3TifM1RYa13GmRu0kMTeiHXmzR4GcSVw%3A1755397816010&q=M%C3%A9todos+para+ordenar+y+manipular+el+orden+de+la+lista&sa=X&ved=2ahUKEwjopOC55pCPAxXqRDABHZfFFLsQxccNegQIShAB&mstk=AUtExfABqbPCuLPhrarJAp2ef4CmdrUObqQAE8TU1wDLo76Ikukj1BjNdFAocHiL8gBtH5QEHeU-x9KYKuMF6GyOAL7CvXzfu9SqTr6uXM1-6_e1F3Ff721uibFkHBoH7FnYlvvk-rwfwobWHYa11FyFxicYVpKa2qtRWNWW-dFoKEv_6yk&csui=3)**:**

* sort(key=None, reverse=False): Ordena los elementos de la lista in-place (modifica la lista original). El parámetro key permite especificar una función de comparación. El parámetro reverse indica si se debe ordenar de forma descendente.
* reverse(): Invierte el orden de los elementos de la lista in-place.
* copy(): Devuelve una copia superficial de la lista.

Ejemplos:

Python

mi\_lista = [1, 2, 3]  
mi\_lista.append(4) *# [1, 2, 3, 4]*  
mi\_lista.insert(1, 5) *# [1, 5, 2, 3, 4]*  
mi\_lista.extend([6, 7]) *# [1, 5, 2, 3, 4, 6, 7]*  
mi\_lista.remove(2) *# [1, 5, 3, 4, 6, 7]*  
mi\_lista.pop(0) *# Devuelve 1, lista ahora es [5, 3, 4, 6, 7]*  
mi\_lista.clear() # []

Además de estos métodos, las listas en Python también se pueden manipular utilizando operadores como + (concatenación) y \* (repetición). También es posible acceder a elementos individuales o sublistas mediante el uso de índices y rebanadas (slicing).