



Cofinanciado por  
la Unión Europea



Fondos Europeos

## Python.

### 1. Entrada y salida por teclado

Input

Print

sys.argv

### 2. Strings

operaciones básicas

Métodos útiles

Formateo de Strings

Verificación

### 3. Caracteres especiales

### 4. Arrays

### 5. Estructuras de datos

Listas

Diccionarios

Tuplas

Conjuntos

### 6. Rangos

### Anexo 1. Inmutabilidad



REGISTRO NACIONAL DE ASOCIACIONES N°611922  
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL  
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA  
C/DOS ACERAS 23, 29012  
MÁLAGA | (+34) 952 300 500  
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ  
TR.<sup>a</sup> ALAMEDA DE SOLANO, 32, 11130  
CHICLANA DE LA FRONTERA | (+34) 956 900 312  
CHICLANA@ARRABALEMPLEO.ORG



## 1. Python entrada y salida por teclado

### Entrada de datos `input()`

Para recibir datos del usuario, usamos la función `input()`. Devuelve siempre una cadena (`str`), por lo que si necesitamos otro tipo de dato (como `int` o `float`), debemos convertirlo.

```
nombre = input("Ingresa tu nombre: ") # Captura una cadena
edad = int(input("Ingresa tu edad: ")) # Convierte la entrada a entero
altura = float(input("Ingresa tu altura en metros: ")) # Convierte a flotante

print(f"Hola {nombre}, tienes {edad} años y mides {altura} metros.")
```

### Salida de datos `print()`

La función `print()` muestra información en la consola. Podemos concatenar valores con `+`, usar `,` o emplear *f-strings* para formatear mejor el texto.

```
nombre = "Ana"
edad = 25

print("Hola", nombre, "tienes", edad, "años.") # Separación con espacios automáticos
print("Hola " + nombre + ", tienes " + str(edad) + " años.") # Concatenación con +
print(f"Hola {nombre}, tienes {edad} años.") # Uso de f-string (recomendado)
```

### Resumen

- Usamos `input()` para leer datos del usuario (siempre como `str`).
- Convertimos los datos con `int()`, `float()`, etc., si es necesario.
- Usamos `print()` para mostrar información, con `+`, `,` o *f-strings* para formateo.





## sys.argv

En Python, **sys.argv** es una lista que contiene los **argumentos de la línea de comandos** que se pasan a un script de Python cuando se ejecuta desde la terminal o línea de comandos.

### ¿Qué es sys.argv?

- **sys.argv** es parte del módulo **sys** de Python, que proporciona acceso a algunas variables y funciones que interactúan con el entorno de ejecución del programa.
- **sys.argv** es una lista donde el primer elemento (`sys.argv[0]`) es el nombre del script Python que se está ejecutando, y los elementos posteriores (`sys.argv[1:]`) son los **argumentos** que se pasan al script desde la línea de comandos.

### Sintaxis de uso:

```
import sys

# Mostrar Los argumentos de La Línea de comandos
print(sys.argv)
```

### Ejemplo:

Supongamos que tienes un archivo llamado `mi_script.py` y lo ejecutas desde la terminal como sigue:

```
python mi_script.py arg1 arg2 arg3
```

En este caso, `sys.argv` será:

<pre>import sys  print(sys.argv)</pre>	Salida: <pre>['mi_script.py', 'arg1', 'arg2', 'arg3']</pre>
--	--

- `sys.argv[0]`: '`mi_script.py`' — El nombre del script.
- `sys.argv[1]`: '`arg1`' — Primer argumento.
- `sys.argv[2]`: '`arg2`' — Segundo argumento.
- `sys.argv[3]`: '`arg3`' — Tercer argumento.





## Cómo usar sys.argv en un script

Los elementos en **sys.argv** se pueden usar dentro de tu script para modificar el comportamiento del programa según los argumentos que se pasen desde la línea de comandos.

### Ejemplo de script usando sys.argv:

```
import sys

if len(sys.argv) < 3:
    print("Por favor, ingresa al menos dos argumentos.")
else:
    arg1 = sys.argv[1]
    arg2 = sys.argv[2]
    print(f"El primer argumento es: {arg1}")
    print(f"El segundo argumento es: {arg2}")
```

### Ejecución:

```
python mi_script.py hola mundo
```

### Salida:

```
El primer argumento es: hola
El segundo argumento es: mundo
```

### Puntos clave:

- **sys.argv[0]** es el nombre del script Python.
- Los **argumentos adicionales** se encuentran en **sys.argv[1:]**.
- **sys.argv** permite que tu script reciba información externa y cambie su comportamiento según los argumentos proporcionados.

### Consideraciones:

- Los elementos de **sys.argv** son **cadenas de texto**, por lo que si esperas un valor numérico o de otro tipo, necesitarás convertirlo explícitamente (por ejemplo, con `int()` o `float()`).





Cofinanciado por  
la Unión Europea



## Ejemplo de conversión:

```
import sys

# Supongamos que el segundo argumento es un número
numero = int(sys.argv[1]) # Convertir de string a entero
print(f"El número ingresado es: {numero}")
```

## Resumen:

- **sys.argv** es una lista que contiene los argumentos que se pasan a un script de Python desde la línea de comandos.
- **sys.argv[0]** es siempre el nombre del script.
- Los **argumentos adicionales** se encuentran en **sys.argv[1:]**.

Es una herramienta muy útil para **personalizar** el comportamiento de un script en función de lo que el usuario ingrese en la línea de comandos.



REGISTRO NACIONAL DE ASOCIACIONES N°611922  
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL  
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA  
C/DOS ACERAS 23, 29012  
MÁLAGA | (+34) 952 300 500  
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ  
TR.ª ALAMEDA DE SOLANO, 32, 11130  
CHICLANA DE LA FRONTERA | (+34) 956 900 312  
CHICLANA@ARRABALEMPLEO.ORG



## 2.Cadenas (Strings) en Python

En Python, los **strings** son secuencias de caracteres **inmutables**. Se definen con comillas simples ('), dobles ("") o triples (""" """" para cadenas multilínea).

```
cadena1 = 'Hola'  
cadena2 = "Mundo"  
cadena3 = '''Este es un  
string multilínea.'''
```

Cuando decimos que un **string** en Python es **inmutable**, significa que **no se puede modificar después de su creación**. Ver anexo 1 Inmutabilidad.

Por ejemplo, si intentamos cambiar un carácter en una cadena, obtendremos un error:

```
texto = "Hola"  
texto[0] = "M" # ✗ Error: TypeError: 'str' object does not support item assignment
```

### ¿Cómo trabajar con strings si son inmutables?

Si necesitas modificar un string, debes crear uno nuevo en lugar de cambiar el original:

```
texto = "Hola"  
nuevo_texto = "M" + texto[1:] # Se crea un nuevo string  
print(nuevo_texto) # "Mola"
```

Aquí, "M" + texto[1:] genera un **nuevo string** sin modificar el original.

- ◊ Si necesitas una estructura modificable, puedes usar **listas**, que sí permiten cambiar sus elementos:

```
lista = list("Hola") # Convertimos el string en lista  
lista[0] = "M" # Modificamos la primera letra  
texto_modificado = "".join(lista) # Volvemos a convertirlo en string  
print(texto_modificado) # "Mola"
```





◇ En resumen:

- **Inmutable** = No puedes cambiar un string después de crearlo.
- Para "modificarlo", debes crear un **nuevo string** con los cambios.
- Si necesitas algo modificable, usa **listas**.

## Operaciones Básicas con Strings

Operación	Ejemplo	Resultado
Concatenación (+)	"Hola" + " Mundo"	"Hola Mundo"
Repetición (*)	"Hola" * 3	"HolaHolaHola"
Índices ([])	"Python"[0]	'P'
Subcadenas ([:])	"Python"[0:3]	'Pyt'
Longitud (len())	len("Hola")	4

```
s = "Python"
print(s[0])      # 'P' (primer carácter)
print(s[-1])     # 'n' (último carácter)
print(s[1:4])    # 'yth' (subcadena)
print(len(s))   # 6 (Longitud)
```

## Métodos Útiles de Strings

Método	Descripción	Ejemplo	Resultado
upper()	Convierte a mayúsculas	"hola".upper()	"HOLA"
lower()	Convierte a minúsculas	"HOLA".lower()	"hola"
title()	Capitaliza cada palabra	"hola mundo".title()	"Hola Mundo"
strip()	Elimina espacios extra	" hola ".strip()	"hola"
replace(old, new)	Reemplaza texto	"Python".replace("P", "J")	"Jython"
split(sep)	Divide en lista	"a,b,c".split(",")	['a', 'b', 'c']
join(iterable)	Une elementos con separador	" ".join(["Hola", "Mundo"])	"Hola Mundo"

```
texto = " Hola Mundo "
print(texto.strip())  # "Hola Mundo"
print(texto.lower())  # " hola mundo "
print(texto.replace("Mundo", "Python")) # " Hola Python "
```





## Formateo de Strings

### ◊ Usando f-strings (recomendado desde Python 3.6)

```
nombre = "Ana"
edad = 25
print(f"Hola, mi nombre es {nombre} y tengo {edad} años.")
```

### ◊ Usando .format()

```
print("Hola, mi nombre es {} y tengo {} años.".format(nombre, edad))
```

### ◊ Usando % (obsoleto pero aún válido)

```
print("Hola, mi nombre es %s y tengo %d años." % (nombre, edad))
```

## Verificación de Strings

Método	Ejemplo	Resultado
startswith(prefijo)	"Python".startswith("Py")	True
endswith(sufijo)	"Python".endswith("on")	True
isdigit()	"123".isdigit()	True
isalpha()	"Python".isalpha()	True
isalnum()	"Python123".isalnum()	True

```
cadena = "12345"
print(cadena.isdigit()) # True
print("Python".isalpha()) # True
print("Python123".isalnum()) # True
```





Cofinanciado por  
la Unión Europea



## Strings como Iterables

Podemos recorrer un string con un for:

```
for letra in "Python":  
    print(letra)
```

**Salida**

P  
y  
t  
h  
o  
n

### Resumen

- Los **strings** son inmutables y pueden manipularse con índices y slicing.
- Se pueden concatenar (+), repetir (\*), y formatear (f-strings).
- Hay muchos métodos útiles como .upper(), .replace(), .split(), .join().
- Podemos verificar su contenido con .isdigit(), .isalpha(), etc.
- Se pueden recorrer como listas con for.



REGISTRO NACIONAL DE ASOCIACIONES N°611922  
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL  
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA  
C/DOS ACERAS 23, 29012  
MÁLAGA | (+34) 952 300 500  
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ  
TR.<sup>a</sup> ALAMEDA DE SOLANO, 32, 11130  
CHICLANA DE LA FRONTERA | (+34) 956 900 312  
CHICLANA@ARRABALEMPLEO.ORG



### 3. Caracteres especiales en Python

Los caracteres especiales (o secuencias de escape) en Python se utilizan dentro de strings para representar símbolos o acciones que no pueden escribirse directamente.

#### Principales caracteres especiales en Python

Carácter	Descripción	Ejemplo	Salida
\n	Salto de línea	"Hola\nMundo"	Hola Mundo (en dos líneas)
\t	Tabulación (espacio grande)	"Hola\tMundo"	Hola Mundo
'	Comilla simple dentro de un string	"Es una \'prueba\'"	Es una 'prueba'
"	Comilla doble dentro de un string	"Dijo: \"Hola\""	Dijo: "Hola"
\\"	Barra invertida \	"C:\\Users\\Juan"	C:\Users\Juan
\r	Retorno de carro (cursor al inicio)	"Hola\rMundo"	Mundo (reemplaza "Hola")
\b	Retroceso (elimina un carácter)	"Hola\b Mundo"	Hol Mundo
\f	Salto de página (poco usado)	"Hola\fMundo"	Hola + (salto de página) + Mundo
\v	Tabulación vertical	"Hola\vMundo"	Hola + (salto vertical) + Mundo
\uXXXX	Unicode de 4 dígitos	"\u2764"	❤ (corazón)
\UXXXXXXXXX	Unicode de 8 dígitos	"\U0001F600"	😊 (emoji de cara feliz)
\xXX	Carácter ASCII en hexadecimal	"\x48\x6F\x6C\x61"	Hola



## 4.Arrays en Python vs Arrays en Java

En Python, [no existe un tipo de dato específico llamado "array"](#) como en otros lenguajes (Java). En su lugar, se utilizan estructuras como **listas (list)**, la **librería array**.

La **librería array en Python** y los **arrays en Java** funcionan de manera muy similar en el sentido de que ambos requieren que todos los elementos sean del **mismo tipo**. La diferencia clave es cómo están definidos en cada lenguaje.

### Python (Librería array)

En Python, el módulo array permite crear arrays que **solo** pueden contener un tipo de dato específico. Al definir un array, debes indicar el tipo de dato de los elementos con un **código tipo**.

```
import array

# Array de enteros
numeros = array.array('i', [1, 2, 3, 4, 5])
print(numeros) # array('i', [1, 2, 3, 4, 5])
```

'i' es el código que indica que el array contiene **enteros**.

### Diferencias clave:

- **Python:** La librería array es opcional y no tan común. Las listas (list) en Python son más flexibles porque pueden contener diferentes tipos de datos.
- **Java:** Los arrays son una estructura de datos fundamental del lenguaje, y todos los elementos deben ser homogéneos en cuanto a tipo (sin necesidad de una librería adicional).
- Tanto los arrays de Python (usando la librería array) como los arrays en Java requieren que sus elementos sean del mismo tipo, pero **Java los maneja de forma más estricta** como una parte fundamental del lenguaje, mientras que en Python se hace mediante una librería adicional.





## 5. Estructuras de datos en Python

### 1. Listas (list)

Las **listas** son colecciones ordenadas y mutables que pueden contener elementos de cualquier tipo (enteros, cadenas, objetos, etc.).

#### Características:

- **Ordenadas:** Los elementos tienen un índice, lo que significa que se mantienen en un orden específico.
- **Mutables:** Puedes cambiar, agregar o eliminar elementos.
- **Pueden contener elementos de diferentes tipos.**

#### Ejemplo:

```
mi_lista = [1, 2, 3, "hola", 5.5]
mi_lista.append(6) # Agregar un elemento
mi_lista[2] = "nuevo valor" # Modificar un elemento
print(mi_lista) # [1, 2, 'nuevo valor', 'hola', 5.5, 6]
```

### 2. Diccionarios (dict)

Los **diccionarios** son colecciones desordenadas de **pares clave-valor**. Cada **clave** es **única** y se asocia con un valor.

#### Características:

- **Desordenados** (en versiones anteriores a Python 3.7, pero a partir de Python 3.7 se mantiene el orden de inserción).
- **Mutables:** Puedes modificar, agregar y eliminar elementos.
- **Claves únicas:** No se permiten claves duplicadas.
- **Pueden contener cualquier tipo de dato como clave o valor.**

#### Ejemplo:

```
mi_diccionario = {"nombre": "Juan", "edad": 30, "ciudad": "Madrid"}
mi_diccionario["edad"] = 31 # Modificar valor
mi_diccionario["ocupacion"] = "Ingeniero" # Agregar nueva clave-valor
print(mi_diccionario) # {'nombre': 'Juan', 'edad': 31, 'ciudad': 'Madrid', 'ocupacion': 'Ingeniero'}
```





### 3. Tuplas (tuple)

Las **tuplas** son colecciones ordenadas e inmutables. Una vez que se crea una tupla, no se puede modificar (no se pueden agregar, eliminar o cambiar elementos).

#### Características:

- **Ordenadas:** Los elementos tienen un índice.
- **Inmutables:** No se pueden cambiar después de la creación.
- **Pueden contener diferentes tipos de datos.**

#### Ejemplo:

```
mi_tupla = (1, 2, 3, "hola")
print(mi_tupla[1]) # 2
# mi_tupla[1] = 4 # Esto da error, porque las tuplas son inmutables
```

### 4. Conjuntos (set)

Los **conjuntos** son colecciones desordenadas de elementos únicos, lo que significa que no permiten duplicados.

#### Características:

- **Desordenados:** No mantienen el orden de los elementos.
- **Elementos únicos:** No pueden tener duplicados.
- **Mutables:** Se pueden agregar y eliminar elementos.

#### Ejemplo:

```
mi_conjunto = {1, 2, 3, 4, 5}
mi_conjunto.add(6) # Agregar un elemento
mi_conjunto.remove(4) # Eliminar un elemento
print(mi_conjunto) # {1, 2, 3, 5, 6}
```



## Operaciones habituales.

Operación	Listas (list)	Conjuntos (set)	Diccionarios (dict)	Tuplas (tuple)
<b>Crear</b>	mi_lista = [1, 2, 3]	mi_set = {1, 2, 3}	mi_dict = {"clave1": "valor1", "clave2": "valor2"}	mi_tupla = (1, 2, 3)
<b>Acceder a un elemento</b>	mi_lista[0] → 1	No se puede acceder a elementos por índice	mi_dict["clave1"] → "valor1"	mi_tupla[0] → 1
<b>Modificar un elemento</b>	mi_lista[1] = 99	No aplicable (set no tiene índice)	mi_dict["clave1"] = "nuevo_valor"	No aplicable (inmutable)
<b>Agregar un elemento</b>	mi_lista.append(4)	mi_set.add(4)	mi_dict["clave3"] = "valor3"	No aplicable (inmutable)
<b>Eliminar un elemento</b>	mi_lista.remove(2) (si existe)	mi_set.remove(2) (si existe)	del mi_dict["clave1"]	No aplicable (inmutable)
<b>Eliminar el último elemento</b>	mi_lista.pop()	No aplicable (set no tiene orden)	No aplicable (diccionarios no tienen orden)	No aplicable (inmutable)
<b>Comprobar existencia</b>	2 in mi_lista → True	2 in mi_set → True	"clave1" in mi_dict → True	2 in mi_tupla → True
<b>Longitud</b>	len(mi_lista) → 3	len(mi_set) → 3	len(mi_dict) → 2	len(mi_tupla) → 3
<b>Concatenar</b>	mi_lista + [4, 5]	No aplicable (set no mantiene orden)	No aplicable (diccionario no mantiene orden)	mi_tupla + (4, 5)
<b>Iterar sobre elementos</b>	for item in mi_lista:	for item in mi_set:	for key, value in mi_dict.items():	for item in mi_tupla:
<b>Convertir a otro tipo</b>	list(mi_set) → Lista	set(mi_lista) → Set	dict([("clave1", "valor1"), ("clave2", "valor2")])	tuple(mi_lista) → Tupla
<b>Vaciar</b>	mi_lista.clear()	mi_set.clear()	mi_dict.clear()	No aplicable (inmutable)

## Resumen:

- Las **tuplas** son útiles cuando necesitas almacenar un conjunto de datos **constantes**, es decir, que no deben cambiar a lo largo del tiempo.
- Listas y diccionarios** son más flexibles, ya que puedes **modificarlas** con facilidad.
- Conjuntos** son útiles cuando necesitas **elementos únicos** sin importar el orden.

## Comparativa rápida de estructuras de datos en Python

Estructura	Ordenada	Mutabilidad	Tipos de datos	Ejemplo
Listas	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Mutable	Cualquier tipo	[1, 2, 3]
Diccionarios	<input checked="" type="checkbox"/> Sí (a partir de Python 3.7)	<input checked="" type="checkbox"/> Mutable	Clave: cualquier tipo, valor: cualquier tipo	{"clave": "valor"}
Tuplas	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Inmutable	Cualquier tipo	(1, 2, 3)
Conjuntos	<input checked="" type="checkbox"/> No	<input checked="" type="checkbox"/> Mutable	Elementos únicos	{1, 2, 3}

- Listas y Diccionarios** son muy flexibles y se usan mucho, ya que permiten almacenar datos de diferentes tipos.
- Tuplas** son útiles cuando necesitas datos que no cambian (inmutables).
- Conjuntos** son buenos para almacenar elementos únicos sin importar el orden.



## 6. Rangos

En Python, el tipo **range** no es exactamente un tipo de datos como **listas** o **tuplas**, sino más bien un tipo de **objeto iterable**. El objeto range se utiliza comúnmente para generar secuencias de números en un rango especificado, y suele ser utilizado en bucles for para iterar sobre una secuencia de números.

### ¿Qué es un range?

El **range** en Python genera una secuencia de números enteros, que se puede utilizar para iterar en bucles o para crear otras estructuras de datos (como listas). A diferencia de las **listas** o **tuplas**, los **objetos range no almacenan todos los valores en memoria**; en cambio, generan los números de forma perezosa (es decir, bajo demanda).

### Sintaxis de range

La sintaxis básica de range es:

```
range(start, stop, step)
```

- **start** (opcional): El valor inicial de la secuencia (por defecto es 0).
- **stop**: El valor final (exclusivo, es decir, no se incluye en la secuencia).
- **step** (opcional): La diferencia entre los valores consecutivos (por defecto es 1).

### Ejemplos:

<b>Rango básico</b> (sin parámetros adicionales):	<b>Rango con start y stop:</b>
<pre>r = range(5) print(list(r)) # [0, 1, 2, 3, 4]</pre>	<pre>r = range(2, 6) print(list(r)) # [2, 3, 4, 5]</pre>
<b>Rango con start, stop y step:</b>	<b>Rango con valores negativos:</b>
<pre>r = range(1, 10, 2) print(list(r)) # [1, 3, 5, 7, 9]</pre>	<pre>r = range(10, 0, -2) print(list(r)) # [10, 8, 6, 4, 2]</pre>





## Características de range:

- **Inmutable:** No puedes cambiar los valores de un objeto range después de haberlo creado.
- **No almacena todos los valores en memoria:** Los valores se generan sobre la marcha, lo que lo hace eficiente en cuanto al uso de memoria, incluso para rangos grandes.
- **Puede ser convertido a otras estructuras de datos:** Como **listas, tuplas o conjuntos.**

## ¿Es range un tipo de datos?

Aunque **range** no es un tipo de datos tradicional (como un **int** o **str**), es un **objeto iterable** en Python que **comporta muchas propiedades** similares a otros tipos de datos. Técnicamente, range es una clase interna de Python que implementa un iterable que produce una secuencia de números en el rango especificado.

Puedes verificar el tipo de un objeto range usando la función `type()`:

```
r = range(5)
print(type(r)) # <class 'range'>
```

## Uso común de range en bucles:

El uso más común de range es en un bucle for para repetir una acción un número determinado de veces.

### Ejemplo con range en un bucle for:

```
for i in range(5):
    print(i)
```

Salida:

```
0
1
2
3
4
```

## Resumen:

- El **range** no es un tipo de datos tradicional, sino un **objeto iterable** que genera una secuencia de números.
- Es muy eficiente en términos de memoria, ya que no almacena todos los números de la secuencia.
- Se usa comúnmente en bucles for para iterar un número de veces.



REGISTRO NACIONAL DE ASOCIACIONES N°611922  
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL  
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA  
C/DOS ACERAS 23, 29012  
MÁLAGA | (+34) 952 300 500  
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ  
TR.<sup>a</sup> ALAMEDA DE SOLANO, 32, 11130  
CHICLANA DE LA FRONTERA | (+34) 956 900 312  
CHICLANA@ARRABALEMPLEO.ORG



## Anexo 1. Inmutabilidad.

Tabla de tipos de datos mutables e inmutables en Python

Tipo de dato	Mutable	Ejemplo	Notas
<b>Tipos básicos</b>			
int (Entero)	✗ No	x = 5	Cualquier operación crea un nuevo objeto.
float (Flotante)	✗ No	y = 3.14	Similar a los enteros, no se puede modificar.
complex (Números complejos)	✗ No	z = 1 + 2j	Inmutable como int y float.
bool (Booleano)	✗ No	b = True	True y False son constantes inmutables.
str (Cadena)	✗ No	s = "Hola"	No se pueden modificar caracteres individuales.
<b>Tipos compuestos</b>			
list (Lista)	✓ Sí	l = [1, 2, 3]	Se pueden modificar elementos, añadir o eliminar.
dict (Diccionario)	✓ Sí	d = {"a": 1, "b": 2}	Se pueden modificar claves y valores.
set (Conjunto)	✓ Sí	s = {1, 2, 3}	Se pueden añadir o eliminar elementos.
tuple (Tupla)	✗ No	t = (1, 2, 3)	No se pueden cambiar los elementos.

### Resumen

**Mutables:** list, dict, set.

**Inmutables:** int, float, bool, str, tuple, range.

Los **tipos mutables** pueden cambiar su contenido sin cambiar su referencia en memoria.

Los **tipos inmutables** no pueden modificarse una vez creados; cualquier "modificación" genera un nuevo objeto.

La **mutabilidad** es un concepto más técnico, pero lo que realmente importa es si pueden cambiar o no el contenido de una variable después de crearla.





## Tipos de datos en Python: ¿Se pueden modificar o no?

Tipo de dato	¿Se puede cambiar su contenido después de crearlo?	Ejemplo
int (Entero)	<input checked="" type="checkbox"/> Sí, pero solo reasignando la variable.	$x = 5 \rightarrow x = 10$
float (Flotante)	<input checked="" type="checkbox"/> Sí, pero solo reasignando la variable.	$y = 3.14 \rightarrow y = 2.71$
bool (Booleano)	<input checked="" type="checkbox"/> Sí, pero solo reasignando la variable.	$b = \text{True} \rightarrow b = \text{False}$
str (Cadena de texto)	<input checked="" type="checkbox"/> No, hay que crear una nueva.	$s = "hola" \rightarrow s = "mundo"$ (nueva cadena)
tuple (Tupla)	<input checked="" type="checkbox"/> No, hay que crear una nueva.	$t = (1, 2, 3) \rightarrow t = (4, 5, 6)$
list (Lista)	<input checked="" type="checkbox"/> Sí, se pueden cambiar elementos.	$l = [1, 2, 3] \rightarrow l[0] = 100$
dict (Diccionario)	<input checked="" type="checkbox"/> Sí, se pueden cambiar valores y agregar claves.	$d["clave"] = "nuevo valor"$
set (Conjunto)	<input checked="" type="checkbox"/> Sí, se pueden agregar y eliminar elementos.	$s.add(4), s.remove(2)$

### Explicación

- Pueden cambiar su valor con una nueva asignación:** int, float, bool, str, tuple  
→ Pero realmente están creando un nuevo valor.
- Pueden modificar su contenido directamente:** list, dict, set, → No necesitan reasignarse.

Ejemplo con str (cadena), int (entero) y list (lista) para ver la diferencia en cómo se modifican o no.

### Ejemplo con str (Cadena) – No se puede modificar, hay que crear una nueva

```
texto = "Hola"
texto[0] = "M" # ✗ Esto da error porque Las cadenas no pueden modificarse.

# Solución: Crear una nueva cadena
texto = "M" + texto[1:] # ✓ Se genera una nueva cadena
print(texto) # "Mola"
```





Cofinanciado por  
la Unión Europea



MINISTERIO  
DE TRABAJO  
Y ECONOMÍA SOCIAL



Fondos Europeos



Junta de Andalucía  
Instituto Andaluz del  
Deporte e Igualdad  
INSTITUTO ANDALUZ DEL DEPORTE



Ayuda  
en Acción  
Impulsa Empleo Joven

- ◇ **Explicación:** No puedes cambiar letras en un string, solo puedes crear una nueva cadena con los cambios.

### Ejemplo con int (Entero) – No se modifica, solo se reasigna

```
x = 5
x = 10 # ✅ Se cambia el valor de x
print(x) # 10
```

- ◇ **Explicación:** No es que 5 cambie a 10, sino que x deja de referirse a 5 y ahora apunta a 10.

### Ejemplo con list (Lista) – Se puede modificar sin reasignar

```
numeros = [1, 2, 3]
numeros[0] = 99 # ✅ Se modifica el primer elemento
print(numeros) # [99, 2, 3]
```

- ◇ **Explicación:** En una lista, podemos cambiar elementos sin crear una nueva lista.

### Resumen

- str (cadena) ❌ No se puede modificar, hay que crear una nueva.
- int (entero) ✅ Se puede cambiar su valor con una nueva asignación.
- list (lista) ✅ Se puede modificar su contenido sin reasignar.



REGISTRO NACIONAL DE ASOCIACIONES N°611922  
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL  
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA  
C/DOS ACERAS 23, 29012  
MÁLAGA | (+34) 952 300 500  
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ  
TR.ª ALAMEDA DE SOLANO, 32, 11130  
CHICLANA DE LA FRONTERA | (+34) 956 900 312  
CHICLANA@ARRABALEMPLEO.ORG