

# Tipos de Datos



# Numéricos

---

Python puede manejar diferentes **tipos** de valores **numéricos**, algunas de las principales son los números **enteros (int)**, los números **reales (float)** y los números **complejos (complex)**.

```
[>>> entero = 4  
[>>> flotante = 3.14  
[>>> imaginario = 4+2j
```

# Numéricos

---

Se cuenta con diferentes **operadores** para poder manejar estos números, algunas de ellos son los siguientes:

+	$a+b$	sumar los números
-	$a-b$	restar los números
*	$a*b$	multiplicar los números
/	$a/b$	dividir los números
//	$a//b$	dividir los números y redondear hacia abajo

# Numéricos

— — —

%	<code>a%b</code>	regresar el residuo de $a/b$
---	------------------	------------------------------

**	<code>a**b</code>	$a$ a la $b$
----	-------------------	--------------

abs	<code>abs(a)</code>	el absoluto de $a$
-----	---------------------	--------------------

round	<code>round(a)</code>	redondeo de $a$
-------	-----------------------	-----------------

# Booleanos

---

El tipo de dato booleano en Python denota si algo es **verdadero o falso**.

Veremos que las palabras reservadas **True y False** son las principales formas para esto, sin embargo no las únicas.

```
[>>> a = True  
[>>> b = False  
[>>> a  
True
```

# Booleanos

---

Los **operadores relacionales** son un tipo de operador que nos sirve para comparar dos cosas. Estos son:

operador	comparación
==	es igual que
!=	es distinto de
<	es menor que
<=	es menor o igual que
>	es mayor que
>=	es mayor o igual que

# Booleanos

---

En el ejemplo siguiente podemos ver como los operadores relacionales nos devuelven un booleano. Ya sea True o False.

```
[>>> 4 < 2
False
>>> 5 > 1
True
>>> 3 == 2
False
>>> 3 != 2
True
>>> "HOLA" == 'HOLA'
True
```

# Booleanos

---

En Python , **otros tipos de datos** pueden ser interpretados como True o False.

Lo siguiente se puede interpretar como False:

- El número 0
- Una cadena vacía
- Una lista vacía
- Una tupla vacía
- Un diccionario vacío
- None



# Booleanos

---

Podemos realizar un cierto tipo de **operaciones** especiales con los **booleanos**. Estos son los **operadores lógicos**.

Los principales son **and** y **or**.

El operador **and** retorna True sólo cuando **todos** los valores son **True**.

El operador **or** retorna True cuando **por lo menos uno** de los valores es **True**.

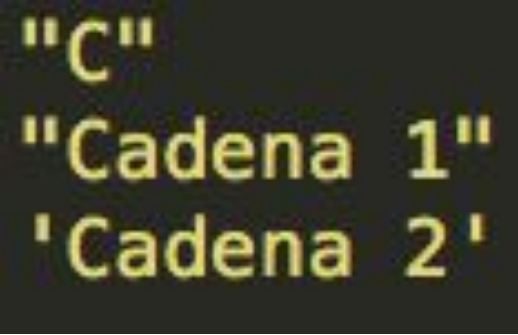
```
[>>> True and False  
False  
[>>> True and True  
True  
[>>> True or False  
True  
[>>> False or False  
False
```

# Cadenas

---

Las cadenas son **secuencias de caracteres**. Un **carácter** representa un **símbolo**, este símbolo puede ser el de una letra, un número u otra cosa. Las cadenas y los caracteres se encuentran encerrados entre comillas dobles o simples.

Carácter ->



"C"

Cadena ->

"Cadena 1"

Cadena ->

'Cadena 2'

# Cadenas

---

Las cadenas son secuencias de caracteres, estas mantienen un **orden de izquierda a derecha** y se numeran de la siguiente forma.

**"Computadora"**

C	o	m	p	u	t	a	d	o	r	a
0	1	2	3	4	5	6	7	8	9	10

Nótese que la numeración **inicia en 0**.

# Cadenas

---

Se puede acceder a los caracteres de una cadena indicando su **posición entre corchetes**.

```
[>>> cadena = "Hola soy Luis!"  
[>>> cadena[0]  
'H'  
[>>> cadena[5]  
's'  
[>>> cadena[-1]  
'!'
```

# Cadenas

---

Se puede acceder a los caracteres de una cadena indicando su **posición entre corchetes**.

```
[>>> cadena = "Hola soy Luis!"  
[>>> cadena[0]  
'H'  
[>>> cadena[5]  
's'  
[>>> cadena[-1]  
'!'
```

De igual forma podemos utilizar **dos puntos** para indicar un **intervalo**.

```
[>>> cadena = "dinosarurio"  
[>>> cadena[0:4]  
'dino'
```

# Cadenas

---

Podemos realizar ciertas operaciones aritméticas con las cadenas.

```
[>>> "Hola" + "Mundo"  
'HolaMundo'  
[>>> "Python" * 3  
'PythonPythonPython']
```

Podemos dar formato a las cadenas.

```
[>>> cadena = "Hoy es %s y está nublado"  
[>>> cadena % ("martes")  
'Hoy es martes y está nublado']
```

Se puede decir que estamos insertando una cadena dentro de otra.

# Cadenas

---

Las cadenas cuentan con algunos ‘**métodos**’ que nos permiten realizar ciertas cosas específicas con ellas. Algunos son:

- Retornar la cadena capitalizada.

```
[>>> cadena = "hola que hace?"  
[>>> cadena.capitalize()  
'Hola que hace?'
```

- Retornar la cadena con un reemplazo.

```
[>>> cadena = "Me gusta la carne"  
[>>> cadena.replace("carne", "pollo")  
'Me gusta la pollo'
```

# Cadenas

---

- Contar el número de apariciones de una subcadena en la cadena.

```
[>>> cadena = "L@ c@dena tiene 3 @"  
[>>> cadena.count("@")  
3
```

- Determinar la posición de una subcadena.

```
[>>> cadena = "PAL@BRA"  
[>>> cadena.find("@")  
3
```



# Cadenas

---

Existen cierto conjunto de caracteres que dentro de una cadena tienen un **significado específico** para el intérprete.

Estos son las **secuencias de escape**. Estas se caracterizan por que **comienzan con una antidiagonal**. (\)

Nos sirven para agregar caracteres que no se pueden introducir por el teclado, como es un salto de línea o una tabulación.

\\	Backslash (stores one \)
\'	Single quote (stores ')
\"	Double quote (stores ")
\a	Bell
\b	Backspace
\f	Formfeed
\n	Newline (linefeed)
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab

# Cadenas

---

En el ejemplo siguiente podemos ver como al imprimir la cadena, `\n` se reemplaza por un salto de línea.

```
>>> cadena = "Cadena de prueba\nAcabo de dar un salto de línea"
>>> print(cadena)
Cadena de prueba
Acabo de dar un salto de línea
```

# Listas

---

Las listas al igual que las cadenas son **secuencias**. Sin embargo, estas no forzosamente son de caracteres. Estas pueden **contener** cualquier '**objeto**' de Python.

Entre algunas de las cosas que pueden contener se encuentran números, cadenas, caracteres, booleanos e incluso otras listas.

Estas se crean usando **corchetes** [].

```
[3, "cadena", "c", True, [1,2,3]]
```

# Listas

---

Al igual que las cadenas es una **secuencia**, por lo que podemos **acceder** a sus **elementos** de igual forma.

```
[>>> lista = [1,"palabra",True]
[>>> lista[0]
1
[>>> lista[-1]
True
[>>> lista[0:2]
[1, 'palabra']
```

# Listas

---

Podemos realizar ciertas **operaciones aritméticas** con las listas.

```
[>>> lista = [1,2,3]
[>>> lista * 3
[1, 2, 3, 1, 2, 3, 1, 2, 3]
[>>> lista + [4,5,6]
[1, 2, 3, 4, 5, 6]
```

# Listas

---

A diferencia de los arreglos de otros lenguajes, las secuencias de Python pueden contener elementos de **diferentes tipos**, las listas pueden **cambiar de tamaño** de forma dinámica.

Para agregar un elemento al final de la lista.

```
[>>> lista = [3,"@",-3+4j,True]
[>>> lista.append("HOLA")
[>>> lista
[3, '@', (-3+4j), True, 'HOLA']
```

# Listas

---

Algunos **métodos** con los que cuentan:

- Retornar el número de elementos con un valor

```
[>>> lista = [1,3,2,1,1,2,4]
[>>> lista.count(1)
3
```

- Invertir el orden de los elementos

```
[>>> lista = ["lunes","martes","miercoles"]
[>>> lista.reverse()
[>>> lista
['miercoles', 'martes', 'lunes']
```



# Listas

---

- Ordenar los elementos de la lista

```
>>> lista = [2,6,1,0,-3]
>>> lista.sort()
>>> lista
[-3, 0, 1, 2, 6]
```

- Eliminar el último elemento de la lista y retornarlo

```
>>> lista = ["a","b","c","d"]
>>> lista.pop()
'd'
>>> lista
['a', 'b', 'c']
```

# Listas

---

Los elementos de una lista se pueden reasignar de la siguiente manera.

```
[>>> lista = [1,3,"a",True]
[>>> lista
[1, 3, 'a', True]
[>>> lista[2] = False
[>>> lista
[1, 3, False, True]
```

# Tuplas

---

Se puede decir que las tuplas son listas que una vez creadas **no se pueden modificar**. Al igual que las listas, estas son **secuencias** y por lo tanto sus elementos se encuentran en una posición específica. Estas se crean usando paréntesis **()**.

```
(True, 4+2j, 6, "Perro", [1,2,3], ("lunes", 4))
```

# Tuplas

---

Podemos acceder a los elementos de las tuplas de la misma forma con la que lo hacíamos con las listas.

```
[>>> tupla = (1,4,6)
[>>> tupla[1]
4
```

De igual forma podemos realizar operaciones algebraicas con ellas.

```
[>>> tupla = ("Hola", True, 4)
[>>> tupla * 3
('Hola', True, 4, 'Hola', True, 4, 'Hola', True, 4)
```

# Tuplas

---

En general las tuplas son idénticas que las listas, la única diferencia es que una vez creadas no se pueden modificar, esto nos ayuda para llevar cierto **control** en programas más avanzados.

```
[>>> tupla = (2, False, 4+6j)
```

```
[>>> tupla[1] = True
```

```
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
```

```
TypeError: 'tuple' object does not support item assignment
```

# Diccionarios

---

Los diccionarios al igual que las listas y las tuplas agrupan elementos, sin embargo estas no son secuencias. A estas se les considera como un mapeo de elementos. En lugar de que a cada elemento se le asigne un número que denote sus posición, se le asignan otro elemento con el que se le van a identificar.

Estos se crean usando llaves {}.

# Diccionarios

---

```
diccionario = {"luis":8,"pato":9,"gabriel":10}
```

Como se puede ver, este diccionario está compuesto por tres **elementos**, y cada uno de estos elementos está **conformado por un valor y una llave**. El valor del primer elemento es 8 y su llave es “luis”, el valor del segundo elemento es 9 y su llave es “pato” y el valor del tercer elemento es 10 y su llave es “gabriel”.

# Diccionarios

---

Se puede observar que los **elementos** están separados por **comas**, y que en cada elemento la **llave y el valor** están separados por **dos puntos**. El **primero** en aparecer es la **llave** y el **segundo** es el **valor**.

```
>>> diccionario = {"llave1":"valor1","llave2":"valor2"}  
>>> diccionario["llave1"]  
'valor1'
```

Como los diccionarios **no son una secuencia ordenada**, para acceder a el valor de los elementos tenemos que **buscarlos por sus llaves**.



# Diccionarios

---

Los **valores** de los elementos de los diccionarios pueden ser de **cualquier tipo de dato**, mientras que las **llaves** pueden ser **cadenas, números o tuplas**.

```
[>>> diccionario = {2:"valor1", 4:"valor2"}  
[>>> diccionario[4]  
'valor2']
```

```
[>>> diccionario = {(2,3,4):[3,6,4], "hola":4}  
[>>> diccionario[(2,3,4)]  
[3, 6, 4]
```

# Diccionarios

---

Al igual que en las listas, podemos modificar el tamaño de los diccionarios una vez creados, se realiza asignando un valor a una llave nueva.

```
>>> diccionario = {"lunes":"pollo","martes":"carne"}  
>>> diccionario["miercoles"] = "pescado"  
>>> diccionario  
{'lunes': 'pollo', 'martes': 'carne', 'miercoles': 'pescado'}
```

# Diccionarios

---

Algunos de los **métodos** con los que cuentan los diccionarios son los siguientes:

- Buscar la llave en un diccionario, en caso de existir, regresar su valor, en caso contrario regresar un valor por default.

```
[>>> diccionario = {"a":1,"b":2}
[>>> diccionario.get("c",20)
20
[>>> diccionario.get("b",20)
2
```

# Diccionarios

---

- Retorna los elementos que tiene un diccionario.

```
[>>> diccionario = {"a":1,"b":2}  
[>>> diccionario.items()  
dict_items([('a', 1), ('b', 2)])
```

- Retorna las llaves que tiene un diccionario.

```
[>>> diccionario.keys()  
dict_keys(['a', 'b'])
```

# Diccionarios

---

- Regresa los valores que tiene un diccionario.

```
[>>> diccionario = {"a":1,"b":2}
[>>> diccionario.values()
dict_values([1, 2])
```

- Elimina el elemento con la llave especificada de un diccionario y regresa su valor.

```
[>>> diccionario = {"a":1,"b":2}
[>>> diccionario.pop("a")
1
[>>> diccionario
{'b': 2}
```