



# Introducción a Python

Estructuras básicas de datos

***Distinguir los tipos de datos  
y sentencias básicas  
del lenguaje para la  
construcción de programas.***

- Unidad 1:  
Introducción a Python
- Unidad 2:  
Sentencias condicionales e  
iterativas
- Unidad 3:  
Estructuras de datos y funciones



Te encuentras aquí



# ¿Qué aprenderás en esta sesión?

*Crea y utiliza aspectos básicos de estructuras de datos en Python*

¿Qué tipos de datos se  
utilizan en Python?



**`/* Listas */`**

# Introducción a listas

Son contenedores que permiten almacenar múltiples datos.

```
# Estos elementos pueden ser de un mismo tipo de datos, por ejemplo solo strings:  
animales = ['gato', 'perro', 'raton']  
  
# O pueden ser de distintos tipos de dato.  
lista_heterogenea = [1, "gato", 3.0, False]
```

Los elementos de la lista se pueden modificar, ya sea cambiando los valores de éstos, agregando o eliminando elementos. Por eso es que se dice que las listas son **mutables**.

# Definir una lista

Para crear una lista utilizaremos la siguiente sintaxis:

```
a = [1, 2, 3, 4]
```

## Elementos de la lista

Son todo lo que esté dentro de los corchetes [], separados por coma.

# Mostrar una lista

Para mostrar una lista podemos ocupar **print** o **llamar a su objeto contenedor**.

```
print(a) # mostramos los valores de la lista a definida arriba  
print([1, 2, 'hola', 4]) # mostramos directo una lista
```

```
[1, 2, 3, 4]
```

```
[1, 2, 'hola', 4]
```



# Índice

Cada elemento en la lista tiene una posición específica, la que se conoce como índice, el que permite acceder al elemento que está dentro de la lista.

En Python los índices parten en **cero** y van hasta  **$n - 1$** , donde  $n$  es la cantidad de elementos en la lista.



# Índice

## Ejemplo

En una lista que contiene 4 elementos:

- el primer elemento está en la posición cero
- el último en la posición 3

Para acceder al elemento de una posición específica de la lista, se debe escribir el nombre de la lista, y entre paréntesis de corchetes, el índice de la posición.

```
colores = ["verde", "rojo", "rosa", "azul"]  
print(colores[0])  
print(colores[1])  
print(colores[3])
```

```
verde  
rojo  
azul
```

# Índice

## *más allá de los límites*

En caso de que el índice sea mayor o igual a la cantidad de elementos en la lista, Python arrojará un **IndexError: list index out of range**, que indica que el índice se posiciona fuera del rango de la lista.

```
colores[8]
```

```
-----  
  
IndexError                                Traceback (most recent call last)  
<ipython-input-4-cf0cdf800f9c> in <module>()  
----> 1 colores[8]  
IndexError: list index out of range
```

# Índice

## *negativo*

Los índices también se pueden utilizar con números negativos y de esta forma referirse a los elementos desde el último al primero.

En Python, el índice del último elemento también se puede denotar con **-1** y el primer elemento corresponde al elemento **-n**.

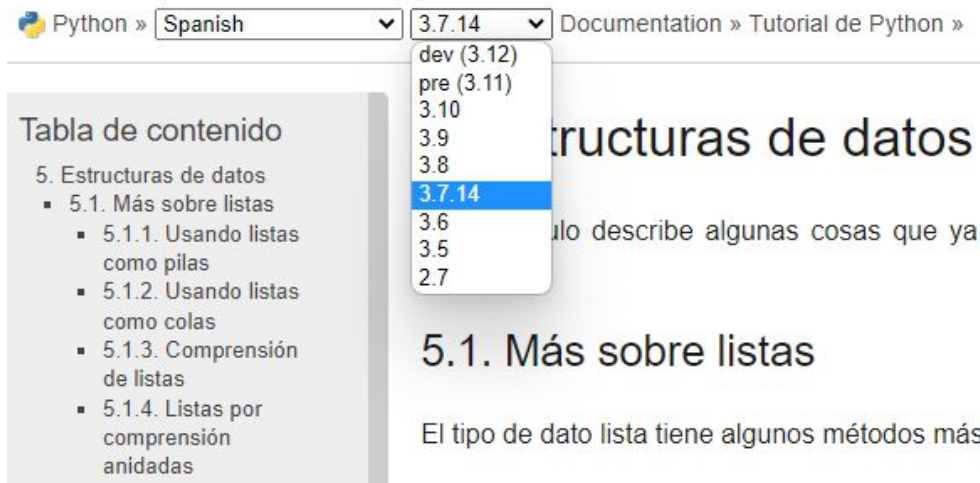
```
a = [1, 2, 3, 4, 5]  
a[-1]
```

5

# Leyendo la documentación de listas

La información oficial de Python sobre las listas (y de manera más general de estructuras de datos) se encuentra en [docs.python.org](https://docs.python.org).

Importante: debes asegurarte que la versión que se esté consultando sea la misma con la que se está trabajando.



The screenshot shows the Python documentation website interface. At the top, there is a navigation bar with "Python »", a language selector set to "Spanish", a version dropdown menu, and "Documentation » Tutorial de Python »". The version dropdown menu is open, showing a list of versions: "dev (3.12)", "pre (3.11)", "3.10", "3.9", "3.8", "3.7.14" (highlighted in blue), "3.6", "3.5", and "2.7". On the left side, there is a "Tabla de contenido" (Table of contents) for the "5. Estructuras de datos" (5. Data Structures) section, listing sub-sections like "5.1. Más sobre listas" and its sub-items. The main content area shows the heading "5.1. Más sobre listas" and the beginning of the text "El tipo de dato lista tiene algunos métodos más."

Python » Spanish 3.7.14 Documentation » Tutorial de Python »

dev (3.12)  
pre (3.11)  
3.10  
3.9  
3.8  
3.7.14  
3.6  
3.5  
2.7

Tabla de contenido

- 5. Estructuras de datos
  - 5.1. Más sobre listas
    - 5.1.1. Usando listas como pilas
    - 5.1.2. Usando listas como colas
    - 5.1.3. Comprensión de listas
    - 5.1.4. Listas por comprensión anidadas

estructuras de datos

5.1. Más sobre listas

El tipo de dato lista tiene algunos métodos más.

**/\* Otras estructuras de datos \*/**

# Tuplas

## ***Par ordenado inmutable***

*(no se pueden modificar partes de ella, en caso de querer actualizarlo se debe modificar la tupla completa)*

```
# definición una tupla
tupla_ej = ("Abril", 2021)
type(tupla_ej)
```

```
tuple
```

Una propiedad útil es lo que se llama **unpacking** o **desempaquetamiento**:

```
# cada valor de la tupla se asigna a month y year respectivamente
month, year = tupla_ej
```

```
# este es una manera alternativa de hacerlo
month, year = "Abril", 2021
```

# Sets

**Permite trabajar similar a lo que es la teoría de conjuntos.**

*No permite valores duplicados, por lo que si se necesita conocer sólo valores únicos, esta es la estructura de datos a utilizar.*

Este tipo de estructura suele ser bastante útil en **análisis de texto**, para conocer las palabras únicas que existen en él.

```
# definición de un set
# se pueden ver que existen valores repetidos
muchos_animales = {'Gato', 'Perro', 'Tortuga',
                  'Gato', 'Perro', 'Tortuga',
                  'Gato', 'Perro', 'Tortuga',
                  'Gato', 'Perro', 'Tortuga',
                  'Hurón', 'Hamster', 'Erizo de Tierra'}
```

```
# no hay duplicados, sólo valores únicos
print(muchos_animales)
```

```
{'Erizo de Tierra', 'Gato', 'Hamster', 'Hurón', 'Perro', 'Tortuga'}
```



**/\* Dictionarios \*/**

# ¿Qué son?

- Estructura de datos compuesta por pares de **clave:valor**, donde:
  - Cada clave se asocia con un elemento del diccionario
  - Como toda estructura de datos, permiten almacenar una gran cantidad de datos en una sola variable.

Reciben este nombre porque se leen igual que uno de la vida real, ya que en un diccionario buscamos una palabra y esta nos lleva a la definición:

- La **clave** es equivalente a la **palabra**
- El **valor** es equivalente a su **definición**

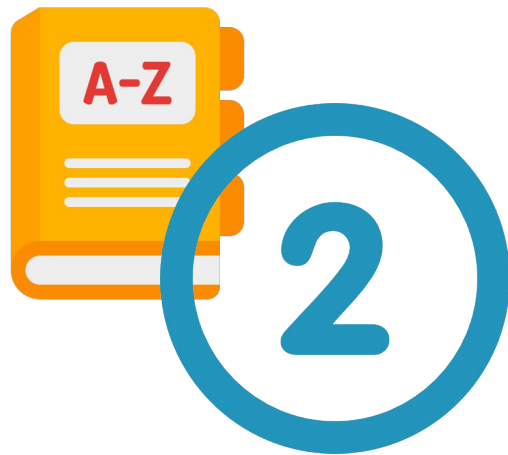
# Listas vs Diccionarios

Para acceder a los elementos de una lista, se hace a través de la posición o el índice asociado al elemento, mientras que en un diccionario, se hace por medio de la clave (o llave).



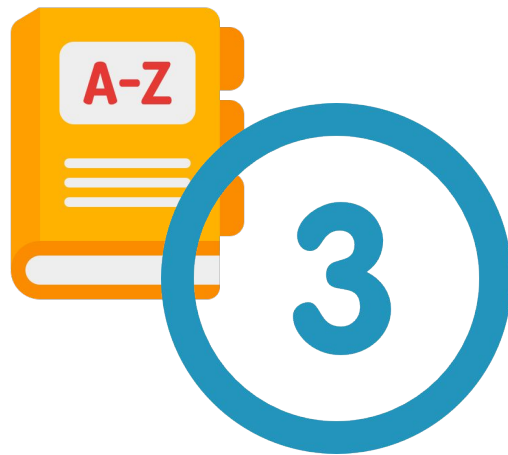
# Listas vs Diccionarios

En una lista, los índices se generan automáticamente para cada elemento, mientras que las claves de un diccionario no se generan automáticamente, sino que son definidas explícitamente al crear el diccionario o al asignar un nuevo elemento a la estructura.



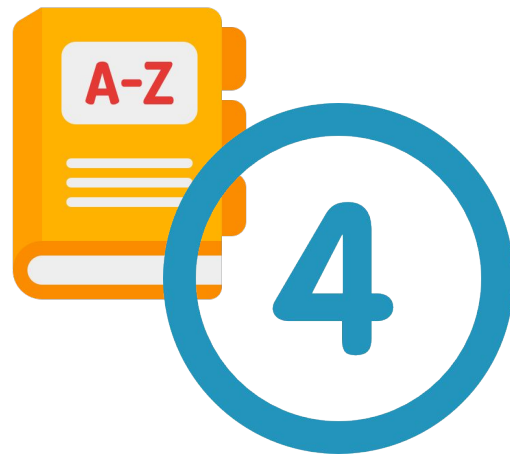
# Listas vs Diccionarios

La lista es un elemento ordenado donde el orden lo entrega el índice, en cambio, en un diccionario es un elemento no ordenado, no hay un elemento que va primero, segundo o último. La única manera de acceder a dicho elemento es mediante su clave.



# Listas vs Diccionarios

Las claves del diccionario pueden ser un string, un número, o incluso un booleano, pero con frecuencia se utilizan los strings.



# Listas vs Diccionarios

```
# En una lista usamos la posición para acceder a un elemento, y los índices se generan en forma implícita
lista = [25, 31, "hola"]
lista[2] # "hola"
```

```
# En un diccionario usamos la clave, y se deben definir de forma explícita
diccionario = {"a": 25, "b": 31, "c": "hola"}
diccionario["c"] # "hola"
```

# Crear un diccionario

En Python, un diccionario (vacío) se define con llaves: {}. Cada par de clave y valor se asocia mediante :, en la forma clave: valor.

```
notas = {"Camila": 7, "Antonio": 5, "Felipe": 6, "Antonia": 7}
```

Si al momento de definir un diccionario tenemos muchas llaves, podemos definirlo en múltiples líneas para facilitar la lectura de código.

```
notas = {  
    "Camila": 7,  
    "Antonio": 5,  
    "Felipe": 6,  
    "Daniela": 5,  
    "Vicente": 7,  
}
```



# Acceder a un elemento

## *dentro de un diccionario*

Se accede a un elemento específico utilizando la **clave** de ese **valor**, donde necesariamente se debe utilizar la misma clave en la que está almacenado el elemento, sino podríamos recibir un valor no deseado, o incluso tener un error si la clave no existe.

```
notas["Felipe"] # 6  
notas["felipe"] # KeyError: 'felipe'
```

# La clave tiene que ser única

En un diccionario, solo puede haber un valor asociado a una clave.

```
duplicados = {"clave": 1, "clave": 2}  
print(duplicados) # {"clave": 2}
```

Al usar dos veces la misma clave, Python se quedará con la última que definimos, ignorando completamente la existencia del primer valor.

¿Qué ventajas tienen las  
listas frente a los  
diccionarios? ¿Y los  
diccionarios frente a las  
listas?





## Próxima sesión...

- *Desafío evaluado*

**{desafío}**  
**latam\_**

*Academia de  
talentos digitales*

