



# Base de datos web con Python

## Unidad IV Colecciones

**Docente: T.S.U Gerardo Alí Ferraro Schelijasch**  
**[gerferr83@soltecferr.com](mailto:gerferr83@soltecferr.com)**  
**<https://soltecferr.com>**

# Unidad IV – Colecciones

---

## Resumen

- **Definición de Colección**
- **Colecciones en Python**
  - **Lista**
  - **Tuplas**
  - **Diccionarios**
  - **Otras**
- **Usos y generalidades**

## Unidad IV – Colecciones

---

### Definición:

Las colecciones de datos son estructuras que permiten almacenar y organizar conjuntos de elementos del mismo tipo y relacionados en un solo contenedor.

En tu vida cotidiana usas el concepto de “colección” constantemente. Normalmente se usan para clasificar un grupo de elementos del mismo tipo de dato que se pueden también repetir y que pueden seguir un orden específico o no.

Por ejemplo:

Los empleados de una empresa, los productos de una compra, los estudiantes de una universidad.

# Unidad IV – Colecciones

---

## Colecciones en Python:

En Python, las colecciones son tipos de datos que permiten agrupar múltiples elementos en una sola estructura. Existen varias colecciones integradas en Python, cada una con sus propias características y usos. A continuación damos una visión general de las mas usadas.

- Listas
- Tuplas
- Diccionarios

## Unidad IV – Colecciones

---

### Listas:

Una lista es una colección ordenada y mutable (se puede modificar después de su creación). Permite elementos duplicados.

Las listas pueden contener cualquier tipo de dato: números, cadenas, booleanos, ... y también listas.

Crear una lista es tan sencillo como indicar entre corchetes, y separados por comas, los valores que queremos incluir en la lista:

```
l = [22, True, "una lista", [1, 2]]
```

# Unidad IV – Colecciones

---

## Uso:

Las listas en Python se usan para indicar una colección de datos, es decir es un contenedor que se usa para almacenar un conjunto de elementos relacionados del mismo tipo o de tipos distintos.

Tienen la características de ser mutables, es decir que su contenido se puede modificar después de haber sido creada.

Para poder acceder a los varios elementos de una lista y poder realizar operaciones con estos, se utilizan los índices.

El índice es un numero entero que indica la posición de un elemento dentro de una colección el primer elemento siempre comienza en 0, y el ultima termina en n-1 donde n representa el numero de elementos de la colección



## Unidad IV – Colecciones

---

### Uso:

Las listas en Python se usan para indicar una colección de datos, es decir es un contenedor que se usa para almacenar un conjunto de elementos relacionados del mismo tipo o de tipos distintos.

Tienen la características de ser mutables, es decir que su contenido se puede modificar después de haber sido creada.

Para poder acceder a los varios elementos de una lista y poder realizar operaciones con estos, se utilizan los índices.

El índice es un numero entero que indica la posición de un elemento dentro de una colección el primer elemento siempre comienza en 0, y el ultima termina en n-1 donde n representa el numero de elementos de la colección

## Unidad IV – Colecciones

---

### Uso:

```
l = [1,4,6[2,3,5], 'mi nombre']
```

```
l[0] = 1
```

```
l[4] = 'mi nombre'
```

También es posible acceder a los elementos de una lista a través de los índices negativos

```
l[-1] = 'mi nombre'
```



# Unidad IV – Colecciones


---

## Métodos:

Python proporciona una variedad de métodos incorporados para manipular listas. A continuación, te presento algunos de los métodos más comunes y útiles para trabajar con listas:

- **append(x)**: Añade un elemento x al final de la lista.

python


 Copiar código

```
mi_lista = [1, 2, 3]
mi_lista.append(4)
print(mi_lista) # [1, 2, 3, 4]
```

## Unidad IV – Colecciones

- **extend(iterable):** Extiende la lista añadiendo todos los elementos de un iterable (otra lista, tupla, etc.).


python

 Copiar código

```
mi_lista = [1, 2, 3]
mi_lista.extend([4, 5, 6])
print(mi_lista) # [1, 2, 3, 4, 5, 6]
```

- **insert(i, x):** Inserta un elemento x en la posición i.

python


 Copiar código

```
mi_lista = [1, 2, 3]
mi_lista.insert(1, 10)
print(mi_lista) # [1, 10, 2, 3]
```

## Unidad IV – Colecciones

- **remove(x)**: Elimina la primera aparición del valor x en la lista. Lanza un error si x no se encuentra en la lista.


python

 Copiar código

```
mi_lista = [1, 2, 3, 2]
mi_lista.remove(2)
print(mi_lista) # [1, 3, 2]
```

- **pop([i])**: Elimina y devuelve el elemento en la posición i. Si i no se proporciona, elimina y devuelve el último elemento.

python

 Copiar código


```
mi_lista = [1, 2, 3]
ultimo_elemento = mi_lista.pop()
print(ultimo_elemento) # 3
print(mi_lista) # [1, 2]
```

## Unidad IV – Colecciones

---

- **clear()**: Elimina todos los elementos de la lista, dejándola vacía.


python

 Copiar código

```
mi_lista = [1, 2, 3]
mi_lista.clear()
print(mi_lista) # []
```

- **count(x)**: Devuelve el número de veces que x aparece en la lista.

python

 Copiar código

```
mi_lista = [1, 2, 2, 3, 2]
cuenta = mi_lista.count(2)
print(cuenta) # 3
```

## Unidad IV – Colecciones

---

- **sorted(key = none, reverse=False):** Ordena los elementos de la lista en su lugar (de forma ascendente por defecto). Puedes usar el parámetro `key` para especificar una función de clave personalizada y `reverse=True` para ordenar en orden descendente.

`sorted(x, reverse = True)`

```
x = [1,4,2,3,9,5]
sorted(x)
[1, 2, 3, 4, 5, 9]
```


```
x = [1,4,2,3,9,5]
sorted(x)
[1, 2, 3, 4, 5, 9]
sorted(x, reverse=True)
[9, 5, 4, 3, 2, 1]
```

## Unidad IV – Colecciones

---

- **reverse()**: Invierte el orden de los elementos en la lista.


python

 Copiar código

```
mi_lista = [1, 2, 3]
mi_lista.reverse()
print(mi_lista) # [3, 2, 1]
```

- **Len()**: Devuelve la longitud (número de elementos) de una secuencia

python

 Copiar código

```
longitud = len(mi_lista)
```



## Unidad IV – Colecciones


- **sum(lista)**: Devuelve la suma de todos los elementos de una secuencia numérica.

```
python Copiar código  
  
mi_lista = [1, 2, 3, 4, 5]  
print(sum(mi_lista)) # 15
```

- **max(lista) y min(lista)** : se utilizan para poder ubicar el valor máximo y mínimo dentro de una lista
- **mean(lista) o sum(lista) / len(lista)**: para calcular el promedio de una lista

# Unidad IV – Colecciones

python


 Copiar código

```
import statistics

mi_lista = [10, 20, 30, 40, 50]

# Calcular la media aritmética usando statistics.mean()
media = statistics.mean(mi_lista)
print("Media:", media) # Media: 30
```

python

 Copiar código

```
mi_lista = [10, 20, 30, 40, 50]

# Calcular la media aritmética
media = sum(mi_lista) / len(mi_lista)
print("Media:", media) # Media: 30.0
```

## Unidad IV – Colecciones

---

- **Tupla:**

Una tupla es similar a una lista, pero es inmutable (no se puede modificar después de su creación). También permite elementos duplicados.

`t = (1,2,3,4,5,6)`

**Uso:**

Usa tuplas cuando trabajes con datos que no deberían cambiar, como coordenadas de un punto en un plano, configuraciones fijas, o claves en un diccionario (ya que las claves deben ser inmutables).

## Unidad IV – Colecciones

---

Los métodos para poder trabajar con estas estructuras son casi los mismos de las lista teniendo en cuenta que métodos como `pop()`, `clear()`, `insert()`, `extend()`, `append()` no se pueden usar ya que estas estructuras son inmutables para poderlos usar lo único que se puede hacer es convertirla en listas y luego aplicarlos los métodos y luego convertir la lista en tupla.

### Ejemplo

```
t = (1,2,4,5,6)
a = list(t)
a.pop()
t = tuple(a)
print (t) » (1,2,4,5)
```


# Unidad IV – Colecciones

## Slicing:

El slicing (corte) es una técnica en Python que permite acceder a una subsección de una secuencia, como listas, tuplas, cadenas de texto, entre otras. Utiliza la notación de corchetes `[]` con dos puntos `:` para especificar el inicio, el final y opcionalmente el paso del corte.

### 1. Cortar una lista:


python

 Copiar código

```
mi_lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sub_lista = mi_lista[2:7]    # [2, 3, 4, 5, 6]
```

### 2. Cortar una cadena de texto:

python


 Copiar código

```
texto = "Hola Mundo"
sub_texto = texto[0:4]    # "Hola"
```

# Unidad IV – Colecciones

## Slicing:


python

 Copiar código

```
mi_lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sub_lista = mi_lista[1:8:2] # [1, 3, 5, 7]
```

Omitir `inicio`, `fin`, o `paso`:

python

 Copiar código

```
mi_lista = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
sub_lista = mi_lista[:5]      # [0, 1, 2, 3, 4] (del inicio al índice 4)
sub_lista = mi_lista[5:]     # [5, 6, 7, 8, 9] (del índice 5 al final)
sub_lista = mi_lista[::2]     # [0, 2, 4, 6, 8] (todos los elementos con paso de 2)
```




# Unidad IV – Colecciones

---

## Diccionarios:

Los diccionarios, también llamados matrices asociativas, deben su nombre a que son colecciones que relacionan una clave y un valor. Por ejemplo, veamos un diccionario de películas y directores:

python

 Copiar código

```
mi_diccionario = {'a': 1, 'b': 2, 'c': 3}
```

## Unidad IV – Colecciones

---

### Diccionarios:

El primer valor se trata de la clave y el segundo del valor asociado a la clave. Como clave podemos utilizar cualquier valor inmutable: podríamos usar números, cadenas, booleanos, tuplas, ... pero no listas o diccionarios, dado que son mutables. Esto es así porque los diccionarios se implementan como tablas hash, y a la hora de introducir un nuevo par clave-valor en el diccionario se calcula el hash de la clave para después poder encontrar la entrada correspondiente rápidamente. Si se modificara el objeto clave después de haber sido introducido en el diccionario, evidentemente, su hash también cambiaría y no podría ser encontrado.

La diferencia principal entre los diccionarios y las listas o las tuplas es que a los valores almacenados en un diccionario se les accede no por su índice, porque de hecho no tienen orden, sino por su clave, utilizando de nuevo el operador `[]`. o el metodo `get()`.

## Unidad IV – Colecciones

---

### Diccionarios:

No es posible usar la técnica del slicing si no mapping es decir pueden ser mapeados.

### Operador in:

El operador in en Python es utilizado para verificar si un elemento se encuentra dentro de una secuencia, como una lista, tupla, cadena, conjunto, o incluso un diccionario (en este caso, verifica si la clave existe en el diccionario). Es un operador muy útil para realizar comprobaciones rápidas y claras sobre la pertenencia de un elemento en una colección de datos.

## Unidad IV – Colecciones

---

### Términos de la licencia.

- This work is licensed under the creative commons Attribution-shareAlike License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/4.0> or send a letter to Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA,
- Este trabajo se otorga bajo los términos de la licencia Creative Commons Attribution-shareAlike License. Para obtener una copia de esta licencia visita <http://creativecommons.org/licenses/by-sa/4.0> o envía una carta a la dirección Creative Commons, 559 Nathan Abbott Way, Stanford, California 94305, USA.