Introducción a la Programación Segura

Colecciones y librerías en Python.

INTRODUCCIÓN A LA PROGRAMACIÓN SEGURA

UNIDAD 3: COLECCIONES Y LIBRERÍAS EN PYTHON.

Contenidos

1. Creación y uso de listas.
2. Métodos de las Listas.
3. Operaciones con Listas.
4. Las rebanadas de listas.
5. Operaciones con Tuplas.
6. Operaciones con Diccionarios.

UNIDAD DE APRENDIZAJE

UNIDAD 3: COLECCIONES Y LIBRERÍAS EN PYTHON.

Aprendizaje Esperado:

3.1. Utiliza estructuras de almacenamiento de datos de Python, para hacer más eficiente el código de programación, considerando el desarrollo de scripts y librerías asociadas a la seguridad.

Creación y uso de listas:

En Python, una lista es una colección ordenada y mutable de elementos. Permite almacenar múltiples valores en una sola variable, y esos valores pueden ser de diferentes tipos, como números, cadenas, listas anidadas, etc. Las listas se crean utilizando corchetes [ ] y los elementos se separan por comas.

Creando listas:

```
# Lista vacía
lista_vacia = []
```

```
# Lista de cadenas
nombres = ["Teresa", "Pedro", "Carlos"]
```

```
# Lista de booleanos
numeros = [True, False, False, True]
```

```
# Lista de números enteros
numeros = [1, 2, 3, 4, 5]
```

```
# Lista de números de punto flotante
numeros = [2.5, 8.1, 0.7, 3.9, 1.5]
```

```
# Lista mixta
mixta = [10, "hola", True, 3.14]
```

Acceder a los elementos de la lista:

Para acceder a elementos individuales de la lista utilizamos el índice del elemento. Los índices en Python comienzan en 0 para el primer elemento, 1 para el segundo elemento, y así sucesivamente.

Acceder a los elementos de la lista:

```
nombres = ["Teresa", "Pedro", "Carlos"]

print(nombres[0])
print(nombres[1])
print(nombres[2])
```

```
Teresa
Pedro
Carlos
```

Métodos de las listas:

Las listas son mutables, lo que significa que puedes modificar, agregar o eliminar elementos después de crear la lista.

Modificar elementos de la lista:

```python
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)
# Modificamos el elemento "Pedro"
nombres[1] = "Karen"
# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
['Teresa', 'Karen', 'Carlos']
```

Agregar elementos a la lista:

```python
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)
# Agregamos el elemento "Diego"
nombres.append("Diego")
# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
['Teresa', 'Pedro', 'Carlos', 'Diego']
```

Métodos de las listas:

Las listas son mutables, lo que significa que puedes modificar, agregar o eliminar elementos después de crear la lista.

Eliminar elementos de la lista:

```python
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)
# Eliminamos el elemento "Pedro"
del nombres[1]
# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
['Teresa', 'Carlos']
```

Operaciones con listas:

Concatenar listas:

```python
nombres = ["Teresa", "Pedro", "Carlos"]
ciudades = ["Temuco", "La Serena", "Vallenar"]

# Mostramos las listas
print(nombres)
print(ciudades)

# Concatenamos las listas
concatenadas = nombres + ciudades
# Mostramos la lista concatenada
print(concatenadas)
```

```
['Teresa', 'Pedro', 'Carlos']
['Temuco', 'La Serena', 'Vallenar']
['Teresa', 'Pedro', 'Carlos', 'Temuco', 'La Serena', 'Vallenar']
```

Obtener el largo de la lista:

```python
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)

# Obtenemos el largo de la lista
largo = len(nombres)

# Mostramos el largo de la lista
print("La lista tiene",largo,"elementos.")
```

```
['Teresa', 'Pedro', 'Carlos']
La lista tiene 3 elementos.
```

Operaciones con listas:

Verificamos si un elemento está en la lista:

```python
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)

# Verificamos si un elemento está en la lista
if "Carlos" in nombres:
    print("Elemento encontrado")
else:
    print("El elemento no encontrado")
```

```
['Teresa', 'Pedro', 'Carlos']
Elemento encontrado
```

Repetir una lista en pantalla:

```python
# Mostramos la lista
print(nombres)

# Retetimos 2 veces la lista
print(nombres * 2)
```

```
['Teresa', 'Pedro', 'Carlos']
['Teresa', 'Pedro', 'Carlos', 'Teresa', 'Pedro', 'Carlos']
```

Operaciones con listas :

Recorrer una lista:

```python
nombres = ["Teresa", "Pedro", "Carlos", "Diego"]

# Mostramos la lista
print(nombres)

contador = 1
# Recorremos la lista
for elemento in nombres:
    print("Elemento",contador,": ",elemento)
    contador += 1
```

```
['Teresa', 'Pedro', 'Carlos', 'Diego']
Elemento 1 :  Teresa
Elemento 2 :  Pedro
Elemento 3 :  Carlos
Elemento 4 :  Diego
```

Operaciones con listas:

Limpiar una lista:

```
nombres = ["Teresa", "Pedro", "Carlos", "Diego"]

# Mostramos la lista
print(nombres)

# Limpiamos la lista
nombres.clear()

# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos', 'Diego']
[]
```

Inserta en una posición específica un elemento a la lista:

```
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)

# insertamos un elemento en la lista
# indicando la posición y su valor
nombres.insert(1,"Rafael")

# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
['Teresa', 'Rafael', 'Pedro', 'Carlos']
```

python™

Inacap

Operaciones con listas:

Eliminar y retornar el último elemento de la lista:

```
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)

# eliminamos el último elemento de la lista
# retornando el valor del elemento eliminado
eliminado = nombres.pop()
print("Se elimino a",eliminado)

# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
Se elimino a Carlos
['Teresa', 'Pedro']
```

Eliminar y retornar un elemento de la lista indicando su índice:

```
nombres = ["Teresa", "Pedro", "Carlos"]

# Mostramos la lista
print(nombres)

# eliminamos el último elemento de la lista
# retornando el valor del elemento eliminado
eliminado = nombres.pop(1)
print("Se elimino a",eliminado)

# Mostramos la lista actualizada
print(nombres)
```

```
['Teresa', 'Pedro', 'Carlos']
Se elimino a Pedro
['Teresa', 'Carlos']
```

Almacenamiento de listas:

Internamente, Python utiliza una estructura dinámica para implementar las listas, lo que significa que puede cambiar su tamaño y contenido durante el tiempo de ejecución.

Al crear una lista en Python, se reserva un bloque de memoria para almacenar los elementos de la lista. Los elementos de la lista se almacenan de forma contigua en la memoria, lo que permite un acceso rápido a cada elemento utilizando su índice.

El almacenamiento de listas en Python se realiza mediante una combinación de punteros y arreglos dinámicos. Los punteros se utilizan para acceder a los elementos individuales de la lista y el arreglo dinámico se encarga de cambiar el tamaño de la lista cuando sea necesario.

Al agregar elementos a una lista, Python verifica si hay suficiente espacio disponible en el bloque de memoria reservado. Si hay suficiente espacio, el nuevo elemento se agrega al final de la lista en la posición correcta. Si no hay suficiente espacio, Python crea un nuevo bloque de memoria más grande, copia los elementos de la lista original al nuevo bloque y libera el bloque de memoria original. Esto asegura que la lista siempre tenga suficiente espacio para almacenar sus elementos y que se pueda ajustar dinámicamente para acomodar nuevos elementos.

Las rebanadas de listas:

En Python, las “rebanadas" de las listas se refieren a la técnica de obtener una porción o subconjunto específico de una lista más grande. Esto se logra mediante el uso de la sintaxis de "slicing" (rebanado) que permite acceder a un rango de elementos de una lista de manera concisa.

lista [inicio : fin : pasos]

inicio: Índice del primer elemento que se incluirá en la rebanada (se incluye en el resultado).

fin: Índice del primer elemento que NO se incluirá en la rebanada (el elemento en el índice fin no se incluirá en el resultado).

pasos: Un número opcional que indica la cantidad de elementos a saltar en cada paso. Si no se proporciona, se asume un valor predeterminado de 1.

Las rebanadas de listas:

Ejemplos:

```
lista = ["uno","dos","tres","cuatro","cinco","seis","siete","ocho","nueve"]

# Obtener una porción de la lista desde el índice 2 hasta el índice 5 (no se incluye el índice 5)
rebanada1 = lista[2:5]
# Resultado: ["tres","cuatro","cinco"]

# Obtener una porción de la lista desde el principio hasta el índice 4 (no se incluye el índice 4)
rebanada2 = lista[:4]
# Resultado: ["uno","dos","tres","cuatro"]

# Obtener una porción de la lista desde el índice 5 hasta el final
rebanada3 = lista[5:]
# Resultado: ["seis","siete","ocho","nueve"]

# Obtener una porción de la lista desde el índice 1 hasta el índice 8, saltando de 2 en 2
rebanada4 = lista[1:8:2]
# Resultado: ["dos","cuatro","seis","ocho"]

# Obtener una copia de toda la lista
copia_lista = lista[:]
# Resultado: ["uno","dos","tres","cuatro","cinco","seis","siete","ocho","nueve"]

# Obtener una porción de la lista desde el índice 8 hasta el índice 0
# (no se incluye el índice 0), de forma regresiva
rebanada5 = lista[8:0:-1]
# Resultado: ["nueve","ocho","siete","seis","cinco","cuatro","tres","dos"]
```

Los operadores in y not in:

En Python, los operadores in y not in se utilizan para verificar si un elemento está presente en una lista o en cualquier otra secuencia. Ambos operadores devuelven un valor booleano (True o False) dependiendo de si el elemento está presente o no en la lista.

```python
nombres = ["Teresa","Pedro","Marta","Josefina","Luis"]

# Verificamos si el elemento esta en la lista
if "Hugo" in nombres:
    print("¡Encontramos el elemento!")
else:
    print("No se encontro el elemento")
```

```
No se encontro el elemento
```

```python
nombres = ["Teresa","Pedro","Marta","Josefina","Luis"]

# Verificamos si el elemento esta en la lista
if "Hugo" not in nombres:
    print("Hugo no esta en la lista")
else:
    print("Hugo esta en la lista")
```

```
Hugo no esta en la lista
```

```python
nombres = ["Teresa","Pedro","Marta","Josefina","Luis"]

# Verificamos si el elemento esta en la lista
if "Marta" in nombres:
    print("¡Encontramos el elemento!")
else:
    print("No se encontro el elemento")
```

```
¡Encontramos el elemento!
```

```python
nombres = ["Teresa","Pedro","Marta","Josefina","Luis"]

# Verificamos si el elemento esta en la lista
if "Marta" not in nombres:
    print("Marta no esta en la lista")
else:
    print("Marta esta en la lista")
```

```
Marta esta en la lista
```

Listas dentro de listas:

En Python, es posible crear listas dentro de listas, lo que se conoce como listas anidadas o listas multidimensionales. Esto significa que los elementos de una lista pueden ser otras listas. Las listas anidadas permiten estructurar datos de manera más compleja y flexible, lo que es especialmente útil cuando se trabaja con datos tabulares o matrices.

```
clientes = [
            ['Pedro','Araya',2245876,'paraya@correo.cl'],
            ['Teresa','Rojas',2365987,'trojas@correo.cl'],
            ['Armando','Roa',2385675,'aroa@correo.cl'],
        ]
```

Declaramos una lista que tiene 3 elementos que también son listas, es decir, tenemos una lista bidimensional.

Podemos acceder a un elemento de la lista por su índice.

```
print(clientes[1])
```

```
['Teresa', 'Rojas', 2365987, 'trojas@correo.cl']
```

```
print(clientes[2])
```

```
['Armando', 'Roa', 2385675, 'aroa@correo.cl']
```

Podemos acceder a un elemento que es a su vez es un elemento de un elemento de la lista, por su índice y subíndice.

```
print(clientes[1][3])
```

```
trojas@correo.cl
```

python™

Inacap

Listas dentro de listas:

También podemos utilizar el slicing para obtener subconjuntos de las listas anidadas.

```
clientes = [
            ['Pedro','Araya',2245876,'paraya@correo.cl'],
            ['Teresa','Rojas',2365987,'trojas@correo.cl'],
            ['Armando','Roa',2385675,'aroa@correo.cl'],
        ]

print(clientes[1][0:3])
```

```
['Teresa', 'Rojas', 2365987]
```

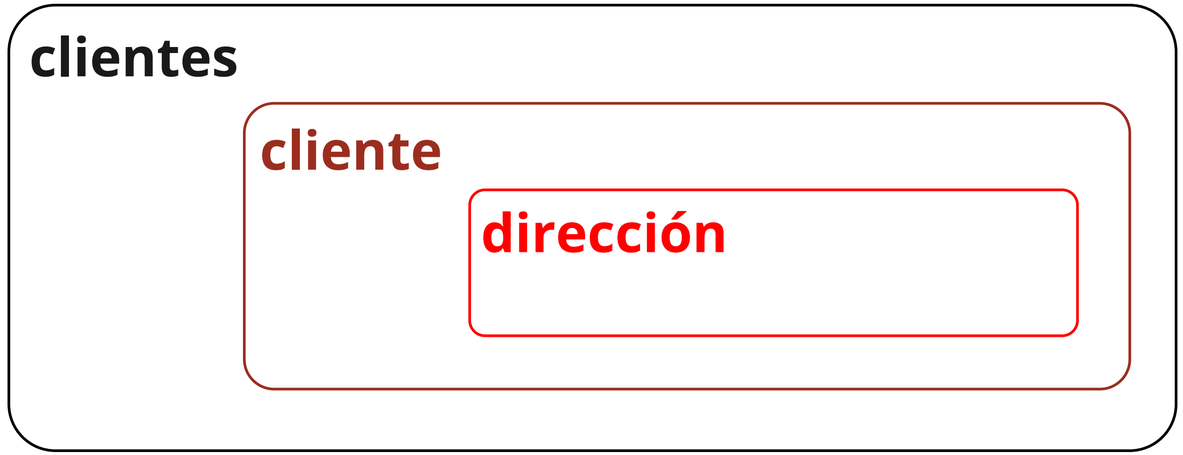
```
clientes = [
            ['Pedro','Araya',2245876,'paraya@correo.cl'],
            ['Teresa','Rojas',2365987,'trojas@correo.cl'],
            ['Armando','Roa',2385675,'aroa@correo.cl'],
        ]

print(clientes[2][::-1])
```

```
['aroa@correo.cl', 2385675, 'Roa', 'Armando']
```

Listas de tres dimensiones:

En Python, las listas de tres dimensiones son listas anidadas que contienen otras listas anidadas. Estas listas anidadas permiten almacenar datos en una estructura tridimensional, lo que puede ser útil para representar datos en un espacio 3D o para trabajar con matrices tridimensionales.

```
clientes = [
            ['Pedro','Araya',2245876,'paraya@correo.cl',['Avenida Aromos',1547,'Valparaiso']],
            ['Teresa','Rojas',2365987,'trojas@correo.cl',['Pasaje Los Robles',9825,'Puerto Varas']],
            ['Armando','Roa',2385675,'aroa@correo.cl',['Balmaceda',640,'La Serena']],
        ]
```



Listas de tres dimensiones:

```
clientes = [
            ['Pedro','Araya',2245876,'paraya@correo.cl',['Avenida Aromos',1547,'Valparaiso']],
            ['Teresa','Rojas',2365987,'trojas@correo.cl',['Pasaje Los Robles',9825,'Puerto Varas']],
            ['Armando','Roa',2385675,'aroa@correo.cl',['Balmaceda',640,'La Serena']],
        ]
```

```
print(clientes[1][4][0])
```

```
Pasaje Los Robles
```

```
print(clientes[0][4][2])
```

```
Valparaiso
```

```
print("Doña",clientes[1][0],clientes[1][1],"vive en",clientes[1][4][0],"#",clientes[1][4][1],"en",clientes[1][4][2])
```

```
Doña Teresa Rojas vive en Pasaje Los Robles # 9825 en Puerto Varas
```

```
print("Don",clientes[0][0],clientes[0][1],"vive en",clientes[0][4][0],"#",clientes[0][4][1],"en",clientes[0][4][2])
```

```
Don Pedro Araya vive en Avenida Aromos # 1547 en Valparaiso
```

Las Tuplas:

¿Qué es una Tupla?

Una tupla es una colección ordenada e inmutable de elementos. Las tuplas se utilizan para almacenar varios elementos relacionados como si fueran un solo objeto. Son similares a las listas, pero con la diferencia clave de que las tuplas son inmutables, lo que significa que una vez creadas, no se pueden modificar, agregar ni eliminar elementos. Esto las hace útiles cuando necesitas datos que no deben cambiar durante la ejecución del programa.

```python
# Declaramos una tupla de números enteros
tupla_1 = (19,25,16,35)

# Declaramos una tupla de puntos flotante
tupla_2 = (3.14,5.47,6.5,2.9)

# Declaramos una tupla de cadenas
tupla_3 = ('auto','camión','bicicleta')

# Declaramos una tupla de booleanos
tupla_4 = (True,False,False,True)

# Declaramos una tupla con diferentes tipos
tupla_5 = ('Marcos',26,False,1.78)
```

Las Tuplas:

Acceder a los elementos de una Tupla:
Al igual que las listas, las tuplas se indexan, lo que significa que puedes acceder a sus elementos mediante un índice numérico. Los índices en Python comienzan desde 0.

```
# Accediendo a los elementos
print(tupla_1[2])
print(tupla_2[2])
print(tupla_3[2])
print(tupla_4[2])
print(tupla_5[2])
```

```
16
6.5
bicicleta
False
False
```

```
# Declaramos una tupla de números enteros
tupla_1 = (19,25,16,35)

# Declaramos una tupla de puntos flotante
tupla_2 = (3.14,5.47,6.5,2.9)

# Declaramos una tupla de cadenas
tupla_3 = ('auto','camión','bicicleta')

# Declaramos una tupla de booleanos
tupla_4 = (True,False,False,True)

# Declaramos una tupla con diferentes tipos
tupla_5 = ('Marcos',26,False,1.78)
```

Las Tuplas:

Acceder a una rebanada de una Tupla:
Puedes obtener una porción (slice) de una tupla utilizando la notación de slicing, que consiste en especificar un rango de índices separados por : igual como lo vimos con las listas.

```
# Accediendo a los elementos
print(tupla_1[2:4])
print(tupla_2[1:4])
print(tupla_3[0:2])
print(tupla_4[3:])
print(tupla_5[::-1])
```

```
(16, 35)
(5.47, 6.5, 2.9)
('auto', 'camión')
(True,)
(1.78, False, 26, 'Marcos')
```

```
# Declaramos una tupla de números enteros
tupla_1 = (19,25,16,35)

# Declaramos una tupla de puntos flotante
tupla_2 = (3.14,5.47,6.5,2.9)

# Declaramos una tupla de cadenas
tupla_3 = ('auto','camión','bicicleta')

# Declaramos una tupla de booleanos
tupla_4 = (True,False,False,True)

# Declaramos una tupla con diferentes tipos
tupla_5 = ('Marcos',26,False,1.78)
```

Las Tuplas:

Obtener la longitud de una Tupla:
Podemos obtener la longitud de una tupla utilizando la función len().

```
# Obteniendo la Longitud
print(len(tupla_1))
print(len(tupla_2))
print(len(tupla_3))
print(len(tupla_4))
print(len(tupla_5))
```

```
4
4
3
4
4
```

```
# Declaramos una tupla de números enteros
tupla_1 = (19,25,16,35)

# Declaramos una tupla de puntos flotante
tupla_2 = (3.14,5.47,6.5,2.9)

# Declaramos una tupla de cadenas
tupla_3 = ('auto','camión','bicicleta')

# Declaramos una tupla de booleanos
tupla_4 = (True,False,False,True)

# Declaramos una tupla con diferentes tipos
tupla_5 = ('Marcos',26,False,1.78)
```

Las Tuplas:

Operaciones con Tuplas:

Verificar que un elemento este en la tupla:

```
# Declaramos una tupla con diferentes tipos
tupla = ('Marcos','Rojas',26,False,1.78,'mrojas@correo.cl')

# Operación x in s
print('Rojas' in tupla)
print('Araya' in tupla)
```

```
True
False
```

Verificar que un elemento no esté en la tupla:

```
# Declaramos una tupla con diferentes tipos
tupla = ('Marcos','Rojas',26,False,1.78,'mrojas@correo.cl')

# Operación x not in s
print('Marcos' not in tupla)
print('Fabian' not in tupla)
```

```
False
True
```

Concatenando tuplas:

```
# Declaramos 2 tuplas
tupla1 = ('auto','camión','bicicleta')
tupla2 = (19,25,16,35)

# Creamos una nueva tupla a partir de las anteriores
nuevaTupla = tupla1 + tupla2

# Mostramos los resultados
print(nuevaTupla)
```

```
('auto', 'camión', 'bicicleta', 19, 25, 16, 35)
```

Las Tuplas:

Operaciones con Tuplas:

Obtener el valor mínimo de la tupla:

```
# Declaramos una tupla
tupla = (19,25,16,35,105,2,37,51,101,3)

# Obtenemos el valor mínimo
print(min(tupla))
```

```
2
```

Obtener el valor máximo de la tupla:

```
# Declaramos una tupla
tupla = (19,25,16,35,105,2,37,51,101,3)

# Obtenemos el valor máximo
print(max(tupla))
```

```
105
```

Obtener la cantidad de ocurrencias:

```
# Declaramos una tupla
tupla = (19,25,16,35,105,2,25,51,25,3)

# Obtenemos la cantidad de ocurrencias
print(tupla.count(25))
```

```
3
```

Borrar una tupla:

```
# Declaramos una tupla
tupla = (19,25,16,35,105,2,25,51,25,3)

# borramos la tupla
del tupla
```

Las Tuplas:

Operaciones con Tuplas:

Acceder a elementos en diferentes dimensiones de una tupla:

```
# Declaramos una tupla anidada
tupla = ('Marcos','Rojas',26,False,(12,41,('Juan','Pedro','Camila'),32),'mrojas@correo.cl')

# Accedemos a elementos dentro de una tuplas anidada
print(tupla)
print(tupla[4])
print(tupla[4][2])
print(tupla[4][2][1])
print(tupla[4][2][1][3])
```

```
('Marcos', 'Rojas', 26, False, (12, 41, ('Juan', 'Pedro', 'Camila'), 32), 'mrojas@correo.cl')
(12, 41, ('Juan', 'Pedro', 'Camila'), 32)
('Juan', 'Pedro', 'Camila')
Pedro
r
```

Obtener el índice de un elemento:

```
# Declaramos una tupla
tupla = (19,25,16,35,105,2,25,51,25,3)

# Buscamos un valor en la tupla
print(tupla.index(105))
```

```
4
```

Los Diccionarios:

¿Qué es un Diccionario?
Son una estructura de datos muy útil que permite almacenar datos en pares clave : valor. A diferencia de las listas y las tuplas, que utilizan índices numéricos para acceder a sus elementos, los diccionarios utilizan claves para acceder a los valores asociados. Los diccionarios son útiles cuando necesitas almacenar datos relacionados de manera estructurada y flexible.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }

# Mostramos los datos del diccionario
print(datos)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
```

Los Diccionarios:

Acceder a los elementos del Diccionario:
A diferencia de las listas y las tuplas, estos no se indexan, por lo que debemos indicar la palabra clave del elemento dentro de corchetes [ ].

```python
# Mostramos los datos del diccionario
print(datos['nombre'])
print(datos['apellido'])
print(datos['edad'])
print(datos['direccion'])
print(datos['ciudad'])
```

```
Marcos
Rojas
26
Balmaceda #1520
La Serena
```

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Modificar los elementos del Diccionario:
Para modificar un valor asociado a una clave, simplemente asigna un nuevo valor a esa clave.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Mostramos los datos del diccionario
print(datos)

# Modificamos los valores
datos['nombre'] = 'Teresa'
datos['apellido'] = 'Novoa'

# Mostramos los datos actualizados
print(datos)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
{'nombre': 'Teresa', 'apellido': 'Novoa', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
```

Los Diccionarios:

Agregar un elemento al Diccionario:

Puedes agregar nuevos pares clave-valor al diccionario asignando un valor a una nueva clave.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Mostramos los datos del diccionario
print(datos)

# Agregamos un elemento
datos['Telefono'] = 2584975

# Mostramos los datos actualizados
print(datos)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena', 'Telefono': 2584975}
```

Los Diccionarios:

Eliminar un elemento del Diccionario:

Para eliminar un elemento del diccionario, utilizamos el método del y especificando la clave del elemento que deseas eliminar.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Mostramos los datos del diccionario
print(datos)

# Eliminamos un elemento
del datos['direccion']

# Mostramos los datos actualizados
print(datos)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'ciudad': 'La Serena'}
```

Los Diccionarios:

Método items():
Devuelve una lista de tuplas, donde cada tupla se compone de 2 elementos: el primero será la clave y el segundo, su valor.

```python
# Mostramos los datos del diccionario
for elemento in datos.items():
    print(elemento)
```

```
('nombre', 'Marcos')
('apellido', 'Rojas')
('edad', 26)
('direccion', 'Balmaceda #1520')
('ciudad', 'La Serena')
```

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Método keys():
Retorna una lista de elementos, los cuales serán las claves de nuestro diccionario.

```python
# Mostramos los datos del diccionario
for clave in datos.keys():
    print(clave)
```

```
nombre
apellido
edad
direccion
ciudad
```

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Método values():
Retorna una lista de elementos, los cuales serán los valores de nuestro diccionario.

```python
# Mostramos los datos del diccionario
for valor in datos.values():
    print(valor)
```

```
Marcos
Rojas
26
Balmaceda #1520
La Serena
```

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Método clear():
Elimina todos los elementos del diccionario dejándolo vacío.

```
# Vaciamos los datos del diccionario
datos.clear()
# Mostramos el diccionario actualizado
print(datos)
```

```
{}
```

```
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Método copy():
Retorna una copia del diccionario original.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Mostramos el diccionario
print(datos)

# Copiamos el diccionario
copia = datos.copy()

# Mostramos la copia del diccionario
print(copia)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena'}
```

Los Diccionarios:

Método get():
Recibe como parámetro una clave, devuelve el valor de la clave, si no la encuentra devuelve un objeto None.

```python
# Obtenemos un valor del diccionario
valor1 = datos.get('nombre')
valor2 = datos.get('fono')

# Mostramos los resultados
print(valor1)
print(valor2)
```

```
Marcos
None
```

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

Los Diccionarios:

Método pop():
Recibe como parámetro una clave, elimina esta y devuelve su valor. Si no lo encuentra devuelve un error.

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Eliminamos un valor del diccionario
eliminado = datos.pop('direccion')

# Mostramos los resultados
print('Eliminamos:',eliminado)
print(datos)
```

```
Eliminamos: Balmaceda #1520
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'ciudad': 'La Serena'}
```

Los Diccionarios:

Método setdefault():
Primera forma de uso: Funciona igual que el método get().

```python
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```python
# Obtenemos un valor del diccionario
encontrado1 = datos.setdefault('direccion')
encontrado2 = datos.setdefault('fono')

# Mostramos los resultados
print('Encontramos:',encontrado1)
print('Encontramos:',encontrado2)
```

```
Encontramos: Balmaceda #1520
Encontramos: None
```

Los Diccionarios:

Método setdefault():
Segunda forma de uso: Agrega un nuevo elemento al Diccionario si le pasamos la clave y el valor.

```
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

```
# Agregamos un valor del diccionario
datos.setdefault('fono',2536987)

# Mostramos los resultados
print(datos)
```

```
{'nombre': 'Marcos', 'apellido': 'Rojas', 'edad': 26, 'direccion': 'Balmaceda #1520', 'ciudad': 'La Serena', 'fono': 2536987}
```

Los Diccionarios:

Recorriendo un Diccionario:

```
# Recorremos el diccionario
for clave,valor in datos.items():
    print(clave,' ==> ',valor)
```

```
nombre  ==>  Marcos
apellido  ==>  Rojas
edad  ==>  26
direccion  ==>  Balmaceda #1520
ciudad  ==>  La Serena
```

```
# Declaramos un Diccionario
datos = {
    'nombre':'Marcos',
    'apellido':'Rojas',
    'edad':26,
    'direccion':'Balmaceda #1520',
    'ciudad':'La Serena'
    }
```

A practicar...

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

1. Desarrolle un script que a través de una función reciba una lista de palabras y devuelva una lista con la longitud de cada palabra:

```python
def longitudPalabras(listaPalabras):
    longitudes = []
    for palabra in palabras:
        longitudes.append(len(palabra))

    return longitudes


palabras = ["Pedro", "Casa", "Murcielago", "Ventana"]
print(longitudPalabras(palabras))
```

```
[5, 4, 10, 7]
```

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

2. Desarrolle un script que a través de una función solicite al usuario el ingreso de números para colocarlos en una lista, pero cuando el usuario ingrese un 0, deje de solicitar más números y muestre los números ingresados en la lista:

```python
def ingresarNumeros():
    numeros = []
    while True:
        numero = int(input("Ingrese un número o ingrese un 0 para detener: "))
        if numero == 0:
            break
        numeros.append(numero)
    return numeros


listaNumeros = ingresarNumeros()
print("Números ingresados:", listaNumeros)
```

```
Ingrese un número o ingrese un 0 para detener: 15
Ingrese un número o ingrese un 0 para detener: 17
Ingrese un número o ingrese un 0 para detener: 5
Ingrese un número o ingrese un 0 para detener: 8
Ingrese un número o ingrese un 0 para detener: 0
Números ingresados: [15, 17, 5, 8]
```

Ejercicios:

De acuerdo al contenido visto en clases, desarrolla scripts en lenguaje Python para los siguientes ejercicios:

3. Desarrolle un script que a través de una función tome una cadena de texto como argumento y cuente la cantidad de veces que aparece cada palabra en la cadena. Debe retornar un diccionario donde las claves son las palabras y los valores son las cantidades de veces que conto a la palabra:

```python
def contarPalabras(cadena):
    palabras = cadena.split()
    contador = {}

    for palabra in palabras:
        palabra = palabra.lower()
        if palabra in contador:
            contador[palabra] += 1
        else:
            contador[palabra] = 1

    return contador


texto = "A Cuesta le cuesta subir la cuesta y en medio de la cuesta va y se acuesta"
resultado = contarPalabras(texto)
print(resultado)
```

```
{'a': 1, 'cuesta': 4, 'le': 1, 'subir': 1, 'la': 2, 'y': 2, 'en': 1, 'medio': 1, 'de': 1, 'va': 1, 'se': 1, 'acuesta': 1}
```

inacap.cl