Python

Estructuras de datos

CertiDevs

Índice de contenidos

1. Listas

Las **listas** en Python son una estructura de datos que almacena elementos ordenados y mutables.

Se pueden utilizar para almacenar una colección de elementos de diferentes tipos, como números, cadenas y otros objetos.

A continuación, se presentan las operaciones y métodos más comunes que se pueden realizar con listas en Python.

1.1. Creación de listas

Para crear una lista, simplemente coloque una serie de elementos entre corchetes `[]`, separados por comas.

También puede crear una lista vacía utilizando corchetes sin elementos.

```
lista_vacia = []
lista_numeros = [1, 2, 3, 4, 5]
lista_mixta = [1, "dos", 3.0, [4, 5]]
```

1.2. Acceso a elementos

Puede acceder a un elemento de una lista utilizando su índice, que comienza en `0`.

Si el índice es negativo, se cuenta desde el final de la lista.

```
lista = [0, 1, 2, 3, 4]

primer_elemento = lista[0]  # 0
segundo_elemento = lista[1]  # 1
ultimo_elemento = lista[-1]  # 4
penultimo_elemento = lista[-2]  # 3
```

1.3. Modificación de elementos

Las listas son **mutables**, lo que significa que puede cambiar sus elementos simplemente asignándoles un nuevo valor utilizando su índice.

```
lista = [0, 1, 2, 3, 4]

lista[1] = 42
print(lista)  # [0, 42, 2, 3, 4]
```

1.4. Añadir elementos

Para agregar elementos a una lista, puede usar el método `append()` para agregar un elemento al final de la lista o el método `insert()` para insertar un elemento en una posición específica.

```
lista = [0, 1, 2, 3, 4]

lista.append(5)  # [0, 1, 2, 3, 4, 5]
lista.insert(2, 42)  # [0, 1, 42, 2, 3, 4, 5]
```

1.5. Eliminar elementos

Puede eliminar elementos de una lista utilizando la palabra clave `del`, el método `remove()` o el método `pop()`.

```
lista = [0, 1, 2, 3, 4]

del lista[1]  # [0, 2, 3, 4]
lista.remove(3)  # [0, 2, 4]
elemento_eliminado = lista.pop(1)  # elemento_eliminado = 2, lista = [0, 4]
```

1.6. Tamaño de la lista

Para obtener el número de elementos en una lista, puede utilizar la función `len()`.

```
lista = [0, 1, 2, 3, 4]
tamaño = len(lista)  # 5
```

1.7. Comprobar si un elemento está en la lista

Para comprobar si un elemento está en una lista, puede utilizar la palabra clave `in`.

```
lista = [0, 1, 2, 3, 4]

resultado = 2 in lista  # True
resultado = 42 in lista  # False
```

1.8. Concatenación de listas

Puede combinar dos listas utilizando el operador `+`.

```
lista1 = [1, 2, 3]
```

```
lista2 = [4, 5, 6]
lista_combinada = lista1 + lista2  # [1, 2, 3, 4, 5, 6]
```

1.9. Repetición de listas

Puede repetir una lista varias veces utilizando el operador *.

```
lista = [1, 2, 3]
lista_repetida = lista * 3  # [1, 2, 3, 1, 2, 3, 1, 2, 3]
```

1.10. Creación de sublistas (slicing)

Puede crear una **sublista** a partir de una lista utilizando el **slicing**.

El **slicing** se realiza proporcionando dos índices separados por dos puntos :.

El **primer índice** indica el inicio del slice (**inclusivo**) y el **segundo índice** indica el final del slice (**exclusivo**).

```
lista = [0, 1, 2, 3, 4, 5]

sublista = lista[1:4]  # [1, 2, 3]
sublista_inicio = lista[:3]  # [0, 1, 2]
sublista_final = lista[3:]  # [3, 4, 5]
```

También puede utilizar un tercer índice para indicar el paso del slicing.

```
lista = [0, 1, 2, 3, 4, 5]

sublista_paso_2 = lista[1:5:2]  # [1, 3]
sublista_invertida = lista[::-1]  # [5, 4, 3, 2, 1, 0]
```

1.11. count()

El método `count()` devuelve el número de veces que un elemento aparece en la lista.

```
lista = [1, 2, 3, 2, 1, 2, 1, 1, 1]

conteo = lista.count(1)  # 5
```

1.12. index()

El método `index()` devuelve el índice del primer elemento que coincide con el valor especificado.

```
lista = [1, 2, 3, 2, 1, 2, 1, 1, 1]

indice = lista.index(2)  # 1
```

1.13. sort()

El método `sort()` ordena los elementos de la lista en su lugar. Para ordenar la lista en orden descendente, puede establecer el argumento reverse en True.

```
lista = [3, 1, 4, 1, 5, 9, 2, 6, 5]

lista.sort()  # [1, 1, 2, 3, 4, 5, 5, 6, 9]
lista.sort(reverse=True)  # [9, 6, 5, 5, 4, 3, 2, 1, 1]
```

1.14. reverse()

El método `reverse()` invierte el orden de los elementos de la lista en su lugar.

```
lista = [1, 2, 3, 4, 5]

lista.reverse()  # [5, 4, 3, 2, 1]
```

1.15. extend()

El método `extend()` añade los elementos de un iterable (por ejemplo, otra lista) al final de la lista.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]

lista1.extend(lista2)  # [1, 2, 3, 4, 5, 6]
```

1.16. Listas por comprensión

Las **comprensiones** o **list comprehension** son una forma concisa de crear listas utilizando una sintaxis similar a las ecuaciones matemáticas de conjuntos. Permiten crear listas a partir de iterables utilizando una expresión que define los elementos y una o más cláusulas for y if.

```
# Crear una lista de los cuadrados de los números del 0 al 9
cuadrados = [x ** 2 for x in range(10)]  # [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# Crear una lista de los números pares del 0 al 9
pares = [x for x in range(10) if x % 2 == 0]  # [0, 2, 4, 6, 8]
```

```
# Crear una lista de los cuadrados de los números pares del 0 al 9
cuadrados_pares = [x ** 2 for x in range(10) if x % 2 == 0]  # [0, 4, 16, 36, 64]
```

2. Tuplas

Las **tuplas** en Python son una estructura de datos similar a las listas, pero son inmutables, lo que significa que no se pueden modificar una vez creadas.

Son útiles para almacenar colecciones de elementos que no deben cambiar durante la vida del programa.

A continuación, se presentan las operaciones y características más comunes de las tuplas en Python.

2.1. Creación de tuplas

Para crear una tupla, coloque una serie de elementos entre paréntesis `()`, separados por **comas**.

También puede crear una tupla vacía utilizando paréntesis sin elementos.

```
tupla_vacia = ()
tupla_numeros = (1, 2, 3, 4, 5)
tupla_mixta = (1, "dos", 3.0, (4, 5))
```

Para crear una tupla con un solo elemento, debe incluir una coma después del elemento, incluso si hay paréntesis alrededor del elemento.

```
tupla_un_elemento = (42,)
```

2.2. Acceso a elementos

Al igual que con las listas, puede acceder a un elemento de una tupla utilizando su índice, que comienza en 0.

Si el índice es negativo, se cuenta desde el final de la tupla.

```
tupla = (0, 1, 2, 3, 4)

primer_elemento = tupla[0]  # 0
segundo_elemento = tupla[1]  # 1
ultimo_elemento = tupla[-1]  # 4
penultimo_elemento = tupla[-2]  # 3
```

2.3. Inmutabilidad

A diferencia de las listas, las tuplas son inmutables, lo que significa que no se pueden modificar una vez creadas.

Esto significa que no puede agregar, eliminar o cambiar elementos en una tupla.

```
tupla = (0, 1, 2, 3, 4)

# Estas operaciones no están permitidas y generarán errores
# tupla[1] = 42
# tupla.append(5)
# del tupla[1]
```

2.4. Tamaño de la tupla

Para obtener el número de elementos en una tupla, puede utilizar la función `len()`.

```
tupla = (0, 1, 2, 3, 4)
tamaño = len(tupla)  # 5
```

2.5. Comprobar si un elemento está en la tupla

Para comprobar si un elemento está en una tupla, puede utilizar la palabra clave `in`.

```
tupla = (0, 1, 2, 3, 4)

resultado = 2 in tupla  # True
resultado = 42 in tupla  # False
```

2.6. Desempaquetado de tuplas

El desempaquetado de tuplas permite asignar los elementos de una tupla a múltiples variables en una sola operación.

```
tupla = (1, 2, 3)
a, b, c = tupla

print(a)  # 1
print(b)  # 2
print(c)  # 3
```

También puede utilizar el desempaquetado parcial de tuplas para asignar solo algunos elementos a

variables y el resto a otra tupla.

```
tupla = (1, 2, 3, 4, 5)
a, b, *resto = tupla

print(a)  # 1
print(b)  # 2
print(resto)  # [3, 4, 5]
```

2.7. Tuplas como claves de diccionarios

Dado que las tuplas son inmutables, pueden utilizarse como claves en los diccionarios de Python. Esto puede ser útil cuando necesita mapear pares o n-tuplas de valores a otros valores.

```
diccionario_tuplas = {
    ("Juan", "Pérez"): "Programador",
    ("María", "García"): "Diseñadora",
}


print(diccionario_tuplas[("Juan", "Pérez")])  # "Programador"
```

3. Diccionarios

Los **diccionarios** en Python son una estructura de datos que permite almacenar **pares clave-valor**.

Son particularmente útiles cuando desea asociar un valor con una clave única, como en una base de datos simple o una tabla de búsqueda.

A continuación, se presentan las operaciones y características más comunes de los diccionarios en Python.

3.1. Creación de diccionarios

Para crear un **diccionario**, utilice llaves `{}` y separe las claves y los valores con dos puntos `:`.

También puede crear un diccionario vacío utilizando llaves sin pares clave-valor.

```
diccionario_vacio = {}
diccionario_numeros = {1: "uno", 2: "dos", 3: "tres"}
diccionario_mixto = {"nombre": "Juan", "edad": 30, "altura": 1.80}
```

3.2. Acceso a los valores

Para acceder al valor asociado con una clave en un diccionario, utilice la clave entre corchetes `[]`.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

nombre = diccionario["nombre"]  # "Juan"
edad = diccionario["edad"]  # 30
altura = diccionario["altura"]  # 1.80
```

Si intenta acceder a una clave que no existe en el diccionario, se generará un error.

Para evitar esto, puede utilizar el método `get()` que devuelve `None` si la clave no se encuentra en el diccionario.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

peso = diccionario.get("peso")  # None
```

También puede proporcionar un valor predeterminado para el método `get()` que se devolverá si la clave no se encuentra en el diccionario.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

peso = diccionario.get("peso", 70)  # 70
```

3.3. Añadir o modificar elementos

Para agregar un nuevo elemento al diccionario o modificar un elemento existente, utilice la siguiente sintaxis:

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

# Añadir un nuevo elemento
diccionario["peso"] = 70

# Modificar un elemento existente
diccionario["edad"] = 31
```

3.4. Eliminar elementos

Para eliminar un elemento del diccionario, utilice la declaración `del`.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}
```

```
# Eliminar un elemento
del diccionario["altura"]
```

Si intenta eliminar una clave que no existe en el diccionario, se generará un error.

Para evitar esto, puede utilizar el método `pop()` que devuelve el valor asociado con la clave y elimina el elemento del diccionario.

Si la clave no se encuentra en el diccionario, devuelve `None`.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

altura = diccionario.pop("altura", None)  # 1.80
```

3.5. Tamaño del diccionario

Para obtener el número de elementos en un diccionario, puede utilizar la función `len()`.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}
tamaño = len(diccionario)  # 3
```

3.6. Comprobar si una clave está en el diccionario

Para comprobar si una clave está en un diccionario, puede utilizar la palabra clave `in`.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

resultado = "nombre" in diccionario  # True
resultado = "peso" in diccionario  # False
```

3.7. Iterar sobre un diccionario

Puede iterar sobre las claves, los valores o los pares clave-valor de un diccionario utilizando los métodos `keys()`, `values()` y `items()` respectivamente.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

# Iterar sobre las claves
for clave in diccionario.keys():
    print(clave)

# Iterar sobre los valores
for valor in diccionario.values():
```

```
    print(valor)

# Iterar sobre los pares clave-valor
for clave, valor in diccionario.items():
    print(clave, valor)
```

Por defecto, si no se utiliza ningún método, la iteración se realizará sobre las claves del diccionario.

```
diccionario = {"nombre": "Juan", "edad": 30, "altura": 1.80}

for clave in diccionario:
print(clave)
```

3.8. Copiar un diccionario

Para crear una copia de un diccionario, puede utilizar el método `copy()`.

```
diccionario_original = {"nombre": "Juan", "edad": 30, "altura": 1.80}
diccionario_copia = diccionario_original.copy()

# Modificar el diccionario original no afecta a la copia
diccionario_original["nombre"] = "Pedro"
print(diccionario_copia["nombre"])  # "Juan"
```

3.9. Combinar diccionarios

Para combinar dos diccionarios, puede utilizar el método `update()`.

Este método actualiza el diccionario original con los pares clave-valor del diccionario que se pasa como argumento.

Si las claves ya existen en el diccionario original, se actualizan con los nuevos valores.

```
diccionario1 = {"nombre": "Juan", "edad": 30}
diccionario2 = {"edad": 31, "altura": 1.80}

diccionario1.update(diccionario2)
print(diccionario1)  # {'nombre': 'Juan', 'edad': 31, 'altura': 1.80}
```