Notas para la Sesión 3: Ciclos, iteradores y estructuras de datos: listas y tuplas

1. Ciclos e iteradores

Función range()

La función `range()` genera una secuencia de números que pueden ser utilizados en un bucle for. Acepta hasta tres argumentos: `range(stop)`, `range(start, stop)` y `range(start, stop, step)`.

- `stop`: El número hasta el cual se generará la secuencia (excluyendo este número).
- `start`: El número inicial de la secuencia (incluido). Por defecto, es 0.
- `step`: La diferencia entre cada número en la secuencia. Por defecto, es 1.

Ejemplos:

```
# Genera una secuencia de 0 a 4
for i in range(5):
    print(i)

# Genera una secuencia de 1 a 5
for i in range(1, 6):
    print(i)

# Genera una secuencia de números pares del 0 al 8
for i in range(0, 10, 2):
    print(i)
```

Iteradores y la función iter()

Los iteradores son objetos que permiten recorrer una colección de elementos, como listas o tuplas, de manera secuencial. Para crear un iterador, utilizamos la función `iter()` y pasamos la colección como argumento. Para acceder al siguiente elemento del iterador, usamos la función `next()`.

Ejemplo:

```
names = ["Alice", "Bob", "Charlie"]
iterator = iter(names)

print(next(iterator))  # Output: Alice
print(next(iterator))  # Output: Bob
print(next(iterator))  # Output: Charlie
```

2. Listas

Las listas son colecciones ordenadas y mutables de elementos. Pueden contener elementos de diferentes tipos, incluso otras listas.

Creación y acceso a elementos

Para crear una lista, utilizamos corchetes `[]` y separamos los elementos con comas. Para acceder a un elemento de la lista, utilizamos índices entre corchetes. Los índices negativos acceden a los elementos desde el final de la lista.

Ejemplo:

```
my_list = [1, 2, 3, 4, 5]
print(my_list[0])  # Output: 1
print(my_list[-1])  # Output: 5
```

Métodos y operaciones básicas

Las listas tienen varios métodos que permiten modificar y manipular sus elementos. Algunos de los métodos más comunes son:

- `append()`: Añade un elemento al final de la lista.
- `insert()`: Inserta un elemento en la posición indicada.
- `remove()`: Elimina la primera aparición del elemento especificado.
- `pop()`: Elimina y devuelve el último elemento de la lista (o el elemento en el índice especificado).
- `reverse()`: Invierte el orden de los elementos de la lista.
- `sort()`: Ordena los elementos de la lista de forma ascendente (o según la función de ordenamiento especificada).

Ejemplo:

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

my_list.append(6)
print(my_list)  # Output: [1, 2, 3, 4, 5, 6]

my_list.insert(0,0)
print(my_list)  # Output: [0, 1, 2, 3, 4, 5, 6]

my_list.remove(2)
print(my_list)  # Output: [0, 1, 3, 4, 5, 6]

pop_element = my_list.pop()
print(pop_element)  # Output: 6
print(my_list)  # Output: [0, 1, 3, 4, 5]

my_list.reverse()
print(my_list)  # Output: [5, 4, 3, 1, 0]

```

```
my_list.sort()
print(my_list)  # Output: [0, 1, 3, 4, 5]
```

Listas anidadas y comprensión de listas

Las listas anidadas son listas dentro de otras listas. Para acceder a los elementos de una lista anidada, utilizamos índices múltiples.

La comprensión de listas es una forma concisa de crear listas utilizando una expresión y un bucle for. También puede incluir una condición opcional para filtrar los elementos.

Ejemplo:

```
nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
print(nested_list[0][1])  # Output: 2

squared_numbers = [x ** 2 for x in range(1, 6)]
print(squared_numbers)  # Output: [1, 4, 9, 16, 25]

even_numbers = [x for x in range(1, 11) if x % 2 == 0]
print(even_numbers)  # Output: [2, 4, 6, 8, 10]
```

3. Tuplas

Las tuplas son colecciones ordenadas e inmutables de elementos. A diferencia de las listas, no se pueden modificar una vez creadas.

Creación y acceso a elementos

Para crear una tupla, utilizamos paréntesis `()` y separamos los elementos con comas. Para acceder a un elemento de la tupla, utilizamos índices entre corchetes.

Ejemplo:

```
my_tuple = (1, 2, 3, 4, 5)
print(my_tuple[0])  # Output: 1
print(my_tuple[-1])  # Output: 5
```

Inmutabilidad y operaciones básicas

A diferencia de las listas, las tuplas son inmutables, lo que significa que no se pueden modificar una vez creadas. Sin embargo, es posible realizar algunas operaciones básicas, como obtener la longitud de una tupla o concatenar dos tuplas.

Ejemplo:

```python
my_tuple = (1, 2, 3, 4, 5)

# my_tuple[0] = 0  # Esto generará un error, ya que las tuplas son inmutables

length = len(my_tuple)
print(length)  # Output: 5

concatenated_tuple = my_tuple + (6, 7, 8)
print(concatenated_tuple)  # Output: (1, 2, 3, 4, 5, 6, 7, 8)
```

4. Ejercicios prácticos de listas y tuplas

1. Crear una lista con los números del 1 al 10 y utilizar un bucle for para imprimir cada número.
2. Crear una función que reciba una lista de números y devuelva una nueva lista con solo los números pares.
3. Crear una función que reciba una lista de strings y devuelva una tupla con dos listas: una que contenga los strings de longitud par y otra con los strings de longitud impar.
4. Utilizar la comprensión de listas para crear una lista con los cuadrados de los números del 1 al 20.
5. Crear una lista de tuplas que contengan el nombre y la edad de cinco personas. Utilizar un bucle for para imprimir solo los nombres de las personas mayores de 18 años.
6. Crear una función que reciba dos listas de números y devuelva una lista con los elementos que están presentes en ambas listas.
7. Crear una función que reciba una lista de números y devuelva una tupla con el menor y el mayor número de la lista.
8. Crear una función que reciba una lista de palabras y devuelva un diccionario con la cantidad de veces que aparece cada palabra en la lista.
9. Crear una función que reciba una lista de números y devuelva una nueva lista con los números que están en posiciones impares de la lista original.
10. Crear una función que reciba una lista de tuplas con dos elementos y devuelva una nueva lista con las tuplas ordenadas por el segundo elemento de cada tupla.