07_arreglos

December 2, 2024

1 Lenguanjes de Programación

1.0.1 Universidad Politecnica Salesiana

1.0.2 Ingeniería en Ciencias de la Computación

1.0.3 Programación

1.1 Autores del Material

- **Instructor**: Ing. Pablo Torres
- **Contacto**: ptorresp@ups.edu.ec

1.2 A. TEMA: Arreglos(Listas) en Python - Vectores

1.2.1 Descripción General

En este tema, aprenderemos sobre los **arreglos**, conocidos como **listas** en Python. Las listas son estructuras de datos fundamentales que permiten almacenar colecciones de elementos ordenados y modificables. Exploraremos cómo crear listas, acceder a sus elementos, realizar operaciones básicas, y utilizar métodos incorporados para manipular y gestionar datos de manera eficiente.

1.2.2 Objetivos de Aprendizaje

- Comprender qué son las listas en Python y sus características principales.
- Aprender a crear y asignar listas a variables.
- Acceder y modificar elementos de una lista utilizando índices.
- Realizar operaciones básicas con listas, como concatenación y repetición.
- Utilizar métodos incorporados para agregar, eliminar y modificar elementos en una lista.
- Iterar sobre los elementos de una lista utilizando bucles.
- Aplicar buenas prácticas en el manejo de listas para escribir código eficiente y legible.

1.2.3 1. Introducción a las Listas

Las **listas** en Python son colecciones ordenadas y mutables que pueden contener elementos de diferentes tipos, como números, cadenas de texto, u otras listas. Se definen utilizando corchetes `[]`, y los elementos están separados por comas.

1.1. Creación de Listas Puedes crear listas de diversas maneras:

```
# Lista vacía
lista_vacia = []

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

# Lista de cadenas de texto
frutas = ["Manzana", "Banana", "Cereza"]

# Lista mixta
mixta = [1, "Hola", 3.14, True]
```

1.2. Acceso a Elementos en una Lista Cada elemento en una lista tiene un índice, comenzando desde `0`. Puedes acceder a elementos individuales utilizando corchetes `[]`.

```
frutas = ["Manzana", "Banana", "Cereza", "Dátil"]

# Primer elemento
print(frutas[0])  # Output: Manzana

# Último elemento
print(frutas[-1])  # Output: Dátil
```

1.3. Modificación de Elementos Las listas son **mutables**, lo que significa que puedes cambiar sus elementos después de haberlas creado.

```
frutas = ["Manzana", "Banana", "Cereza"]
frutas[1] = "Mango"
print(frutas)  # Output: ['Manzana', 'Mango', 'Cereza']
```

Output:

- Frase: “Python Es Genial!”
- Mayúsculas: 3, Minúsculas: 11
- Frase: “123 ABC abc!”
- Mayúsculas: 3, Minúsculas: 3

1.2.4 2. Operaciones Básicas con Listas

2.1. Concatenación de Listas Puedes unir dos o más listas utilizando el operador `+`.

```
lista1 = [1, 2, 3]
lista2 = [4, 5, 6]
concatenada = lista1 + lista2
print(concatenada)  # Output: [1, 2, 3, 4, 5, 6]
```

2.2. Repetición de Listas Puedes repetir una lista utilizando el operador `*`.

```
lista = ["Hola"]
repetida = lista * 3
print(repetida)  # Output: ['Hola', 'Hola', 'Hola']
```

2.3. Slicing (Corte) de Listas Puedes obtener una porción de una lista utilizando slicing.

“‘python numeros = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

2 Desde el índice 2 hasta el 5 (excluyendo el 5)

sublista = numeros[2:5] print(sublista) # Output: [2, 3, 4]

3 Desde el inicio hasta el índice 4 (excluyendo el 4)

sublista = numeros[:4] print(sublista) # Output: [0, 1, 2, 3]

4 Desde el índice 6 hasta el final

sublista = numeros[6:] print(sublista) # Output: [6, 7, 8, 9]

5 Cada segundo elemento

sublista = numeros[::2] print(sublista) # Output: [0, 2, 4, 6, 8]

5.0.1 3. Métodos Comunes de Listas

Python proporciona una variedad de **métodos incorporados** para facilitar la manipulación de listas.

3.1. Agregar Elementos

- `append()`: Añade un elemento al final de la lista.
- `insert()`: Inserta un elemento en una posición específica.

```
frutas = ["Manzana", "Banana"]

# Agregar al final
frutas.append("Cereza")
print(frutas)  # Output: ['Manzana', 'Banana', 'Cereza']

# Insertar en la posición 1
frutas.insert(1, "Mango")
print(frutas)  # Output: ['Manzana', 'Mango', 'Banana', 'Cereza']
```

3.2. Eliminar Elementos

- `remove()`: Elimina la primera ocurrencia de un elemento específico.
- `pop()`: Elimina un elemento en una posición específica y lo devuelve.
- `clear()`: Elimina todos los elementos de la lista.

```
frutas = ["Manzana", "Mango", "Banana", "Cereza"]

# Eliminar 'Mango'
frutas.remove("Mango")
print(frutas)  # Output: ['Manzana', 'Banana', 'Cereza']

# Eliminar el último elemento
ultimo = frutas.pop()
print(ultimo)   # Output: Cereza
print(frutas)  # Output: ['Manzana', 'Banana']

# Limpiar la lista
frutas.clear()
print(frutas)  # Output: []
```

3.3. Buscar Elementos

- `index()`: Devuelve el índice de la primera ocurrencia de un elemento.
- `count()`: Cuenta cuántas veces aparece un elemento en la lista.

```
frutas = ["Manzana", "Banana", "Cereza", "Banana"]

# Índice de 'Cereza'
indice = frutas.index("Cereza")
print(indice)  # Output: 2

# Contar 'Banana'
contador = frutas.count("Banana")
print(contador)  # Output: 2
```

3.4. Ordenar y Revertir

- `sort()`: Ordena la lista en orden ascendente.
- `reverse()`: Invierte el orden de los elementos en la lista.

```
numeros = [5, 2, 9, 1, 5, 6]

# Ordenar
numeros.sort()
print(numeros)  # Output: [1, 2, 5, 5, 6, 9]

# Revertir
numeros.reverse()
print(numeros)  # Output: [9, 6, 5, 5, 2, 1]
```

5.0.2 4. Iteración sobre Listas

Puedes **iterar** sobre cada elemento de una lista utilizando bucles como `for`.

```
frutas = ["Manzana", "Banana", "Cereza"]

# Iterar sobre elementos
for fruta in frutas:
    print(fruta)

# Output:
# Manzana
# Banana
# Cereza
```

También puedes iterar sobre los índices utilizando `range()` y `len()`.

```
frutas = ["Manzana", "Banana", "Cereza"]

# Iterar sobre índices
for i in range(len(frutas)):
    print(f"Índice {i}: {frutas[i]}")

# Output:
# Índice 0: Manzana
# Índice 1: Banana
# Índice 2: Cereza
```

5.0.3 5. Comprensión de Listas (List Comprehensions)

La **comprensión de listas** es una forma concisa de crear listas basadas en iteraciones y condiciones.

“‘python # Crear una lista de cuadrados cuadrados = [x**2 for x in range(10)] print(cuadrados) # Output: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

6 Filtrar números pares

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

6.1 B. Ejercicios Prácticos

A continuación, se presentan **5 ejercicios** que te permitirán practicar y consolidar tu comprensión sobre las listas en Python.

6.1.1 Ejercicio 1: Función para Encontrar el Elemento Máximo en una Lista

Descripción: Crea una función que encuentre y devuelva el elemento máximo en una lista de números proporcionada por el usuario.

Instrucciones: 1. Define una función llamada `encontrar_maximo` que reciba una lista de números. 2. Itera sobre los elementos de la lista para encontrar el número más grande. 3. Devuelve el número máximo encontrado. 4. Llama a la función con diferentes listas y muestra los resultados.

Output:

```
Lista: [3, 7, 2, 9, 4] -> Máximo: 9
Lista: [10, 20, 30] -> Máximo: 30
Lista: [-5, -1, -10] -> Máximo: -1
Lista: [] -> Máximo: None
```

6.1.2 Ejercicio 2: Función para Eliminar Elementos Duplicados en una Lista

Descripción: Crea una función que elimine los elementos duplicados de una lista y devuelva una nueva lista con solo elementos únicos.

Instrucciones: 1. Define una función llamada `eliminar_duplicados` que reciba una lista. 2. Itera sobre los elementos de la lista original y agrega solo los elementos que no hayan sido añadidos previamente. 3. Devuelve la nueva lista sin duplicados. 4. Llama a la función con diferentes listas y muestra los resultados.

Output:

```
Original: [1, 2, 2, 3, 4, 4, 5] -> Sin duplicados: [1, 2, 3, 4, 5]
Original: ['manzana', 'banana', 'manzana', 'cereza'] -> Sin duplicados: ['manzana', 'banana',
Original: [True, False, True, True] -> Sin duplicados: [True, False]
Original: [] -> Sin duplicados: []
```

6.1.3 Ejercicio 3: Función para Verificar si Dos Listas son Iguales

Descripción: Crea una función que verifique si dos listas son iguales, es decir, tienen los mismos elementos en el mismo orden.

Instrucciones: 1. Define una función llamada `listas_iguales` que reciba dos listas. 2. Compara la longitud de ambas listas; si son diferentes, retorna `False`. 3. Itera sobre los elementos y verifica que cada elemento en una posición sea igual al de la otra lista. 4. Retorna `True` si todas las comparaciones son iguales, de lo contrario, `False`. 5. Llama a la función con diferentes pares de listas y muestra los resultados.

Output:

```
Lista1: [1, 2, 3] | Lista2: [1, 2, 3] -> Iguales
Lista1: ['a', 'b', 'c'] | Lista2: ['a', 'b', 'c'] -> Iguales
Lista1: [True, False] | Lista2: [True, True] -> Diferentes
Lista1: [1, 2] | Lista2: [1, 2, 3] -> Diferentes
Lista1: [] | Lista2: [] -> Iguales
```

6.1.4 Ejercicio 4: Función para Invertir una Lista

Descripción: Crea una función que invierta el orden de los elementos en una lista sin utilizar métodos incorporados como `reverse()`.

Instrucciones: 1. Define una función llamada `invertir_lista` que reciba una lista. 2. Crea una nueva lista vacía. 3. Itera sobre la lista original desde el último elemento hasta el primero y agrega cada elemento a la nueva lista. 4. Devuelve la lista invertida. 5. Llama a la función con diferentes listas y muestra los resultados.

Output:

```
Original: [1, 2, 3, 4, 5] -> Invertida: [5, 4, 3, 2, 1]
Original: ['manzana', 'banana', 'cereza'] -> Invertida: ['cereza', 'banana', 'manzana']
Original: [True, False, True] -> Invertida: [True, False, True]
Original: [] -> Invertida: []
```

6.1.5 Ejercicio 5: Función para Calcular la Suma de Elementos en una Lista

Descripción: Crea una función que calcule y devuelva la suma de todos los elementos numéricos en una lista.

Instrucciones: 1. Define una función llamada `sumar_elementos` que reciba una lista de números. 2. Inicializa una variable para almacenar la suma. 3. Itera sobre los elementos de la lista y acumula su valor en la variable suma. 4. Devuelve el resultado final. 5. Llama a la función con diferentes listas y muestra los resultados.

Output:

```
Lista: [10, 20, 30] -> Suma: 60
Lista: [1.5, 2.5, 3.0] -> Suma: 7.0
Lista: [-5, 15, 10] -> Suma: 20
Lista: [] -> Suma: 0
```

6.2 C. Resumen de Conceptos Clave

1. **Listas**: Colecciones ordenadas y mutables que pueden contener elementos de diferentes tipos.
2. **Creación de Listas**: Definidas utilizando corchetes `[]` y elementos separados por comas.
3. **Acceso y Modificación**: Acceder a elementos mediante índices y modificarlos directamente.
4. **Operaciones Básicas**: Concatenación (`+`), repetición (`*`), y slicing para obtener sublistas.
5. **Métodos Comunes**: `append()`, `insert()`, `remove()`, `pop()`, `clear()`, `index()`, `count()`, `sort()`, y `reverse()`.
6. **Iteración**: Utilizar bucles `for` para recorrer elementos o índices.
7. **Comprensión de Listas**: Crear listas de manera concisa y eficiente utilizando expresiones dentro de una estructura de lista.
8. **Mutabilidad**: Las listas pueden ser modificadas después de su creación.

6.3 D. Recursos Adicionales

- Documentación Oficial de Python - Estructuras de Decisión
- Tutorial de Estructuras de Decisión en Real Python

- Ejemplos de Estructuras de Decisión en W3Schools

6.4 E. Referencias

1. Van Rossum, G. (1991). *Python Tutorial*. Python Software Foundation.
2. Lutz, M. (2013). *Learning Python*. O'Reilly Media.
3. Sweigart, A. (2015). *Automate the Boring Stuff with Python*. No Starch Press.
