Estructuras de dato en Python y sentencias iterativas

M2: FUNDAMENTOS DE INGENIERÍA DE DATOS

AE1: CODIFICAR UNA RUTINA UTILIZANDO ESTRUCTURAS DE DATO Y SENTENCIAS ITERATIVAS ACORDE AL LENGUAJE PYTHON PARA RESOLVER UN PROBLEMA DE MEDIANA COMPLEJIDAD.

alkemy

Introducción 👩‍💻

Python es un lenguaje de programación multiparadigma y de alto nivel que ha ganado popularidad por su sintaxis clara y legible. Una de las características que lo hacen tan poderoso es su amplia variedad de estructuras de datos incorporadas, como listas, diccionarios, tuplas y sets. Estas estructuras permiten almacenar y manipular datos de manera eficiente y flexible, adaptándose a diversas necesidades en el desarrollo de aplicaciones.

En esta lección, exploraremos en profundidad estas estructuras de datos fundamentales. Aprenderemos cómo crearlas, modificarlas y utilizarlas en diferentes contextos. Además, abordaremos las sentencias iterativas como `while` y `for`, que son esenciales para recorrer y manipular colecciones de datos. A través de ejemplos prácticos, tablas comparativas y ejercicios, te brindaremos las herramientas necesarias para dominar estos conceptos y aplicarlos en tus proyectos de programación.

Aprendizajes esperados

Cuando finalices la lección serás capaz de:

- Comprender las características y usos de listas, diccionarios, tuplas y sets en Python.
- Crear y manipular listas, incluyendo agregar y rescatar elementos o rangos de elementos.
- Entender la relación entre listas y cadenas de caracteres, así como trabajar con listas anidadas y matrices.
- Crear y manejar diccionarios, incluyendo cómo agregar y acceder a elementos, y trabajar con diccionarios anidados.
- Comprender las características y usos de las tuplas, incluyendo su creación y cómo acceder a sus elementos, además de empacar y desempaquetar tuplas.
- Crear y utilizar sets, y realizar operaciones de conjunto con ellos.
- Entender qué son las sentencias iterativas y su importancia en la programación.
- Utilizar las sentencias `while` y `for` para iterar sobre diferentes estructuras de datos.
- Iterar listas y diccionarios de elementos eficientemente.
- Comprender y utilizar la función `range` para controlar ciclos de iteración.

Desarrollo

LISTAS

Las listas en Python son estructuras de datos ordenadas y mutables que permiten almacenar colecciones de elementos de diferentes tipos. Son equivalentes a los arreglos en otros lenguajes de programación, pero con mayor flexibilidad.

CARACTERÍSTICAS DE UNA LISTA

- **Ordenadas**: Mantienen el orden en que se agregan los elementos.
- **Mutables**: Los elementos pueden ser modificados después de la creación de la lista.
- **Heterogéneas**: Pueden contener elementos de distintos tipos (enteros, cadenas, objetos, etc.).
- **Indexación**: Los elementos se acceden mediante índices, comenzando desde 0.

CREAR UNA LISTA, AGREGAR, RESCATAR UN ELEMENTO

Para crear una lista, se utilizan corchetes `[]` o la función `list()`.

```
mi_lista = [1, "dos", 3.0, True]
```

Agregar elementos:

`append()`: Agrega un elemento al final de la lista.

```
mi_lista.append("nuevo elemento")
```

`insert()`: Inserta un elemento en una posición específica.

```
mi_lista.insert(2, "insertado en posición 2")
```

Rescatar un elemento:

Acceso por índice:

```
elemento = mi_lista[0]  # Primer elemento
```

Índices negativos para acceder desde el final:

```
ultimo_elemento = mi_lista[-1]
```

RESCATAR UN RANGO DE ELEMENTOS

Utilizando el slicing (rebanado), puedes obtener sublistas:

```
sub_lista = mi_lista[1:3]  # Elementos desde el índice 1 hasta el 2
```

Ejemplos de slicing:

- **`mi_lista[:3]`**: Desde el inicio hasta el índice 2.
- **`mi_lista[2:]`**: Desde el índice 2 hasta el final.
- **`mi_lista[-3:-1]`**: Desde el tercer último hasta el penúltimo elemento.

LISTAS Y CADENAS DE CARACTERES

Las cadenas de caracteres (`str`) pueden ser tratadas como listas de caracteres.

```
texto = "Python"
letra = texto[0]  # 'P'
sub_texto = texto[1:4]  # 'yth'
```

Además, puedes convertir cadenas en listas y viceversa:

`list()`: Convierte una cadena en una lista de caracteres.

```
lista_caracteres = list(texto)
```

`join()`: Une elementos de una lista en una cadena.

```
nueva_cadena = ''.join(lista_caracteres)
```

LISTAS ANIDADAS Y MATRICES

Una lista puede contener otras listas, permitiendo crear estructuras multidimensionales como matrices.

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
```

```
elemento = matriz[1][2]  # Accede al elemento en fila 2, columna 3 (valor 6)
```

Las listas anidadas son útiles para representar tablas de datos, gráficos y más.

DICCIONARIOS

Los diccionarios son colecciones desordenadas de pares clave-valor. Son mutables y permiten un acceso rápido a los valores a través de sus claves únicas.

CARACTERÍSTICAS DE UN DICCIONARIO

- **Desordenados**: No mantienen un orden específico (aunque desde Python 3.7 se preserva el orden de inserción).
- **Mutables**: Pueden modificarse después de su creación.
- **Claves únicas**: Cada clave es única y se utiliza para acceder a su valor asociado.
- **Claves inmutables**: Las claves deben ser tipos inmutables (cadenas, números, tuplas inmutables).

CREAR UN DICCIONARIO, AGREGAR, RESCATAR ELEMENTOS

Crear un diccionario:

```
mi_diccionario = {'nombre': 'Juan', 'edad': 30, 'ciudad': 'Madrid'}
```

Agregar o modificar elementos:

```
mi_diccionario['profesión'] = 'Ingeniero'  # Agregar nueva clave-valor
mi_diccionario['edad'] = 31  # Modificar valor existente
```

Rescatar elementos:

Acceder por clave:

```
nombre = mi_diccionario['nombre']
```

Método `get()` para evitar errores si la clave no existe:

```
pais = mi_diccionario.get('país', 'No especificado')
```

DICCIONARIOS ANIDADOS

Los diccionarios pueden contener otros diccionarios, lo que permite estructuras más complejas.

```
empleados = {
    'emp1': {'nombre': 'Ana', 'edad': 28},
    'emp2': {'nombre': 'Luis', 'edad': 35}
}

edad_emp1 = empleados['emp1']['edad']  # 28
```

Esta estructura es útil para representar objetos con múltiples atributos.

TUPLAS

Las tuplas son colecciones ordenadas e inmutables. Una vez creadas, no pueden ser modificadas, lo que las hace ideales para datos constantes.

CARACTERÍSTICAS DE UNA TUPLA

- **Ordenadas**: Mantienen el orden de los elementos.
- **Inmutables**: No pueden cambiar después de su creación.
- **Heterogéneas**: Pueden contener elementos de distintos tipos.
- **Eficientes**: Consumen menos memoria que las listas.

CREACIÓN DE UNA TUPLA, RESCATAR ELEMENTOS

Crear una tupla:

```
mi_tupla = (1, 'dos', 3.0)
```

También se pueden crear sin paréntesis (tuple packing):

```
mi_tupla = 1, 'dos', 3.0
```

Rescatar elementos:

Acceso por índice:

```
elemento = mi_tupla[0]  # 1
```

Slicing:

```
sub_tupla = mi_tupla[1:]  # ('dos', 3.0)
```

EMPAQUETADO Y DESEMPAQUETADO DE TUPLAS

Empaquetado: Asignar múltiples valores a una tupla.

```
tupla = 1, 2, 3
```

Desempaquetado: Asignar elementos de una tupla a variables.

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

Si el número de variables es menor, se puede usar * para agrupar el resto:

```
a, *b = tupla
print(a)  # 1
print(b)  # [2, 3]
```

SETS

Los sets son colecciones desordenadas de elementos únicos. Se utilizan para operaciones de conjuntos matemáticos.

CARACTERÍSTICAS DE UN SET

- **Desordenados**: No mantienen un orden específico.
- **Elementos únicos**: No permiten duplicados.
- **Mutables**: Pueden agregar o eliminar elementos.

CREACIÓN DE UN SET

Crear un set:

```
mi_set = {1, 2, 3}
```

O usando la función `set()`:

```
mi_set = set([1, 2, 2, 3])  # Duplicados se eliminan
```

Agregar elementos:

python
Copiar código

```
mi_set.add(4)
```

Eliminar elementos:

python

Copiar código

```
mi_set.remove(2)
```

OPERACIONES DE CONJUNTO CON SETS

Los sets permiten realizar operaciones como unión, intersección y diferencia.

Unión (`|`):

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union = set1 | set2  # {1, 2, 3, 4, 5}
```

Intersección (`&`):

```
interseccion = set1 & set2  # {3}
```

Diferencia (`-`):

```
diferencia = set1 - set2  # {1, 2}
```

Diferencia simétrica (`^`):

```
dif_simetrica = set1 ^ set2  # {1, 2, 4, 5}
```

QUÉ ES UNA SENTENCIA ITERATIVA Y POR QUÉ SE NECESITAN

Las sentencias iterativas permiten ejecutar un bloque de código repetidamente. Son esenciales para:

- **Recorrer estructuras de datos**: Como listas, diccionarios, tuplas y sets.
- **Automatizar tareas repetitivas**: Evitando código redundante.
- **Controlar flujos de ejecución**: Basados en condiciones dinámicas.

Las dos sentencias iterativas principales en Python son `while` y `for`.

LA SENTENCIA WHILE

La sentencia `while` ejecuta un bloque de código mientras una condición sea verdadera.

Sintaxis:

```
while condición:
    # Bloque de código
```

Ejemplo:

```
contador = 0
while contador < 5:
    print(f"El contador es {contador}")
    contador += 1
```

Salida:

```
El contador es 0
El contador es 1
El contador es 2
El contador es 3
El contador es 4
```

Precaución: Asegurarse de que la condición pueda volverse falsa para evitar bucles infinitos.

LA SENTENCIA FOR

La sentencia `for` se utiliza para iterar sobre una secuencia (lista, tupla, diccionario, set o cadena).

Sintaxis:

```
for elemento in secuencia:
    # Bloque de código
```

Ejemplo:

```
frutas = ['manzana', 'banana', 'cereza']
for fruta in frutas:
    print(f"Me gusta la {fruta}")
```

Salida:

```
Me gusta la manzana
Me gusta la banana
Me gusta la cereza
```

ITERANDO LISTAS DE ELEMENTOS

Puedes iterar sobre listas para acceder o manipular sus elementos.

Ejemplo:

```
numeros = [1, 2, 3, 4, 5]
for numero in numeros:
    cuadrado = numero ** 2
    print(f"El cuadrado de {numero} es {cuadrado}")
```

ITERANDO DICCIONARIOS DE ELEMENTOS

Al iterar sobre un diccionario, por defecto se recorren las claves. Puedes acceder a claves y valores utilizando métodos específicos.

Ejemplo:

```
capitales = {'España': 'Madrid', 'Francia': 'París', 'Italia':
'Roma'}

for pais, ciudad in capitales.items():
    print(f"La capital de {pais} es {ciudad}")
```

Métodos útiles:

- **`keys()`**: Devuelve las claves.
- **`values()`**: Devuelve los valores.
- **`items()`**: Devuelve pares clave-valor.

LA FUNCIÓN RANGE

La función `range()` genera una secuencia de números, muy útil en bucles `for`.

Sintaxis:

- **`range(stop)`**: Desde 0 hasta `stop` - 1.
- **`range(start, stop[, step])`**: Desde `start` hasta `stop` - 1, con incrementos de `step`.

Ejemplo:

```
for i in range(5):
    print(i)
```

Salida:

```
0
1
2
```

```
3
4
```

ITERANDO CON LA FUNCIÓN RANGE

Puedes combinar `for` y `range()` para iterar un número específico de veces.

Ejemplo:

```
for i in range(1, 11):
    print(f"Tabla del 5: 5 x {i} = {5 * i}")
```

Cierre ✍️

Las estructuras de datos y las sentencias iterativas son componentes fundamentales en la programación con Python. Las listas, diccionarios, tuplas y sets te permiten almacenar y manipular datos de formas diversas y eficientes. Comprender sus características y cómo utilizarlos es esencial para desarrollar aplicaciones robustas y flexibles.

Las sentencias `while` y `for`, junto con la función `range()`, te proporcionan las herramientas necesarias para recorrer y procesar estas estructuras de datos. Al dominar estos conceptos, podrás escribir código más limpio, eficiente y fácil de mantener.

Te animamos a practicar estos conceptos con ejercicios y proyectos propios. La práctica constante te ayudará a internalizar estas herramientas y a descubrir nuevas formas de aplicarlas en tus desarrollos.

Referencias 📚

- **Documentación oficial de Python**: https://docs.python.org/es/3/
    - Referencia completa y en español sobre las características del lenguaje y sus bibliotecas estándar.
- **Tutorial de Python en W3Schools**: https://www.w3schools.com/python/
    - Recursos interactivos y ejemplos prácticos para aprender Python.
- **Programación en Python de manera efectiva** por Luciano Ramalho.
    - Libro que profundiza en el uso avanzado de Python y sus buenas prácticas.
- **Aprenda a pensar como un programador con Python** por Allen Downey.
    - Enfoque en el pensamiento computacional y la resolución de problemas utilizando Python.
- **Real Python**: https://realpython.com/
    - Artículos y tutoriales detallados sobre diversos temas de Python.

¡Muchas gracias!

Nos vemos en la próxima lección