


Lectura sesión Estructuras de Datos y Sentencias Iterativas

En la programación, una estructura de datos es una forma de organizar y almacenar la información para que pueda ser utilizada de manera eficiente. Python, al ser un lenguaje moderno y versátil, proporciona una variedad de estructuras de datos incorporadas que permiten trabajar con diferentes tipos de colecciones: secuencias, asociaciones clave-valor, agrupaciones inmutables y conjuntos no ordenados.

 **por Kibernetum Capacitación S.A.**

Estructuras de datos fundamentales en Python

En esta sección aprenderás a utilizar cuatro de las estructuras de datos fundamentales en Python:

- Listas, para almacenar elementos ordenados y modificables.
- Diccionarios, para asociar claves con valores.
- Tuplas, para representar colecciones ordenadas que no deben cambiarse.
- Sets (conjuntos), para manejar agrupaciones sin duplicados.

Cada una de estas estructuras tiene características únicas que las hacen más adecuadas para ciertas tareas. Comprenderlas y saber cuándo utilizarlas es una habilidad esencial para escribir código limpio, claro y eficiente.

Listas en Python

¿Qué es una lista?

Una lista en Python es una colección ordenada de elementos que se pueden modificar. Es una de las estructuras más utilizadas gracias a su versatilidad. Puede contener números, cadenas, otras listas, e incluso objetos personalizados.

Características clave:

- Ordenada: los elementos tienen un índice que empieza desde 0.
- Mutable: se puede modificar después de creada.
- Permite duplicados.
- Usa corchetes [] para su declaración.

Ejemplos prácticos

Crear una lista

```
frutas = ['manzana', 'banana', 'naranja']
print(frutas)
```

Acceder a elementos

```
print(frutas[0])    # manzana
print(frutas[-1])   # naranja (último)
```

Agregar elementos

```
frutas.append('pera')    # Agrega al final
frutas.insert(1, 'uva')   # Inserta en la posición 1
```

Modificar elementos

```
frutas[0] = 'durazno'
```

Eliminar elementos

```
frutas.remove('banana')
del frutas[2]
```

Listas anidadas (matrices)

```
matriz = [[1, 2], [3, 4], [5, 6]]
print(matriz[1][0])  # 3
```

Diccionarios en Python

¿Qué es un diccionario?

Un diccionario en Python es una colección de pares clave-valor, donde cada clave está asociada a un valor. Es muy útil cuando necesitas acceder a información por nombre o etiqueta en lugar de un índice numérico.

Características clave:

- No ordenado (hasta Python 3.6).
- Clave única e inmutable.
- Mutable.
- Usa llaves {}.

Ejemplos prácticos

Crear un diccionario

```
persona = {'nombre': 'Ana', 'edad': 30}
```

Acceder a valores

```
print(persona['nombre'])    # Ana
print(persona.get('edad'))  # 30
```

Agregar/modificar claves

```
persona['ciudad'] = 'Santiago'
persona['edad'] = 31
```

Eliminar claves

```
del persona['nombre']
```

Diccionario anidado

```
alumno = {
    'nombre': 'Carlos',
    'notas': {'matemáticas': 6.5, 'historia': 5.8}
}
print(alumno['notas']['historia'])  # 5.8
```

Tuplas en Python

¿Qué es una tupla?

Una tupla es similar a una lista, pero inmutable. Es útil para almacenar datos que no deberían cambiar durante la ejecución del programa, como coordenadas o constantes.

Características clave:

- Ordenada.
- Inmutable.
- Permite duplicados.
- Usa paréntesis ().

Ejemplos prácticos

Crear una tupla

```
coordenadas = (10.5, 20.3)
```

Acceder a elementos

```
print(coordenadas[0]) # 10.5
```

Empaquetado y desempaquetado

```
persona = ('Luis', 28, 'Chile')  
nombre, edad, pais = persona  
print(nombre) # Luis
```

Nota: Aunque la tupla no se puede modificar, sí puedes reasignar la variable.

Sets (Conjuntos) en Python

¿Qué es un set?

Un set es una colección no ordenada de elementos únicos. Es útil para realizar operaciones matemáticas de conjuntos como unión, intersección y diferencia.

Características clave:

- No ordenado.
- No permite duplicados.
- Mutable.
- Usa {} o set().

Ejemplos prácticos

Crear un set

```
colores = {'rojo', 'verde', 'azul'}
```

Agregar elementos

```
colores.add('amarillo')
```

Eliminar elementos

```
colores.discard('verde')
```

Operaciones de conjuntos

```
a = {1, 2, 3}
b = {3, 4, 5}

print(a | b)  # Unión → {1, 2, 3, 4, 5}
print(a & b)  # Intersección → {3}
print(a - b)  # Diferencia → {1, 2}
```

Operaciones con conjuntos (sets) en Python

Este código muestra cómo usar tres operaciones básicas entre conjuntos (set) en Python:

- $a \mid b$: Unión – combina todos los elementos de a y b, sin duplicados.
- $a \& b$: Intersección – devuelve los elementos que están en ambos conjuntos.
- $a - b$: Diferencia – devuelve los elementos que están en a pero no en b.

Gracias a estas operaciones, los conjuntos son ideales cuando trabajamos con agrupaciones sin repeticiones y queremos comparar sus contenidos fácilmente.

Resumen de estructuras de datos en Python

Estructura	Ordenada	Mutable	Permite duplicados	Sintaxis
Lista	✓	✓	✓	[]
Diccionario	✗ (3.6-)	✓	✗ (claves únicas)	{}
Tupla	✓	✗	✓	()
Set	✗	✓	✗	{}/set()

Esta tabla compara las principales estructuras de datos en Python según sus propiedades. Es una excelente referencia rápida para decidir qué estructura usar en cada caso:

- Ordenada: los elementos mantienen el orden en que fueron agregados.
- Mutable: permite modificar su contenido después de ser creada.
- Permite duplicados: puede contener elementos repetidos.

Por ejemplo:

- Usa listas cuando necesites una secuencia ordenada que pueda modificarse.
- Prefiere diccionarios si necesitas asociar claves únicas a valores.
- Usa tuplas para datos que no deben cambiar, como coordenadas o constantes.
- Elige sets cuando quieras asegurarte de que los elementos no se repitan y necesites realizar operaciones de conjuntos como unión e intersección.

Sentencias Iterativas en Python

En muchos programas, necesitamos realizar una misma acción varias veces. Esto puede incluir recorrer los elementos de una lista, repetir cálculos hasta que se cumpla una condición, o ejecutar un bloque de instrucciones una cantidad específica de veces.

Para este tipo de situaciones, Python nos ofrece las sentencias iterativas, también conocidas como bucles. Estas estructuras permiten repetir instrucciones de manera automática, reduciendo la necesidad de código duplicado y mejorando la eficiencia.

En esta sección aprenderás a utilizar:

- El bucle while, que se repite mientras una condición sea verdadera.
- El bucle for, ideal para recorrer colecciones de datos.
- La función range(), que permite generar secuencias numéricas útiles en iteraciones.
- Cómo iterar listas y diccionarios de forma limpia y eficiente.

¿Qué es una sentencia iterativa?

Una sentencia iterativa es una instrucción que ejecuta repetidamente un bloque de código mientras se cumpla una condición, o por cada elemento de una colección.

¿Por qué se necesitan?

- Automatizan tareas repetitivas.
- Evitan código duplicado.
- Permiten recorrer colecciones como listas o diccionarios.

Sentencia while

¿Qué es?

while ejecuta un bloque de código mientras una condición sea verdadera. Debe tener cuidado de no generar bucles infinitos.

Ejemplo con buenas prácticas:

```
contador = 0
limite = 5

while contador < limite:
    print(f"Contador: {contador}")
    contador += 1
```

Buenas prácticas:

- Asegúrate de modificar la condición dentro del bucle.
- Usa nombres de variables claros.
- Si necesitas romper el bucle anticipadamente, usa break.

Sentencia for y función range()

Sentencia for

¿Qué es?

for se utiliza para recorrer elementos de una secuencia como una lista, tupla, cadena o diccionario.

Ejemplo recorriendo una lista:

```
frutas = ['manzana', 'banana', 'naranja']

for fruta in frutas:
    print(f"Me gusta la {fruta}")
    ....
```

Iterar con range()

¿Qué es?

range() genera una secuencia de números que puedes usar en un bucle for. Muy útil cuando necesitas repetir algo un número fijo de veces.

Ejemplos:

```
# De 0 a 4
for i in range(5):
    print(f"Iteración {i}")

# De 1 a 5
for i in range(1, 6):
    print(f"Número: {i}")

# De 10 a 2, en pasos de -2
for i in range(10, 1, -2):
    print(i)
    ....
```

Iterando colecciones en Python

Iterar listas con for

Ejemplo práctico:

```
lenguajes = ['Python', 'JavaScript', 'C#']

for lenguaje in lenguajes:
    print(f"Lenguaje: {lenguaje}")
    ....
```

Iterar con índice usando enumerate:

```
for index, lenguaje in enumerate(lenguajes):
    print(f"{index + 1}. {lenguaje}")
    ....
```

Iterar diccionarios

Recorriendo claves:

```
persona = {'nombre': 'Ana', 'edad': 30}

for clave in persona:
    print(clave)
    ....
```

Clave y valor con items():

```
for clave, valor in persona.items():
    print(f"{clave.capitalize()}: {valor}")
    ....
```

Buenas prácticas y actividad práctica

Buenas prácticas (Clean Code):

- Usa nombres descriptivos (nunca x, y, data si puedes evitarlo).
- Evita repetir código dentro del bucle.
- Controla condiciones para evitar bucles infinitos.
- Prefiere for cuando iteras colecciones y while cuando no sabes cuántas veces se repetirá.

Actividad Práctica Guiada – Explorando Estructuras de Datos e Iteraciones en Python

Objetivo:

Aplicar estructuras de datos (listas, diccionarios, tuplas, sets) y sentencias iterativas (for, while, range) mediante la implementación de un sistema simple para registrar estudiantes, calcular promedios y clasificar resultados.

Paso a paso detallado:

1. Crear una lista vacía llamada estudiantes. Esta lista almacenará los datos de cada estudiante.
2. Crear un bucle while que permita ingresar los datos de varios estudiantes (nombre y 3 notas). Usa un diccionario para almacenar los datos de cada estudiante.

```
{
  "nombre": "Laura",
  "notas": [6.5, 7.0, 5.8],
  "promedio": 6.43
}
```

3. Al finalizar cada ingreso, preguntar si se desea agregar otro estudiante.
4. Una vez finalizado el ingreso, iterar la lista de estudiantes con for y mostrar:
 - Nombre del estudiante
 - Notas
 - Promedio
 - Mensaje: "Aprobado" si el promedio ≥ 4.0 , "Reprobado" si el promedio < 4.0
5. Opcional: mostrar totales y estadísticas al final
 - Total de estudiantes ingresados
 - Porcentaje de aprobados y reprobados

Esto permite practicar el uso de len(), contadores y condiciones dentro del ciclo.