04_estructuras_de_iteracion

November 13, 2024

1 Lenguanje 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 Temas

Estructuras de Iteración en Python E

- While
- For

1.2.1 Descripción General

En este tema, exploraremos las estructuras de iteración en Python, que nos permiten ejecutar bloques de código de manera repetitiva. Analizaremos las estructuras `while` y `for`, comprendiendo su sintaxis y aplicabilidad mediante ejemplos prácticos. Además, veremos cómo combinar estructuras de iteración con estructuras de decisión para resolver problemas más complejos.

1.2.2 Objetivos de Aprendizaje

- Comprender el uso de la estructura while en Python.
- Implementar estructuras for para iterar sobre secuencias.
- Combinar estructuras de iteración con estructuras de decisión.
- Aplicar las estructuras de iteración en la resolución de problemas

---

1.3 A. Estructuras de Iteración en Python

Las estructuras de iteración permiten que el programa ejecute repetidamente un bloque de código mientras se cumpla una condición o recorra una secuencia de elementos.

1.3.1 1.1. La Estructura `while`

Definición y Propósito

- **Definición**: Definición: La estructura `while` se utiliza para ejecutar un bloque de código mientras una condición específica se evalúa como verdadera.
- **Propósito**: Permite realizar repeticiones indefinidas hasta que una condición deje de cumplirse.

Sintaxis

```
while condición:
    # Bloque de código a ejecutar mientras la condición sea verdadera
```

Ejemplo de Uso

```
# Ejemplo de estructura while
contador = 1

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

En este ejemplo, el mensaje se imprimirá cinco veces, incrementando el valor de `contador` *en cada iteración.*

[14]:
```
contador = 1

while contador <= 5 or contador == 6:
    print(f"Contador: {contador}")
    contador = contador + 1
print(f"Contador: {contador}")
```

```
Contador: 1
Contador: 2
Contador: 3
Contador: 4
Contador: 5
Contador: 6
Contador: 7
```

1.3.2 1.2. La Estructura `for`

Definición y Propósito

- **Definición**: La estructura `for` se utiliza para iterar sobre los elementos de una secuencia (como listas, tuplas, cadenas, etc.) y ejecutar un bloque de código para cada elemento.
- **Propósito**: Facilita la iteración sobre colecciones de datos de manera eficiente y legible.### 1.2. La Estructura `for`

Definición y Propósito

- **Definición**: La estructura `for` se utiliza para iterar sobre los elementos de una secuencia (como listas, tuplas, cadenas, etc.) y ejecutar un bloque de código para cada elemento.
- **Propósito**: Facilita la iteración sobre colecciones de datos de manera eficiente y legible.

Ejemplo de Uso

```
# Ejemplo de estructura for
frutas = ["manzana", "banana", "cereza"]

for fruta in frutas:
    print(f"Fruta: {fruta}")
```

En este ejemplo, el mensaje se imprimirá una vez por cada elemento en la lista ***frutas****.*

Sintaxis

```
for elemento in secuencia:
    # Bloque de código a ejecutar para cada elemento
```

1.3.3 1.3. Combinación de Iteración y Decisión

Definición y Propósito

- **Definición**: Combinar estructuras de iteración (`while`, `for`) con estructuras de decisión (`if`, `elif`, `else`) para manejar condiciones dentro de ciclos.
- **Propósito**: Permite controlar el flujo de ejecución dentro de bucles basándose en condiciones dinámicas.

Ejemplo de Uso

```
# Ejemplo de combinación de for y if..elif..else
numeros = [1, 2, 3, 4, 5, 6]

for numero in numeros:
    if numero % 2 == 0:
        print(f"{numero} es par.")
    else:
        print(f"{numero} es impar.")
```

En este ejemplo, se iteran los números de la lista y se determina si cada número es par o impar.

1.4 B. Ejercicios Prácticos

1.4.1 Ejercicio 1: Tabla de Multiplicar

Descripción: Crea un programa que solicite al usuario que ingrese un número entero y luego muestre su tabla de multiplicar hasta el 10.

Instrucciones: 1. Utiliza la función `input()` para solicitar al usuario que ingrese un número entero. 2. Convierte la entrada a tipo `int` usando la función `int()`. 3. Utiliza un ciclo `for` para iterar del 1 al 10. 4. En cada iteración, calcula el producto del número ingresado por el iterador. 5. Imprime el resultado utilizando `print()` con una f-string.

[30]:
```
numero_ingresado = int(input("Ingrese un numero: "))
for numero_rango in range(1,11):
    resultado = numero_ingresado * numero_rango
    print(f"{numero_ingresado} x {numero_rango} = {resultado}")
```

```
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
```

[16]:
```
# Solicitar al usuario que ingrese un número entero
numero_str = input("Ingresa un número entero para ver su tabla de multiplicar:␣
↪")
numero = int(numero_str)

# Mostrar la tabla de multiplicar
print(f"Tabla de multiplicar del {numero}:")
for i in range(10):

    print(f"{i} ")
```

```
Tabla de multiplicar del 10:
0
1
2
3
4
5
6
7
```

8
9

1.4.2 Ejercicio 2: Suma de Números Pares

Descripción: Crea un programa que calcule la suma de todos los números pares entre 1 y un número ingresado por el usuario.

Instrucciones: 1. Utiliza la función `input()` para solicitar al usuario que ingrese un número entero. 2. Convierte la entrada a tipo `int` usando la función `int()`. 3. Inicializa una variable para la suma. 4. Utiliza un ciclo `for` para iterar desde 1 hasta el número ingresado. 5. En cada iteración, verifica si el número es par. 6. Si es par, suma el número a la variable de suma. 7. Imprime el resultado utilizando `print()` con una f-string.

```
# Solicitar al usuario que ingrese un número entero
limite_str = input("Ingresa un número entero para sumar los números pares hasta él: ")
limite = int(limite_str)

# Inicializar la suma
suma_pares = 0

# Iterar desde 1 hasta el límite
for numero in range(1, limite + 1):
    if numero % 2 == 0:
        suma_pares += numero

# Mostrar el resultado
print(f"La suma de los números pares hasta {limite} es: {suma_pares}")
```

[ ]:
```
numero_str = input("Ingresa un número entero para ver su tabla de multiplicar:␣
↪")
numero= int(numero_str)
resultado = 0
for num in range(0,numero_str+1,2):
    resultado = resultado + num
print(f"La suma de los números pares es: {resultado}")
```

[3]:
```
# Solicitar al usuario que ingrese un número entero
limite_str = input("Ingresa un número entero para sumar los números pares hasta␣
↪él: ")
limite = int(limite_str)

# Inicializar la suma
suma_pares = 0

# Iterar desde 1 hasta el límite
for numero in range(1, limite + 1):
    if numero % 2 == 0:
        suma_pares += numero
```

```
# Mostrar el resultado
print(f"La suma de los números pares hasta {limite} es: {suma_pares}")
```

```
La suma de los números pares hasta 32 es: 272
```

1.4.3 Ejercicio 4: Factorial de un Número

Descripción: Crea un programa que calcule el factorial de un número ingresado por el usuario utilizando un ciclo `while`.

Instrucciones: 1. Utiliza la función `input()` para solicitar al usuario que ingrese un número entero positivo. 2. Convierte la entrada a tipo `int` usando la función `int()`. 3. Inicializa una variable para el factorial con el valor 1. 4. Utiliza un ciclo `while` para multiplicar la variable factorial por cada número desde 1 hasta el número ingresado. 5. Imprime el resultado utilizando `print()` con una f-string.

```
# Solicitar al usuario que ingrese un número entero positivo
numero_str = input("Ingresa un número entero positivo para calcular su factorial: ")
numero = int(numero_str)

# Inicializar el factorial
factorial = 1
contador = 1

# Calcular el factorial usando while
while contador <= numero:
    factorial *= contador
    contador += 1

# Mostrar el resultado
print(f"El factorial de {numero} es: {factorial}")
```

[4]:
```
# Solicitar al usuario que ingrese un número entero positivo
numero_str = input("Ingresa un número entero positivo para calcular su␣
 ↪factorial: ")
numero = int(numero_str)

# Inicializar el factorial
factorial = 1
contador = 1

# Calcular el factorial usando while
while contador <= numero:
    factorial *= contador
    contador += 1

# Mostrar el resultado
print(f"El factorial de {numero} es: {factorial}")
```

```
El factorial de 34 es: 295232799039604140847618609643520000000
```

1.4.4 Ejercicio 4: Contador de Vocales

Descripción: Crea un programa que solicite al usuario que ingrese una cadena de texto y cuente el número de vocales presentes en la cadena.

Instrucciones: 1. Utiliza la función `input()` para solicitar al usuario que ingrese una cadena de texto. 2. Inicializa un contador para las vocales. 3. Utiliza un ciclo `for` para iterar sobre cada carácter de la cadena. 4. Utiliza una estructura `if` para verificar si el carácter es una vocal. 5. Incrementa el contador cada vez que se encuentre una vocal. 6. Imprime el número total de vocales utilizando `print()` con una f-string.

```
# Solicitar al usuario que ingrese una cadena de texto
texto = input("Ingresa una cadena de texto: ")

# Inicializar el contador de vocales
contador_vocales = 0

# Definir las vocales
vocales = "aeiouAEIOU"

# Iterar sobre cada carácter en la cadena
for caracter in texto:
    if caracter in vocales:
        contador_vocales += 1

# Mostrar el resultado
print(f"El número total de vocales en la cadena es: {contador_vocales}")
```

[1]:
```
# Solicitar al usuario que ingrese una cadena de texto
texto = input("Ingresa una cadena de texto: ")

# Inicializar el contador de vocales
contador_vocales = 0

# Definir las vocales
vocales = "aeiouAEIOU"

# Iterar sobre cada carácter en la cadena
for caracter in texto:
    if caracter in vocales:
        contador_vocales += 1

# Mostrar el resultado
print(f"El número total de vocales en la cadena es: {contador_vocales}")
```

```
El número total de vocales en la cadena es: 2
```

1.5 C. Resumen de Conceptos Clave

1. **Estructuras de Iteración**: Permiten ejecutar bloques de código de manera repetitiva, ya sea mientras se cumpla una condición (`while`) o iterando sobre una secuencia (`for`).
2. **Estructura `while`**: Ejecuta un bloque de código mientras una condición específica sea verdadera.
3. **Estructura `for`**: Itera sobre los elementos de una secuencia (como listas, tuplas, cadenas) y ejecuta un bloque de código para cada elemento.
4. **Combinación de Iteración y Decisión**: Permite controlar el flujo de ejecución dentro de bucles basándose en condiciones dinámicas utilizando estructuras de decisión como `if`, `elif`, y `else`.
5. **`range()`**: Función utilizada comúnmente con ciclos `for` para generar una secuencia de números.
6. **f-strings**: Facilitan el formateo de cadenas de texto, permitiendo incorporar variables y expresiones dentro de las cadenas de manera legible y eficiente.
7. **Conversión de Tipos**:
   - `int()`: Convierte una cadena de texto a un entero.
   - `float()`: Convierte una cadena de texto a un número de punto flotante.
   - `str()`: Convierte un valor a una cadena de texto.

1.6 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
- Curso de Python en Coursera

1.7 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.
