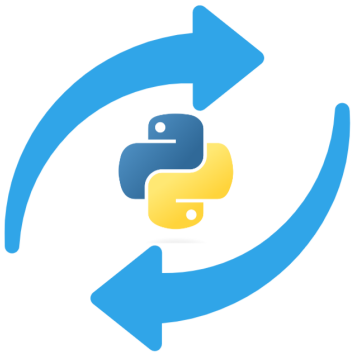
Tabla de contenido

1. Objetivo de la unidad 2

Reconocer la sintaxis de Python, incluyendo la declaración de variables, tipos de datos, expresiones, estructuras de datos, control de flujo y bucles, para implementar programas simples y realizar tareas repetitivas. Asimismo, comprender la creación y el uso de funciones en Python, para desarrollar programas con código más modular, eficiente y reutilizable.

2. Tema: Estructuras de control de flujo

Son las estructuras y mecanismos que permiten dirigir la ejecución de un programa de manera condicional o repetitiva. Las estructuras fundamentales de control de flujo en Python incluyen condicionales (if, elif, else) y bucles (for, while). Estas características son esenciales en la programación para tomar decisiones, ejecutar bloques de código específicos en función de condiciones y repetir tareas.



2.1 Condicionales

if: La instrucción if se utiliza para ejecutar un bloque de código si una condición es verdadera.

elif: La instrucción elif (abreviatura de "else if") se utiliza para evaluar múltiples condiciones después de un if. Se ejecutará el bloque de código asociado a la primera condición verdadera encontrada.

else: La instrucción else se utiliza para ejecutar un bloque de código cuando ninguna de las condiciones anteriores (if o elif) es verdadera.

Sintaxis de if, elif, else

A continuación, puedes observar los componentes en la sintaxis para las declaraciones "if", "elif" y "else":

```
if condicion_1:
    # código a ejecutar si condicion_1 es verdadera
elif condicion_2:
    # código a ejecutar si condicion_1 es falsa y condicion_2 es
verdadera
else:
    # código a ejecutar si ninguna de las condiciones anteriores es
verdadera
```

if se utiliza para iniciar la declaración. Después de **if**, se coloca la condición que se evaluará. Si esta condición es verdadera, el bloque de código indentado debajo de **if** se ejecutará. Si la condición es falsa, se pasará a la siguiente parte de la declaración.

elif se utiliza para agregar una nueva condición que se evaluará si todas las condiciones anteriores (**if** y **elif** previos) son falsas y esta nueva condición es verdadera. Si la condición **elif** es verdadera, el bloque de código indentado debajo de **elif** se ejecutará y el flujo del programa saldrá de la declaración **if**.

else se utiliza al final de la declaración para ejecutar un bloque de código si todas las condiciones anteriores (**if** y **elif**) son falsas. No se especifica ninguna condición después de **else**, ya que else representa "cualquier otra cosa". Por lo tanto, el bloque de código indentado debajo de **else** se ejecutará sólo si ninguna de las condiciones anteriores es verdadera.

Ejemplo:

El siguiente código identifica si un número dado es positivo, negativo o cero.

```
# Ejemplo de uso de if, elif y else en Python

# Definir una variable con un número
numero = int(input("Introduce un número: "))

# Verificar si el número es positivo, negativo o cero
if numero > 0:
    print("El número es positivo.")
```

```
elif numero < 0:
    print("El número es negativo.")
else:
    print("El número es cero.")
```

La variable "numero" se evalúa con una serie de condiciones:

- Si numero es mayor que cero, imprime "El número es positivo".
- Si numero es menor que cero, imprime "El número es negativo".
- Si ninguna de las condiciones anteriores es verdadera, significa que numero es igual a cero, por lo que imprime "El número es cero".

Ejemplo:

El siguiente código clasifica los estudiantes según las calificaciones.

```
# Ejemplo de uso de if, elif y else en Python
calificacion = int(input("Introduce la calificación: "))

# Clasificar estudiante según la calificación
if calificacion >= 90:
    print("Excelente")
elif calificacion >= 80:
    print("Muy bueno")
elif calificacion >= 70:
    print("Bueno")
elif calificacion >= 60:
    print("Satisfactorio")
else:
    print("Insuficiente")
```

En este ejemplo, se evalúa la variable "calificacion" para determinar la calificación de un estudiante:

- Si calificacion es mayor o igual a 90, imprime "Excelente".
- Si no se cumple la primera condición, pero calificacion es mayor o igual a 80, imprime "Muy bueno".

- Y así sucesivamente, se asigna una letra de acuerdo con el rango de calificaciones.

Ejemplo:

El siguiente código comprueba si una contraseña tiene al menos 8 caracteres de longitud.

```
# Solicitar al usuario que ingrese una contraseña
contrasena = input("Por favor, ingrese una contraseña: ")

# Comprobar si la contraseña tiene al menos 8 caracteres
if len(contrasena) >= 8:
    print("La contraseña es válida.")
else:
    print("La contraseña es demasiado corta. Debe tener al menos 8 caracteres.")
```

Este ejemplo verifica si una contraseña tiene al menos 8 caracteres. Si la longitud de la contraseña (len(contrasena)) es mayor o igual a 8, imprime "Contraseña válida"; de lo contrario, imprime "La contraseña debe tener al menos 8 caracteres".

Ejemplo:

El siguiente código solicita al usuario que ingrese su peso en kilogramos y su altura en metros. Luego, calcula el IMC utilizando la fórmula IMC = peso / (altura^2), donde el peso se da en kilogramos y la altura en metros al cuadrado. Después de calcular el IMC, el programa utiliza una serie de declaraciones elif para clasificar el estado ponderal del individuo según el valor de su IMC.

```
# Solicitar al usuario que ingrese su peso en kilogramos
peso = float(input("Por favor, ingrese su peso en kilogramos: "))

# Solicitar al usuario que ingrese su altura en metros
altura = float(input("Por favor, ingrese su altura en metros: "))

# Calcular el IMC
imc = peso / (altura ** 2)
```

```python
# Imprimir el valor del IMC
print(f"Su IMC es: {imc:.2f}")

# Clasificar el estado ponderal según el valor del IMC
if imc < 18.5:
    print("Clasificación: Bajo peso")
elif 18.5 <= imc < 24.9:
    print("Clasificación: Peso normal")
elif 25 <= imc < 29.9:
    print("Clasificación: Sobrepeso")
else:
    print("Clasificación: Obesidad")
```

2.2 Introducción a los Bucles

son estructuras que permiten ejecutar un bloque de código repetidamente, ya sea un número específico de veces o mientras se cumple una condición. Los bucles son fundamentales en la programación y tienen una importancia significativa.

Ventajas

- **Automatización de tareas repetitivas:** Los ciclos permiten ejecutar el mismo bloque de código varias veces sin tener que escribirlo múltiples veces, lo que ahorra tiempo y reduce la posibilidad de cometer errores.
- **Legibilidad del código:** Utilizar ciclos hace que el código sea más fácil de leer, entender y mantener. En lugar de repetir el mismo bloque de código múltiples veces, se utiliza una estructura de ciclo para ejecutarlo las veces necesarias.
- **Flexibilidad en la manipulación de datos:** Los ciclos **"for"** son particularmente útiles para iterar sobre listas, tuplas, diccionarios y otros tipos de datos en Python, permitiendo realizar operaciones en cada elemento de la colección de manera eficiente.

- **Control sobre la ejecución del código:** Los ciclos "while" permiten ejecutar un bloque de código mientras se cumpla una condición específica, lo que brinda un mayor control sobre la ejecución del programa.
- **Facilita la implementación de algoritmos:** Muchos algoritmos y soluciones de problemas requieren la repetición de tareas para procesar datos o realizar cálculos. Los ciclos facilitan la implementación de estos algoritmos, lo que puede resultar en soluciones más eficientes y optimizadas.
- **Adaptable a diversas situaciones:** Los ciclos son versátiles y se pueden adaptar a una amplia variedad de situaciones y problemas, lo que los convierte en una herramienta esencial en la programación.

Bucle "for"

Un bucle "**for**" es una estructura de control que le permite ejecutar un bloque de código una cantidad específica de veces. Esto es especialmente útil cuando sabes de antemano cuántas veces se repetirá una acción. En Python, los bucles **for** se utilizan ampliamente para iterar a través de secuencias como listas, tuplas, cadenas y diccionarios.

Ejemplo:

El siguiente código usa un ciclo for para sumar los numéros almacenados en una lista

```
# Crear una lista de números
numeros = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Inicializar una variable para almacenar la suma
suma = 0

# Utilizar un ciclo for para iterar a través de la lista de números
for numero in numeros:
    # Sumar cada número al total acumulado
    suma += numero

# Imprimir el resultado de la suma
print("La suma de los números en la lista es:", suma)
```

Ejemplo:

El siguiente código usa un ciclo for para calcular el número n factorial para un número n dado

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

# Inicializar una variable para almacenar el factorial
factorial = 1

# Utilizar un ciclo for para calcular el factorial de n
for i in range(1, n + 1):
    # Multiplicar el valor actual de factorial por el número de la iteración actual
    factorial *= i

# Imprimir el resultado del factorial
    print(f"El factorial de {n} es: {factorial}")
```

Bucle "While"

El bucle "**while**" en Python se utiliza para repetir un bloque de código mientras una condición específica sea verdadera. Es decir, la repetición depende de una condición. Es útil cuando no se sabe cuántas veces se debe repetir una acción, pero se conoce la condición bajo la cual se debe continuar la ejecución del ciclo.

Ejemplo:

El siguiente código usa un ciclo while para imprimir los números del 1 al 10.

```
# Inicializar la variable contador con el valor 1
contador = 1

# Utilizar un bucle while para imprimir los números del 1 al 10
while contador <= 10:
    # Imprimir el valor actual de contador
```

```
    print(contador)
    # Incrementar el valor de contador en 1
    contador += 1
```

Ejemplo:

El siguiente código usa un ciclo while para contar los números pares hasta el 20.

```
# Inicializar la variable suma con el valor 0
suma = 0

# Inicializar la variable número con el valor 1
numero = 1

# Utilizar un bucle while para recorrer los números del 1 al 20
while numero <= 20:
    # Verificar si el número es par
    if numero % 2 == 0:
        # Si es par, añadirlo a la suma
        suma += numero
   # Incrementar el valor de número en 1
   numero += 1

# Imprimir el resultado de la suma de los números pares
print("La suma de los números pares del 1 al 20 es:", suma)
```

Sintaxis de bucles For y While

Bucle for

```
for elemento in secuencia:
    # Código a ejecutar en cada iteración
```

El bucle for se utiliza para iterar sobre una secuencia (como una lista, tupla, diccionario, etc.) y ejecutar un bloque de código una vez para cada elemento de esa secuencia.

- **elemento**: Es una variable que toma el valor de cada elemento de la secuencia en cada iteración del bucle.
- **secuencia**: Es la colección de elementos sobre la que se iterará.

Bucle while

```
while condicion:
    # Código a ejecutar mientras condicion sea verdadera
```

El bucle while se utiliza para ejecutar un bloque de código repetidamente siempre que una condición especificada sea verdadera.

- **condicion**: Es una expresión booleana que se evalúa antes de cada iteración del bucle. Si es verdadera, el bloque de código dentro del bucle se ejecutará; si es falsa, el bucle se detendrá y la ejecución continuará después del bucle.

Ejemplos uso de bucles For y While

Ejemplo 1:

Se tiene una lista de números [1, 2, 3, 4, 5]. El bucle for itera sobre cada elemento de la lista (num) y calcula su cuadrado (num ** 2). Luego, imprime el número original y su cuadrado.

```
# Definimos una lista de números
numeros = [1, 2, 3, 4, 5]

# Iteramos sobre cada elemento de la lista
for num in numeros:
    # Calculamos el cuadrado de cada número
    cuadrado = num ** 2
    # Imprimimos el número original y su cuadrado
    print("El cuadrado de ", num, "es ", cuadrado)
```

Ejemplo 2:

Se tiene un diccionario estudiante con información sobre un estudiante. El método items() se utiliza para obtener tanto las claves como los valores del diccionario. El bucle for itera sobre cada par clave-valor y los imprime en formato "clave: valor".

```
# Definimos un diccionario con información sobre un estudiante
estudiante = {
    "nombre": "Juan",
    "edad": 20,
    "carrera": "Ingeniería",
    "promedio": 4.5
}

# Iteramos sobre cada par clave-valor del diccionario
for clave, valor in estudiante.items():
    # Imprimimos la clave y el valor en formato "clave: valor"
    print(clave + ": " + str(valor))
```

Ejemplo 3:

Aquí, se tiene una cadena de texto "Python". El bucle for itera sobre cada carácter de la cadena (letra) e imprime cada carácter por separado en una línea.

```
# Definimos una cadena de texto
cadena = "Python"

# Iteramos sobre cada carácter de la cadena
for letra in cadena:
    # Imprimimos cada carácter por separado en una línea
    print(letra)
```

2.3 Autoevaluación

Pregunta 1:

¿Cuál es la palabra clave en Python que se utiliza para iniciar una declaración condicional?

Opción 1: if

Opción 2: else

Opción 3: for

Opción 4: while

Pregunta 2:

¿Cuál de las siguientes palabras clave se utiliza para agregar una nueva condición en una declaración condicional después de la primera?

Opción 1: else

Opción 2: elif

Opción 3: while

Opción 4: for

Pregunta 3:

¿Qué estructura de control se utiliza para iterar sobre una secuencia en Python?

Opción 1: if

Opción 2: while

Opción 3: for

Opción 4: do-while

Pregunta 4:

¿Qué hace la palabra clave "else" en una declaración condicional en Python?

Opción 1: Se utiliza para agregar una nueva condición.

Opción 2: Se ejecuta si la condición en la declaración condicional es verdadera.

Opción 3: Se ejecuta si la condición en la declaración condicional es falsa.

Opción 4: Se ejecuta si ninguna de las condiciones anteriores es verdadera.

Pregunta 5:

¿Cuál es la diferencia entre "for" y "while" en Python?

Opción 1: for se utiliza para ejecutar un bloque de código mientras una condición sea verdadera, mientras que while se utiliza para iterar sobre una secuencia.

Opción 2: for se utiliza para iterar sobre una secuencia, mientras que while se utiliza para ejecutar un bloque de código mientras una condición sea verdadera.

Opción 3: No hay diferencia entre for y while, ambos se pueden usar indistintamente.

Opción 4: for y while son lo mismo en Python.

3. Bibliografía

- ✓ Álvarez, C. A. (2020). Introducción al Jupyter Notebook y aplicaciones básicas. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/35681
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 21-24). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 39). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 49-88). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 89-104). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1
- ✓ Hinojosa Gutiérrez, Á. (2015). Python paso a paso: (ed.). RA-MA Editorial, (pp. 105-122). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/107213?page=1
- ✓ Cuevas Álvarez, A. (2016). Python 3: curso práctico. RA-MA Editorial, (pp. 145-182).

https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/106404?page=1

- ✓ Código Marzal Varó, A. García Sevilla, P. & Gracia Luengo, I. (2016). Introducción a la programación con Python 3. D - Universitat Jaume I. Servei de Comunicació i Publicacions, (pp. 228-336). https://elibro-net.bibliotecavirtual.unad.edu.co/es/ereader/unad/51760?page=1
- ✓ Noguera, A. d. (2021). Lenguaje de programación Python. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/39346
- ✓ Aldana, J. M. (2023). *Estructuras de control cíclicas*. [Objeto_virtual_de_aprendizaje_OVA]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/58883
- ✓ Amórtegui, M. P. (2021). Algoritmos y Programación. [Objeto_virtual_de_Informacion_OVI]. Repositorio Institucional UNAD. https://repository.unad.edu.co/handle/10596/44535