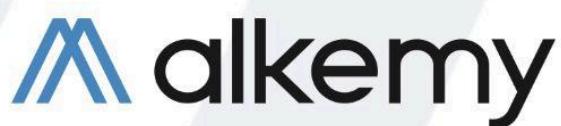


Sentencias condicionales

Módulo Fundamentos de programación python para el análisis de datos

|AE3: Codificar una rutina utilizando estructuras condicionales y expresiones booleanas para resolver un problema de baja complejidad de acuerdo al lenguaje python.



Introducción



Las sentencias condicionales son fundamentales en cualquier lenguaje de programación, ya que permiten ejecutar diferentes bloques de código en función de si una condición es verdadera o falsa. En Python, las estructuras condicionales son especialmente importantes para controlar el flujo de un programa, permitiendo tomar decisiones lógicas basadas en datos o en los resultados de operaciones previas. Estas sentencias ayudan a que el programa sea más interactivo, adaptable y capaz de responder a diversas situaciones.

Las sentencias condicionales de Python se basan en operadores de comparación y operadores booleanos que evalúan expresiones y devuelven valores de verdad. Con estas herramientas, es posible construir decisiones simples y complejas, adecuadas para gestionar cualquier flujo lógico en el programa. Este manual presenta las sentencias condicionales y los operadores necesarios, explicando cómo utilizarlos con ejemplos prácticos.

Aprendizaje esperado

Cuando finalices la lección serás capaz de:

- Comprender qué son las sentencias condicionales y su importancia en la programación.
- Utilizar operadores booleanos y de comparación para construir condiciones.
- Implementar sentencias `if`, `if-else` y `if-elif-else` para controlar el flujo de un programa en Python.
- Trabajar con expresiones booleanas y paréntesis para mejorar la claridad de las condiciones.
- Aplicar expresiones ternarias para crear condiciones compactas y eficientes.

Desarrollo

Sección 1: Qué es una Sentencia Condicional y Por Qué se Necesitan

Concepto de Sentencia Condicional

Una sentencia condicional es una estructura que permite ejecutar un bloque de código solo si se cumple una condición específica. En Python, las sentencias condicionales se expresan mediante palabras clave como `if`, `else` y `elif`. Estas estructuras permiten al programa tomar decisiones, lo cual es fundamental para crear aplicaciones dinámicas y adaptativas. Sin las sentencias condicionales, un programa no podría responder a diferentes situaciones ni personalizar la ejecución en función de los datos de entrada o de los resultados de operaciones previas.

Por ejemplo, una sentencia condicional permite verificar si el usuario tiene la edad suficiente para acceder a un contenido, realizar operaciones en función del estado de una variable o calcular resultados específicos basados en ciertas condiciones.

Ejemplo Básico de Condicional

```
edad = 18
if edad >= 18:
    print("Eres mayor de edad.")
else:
    print("Eres menor de edad.")
# Salida: Eres mayor de edad.
```

En este ejemplo, el programa verifica si la variable `edad` es mayor o igual a 18 para determinar el mensaje adecuado. Dependiendo del valor de `edad`, el programa elige uno de los bloques para ejecutarse.

La Importancia de las Condiciones en la Lógica de un Programa

Las condiciones son esenciales para controlar el flujo de ejecución de un programa y determinar qué acciones deben realizarse en cada caso. Esto permite crear programas interactivos y personalizados que reaccionan a los datos de entrada, lo cual es crucial en aplicaciones como juegos, sistemas de gestión de usuarios, formularios en línea y mucho más.

Adaptabilidad y Flexibilidad en Programación

Gracias a las sentencias condicionales, los programas pueden manejar excepciones, validar entradas de usuario y ejecutar diferentes lógicas de procesamiento según el contexto. Esto mejora la adaptabilidad de las aplicaciones y permite ofrecer experiencias de usuario más ricas y personalizadas.

Sección 2: Operadores Booleanos

Concepto de Operadores Booleanos

Los operadores booleanos son operadores lógicos que permiten construir expresiones de verdad o falsedad en un programa. Estos operadores son fundamentales para combinar y evaluar múltiples condiciones en una sola expresión lógica. En Python, los operadores booleanos más utilizados son **and**, **or** y **not**. Cada uno de ellos permite combinar o modificar condiciones de manera diferente, ofreciendo flexibilidad en la evaluación lógica.

Operador **and**

El operador **and** devuelve **True** solo si ambas condiciones en una expresión son verdaderas. Si alguna de las condiciones es **False**, la expresión completa será **False**.

Ejemplo de uso de **and**:

```
edad = 20
tiene_identificacion = True

if edad >= 18 and tiene_identificacion:
    print("Puedes entrar al club.")
else:
    print("No puedes entrar.")
# Salida: Puedes entrar al club.
```

En este caso, la persona puede entrar solo si cumple con ambas condiciones: ser mayor de edad y tener identificación.

Operador **or**

El operador **or** devuelve **True** si al menos una de las condiciones en la expresión es verdadera. Solo será **False** si ambas condiciones son falsas.

Ejemplo de uso de **or**:

```
es_estudiante = True
es_docente = False

if es_estudiante or es_docente:
    print("Tienes acceso al descuento.")
else:
    print("No tienes descuento.")
# Salida: Tienes acceso al descuento.
```

Aquí, la persona tiene descuento si es estudiante o docente, o ambos.

Operador **not**

El operador **not** invierte el valor de verdad de una condición. Si la condición es **True**, **not** la convierte en **False**, y viceversa.

Ejemplo de uso de **not**:

```
es_mayor = False

if not es_mayor:
    print("No eres mayor de edad.")
# Salida: No eres mayor de edad.
```

Sección 3: Operadores de Comparación

Concepto de Operadores de Comparación

Los operadores de comparación permiten comparar dos valores y devolver un resultado booleano (**True** o **False**). Son esenciales para construir condiciones en sentencias **if**, ya que permiten determinar si una expresión cumple con ciertos criterios.

Operador Mayor Que y Mayor o Igual Que

El operador **>** se usa para verificar si un valor es mayor que otro, mientras que **\geq** verifica si es mayor o igual. Estos operadores son comunes en comparación de valores numéricos.

Ejemplo de mayor que y mayor o igual que:

```
edad = 21

if edad > 18:
    print("Puedes votar.")
if edad >= 21:
    print("Puedes beber alcohol legalmente en EE. UU.")
# Salida: Puedes votar.
#         Puedes beber alcohol legalmente en EE. UU.
```

Operador Menor Que y Menor o Igual Que

El operador `<` verifica si un valor es menor que otro, mientras que `<=` evalúa si es menor o igual. Se usa principalmente en validaciones de límites inferiores.

Ejemplo de menor que y menor o igual que:

```
nota = 7

if nota < 5:
    print("Suspensos")
elif nota <= 7:
    print("Aprobado")
# Salida: Aprobado
```

Operador Igual Que

El operador `==` evalúa si dos valores son iguales. Es uno de los operadores más utilizados en condiciones.

Ejemplo de comparación de igualdad:

```
respuesta = "sí"

if respuesta == "sí":
    print("Aceptado")
else:
    print("Rechazado")
# Salida: Aceptado
```

Operador Distinto Que

El operador `!=` devuelve `True` si dos valores son distintos. Es útil para asegurar que un valor no sea igual a otro.

Ejemplo de comparación de diferencia:

```
usuario = "admin"

if usuario != "invitado":
    print("Tienes acceso completo.")
# Salida: Tienes acceso completo.
```

Sección 4: Paréntesis y Expresiones Booleanas

Uso de Paréntesis para Agrupar Condiciones

En expresiones complejas, los paréntesis permiten agrupar condiciones para controlar el orden de evaluación. Esto facilita la claridad y la precisión en las expresiones booleanas, asegurando que el programa evalúe primero las condiciones agrupadas.

Ejemplo de uso de paréntesis en una condición:

```
edad = 25
es_miembro = True
codigo_descuento = "VIP"

if (edad > 18 and es_miembro) or codigo_descuento == "VIP":
    print("Descuento aplicado.")
else:
    print("No tienes descuento.")
# Salida: Descuento aplicado.
```

Precedencia de Operadores Booleanos

En Python, los operadores tienen una jerarquía de evaluación: primero se evalúa `not`, luego `and` y finalmente `or`. Sin embargo, el uso de paréntesis permite anular esta precedencia.

Combinar Múltiples Condiciones con Paréntesis

Los paréntesis ayudan a combinar múltiples condiciones para definir expresiones complejas. Esto es especialmente útil en programas que requieren verificar varias condiciones antes de ejecutar un bloque de código.

Ejemplo de expresión compleja:

```
salario = 3000
es_tiempo_completo = True
antiguedad = 2

if (salario > 2500 and es_tiempo_completo) or antiguedad > 5:
    print("Eres elegible para el bono.")
else:
    print("No eres elegible para el bono.")
# Salida: Eres elegible para el bono.
```

Mejorar la Claridad en Expresiones Booleanas

Los paréntesis no solo definen el orden de evaluación, sino que también mejoran la claridad y legibilidad del código. Esto es esencial en equipos de desarrollo, ya que facilita la comprensión de expresiones complejas.

Sección 5: La Sentencia `if`

Estructura Básica de `if`

La sentencia `if` permite ejecutar un bloque de código si se cumple una condición específica. Es la estructura condicional más simple y se utiliza cuando solo hay una condición que determinará la ejecución de un bloque.

Ejemplo básico de `if`:

```
temperatura = 30

if temperatura > 25:
    print("Hace calor.")
# Salida: Hace calor.
```

Condiciones Booleanas en `if`

La condición en un `if` puede incluir comparaciones simples o expresiones booleanas. Si la condición es verdadera, se ejecuta el bloque de código dentro del `if`; si no, se omite.

Ejemplo de `if` con expresión compleja:

```
edad = 20
tiene_permiso = True

if edad >= 18 and tiene_permiso:
    print("Puedes conducir.")
# Salida: Puedes conducir.
```

Uso Práctico de `if` en Validaciones

El `if` es útil para verificar entradas del usuario, controlar flujos de datos y validar requisitos antes de proceder con operaciones. Esto asegura que solo se ejecute el código si las condiciones son adecuadas.

Buena Práctica: Identación en el Bloque `if`

Python requiere una identación adecuada en los bloques `if`. Todo el código dentro del `if` debe estar correctamente identado para evitar errores y asegurar la ejecución adecuada del bloque.

Sección 6: La Sentencia `if-else`

Estructura de `if-else`

La sentencia `if-else` permite manejar dos posibles caminos de ejecución. Si la condición es verdadera, se ejecuta el bloque `if`; si es falsa, se ejecuta el bloque `else`. Esto permite que el programa maneje ambas opciones de manera estructurada.

Ejemplo de `if-else`:

```
hora = 10

if hora < 12:
    print("Buenos días.")
else:
    print("Buenas tardes.")
# Salida: Buenos días.
```

Cuando Usar `if-else`

`if-else` es útil en casos donde el programa debe responder a una condición de manera exclusiva, permitiendo un flujo claro y alternativo en el código.

Comparaciones y Alternativas en **if-else**

if-else también permite comparar variables y ejecutar distintos bloques de acuerdo a los resultados, mejorando la adaptabilidad de las aplicaciones.

Ejemplo de Validación de Edad con **if-else**

```
edad = 17

if edad >= 18:
    print("Acceso permitido.")
else:
    print("Acceso denegado.")
# Salida: Acceso denegado.
```

Sección 7: La Sentencia **if-elif-else**

Estructura de **if-elif-else**

La sentencia **if-elif-else** permite evaluar múltiples condiciones de forma secuencial. Si la primera condición es verdadera, se ejecuta su bloque; si no, se evalúa la siguiente condición **elif**, y así sucesivamente.

Ejemplo de **if-elif-else**:

```
dia = "miércoles"

if dia == "lunes":
    print("Hoy es lunes.")
elif dia == "martes":
    print("Hoy es martes.")
else:
    print("Es otro día.")
# Salida: Es otro día.
```

Cuando Usar **if-elif-else**

Es útil cuando hay varias condiciones posibles y solo una debe ejecutarse, permitiendo manejar casos alternativos.

Verificación de Rangos de Valores

En programación, **if-elif-else** es común para verificar rangos de valores o categorías.

Ejemplo:

```
nota = 85

if nota >= 90:
    print("Excelente")
elif nota >= 75:
    print("Bueno")
else:
    print("Necesita mejorar")
# Salida: Bueno
```

Flujo de Evaluación de Condiciones

El **if-elif-else** evalúa de forma secuencial, permitiendo un flujo estructurado para distintas situaciones.

Sección 8: Expresiones Ternarias

Concepto de Expresiones Ternarias

Las expresiones ternarias permiten escribir una condición **if-else** en una sola línea. Son útiles para casos simples y mejoran la legibilidad del código en comparaciones cortas.

Ejemplo de expresión ternaria:

```
edad = 20
mensaje = "Mayor de edad" if edad >= 18 else "Menor de edad"
print(mensaje)
# Salida: Mayor de edad
```

Cuando Usar Expresiones Ternarias

Son ideales para asignaciones rápidas o impresiones condicionales, evitando la necesidad de una estructura **if-else** extensa.

Ventajas en Código Compacto

La ternaria mejora la legibilidad cuando se usa con moderación, evitando estructuras largas.

Ejemplo de Condición de Aprobación con Ternaria

```
nota = 70
resultado = "Aprobado" if nota >= 60 else "Reprobado"
print(resultado)
# Salida: Aprobado
```

Cierre



Las sentencias condicionales en Python son fundamentales para permitir que un programa tome decisiones basadas en condiciones específicas. Al utilizar estructuras como `if`, `if-else`, y `if-elif-else`, los programadores pueden controlar qué bloques de código se ejecutan en función de los datos o del contexto, lo que otorga flexibilidad y adaptabilidad a las aplicaciones. Estas estructuras condicionales, junto con los operadores de comparación y booleanos (`and`, `or`, `not`), permiten evaluar expresiones lógicas complejas, lo cual es esencial para manejar una amplia variedad de escenarios y casos en la ejecución de un programa.

La comprensión y uso adecuado de los operadores y expresiones booleanas mejora la claridad y eficiencia del código, ayudando a evitar errores comunes y a hacer el código más mantenable. Las condicionales también permiten manejar validaciones de entrada, personalización de respuestas y adaptación de comportamientos en función de diversas condiciones, lo cual es crucial en aplicaciones interactivas o sistemas complejos. Mediante el uso de paréntesis y expresiones ternarias, los desarrolladores pueden crear condiciones más legibles y eficientes, optimizando tanto el flujo de trabajo como la experiencia del usuario.

En conclusión, las sentencias condicionales son la base para una programación lógica y adaptable, y su dominio es clave para progresar hacia estructuras de control más complejas, como bucles y funciones. Practicar con diferentes tipos de condicionales y situaciones permite desarrollar una lógica de programación más sólida y ayuda a resolver problemas de forma estructurada y efectiva. Con una base fuerte en el uso de sentencias condicionales, los programadores pueden abordar proyectos más complejos y crear aplicaciones que respondan de manera eficaz a las necesidades de los usuarios.

Referencias



Python Software Foundation. (s.f.). 4. More control flow tools.

<https://docs.python.org/3/tutorial/controlflow.html>

Real Python. (s.f.). Learn Python programming. <https://realpython.com/>

¡Muchas gracias!

Nos vemos en la próxima lección

