Introducción

Los booleanos son un tipo común en Python. Su valor solo puede ser una de dos cosas: verdadero o falso. Comprender cómo usar los valores booleanos es fundamental, ya que los necesitarás para escribir *lógica condicional*.

Escenario: Imprimir mensajes de advertencia

Supongamos que estás creando un programa que alertará a las personas de todo el mundo de que un asteroide grande y rápido se está acercando a la Tierra. Si el asteroide está lo suficientemente cerca de la Tierra como para representar un peligro, debe imprimir un mensaje de advertencia en los dispositivos de las personas. Si no hay peligro, debes hacer que todos lo sepan, para que puedan continuar con su día. Para poder proporcionar estos mensajes, tendrías que encontrar una manera de razonar si una condición es verdadera o falsa.

En este módulo, usarás palabras clave y operadores booleanos para escribir varios tipos de expresiones condicionales.

¿Qué aprenderás?

Al final de este módulo, podrás:

- Ejecutar código en una variedad de condiciones mediante sentencias `if`, `else`, y `elif`.
- Combinar la lógica condicional y crear condiciones más complejas mediante el uso de operadores `and` y operadores `or`.

¿Cuál es el objetivo principal?

Este módulo te enseña cómo usar la lógica condicional para crear programas basados en decisiones.

Lógica Booleana

Escribir declaraciones 'if'

Para expresar la lógica condicional en Python, se utilizan instrucciones `if`. Cuando escribes una declaración `if`, confías en otro concepto que cubrimos en este módulo, los operadores matemáticos. Python admite los operadores lógicos comunes de las matemáticas: igual, no igual, menor que, menor que o igual a, mayor que y mayor que o igual a. Probablemente estés acostumbrado a ver estos operadores mostrados usando símbolos, que es la forma en que también se representan en Python.

- Iguales: `a == b`
- No es igual: `a != b`
- Menos que: `a < b`
- Menor o igual que: `a <= b`
- Mayor que: `a > b`
- Mayor o igual que: `a >= b`

Expresiones de prueba

Debes usar una instrucción `if` para ejecutar código solo si se cumple una determinada condición. Lo primero que se hace cuando se escribe una instrucción `if` es comprobar la condición mediante una *expresión de prueba*. A continuación, determins si la instrucción evalúa a `True (Verdadero)` o `False (Falso)`. Si es `True (Verdadero)`, se ejecuta el siguiente bloque de código con sangría:

```
# Tip de práctica 1: Intenta ejecutarlo en un notebook.
a = 97
b = 55
# test expression / expresión de prueba
if a < b:
    # statement to be run / instrucción a ejecutar
    print(b)
```

En este ejemplo, `a < b` es la expresión de prueba. El programa evalúa la expresión de prueba y, a continuación, ejecuta el código dentro de la instrucción `if` sólo si la expresión de prueba es `True (Verdadera)`. Si evalúa la expresión, sabe que es `False (Falso)`, no se ejecutará ningún código que escriba en la instrucción `if`.

Escribir declaraciones `if`

Utiliza una instrucción `if` si deseas ejecutar código sólo si se cumple una determinada condición. La sintaxis de una instrucción `if` es siempre:

```
if expresion_prueba:
    # intrucción(es) a ejecutar
```

Por ejemplo:

Tip de práctica 2: Antes de ejecutarlo en un notebook, intenta deducir cuál será el resultado y compruébalo.

```
# Aplica el tip de práctica 1.
a = 93
b = 27
if a >= b:
    print(a)
```

El fragmento anterior se lee de la siguiente manera (Línea por línea):

```
A la letra 'a' le asigno el valor de 93.
A la letra 'b' le asigno el valor de 27.
SI a(93) es mayor o igual a b(27) entonces:
Muestra (print) el valor de a(93)
```

 Estamos aprendiendo a leer el código de una forma más sencilla.

En Python, el cuerpo de una instrucción `if` debe tener sangría . Siempre se ejecutará cualquier código que siga a una expresión de prueba que no tenga sangría:

Tips de práctica 1 y 2

```
a = 24
b = 44
if a <= 0:
    print(a)
print(b)
```

En este ejemplo, el `44` como resultado se debe a que la expresión de prueba es `False` y la instrucción `print(b)` no tiene sangría en el mismo nivel que la instrucción `if` .

¿Qué son las declaraciones "else" y "elif"?

¿Qué sucede si también deseas que tu programa ejecute una pieza de código cuando tu expresión de prueba es `False` ? ¿O qué pasa si deseas incluir otra expresión de prueba?.

Python tiene otras palabras clave que puedes usar para hacer declaraciones `if` más complejas, `else` y `elif` . Cuando se utiliza `if` , `else` , y `elif` en combinación, se pueden escribir programas complejos con varias expresiones de prueba y sentencias para ejecutar.

Trabajando con else

Cuando utilizas una instrucción, el cuerpo del programa sólo se ejecutará si la expresión de prueba es `True` . Para agregar más código que se ejecutará cuando la expresión de prueba sea `False` , debes agregar una instrucción `else` .

Volvamos al ejemplo de la sección anterior:

```
a = 93
b = 27
if a >= b:
    print(a)
```

En este ejemplo, si `a` no es mayor o igual que `b` , no pasa nada. Supongamos que desea imprimir `b` si la expresión de prueba es `False` :

```
a = 93
b = 27
```

```
if a >= b:
    print(a)
else:
    print(b)
```

Si la expresión de prueba es `False`, se omite el código del cuerpo de la instrucción `if` y el programa continúa ejecutándose desde la instrucción `else`. La sintaxis de una instrucción `if/else` es siempre:

```
if expresion_prueba:
    # instrucción(es) a ejecutar
else:
    # instrucción(es) a ejecutar
```

Trabajando con elif

289 lines (220 sloc) | 11.7 KB

le permite agregar varias expresiones de prueba al programa. Estas instrucciones se ejecutan en el orden en que están escritas, por lo que el programa ingresará una instrucción `elif` solo si la primera instrucción `if` es `False`. Por ejemplo:

```
a = 93
b = 27
if a >= b:
    print("a es mayor o igual que b")
elif a == b:
    print("a es igual que b")
```

La instrucción `elif` de este bloque de código no se ejecutará, porque la instrucción `if` es `True`.

La sintaxis de una instrucción `if/elif` es:

```
if expresion_prueba:
    # instrucción(es) a ejecutar
elif expresion_prueba:
    # instrucción(es) a ejecutar
```

Combinar declaraciones `if`, `elif`, y `else`

Puedes combinar sentencias `if` , `elif` , y `else` para crear programas con lógica condicional compleja. Recuerda que una instrucción `elif` sólo se ejecuta cuando la condición `if` es `False` . También ten en cuenta que un bloque `if` puede tener solo un bloque `else` , pero puede tener varios bloques `elif` .

Veamos el ejemplo de nuevo con una declaración `elif` añadida:

```
a = 93
b = 27
if a > b:
    print("a es mayor que b")
elif a < b:
    print("a es menor que b")
else:
    print ("a es igual que b")
```

Un bloque de código que utiliza los tres tipos de instrucciones tiene la sintaxis siguiente:

```
if expresion_prueba:
    # instrucción(es) a ejecutar
elif expresion_prueba:
    # instrucción(es) a ejecutar
elif expresion_prueba:
    # instrucción(es) a ejecutar
else:
    # instrucción(es) a ejecutar
```

Trabajar con lógica condicional anidada

Python también admite lógica condicional anidada, lo que significa que puedes anidar sentencias `if` , `elif` , y `else` , para crear programas aún más complejos. Para anidar condiciones, indenta las condiciones internas y todo lo que esté en el mismo nivel de sangría se ejecutará en el mismo bloque de código:

```
a = 16
b = 25
c = 27
if a > b:
    if b > c:
        print ("a es mayor que b y b es mayor que c")
    else:
        print ("a es mayor que b y menor que c")
elif a == b:
```

```
    print ("a es igual que b")
else:
    print ("a es menor que b")
```

Este fragmento de código produce la salida `"a es menor que b"`.

La lógica condicional anidada sigue las mismas reglas que la lógica condicional normal dentro de cada bloque de código. Aquí hay un ejemplo de la sintaxis:

```
if expresion_prueba:
    # instrucción(es) a ejecutar
    if expresion_prueba:
        # instrucción(es) a ejecutar
    else:
        # instrucción(es) a ejecutar
elif expresion_prueba:
    # instrucción(es) a ejecutar
    if expresion_prueba:
        # instrucción(es) a ejecutar
    else:
        # instrucción(es) a ejecutar
else:
    # instrucción(es) a ejecutar
```

¿Qué son los operadores 'and' y 'or'?

Es posible que ocasionalmente desees combinar expresiones de prueba para evaluar varias condiciones `if`, `elif`, y `else` en una instrucción. Para ello, utilizaremos los operadores booleanos `and` y `or`.

El operador `or`

Puede conectar dos expresiones booleanas o de prueba mediante el operador booleano `or`. Para que toda la expresión se evalúe en `True`, al menos una de las subexpresiones debe ser verdadera. Si ninguna de las subexpresiones es verdadera, toda la expresión se evalúa en `False`. Por ejemplo, en la siguiente expresión, toda la expresión de prueba se evalúa en `True`, porque se ha cumplido una de las condiciones de las subexpresiones:

```
a = 23
b = 34
if a == 34 or b == 34:
    print(a + b)
```

Si ambas subexpresiones son verdaderas, toda la expresión de prueba también evalúa a `True`.

Una expresión booleana que utiliza `or` tiene la sintaxis siguiente:

```
subexpresión1 or subexpresión2
```

El operador `and`

También puedes conectar dos expresiones de prueba mediante el operador booleano `and`.

Ambas condiciones de la expresión de prueba deben ser verdaderas para que toda la expresión de prueba se evalúe en `True`. En cualquier otro caso, la expresión de prueba es `False`. En el ejemplo siguiente, toda la expresión de prueba se evalúa en `False`, porque sólo una de las condiciones de las subexpresiones es `True`:

```
a = 23
b = 34
if a == 34 and b == 34:
    print (a + b)
```

Una expresión booleana que utiliza `and` tiene la sintaxis siguiente:

```
subexpresión1 and subexpresión2
```

La diferencia entre `and` y `or`

Para resaltar la diferencia entre los dos operadores booleanos, puedes utilizar una tabla de verdad. Una tabla de verdad muestra a qué se evalúa toda la expresión de prueba en función de las dos subexpresiones.

Aquí está la tabla de la verdad para: `and`

subexpresión1	Operador	subexpresión2	Resultado
`True`	`and`	`True`	`True`
`True`	`and`	`False`	`False`
`False`	`and`	`True`	`False`

`False` **ɔresión1**	`and` **rador**	`False` **ɔresión2**	`False` **ado**

Aquí está la tabla de la verdad para: `or`

subexpresión1	Operador	subexpresión2	Resultado
`True`	`or`	`True`	`True`
`True`	`or`	`False`	`True`
`False`	`or`	`True`	`True`
`False`	`or`	`False`	`False`

Curso Propedútico de Python para Launch X - Innovacción Virtual.

Material desarrollado con base en los contenidos de MSLearn y la metáfora de LaunchX, traducción e implementación por: Fernanda Ochoa - Learning Producer de LaunchX.

Redes:

- GitHub: FernandaOchoa
- Twitter: @imonsh
- Instagram: fherz8a