Sentencias condicionales

March 29, 2022

Ejecuta el siguiente bloque de código siempre antes de ejecutar el resto del notebook.

[ ]:
```
from IPython.core.magic import  Magics, magics_class, cell_magic, line_magic

@magics_class
class Helper(Magics):

  def __init__(self, shell=None,  **kwargs):
    super().__init__(shell=shell, **kwargs)

  @cell_magic
  def debug_cell_with_pytutor(self, line, cell):
    import urllib.parse
    url_src = urllib.parse.quote(cell)
    str_begin = '<iframe width="1000" height="500" frameborder="0" src="https://
↪pythontutor.com/iframe-embed.html#code='
    str_end   = '&cumulative=false&py=3&curInstr=0"></iframe>'
    import IPython
    from google.colab import output
    display(IPython.display.HTML(str_begin+url_src+str_end))

get_ipython().register_magics(Helper)
```

1 Estructuras de control condicionales

La toma de decisiones es inherente a nuestra propia vida. Constantemente tenemos que tomar alternativas dependiendo de algún suceso o acontecimiento o de la ocurrencia de un evento u objetivo. La programación propone las estructuras de control condicionales para modificar el flujo de ejecución de las instrucciones de un programa, dentro de las estructuras condicionales tenemos la estructura `if`, `elif` y `else`, además de combinaciones de estos.

1.1 Sentencia if...else en Python

La sentencia `if` nos permite ejecutar un bloque de código únicamente si una condición dada se cumple, su sintaxis es muy simple:

```
if condicion:
    instruccion_1
```

```
    instruccion_2
    #.
    #.
    #.
    instruccion_n

instruccion_fuera_del_if
```

La instrucción `if` precede una condición lógica (la cual se construye a partir de los operadores lógicos y de comparación vistos en la sección anterior) Al finalizar esta condición se debe poner dos puntos `:` este operador le indica a Python que se iniciará un bloque de código que estará subordinado a la sentencia if, todas las instrucciones que queramos ejecutar cuando se cumpla la condición deben ir indentadas, es decir, corridas a la derecha por 4 espacios o un tabulador. Para cerrar el bloque de instrucciones del condicional, basta eliminar la indentación y seguir codificando de manera natural.

Para las personas que tengan experiencia programando en otros lenguajes, notarán que en Python no se usan `{}` para rodear los bloques de código, por lo tanto es muy importante respetar los niveles de indentación dentro del programa, de lo contrario se pueden encontrar errores como los siguientes:

[ ]:
```
x = 5
if x > 3:
    x += 1
  print(x)
    x += 3
```

```
  File "<tokenize>", line 4
    print(x)
    ^
IndentationError: unindent does not match any outer indentation level
```

[ ]:
```
x = 5
if x > 3:
    x += 1
print(x)
    x += 3
```

```
  File "<ipython-input-2-f04dc12ce01d>", line 5
    x += 3
    ^
IndentationError: unexpected indent
```

Veamos un pequeño ejemplo de su funcionamiento. Digitemos un número, el siguiente script nos dirá si el número es positivo o negativo.

```
[ ]: num = int(input("digite un número: "))
     if num > 0:
         print(f"{num} es un número positivo.")
     print("esta línea siempre se ejecuta.")
     if num < 0:
         print(f"{num} es un número negativo.")
     print("esta línea también se ejecuta siempre.")
```

```
digite un número: 55
55 es un número positivo.
esta línea siempre se ejecuta.
esta línea también se ejecuta siempre.
```

En el ejemplo anterior, `num > 0` y `num < 0` son expresiones de prueba, el cuerpo del `if` solo se ejecuta si se evalúan como `True`.

Cuando la variable `num` recibe un número mayor a 0, la expresión de prueba del primer `if` es verdadera y se ejecutan las sentencias dentro del cuerpo de este, sin embargo, la expresión de prueba del segundo será falsa, pues espera un número menor a 0 y las sentencias dentro este se omiten.

Las sentencias print() de las líneas 4 y 7 quedan fuera de los bloques `if` ya que no tienen el mismo nivel de indentación, como se mencionó arriba. Por lo tanto, se ejecutan independientemente de la expresión de prueba.

Veamos cómo el intérprete ejecuta las líneas una a una, ejecuta el siguiente bloque de código, la herramienta Python Tutor ofrece un entorno donde podemos ver los pasos que realiza el programa para ejecutar el código.

```
[ ]: %%debug_cell_with_pytutor
     num = 5 #Cambia este valor y ejecuta la línea
     if num > 0:
         print(f"{num} es un número positivo.")
     print("esta línea siempre se ejecuta.")
     if num < 0:
         print(f"{num} es un número negativo.")
     print("esta línea también se ejecuta siempre.")
```

```
<IPython.core.display.HTML object>
```

Este código no es muy eficiente en la realidad, puesto que debe evaluar dos condiciones sumamente similares, si un número es mayor que 0, por lógica sabemos que el caso contrario (que sea menor) es falso, por esta razón, al evaluar el primer `if` obtendremos un valor lógico (`True` o `False`) e ingresaremos a su bloque de código únicamente si la condición fue verdadera, aquí es donde entra la sentencia `else`, esta nos permite definir un bloque de código que deseemos ejecutar en caso de que no se cumpla la condición, es decir, una ruta alternativa.

Su sintaxis es la siguiente:

```
if condicion:
```

```
    instrucciones_caso_verdadero

else:

    instrucciones_caso_contrario

instruccion_fuera_del_if
```

La sentencia `if..else` evalúa la expresión de prueba y ejecutará el cuerpo del `if` sólo cuando la condición de prueba sea `True`. Si la condición es `False`, se ejecuta el cuerpo de else. La indentación se utiliza para separar los bloques. Así pues, nuestro ejemplo queda de la siguiente manera usando la sentencia `else` > **Es posible tener sentencias `if` sin estar acompañadas de un `else`, pero no se puede tener un `else` por sí solo, recordemos que esta sentencia se usa para definir caminos alternativos, si no se tiene una condición, ¿de qué camino nos podríamos desviar?**

[ ]:
```python
num = int(input("digite un número: "))
if num > 0:
    print(f"{num} es un número positivo.")
else:
    print(f"{num} es un número negativo.")
```

```
digite un número: -67
-67 es un número negativo.
```

Si deseas ver su ejecución paso a paso, ejecuta el siguiente bloque:

[ ]:
```python
%%debug_cell_with_pytutor
num = 5 #Cambia este valor por números positivos y negativos
if num > 0:
    print(f"{num} es un número positivo.")
else:
    print(f"{num} es un número negativo.")
```

1.2 If anidados

En ocasiones queremos verificar más de una condición para las variables suministradas, en este caso, una de las opciones con las que contamos para solucionar esto son los `if` anidados, Python nos permite construir cuantos condicionales deseemos al interior de otros condicionales, sin embargo, no es una buena práctica anidar demasiados `if`, ya que el código será sumamente complejo de leer, si nuestra solución tiene ese tipo de estructuras demasiado profundas, tal vez lo mejor sea sentarse y pensar en una alternativa distinta. Veamos un ejemplo de cómo usar los if anidados.

Supongamos que queremos saber si un número dado es par o impar, si es par, queremos saber si también es múltiplo de 5 y si es impar, verificaremos si también es múltiplo de 3.

- Un número es par si al dividirlo por 2, la división arroja un residuo 0.
- Un número es múltiplo de 5 o de 3 si al dividirlo por uno de estos valores, el residuo también es 0.

Como vemos, estas condiciones se pueden construir a partir de la operación de módulo (el cual, recordemos que nos retorna el residuo de la división de dos números enteros) y la comparación ==

[ ]:
```
num = int(input("digite un número: "))
if num % 2 == 0: #permite evaluar si un número es par o no
    if num % 5 == 0:#permite evaluar si un número es divisible por 5
        print(f"{num} es un número par y múltiplo de 5")
    else:
        print(f"{num} es un número par")
else:
    if num % 3 == 0:#permite evaluar si un número es divisible por 3
        print(f"{num} es un número impar y múltiplo de 3")
    else:
        print(f"{num} es un número impar")
```

```
digite un número: 50
50 es un número par y múltiplo de 5
```

Y ahora, ¿qué pasaría si nos interesa saber si el número es par **Y/O** múltiplo de 5?... Si tenemos combinaciones de condiciones, nos conviene en este caso usar operadores lógicos, ya que permiten realizar un único bloque de código y de este modo hacemos más eficiente el programa.

Para esto es importante recordar las tablas de verdad de dichos operadores:

A	B	A and B	A or B	not A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

El operador `and` corresponde a una conjunción $\bigwedge$ y solo arroja un valor `True` si ambas condiciones son `True`, mientras que el operador `or` corresponde a la disyunción y retorna un valor `True` si alguna de las condiciones es `True`. Finalmente el operador `not` equivale a una negación y retorna el valor opuesto al que tenía originalmente la condición. Tomando el ejemplo anterior, si quisiéramos detectar los números que sean pares **Y** múltiplos de 5 únicamente, podríamos realizar el siguiente código.

[ ]:
```
num = int(input("digite un número: "))
if num % 2 == 0 and num % 5 == 0:
    print(f"{num} es un número par y múltiplo de 5")
else:
    print(f"{num} el número no cumple la condición")
```

Si quisiéramos complicar un poco más el ejercicio y solicitar que imprima un mensaje cuando los números sean pares **Y** múltiplos de 5 o cuando sean impares **O** múltiplos de 3, nos veríamos en la necesidad de usar ifs anidados nuevamente, debido a que la disyunción de la segunda condición implica que también debería imprimir un mensaje si el número es par y múltiplo de 3. En la

programación es importante comprender muy bien el objetivo del código que vamos a realizar y a partir del mismo buscar cuales son las herramientas que harán más fácil el alcanzar dicho objetivo. Este sería el código.

[ ]:
```
num = int(input("digite un número: "))
if num % 2 == 0:
    if num % 5 == 0:
        print(f"{num} es un número par y múltiplo de 5")
    if num % 3 == 0:
        print(f"{num} es un número par y múltiplo de 3")
else:
    if num % 3 == 0:
        print(f"{num} es un número impar y múltiplo de 3")
    else:
        print(f"{num} es un número impar")
```

```
digite un número: 6
6 es un número par y múltiplo de 3
```

1.3 Sentencia if...elif...else

Finalmente, ¿qué pasa si tenemos la necesidad de que nuestro programa ejecute una única alternativa de muchas? Podríamos usar muchos `if` consecutivos, sin embargo, proceder de esta forma resulta costoso, debido a que el programa tendrá que verificar cada una de las condiciones, aún cuando sabemos que la mayoría fallarán. Para evitar esto Python cuenta con la sentencia `elif`; con ella, si alguna de las condiciones es válida, ejecutará su bloque y luego saltará hasta el final de las condiciones. El único caso en el que Python tendrá que probar todos los condicionales, es cuando ninguno de ellos es válido y deberá ejecutar el bloque de la sentencia `else`.

Un ejemplo perfecto para la aplicación de esta estructura es un menú, cuando el usuario elija una opción, no nos interesa que el programa intente ejecutar las demás. Creemos una calculadora que opere dos números a partir de un menú, ejecuta el siguiente bloque de código con python tutor, prueba con diferentes combinaciones de valores y mira cómo se comporta el programa, ¿qué sucede si ingresas un número que no esté en la lista?

[ ]:
```
%%debug_cell_with_pytutor
a = float(input('digite el número 1: '))
b = float(input('digite el número 2: '))
print(
    """Menú:
    1. Sumar
    2. Restar
    3. Multiplicar
    4. Dividir
    5. Módulo
    6. Potencia
""")
opcion = int(input('Digite la opción: '))
if(opcion == 1):
```

```
    print(f"{a} + {b} = ", a + b)
elif(opcion == 2):
    print(f"{a} - {b} = ", a - b)
elif(opcion == 3):
    print(f"{a} * {b} = ", a * b)
elif(opcion == 4):
    print(f"{a} / {b} = ", a / b)
elif(opcion == 5):
    print(f"{a} % {b} = ", a % b)
elif(opcion == 6):
    print(f"{a} ^ {b} = ", a ** b)
else:
    print("opción inválida")
```