Sentencias condicionales: if ... elif ... else ...

Sentencias condicionales: if ...

La estructura de control `if ...` permite que un programa ejecute unas instrucciones cuando se cumpla una condición. En inglés "if" significa "si" (condición). Una sentencia `if` en Pyhton se escribe así:

```
if condición:
    aquí van las órdenes que se ejecutan si la condición es cierta
    y pueden ocupar varias líneas
```

La primera línea contiene la **condición a evaluar** y es una **expresión lógica**. Esta línea debe terminar siempre por dos puntos (`:`).

A continuación viene el **bloque de órdenes** que se ejecutan cuando la condición se cumple (es decir, cuando la condición es verdadera). Es importante señalar que este bloque debe ir **sangrado**, puesto que Python utiliza el sangrado para reconocer las líneas que forman un **bloque de instrucciones**.

El sangrado que se suele utilizar en Python es de **cuatro espacios**, pero se pueden utilizar más o menos espacios.

Al escribir dos puntos (:) al final de una línea, El editor sangrará automáticamente las líneas siguientes. Para terminar un bloque, basta con volver al principio de la línea.

Ejemplo

El programa siguiente pide un número positivos al usuario y almacena la respuesta en la variable `numero`. Después comprueba si el número es negativo. Si lo es, el programa avisa que no era eso lo que se había pedido. Finalmente, el programa imprime siempre el valor introducido por el usuario. A continuación se pueden ver dos ejecuciones paso a paso de ese programa. En la primera el usuario escribe un valor negativo y en la segunda el usuario escribe un valor positivo:

```
numero = int(input("Introduce un número positivo: "))
if numero < 0:
    print("¡Le he dicho que Escriba un número positivo!")
print("Ha escrito el número", numero)
```

Ejemplo de ejecución if ... 1

```
Introduce un número positivo: -5
¡Le he dicho que escriba un número positivo!
Ha escrito el número -5
```

1. Primero se muestra el mensaje para introducir un número, el valor leído es convertido a entero y luego se guarda en la variable `numero` después de pulsar `<ENTER>`. En este caso

introducimos un número negativo.
2. A continuación se evalúa la condición (`numero < 0`)
3. En este caso, la condición es cierta (`True`), puesto que -5 es inferior a 0.
4. Como la condición es cierta (`True`), a continuación se ejecutan las **instrucciones del bloque `if`**.
   En este caso el bloque consta de una sola instrucción que imprime el texto **"¡Le he dicho que escriba un número positivo!"**.
5. Una vez completado el **bloque if**, el programa salta a la instrucción siguiente al bloque `if ..` La última instrucción del programa imprime el valor introducido y el **programa termina**.

Ejemplo de ejecución if ... 2 :

```
Introduce un número positivo: 10
Ha escrito el número 10
```

1. Primero se muestra el mensaje para introducir un número, el valor leído es convertido a entero y luego se guarda en la variable `numero` después de pulsar `<ENTER>`
2. A continuación se evalúa la condición (`numero < 0`)
3. En este caso, la condición es falsa (`False`), puesto que 10 no es inferior a 0.
4. Como la condición es falsa (`False`), **no se ejecutan las instrucciones del bloque `if`**.
5. El programa salta a la instrucción siguiente al bloque `if ..` La última instrucción del programa imprime el valor introducido y el **programa termina**.

Bifurcaciones: if ... else ...

La estructura de control `if ... else ...` permite que un programa ejecute unas instrucciones **cuando se cumple** una condición y otras instrucciones **cuando no se cumple** esa condición. En inglés "if" significa "si" (condicional) y "else" significa "si no". La orden en Pyhton se escribe así:

```
if condición:
    aquí van las órdenes que se ejecutan si la condición es cierta
    y que pueden ocupar varias líneas
else:
    y aquí van las órdenes que se ejecutan si la condición es falsa
    y que también pueden ocupar varias líneas
```

La primera línea contiene la instrucción `if` y a continuación la **condición** a evaluar. Esta línea debe terminar siempre por dos puntos `:`.

A continuación viene el bloque de órdenes que se ejecutan cuando la condición se cumple (es decir, cuando la condición es verdadera). Como vimos en el apartado anterior este bloque debe ir sangrado

Después viene la línea con la orden `else`, que indica a Python que el bloque que viene a continuación se tiene que ejecutar cuando la condición no se cumpla (es decir, cuando sea falsa).

Esta línea también debe terminar siempre por dos puntos (:). La línea con la orden `else` no debe incluir nada más que el else y los dos puntos.

En último lugar está el bloque de instrucciones sangrado que corresponde al else.

Ejemplo

El programa siguiente pregunta la edad al usuario y almacena la respuesta en la variable "edad". Después comprueba si la edad es inferior a 18 años. Si esta comparación es cierta, el programa escribe que es menor de edad y si es falsa escribe que es mayor de edad. Finalmente el programa siempre se despide, ya que la última instrucción está fuera de cualquier bloque y por tanto se ejecuta siempre.

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
print("¡Hasta la próxima!")
```

Ejemplo de ejecución if ... else 1

```
¿Cuántos años tiene? 17
Es usted menor de edad
¡Hasta la próxima!
```

Primero se muestra por pantalla la pregunta y se espera que el usuario escriba la respuesta y pulse `<ENTER>`. La respuesta es convertida a entero y almacenada en la variable `edad`

En este caso la condición es cierta (`True`), puesto que 17 es menor a 18.

Como la condición es cierta (`True`), a continuación se ejecutan las instrucciones del bloque `if`. En este caso el bloque consta de una sola instrucción que imprime el texto "Es usted menor de edad".

Una vez completado el bloque `if`, el programa salta a la instrucción siguiente al bloque `if .. else ...` La última instrucción del programa imprime la despedida y el programa termina

Ejemplo de ejecución if ... else 1

```
¿Cuántos años tiene? 22
Es usted menor de edad
¡Hasta la próxima!
```

1. Primero se muestra por pantalla la pregunta y se espera que el usuario escriba la respuesta y pulse `<ENTER>`. La respuesta es convertida a entero y almacenada en la variable `edad`
2. En este caso la condición es cierta (`False`), puesto que 22 no es menor a 18.
3. Como la condición es cierta (False), a continuación se ejecutan las instrucciones del bloque `else`. En este caso el bloque consta de una sola instrucción que imprime el texto "Es usted mayor de edad".
4. Una vez completado el bloque `else`, el programa salta a la instrucción siguiente al bloque `if .. else ...` La última instrucción del programa imprime la despedida y el programa termina

Aunque no es aconsejable, en vez de un bloque `if ... else ...` se podría escribir el mismo programa con dos
bloques `if ...`

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
if edad >= 18:
    print("Es usted mayor de edad")
print("¡Hasta la próxima!")
```

Es mejor no hacerlo así por dos motivos:

- Al poner dos bloques if estamos obligando a Python a evaluar siempre las dos condiciones, mientras que en un bloque `if ... else ...` sólo se evalúa una condición. En un programa sencillo la diferencia no es apreciable, pero en programas que ejecutan muchas comparaciones, el impacto si que puede ser apreciable.
- Utilizando `else` nos ahorramos escribir una condición (además, escribiendo la condición nos podemos equivocar, pero escribiendo else no). Si, por ejemplo, en la condición `edad >= 18` nos equivocamos y ponemos `edad > 18` si introducimos como edad el valor **18** el programa no mostraría si la persona es mayor o menor de edad.

Instrucción pass

Si por algún motivo no se quisiera ejecutar ninguna orden en alguno de los bloques de `if ... else ...`, el bloque de órdenes debe contener al menos la orden `pass` (esta orden le dice a Python que no tiene que hacer nada).

Ejemplo

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 120:
    pass
else:
    print("¡No me lo creo!")
print("Usted dice que tiene", edad, "años.")
```

Evidentemente este programa podría simplificarse cambiando la desigualdad. Era sólo un ejemplo para mostrar cómo se utiliza la orden `pass`.

```
edad = int(input("¿Cuántos años tiene? "))
if edad >= 120:
    print("¡No me lo creo!")
print("Usted dice que tiene", edad, "años."
```

Sangrado de los bloques

Un bloque de instrucciones puede contener varias instrucciones. Todas las instrucciones del bloque tener el mismo sangrado (mismo número de espacios al principio):

Ejemplo

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
    print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

Se aconseja utilizar siempre el mismo número de espacios en el sangrado, aunque Python permite que cada bloque tenga un número distinto. El siguiente programa es correcto:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
        print("Es usted menor de edad")
        print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

Lo que no se permite es que en un mismo bloque haya instrucciones con distintos sangrados. Dependiendo del orden de los sangrados, el mensaje de error al intentar ejecutar el programa será diferente:

Sangrado incorrecto. Ejemplo 1

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
        print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

En este caso, el mensaje de error será:

```
IndentationError: unexpected indent
```

La primera instrucción determina el sangrado de ese bloque, por lo que al encontrar la segunda instrucción, con un sangrado mayor, se produce el error "**unexpected ident**" (sangrado inesperado).

Sangrado incorrecto. Ejemplo 2

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
  print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

En este caso, el mensaje de error será:

```
unindent does not match any outer indentation level
```

La primera instrucción determina el sangrado de ese bloque, por lo que al encontrar la segunda instrucción, con un sangrado menor, Python entiende que esa instrucción pertenece a otro bloque, pero como no hay ningún bloque con ese nivel de sangrado, se produce el eror "**unindent does not match any outer indentation level**" (el sangrado no coincide con el de ningún nivel superior).

Sangrado incorrecto. Ejemplo 3

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
print("Recuerde que está en la edad de aprender")
else:
    print("Es usted mayor de edad")
    print("Recuerde que debe seguir aprendiendo")
print("¡Hasta la próxima!")
```

En este caso, el mensaje de error será:

```
invalid syntax
```

Como la segunda instrucción no tiene sangrado, Python entiende que la bifurcación `if` ha terminado, por lo que al encontrar un `else` sin su `if` correspondiente se produce el error ""**invalid syntax**"" (sintaxis no válida).

Sentencias condicionales anidadas

Una sentencia condicional puede contener a su vez otra sentencia anidada.

Ejemplo

```
print("Piense un número de 1 a 4.")
print("Conteste S (sí) o N (no)a mis preguntas.")
primera = input("¿El número pensado es mayor que 2? ")
if primera == "S":
    segunda = input("¿El número pensado es mayor que 3? ")
    if segunda == "S":
        print("El número pensado es 4.")
    else:
        print("El número pensado es 3")
else:
    segunda = input("¿El número pensado es mayor que 1? ")
    if segunda == "S":
        print("El número pensado es 2.")
    else:
        print("El número pensado es 1.")
print("¡Hasta la próxima!")
```

Ejemplo de ejecución

```
Piense un número de 1 a 4.
Conteste S (sí) o N (no) a mis preguntas.
¿El número pensado es mayor que 2? S
¿El número pensado es mayor que 3? N
El número pensado es 3
¡Hasta la próxima!
```

Más de dos alternativas: if ... elif ... else ...

La estructura de control `if ... elif ... else ...` permite encadenar varias condiciones. `elif` es una contracción de "**else if**". La orden en Pyhton se escribe así:

```
if condición_1:
    bloque 1
elif condición_2:
    bloque 2
else:
    bloque 3
```

- Si se cumple la condición 1, se ejecuta el bloque 1
- Si no se cumple la condición 1 pero sí que se cumple la condición 2, se ejecuta el bloque 2
- Si no se cumplen ni la condición 1 ni la condición 2, se ejecuta el bloque 3.

Esta estructura es equivalente a la siguiente estructura de `if ... else ...`

```
if condición_1:
    bloque 1
else:
    if condición_2:
        bloque 2
    else:
        bloque 3
```

Se pueden escribir tantos bloques `elif` como sean necesarios. El bloque `else` (que es opcional) se ejecuta si no se cumple ninguna de las condiciones anteriores.

En las estructuras `if ... elif ... else ...` el orden en que se escriben los casos es importante y, a menudo, se pueden simplificar las condiciones ordenando adecuadamente los casos.

Podemos distinguir dos tipos de situaciones:

1. Cuando los casos son mutuamente excluyentes.

Consideremos un programa que pide la edad y en función del valor recibido da un mensaje diferente. Podemos distinguir, por ejemplo, tres situaciones:

- si el valor es negativo, se trata de un error
- si el valor está entre 0 y 17, se trata de un menor de edad
- si el valor es superior o igual a 18, se trata de un menor de edad

Los casos son mutuamente excluyentes, ya que un valor sólo puede estar en uno de los 3 casos.

Un posible programa es el siguiente:

```
edad = int(input("¿Cuántos años tiene? "))
if edad >= 18:
    print("Es usted mayor de edad")
elif edad < 0:
    print("No se puede tener una edad negativa")
else:
     print("Es usted menor de edad")
```

El programa anterior funciona correctamente, pero los casos están desordenados. Es mejor escribirlos en orden, para asegurarnos de no olvidar ninguna de las posibles situaciones. Por ejemplo, podríamos escribirlos de menor a mayor edad, aunque eso obliga a escribir otras condiciones:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 0:
    print("No se puede tener una edad negativa")
elif edad >= 0 and edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
```

En el programa anterior se pueden simplificar las comparaciones:

```
edad = int(input("¿Cuántos años tiene? "))
if edad < 0:
    print("No se puede tener una edad  negativa")
elif edad < 18:
    print("Es usted menor de edad")
else:
    print("Es usted mayor de edad")
```

Este último programa también funciona correctamente porque, en una estructura `if ... elif .. else ...`, cuando **se cumple** una de las comparaciones Python ya **no considera** las siguientes condiciones, se cumplan o no. Por ejemplo, en la segunda comparación no hace falta comprobar que la edad sea mayor que 0 porque si hubiera sido menor que 0 hubiera cumplido la primera comparación y no estaría evaluando la segunda comparación, así que si está evaluando la segunda comparación es que no se cumple la primera, es decir, que la edad es mayor que 0.

Pero hay que tener cuidado, porque si los casos del programa anterior se ordenan al revés manteniendo las condiciones, el programa no funcionaría como se espera, puesto que al escribir un valor negativo mostraría el mensaje "Es usted menor de edad".

```
# Este programa no funciona correctamente

edad = int(input("¿Cuántos años tiene? "))
if edad < 18:
    print("Es usted menor de edad")
elif edad < 0:
    print("No se puede tener una edad negativa")
else:
    print("Es usted mayor de edad")
```

2. Cuando unos casos incluyen a otros

Consideremos un programa que pide un valor y nos dice si es:

- múltiplo de dos
- múltiplo de cuatro (y de dos)
- si no es múltiplo de dos

> Nota: El valor 0 se considerará múltiplo de 4 y de 2.

Los casos no son mutuamente excluyentes, puesto que los múltiplos de cuatro son también múltiplos de dos. El siguiente programa no sería correcto:

```
# Este programa no funciona correctamente
numero = int(input("Escriba un número: "))
if numero % 2 == 0:
    print(numero, "es múltiplo de dos")
elif numero % 4 == 0:
    print(numero, "es múltiplo de cuatro y de dos")
else:
    print(numero, "no es múltiplo de dos")
```

El error de este programa es que el número que si se escribe un múltiplo de cuatro, como también es múltiplo de dos, cumple la primera condición y el programa ejecuta el primer bloque de instrucciones, sin llegar a comprobar el resto de condiciones.

Una manera de corregir ese error es añadir en la primera condición (la de if) que el número no sea múltiplo de cuatro.

```
numero = int(input("Escriba un número: "))
if numero % 2 == 0 and numero % 4 != 0:
    print(numero, "es múltiplo de dos")
elif numero % 4 == 0:
    print(numero, "es múltiplo de cuatro y de dos")
else:
    print(numero, "no es múltiplo de dos")
```

Este programa funciona porque los múltiplos de cuatro también son múltiplos de dos y el programa sólo evalúa la segunda condición (la de elif) si no se ha cumplido la primera.

Pero todavía podemos simplificar más el programa, ordenando de otra manera los casos:

```
numero = int(input("Escriba un número: "))
if numero % 4 == 0:
    print(numero, "es múltiplo de cuatro y de dos")
elif numero % 2 == 0:
    print(numero, "es múltiplo de dos")
else:
    print(numero, "no es múltiplo de dos")
```

Este programa funciona correctamente ya que aunque la segunda condición (la de `elif`) no distingue entre múltiplos de dos y de de cuatro, si se escribe un múltiplo de cuatro, el programa no llega a evaluar la segunda condición porque se cumple la primera (la de if).

En general, el orden que permite simplificar más las expresiones suele ser considerar **primero los casos particulares** y **después los casos generales**.

Si las condiciones `if ... elif ...` cubren todas las posibilidades, se puede no escribir el bloque else:

```
numero = int(input("Escriba un número: "))
if numero >= 0:
    print("Ha escrito un número positivo")
elif numero < 0:
    print("Ha escrito un número negativo")
```

A menudo el último bloque `elif ...` se sustituye por un bloque else:

```
numero = int(input("Escriba un número: "))
if numero >= 0:
    print("Ha escrito un número positivo")
else:
    print("Ha escrito un número negativo")
```

Condiciones no booleanas

Dado que **cualquier variable** puede interpretarse como una variable booleana, si la condición es una comparación con cero, podemos omitir la comparación.

El programa siguiente:

```
numero = int(input("Escriba un número: "))
if numero % 2 != 0:
    print(numero, "es impar")
else:
    print(numero, "es par")
```

Se podría escribir omitiendo la comparación:

```
numero = int(input("Escriba un número: "))
if numero % 2:
    print(numero, "es impar")
else:
    print(numero, "es par")
```

En este programa, si el número es impar, `numero % 2` da como resultado **1**. Y como el valor booleano de un número diferente de cero es `True` (es decir, `bool(1)` es `True`), la condición se estaría cumpliendo.

Si la comparación es una **igualdad**, se puede utilizar el operador `not`. Por ejemplo, el programa siguiente:

```
numero = int(input("Escriba un número: "))
if numero % 2 == 0:
    print(numero, "es par")
else:
    print(numero, "es impar")
```

Se podría escribir omitiendo la comparación:

```python
numero = int(input("Escriba un número: "))
if not numero % 2:
    print(numero, "es par")
else:
    print(numero, "es impar")
```

En este programa, si el número es par, `numero % 2` da como resultado **0**. El valor booleano de cero es `False` (es decir, `bool(0)` es `False`), pero al negarse con `not`, la condición se estaría cumpliendo (ya que `not False` es `True`).

Cuando se está aprendiendo a programar, esta notación puede resultar un poco crítpica, por lo que se recomienda empezar escribiendo las comparaciones completas. Más adelante, cuando nos hayamos familiarizado con las expresiones lógicas, nos resultará más natural utilizarla.

Referencias