😎 Condicionales

Sentencias condicionales: if ...

La estructura de control **if ...** permite que un programa ejecute una o más instrucciones cuando se cumpla una condición o más.

Sintaxis de la sentencia condicional if ...

La sintaxis de la construcción if es la siguiente:

if condición:

```
    bloque de instrucciones
```

La ejecución de esta construcción es la siguiente:

- **La condición se evalúa siempre**

> **Si el resultado es *True* se ejecuta el bloque de instrucciones.**

> **Si el resultado es *False* no se ejecuta el bloque de instrucciones.**

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 instrucciones que se ejecutan cuando la condición es verdadera. Es importante indicar que este bloque debe ir **indentado**, porque Python utiliza el indentado para reconocer las líneas que forman un bloque de instrucciones.

Al escribir dos puntos (:) al final de una línea, el editor identará automáticamente las líneas siguientes.

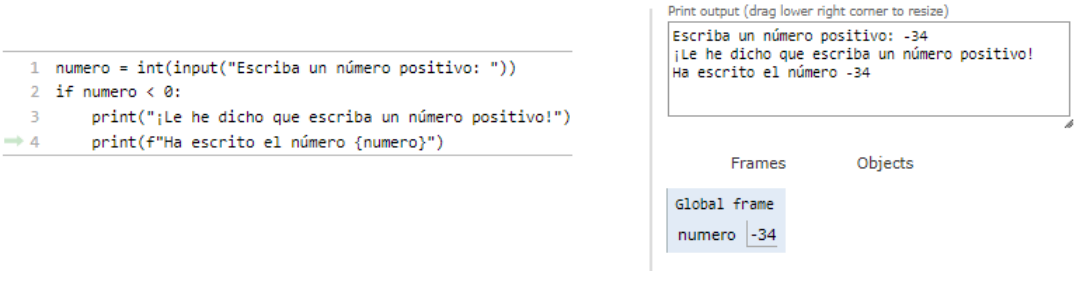
Programas de ejemplo con sentencias condicionales if ...

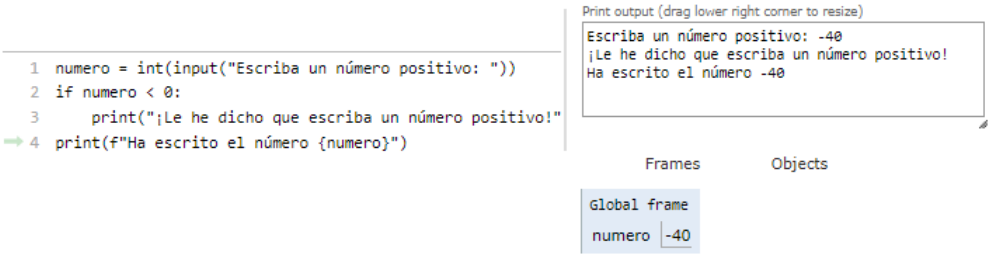
Ejemplo. El programa siguiente pide un número positivo al usuario y almacena la respuesta en la variable "numero". Después comprueba si el número es negativo. Si lo es, el programa avisa que no era eso lo que se había pedido. Finalmente, el programa imprime siempre el valor introducido por el usuario.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

Ejemplo de if ... con la condición verdadera

In [1]:
```
numero = int(input("Escriba un número positivo: "))
if numero < 0:
    print("¡Le he dicho que escriba un número positivo!")
    print(f"Ha escrito el número {numero}")
```

```
Escriba un número positivo: 15
```

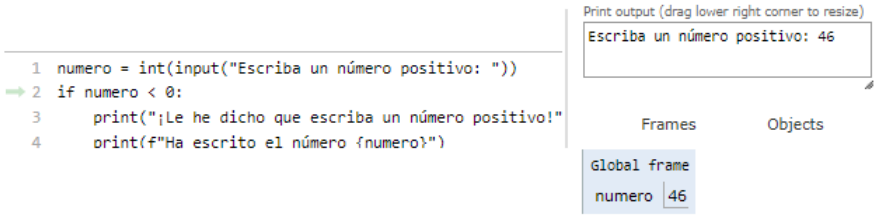


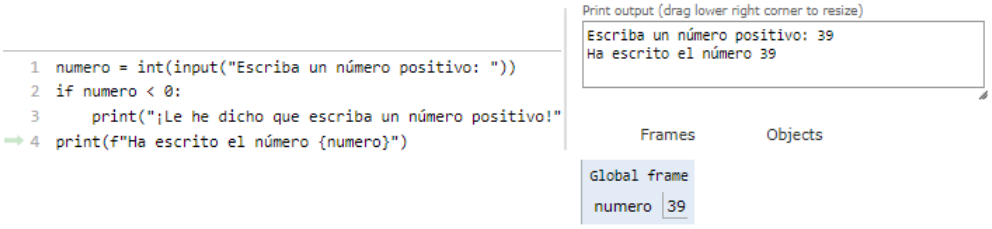


Ejemplo de if ... con la condición falsa

In [2]:
```
numero = int(input("Escriba un número positivo: "))
if numero < 0:
    print("¡Le he dicho que escriba un número positivo!")
    print(f"Ha escrito el número {numero}")
```

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





Bifurcaciones

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" (condición) y "else" significa "si no".

Sintaxis de la sentencia condicional if ... else ...

La sintaxis de la construcción if ... else ... es la siguiente:

if condición:

```
    bloque de instrucciones (1)
```

else:

```
    bloque de instrucciones (2)
```

La ejecución de esta construcción es la siguiente:

- **La condición se evalúa siempre.**
  - **Si el resultado es *True* se ejecuta solamente el bloque de instrucciones 1**
  - **Si el resultado es *False* se ejecuta solamente el bloque de instrucciones 2**

La primera línea contiene la condición a evaluar. Esta línea debe terminar siempre por dos puntos (:).

A continuación viene el bloque de instrucciones que se ejecutan cuando la condición es verdadera.

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 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.

Programas de ejemplo con sentencias condicionales if ... 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.***

Ejemplo de if ... else ...

In [3]:
```
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!")
```

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

In [4]:
```
edad = int(input("¿Cuántos años tiene? "))
if edad >= 18:
    print("Es usted mayor de edad")
else:
    print("Es usted menor de edad")
print("¡Hasta la próxima!")
```

```
¿Cuántos años tiene? 34
Es usted mayor de edad
¡Hasta la próxima!
```

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

In [5]:
```
edad = int(input("¿Cuántos años tiene? "))
if edad < 120:
    pass
else:
    print("¡No me lo creo!")
print(f"Usted dice que tiene {edad} años.")
```

```
¿Cuántos años tiene? 56
Usted dice que tiene 56 años.
```

Evidentemente este programa podría simplificarse cambiando la desigualdad.

In [6]:
```
edad = int(input("¿Cuántos años tiene? "))
if edad >= 120:
    print("¡No me lo creo!")
print(f"Usted dice que tiene {edad} años.")
```

```
¿Cuántos años tiene? 48
Usted dice que tiene 48 años.
```

El primer ejemplo era sólo un ejemplo para mostrar cómo se utiliza la instrucción **pass**. Normalmente, la instrucción pass se utiliza para "llenar" un bloque de instrucciones que todavía no hemos escrito y poder ejecutar el programa, ya que Python requiere que se escriba alguna instrucción en cualquier bloque.

Indentado de los bloques

Un bloque de instrucciones puede contener varias instrucciones. Todas las instrucciones del bloque deben tener el mismo indentado:

In [7]:
```
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!")
```

```
¿Cuántos años tiene? 55
Es usted mayor de edad
Recuerde que debe seguir aprendiendo
¡Hasta la próxima!
```

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

In [8]:
```
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!")
```

```
¿Cuántos años tiene? 55
Es usted mayor de edad
Recuerde que debe seguir aprendiendo
¡Hasta la próxima!
```

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

En este primer caso, la primera instrucción determina el indentado de ese bloque, por lo que al encontrar la segunda instrucción, con un sangrado mayor, se produce el error **"unexpected indent"** (sangrado inesperado).

In [9]:
```
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!")
```

```
  Input In [9]
    print("Recuerde que está en la edad de aprender")
    ^
IndentationError: unexpected indent
```

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

In [11]:
```
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!")
```

```
  File <tokenize>:4
    print("Recuerde que está en la edad de aprender")
    ^
IndentationError: unindent does not match any outer indentation level
```

En este tercer caso, como la segunda instrucción no tiene indentado, Python entiende que la bifurcación if ha terminado, por lo que al encontrar un else sin su if correspondiente se produce el error **"invalid sintax"** (sintaxis no válida).

In [12]:
```
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!")
```

```
  Input In [12]
    else:
    ^
SyntaxError: invalid syntax
```

Sentencias condicionales anidadas

Una sentencia condicional puede contener a su vez otra sentencia anidada. Por ejemplo, el programa siguiente muestra el color obtenido al mezclar dos colores en la pantalla.

.

Ejemplo de sentencias condicionales anidadas

In [13]:
```
print("Este programa mezcla dos colores.")
print(" r. Rojo a. Azul")
primera = input(" Elija un color (r o a): ")
if primera == "r":
    print(" a. Azul v. Verde")
    segunda = input(" Elija otro color (a o v): ")

    if segunda == "a":
        print("La mezcla de Rojo y Azul produce Magenta.")
    else:
        print("La mezcla Rojo y Verde produce Amarillo.")

else:
    print(" v. Verde r. Rojo")
    segunda = input(" Elija otro color (v o r): ")

    if segunda == "v":
        print("La mezcla de Azul y Verde produce Cian.")
    else:
        print("La mezcla Azul y Rojo produce Magenta.")

print("¡Hasta la próxima!")
```

```
Este programa mezcla dos colores.
 r. Rojo a. Azul
 Elija un color (r o a): a
 v. Verde r. Rojo
 Elija otro color (v o r): r
La mezcla Azul y Rojo produce Magenta.
¡Hasta la próxima!
```

Se pueden anidar tantas sentencias condicionales como se desee.

Más de dos alternativas:

La construcción **if ... else ...** se puede extender agregando la instrucción **elif**. La estructura de control if ... elif ... else ... permite encadenar varias condiciones. elif es una contracción de else if.

Sintaxis de la sentencia condicional if ... elif ... else ...

La sintaxis de la construcción if ... elif ... else ... es la siguiente:

if condición_1:

    bloque (1)

elif condición_2:

    bloque (2)

else:

    bloque (3)

La ejecución de esta construcción es la siguiente:

- **Si se cumple la condición 1, se ejecuta el bloque 1**
- **Si no se cumple la condición 1 pero sí 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 ... anidados:

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:

- **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 mayor de edad

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

Un posible programa es el siguiente:

In [14]:
```
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")
```

```
¿Cuántos años tiene?-5
No se puede tener una edad negativa
```

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:

In [15]:
```
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")
```

```
¿Cuántos años tiene? 17
Es usted menor de edad
```

En el programa anterior se pueden simplificar las comparaciones:

In [16]:
```
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")
```

```
¿Cuántos años tiene? 17
Es usted menor de edad
```

Estos dos programas son equivalentes porque en una estructura if ... elif .. else cuando se cumple una de las comparaciones Python ya no evalúa las siguientes condiciones.

En este caso, si el programa tiene que comprobar la segunda comparación (la del elif), es porque no se ha cumplido la primera (la del if). Y si no se ha cumplido la primera es que edad es mayor que 0, por lo que no es necesario comprobarlo en la segunda condición.

- 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".**

In [17]:
```
# 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")
```

```
¿Cuántos años tiene? 17
Es usted menor de edad
```

- Cuando unos casos incluyen a otros. Consideremos un programa que pide un valor y nos dice:

```
- si es múltiplo de dos,
- si es 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 4 son también múltiplos de 2.

El siguiente programa no sería correcto:

In [18]:
```
# Este programa no funciona correctamente
numero = int(input("Escriba un número: "))
if numero % 2 == 0:
    print(f"{numero} es múltiplo de dos")
elif numero % 4 == 0:
    print(f"{numero} es múltiplo de cuatro y de dos")
else:
    print(f"{numero} no es múltiplo de dos")
```

```
Escriba un número: 16
16 es múltiplo de dos
```

El error de este programa es que si la variable numero cumple la segunda condición, cumple también la primera.

Es decir, si numero es un múltiplo de 4, como también es múltiplo de 2, 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 agregar en la primera condición (la de if) que numero no sea múltiplo de 4

In [19]:
```
numero = int(input("Escriba un número: "))
if numero % 2 == 0 and numero % 4 != 0:
    print(f"{numero} es múltiplo de dos")
elif numero % 4 == 0:
    print(f"{numero} es múltiplo de cuatro y de dos")
else:
    print(f"{numero} no es múltiplo de dos")
```

```
Escriba un número: 4
4 es múltiplo de cuatro y de dos
```

También se podría escribir de esta manera:

In [20]:
```
numero = int(input("Escriba un número: "))
if numero % 2 == 0 and numero % 4 != 0:
    print(f"{numero} es múltiplo de dos")
elif numero % 2 == 0:
    print(f"{numero} es múltiplo de cuatro y de dos")
else:
    print(f"{numero} no es múltiplo de dos")
```

```
Escriba un número: 4
4 es múltiplo de cuatro y de dos
```

Este programa funciona porque los múltiplos de 4 también son múltiplos de 2 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:

In [21]:
```
numero = int(input("Escriba un número: "))
if numero % 4 == 0:
    print(f"{numero} es múltiplo de cuatro y de dos")
elif numero % 2 == 0:
    print(f"{numero} es múltiplo de dos")
else:
    print(f"{numero} no es múltiplo de dos")
```

```
Escriba un número: 4
4 es múltiplo de cuatro y de dos
```

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

Nota: 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:

In [23]:
```
numero = int(input("Escriba un número: "))
if numero >= 0:
    print("Ha escrito un número positivo")
elif numero < 0:
    print("Ha escrito un número negativo")
```

```
Escriba un número: -1
Ha escrito un número negativo
```

Pero es más habitual sustituir el último bloque elif ... por un bloque else:

In [24]:
```
numero = int(input("Escriba un número: "))
if numero >= 0:
    print("Ha escrito un número positivo")
else:
    print("Ha escrito un número negativo")
```

```
Escriba un número: -1
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. Por ejemplo, el programa siguiente:

In [25]:
```
numero = int(input("Escriba un número: "))
if numero % 2 != 0:
    print(f"{numero} es impar")
else:
    print(f"{numero} es par")
```

```
Escriba un número: 3
3 es impar
```

Se podría escribir omitiendo la comparación:

In [26]:
```
numero = int(input("Escriba un número: "))
if numero % 2:
    print(f"{numero} es impar")
else:
    print(f"{numero} es par")
```

```
Escriba un número: 3
3 es impar
```

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:

In [27]:
```
numero = int(input("Escriba un número: "))
if numero % 2 == 0:
    print(f"{numero} es par")
else:
    print(f"{numero} es impar")
```

```
Escriba un número: 3
3 es impar
```

Se podría escribir omitiendo la comparación:

In [28]:
```
numero = int(input("Escriba un número: "))
if not numero % 2:
    print(f"{numero} es par")
else:
    print(f"{numero} es impar")
```

```
Escriba un número: 3
3 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.