GOBIERNO DE ESPAÑA
MINISTERIO DE HACIENDA Y FUNCIÓN PÚBLICA
INAP
INSTITUTO NACIONAL DE ADMINISTRACIÓN PÚBLICA

Introducción a la programación en Python

Tema 3. Toma de decisiones y bucles

Autor: Jesús Moreno Jabato

Contenido

1. Condicionales

La estructura de control if ... permite que un programa ejecute unas instrucciones cuando se cumplan una condición. En inglés "if" significa "si" (condición).

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
```

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.

En 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 puede ver el caso que ha escrito un número negativo y devuelve el aviso.

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

Escriba un número positivo: -5
¡Te he dicho que escribas un número positivo!
Has escrito el número -5
```

1.1. 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" (condición) 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 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. 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.

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.

En 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("Eres mayor de edad")
print("¡Hasta luego!")

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

1.2. Varios if

Aunque no es aconsejable, en vez de un bloque if ... else ... se podría escribir un 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 luego!")
```

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 puede ser apreciable.
- Utilizando else nos ahorramos escribir una condición. Además escribiendo la condición nos podemos equivocar pero escribiendo else no.

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

```
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.

1.3. Sangrado de los bloques

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

```
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 de la vida")
print("¡Hasta luego!")
```

Se aconseja utilizar siempre el mismo número de espacios en el sangrado, aunque Python permite que cada bloque tenga un número distinto.

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:

- En este primer caso, 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 indent" (sangrado inesperado).

```
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 de la vida")
print("¡Hasta luego!")
```

- En este segundo caso, 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 error "unindent does not match any outer indentation level" (el sangrado no coincide con el de ningún nivel superior).

```
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 de la vida")
print("¡Hasta luego!")
```

- En este tercer caso, 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).

```
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 de la vida ")
print("¡Hasta la próxima!")
```

1.4. Sentencias condicionales anidadas

Una sentencia condicional puede contener a su vez otra sentencia anidada. Se pueden anidar tantas sentencias condicionales como se desee.

Por ejemplo, el programa siguiente "adivina" el número pensado por el usuario:

```
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!")

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!
```

1.5. 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 ... 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.

Un ejemplo:

```
print("COMPARADOR DE NÚMEROS")
numero_1 = float(input("Escriba un número: "))
numero_2 = float(input("Escriba otro número: "))

if numero_1 > numero_2:
    print(f"Menor: {numero_2} Mayor: {numero_1}")
elif numero_1 < numero_2:
    print(f"Menor: {numero_1} Mayor: {numero_2}")
else:
    print("Los dos números son iguales")
```

2. Iteraciones

2.1. El bucle FOR

Un bucle es una estructura de control que repite un bloque de instrucciones. Un bucle for es un bucle que repite el bloque de instrucciones un número determinado de veces.

El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

La sintaxis de un bucle for es la siguiente:

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.

El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento que se pueda recorrer. Por ejemplo:

```
print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ")
print()
print("Final")

Comienzo
Hola Hola Hola
Final
```

En el ejemplo anterior, los valores que toma la variable no son importantes, lo que importa es que la lista tiene tres elementos y por tanto el bucle se ejecuta tres veces. El siguiente programa produciría el mismo resultado que el anterior:

```
print("Comienzo")
for i in [1, 1, 1]:
    print("Hola ")
print()
print("Final")

Comienzo
Hola Hola Hola
Final
```

Si la lista está vacía, el bucle no se ejecuta ninguna vez. Por ejemplo:

```
print("Comienzo")
for i in []:
    print("Hola ")
print()
print("Final")

Comienzo

Final
```

En los ejemplos anteriores, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza.

Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento recorrible. Por ejemplo:

```
print("Comienzo")
for i in [3, 4, 5]:
    print(f"Hola. Ahora i vale {i} y su cuadrado {i ** 2}")
print("Final")

Comienzo
Hola. Ahora i vale 3 y su cuadrado 9
Hola. Ahora i vale 4 y su cuadrado 16
Hola. Ahora i vale 5 y su cuadrado 25
Final
```

La lista puede contener cualquier tipo de elementos, no sólo números. El bucle se repetirá siempre tantas veces como elementos tenga la lista y la variable irá tomando los valores de uno en uno. Por ejemplo:

```
print("Comienzo")
for i in ["Alba", "Benito", 27]:
    print(f"Hola. Ahora i vale {i}")
print("Final")

Hola. Ahora i vale Alba
Hola. Ahora i vale Benito
Hola. Ahora i vale 27
Final
```

La costumbre más extendida es utilizar la letra *i* como nombre de la variable de control, pero se puede utilizar cualquier otro nombre válido.

La variable de control puede ser una variable empleada antes del bucle. El valor que tuviera la variable no afecta a la ejecución del bucle, pero cuando termina el bucle, la variable de control conserva el último valor asignado:

```
i = 10
print(f"El bucle no ha comenzado. Ahora i vale {i}")
for i in [0, 1, 2, 3, 4]:
    print(f"{i} * {i} = {i ** 2}")
print(f"El bucle ha terminado. Ahora i vale {i}")

0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
El bucle ha terminado. Ahora i vale 4
```

Cuando se escriben dos o más bucles seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada bucle establece los valores de la variable sin importar los valores anteriores.

En vez de una lista se puede escribir una cadena, en cuyo caso la variable de control va tomando como valor cada uno de los caracteres:

```
for i in "ALGO":
    print("Dame una", {i})
print("¡ALGO!")

Dame una {'A'}
Dame una {'L'}
Dame una {'G'}
Dame una {'O'}
¡ALGO!
```

2.2. Range

En los ejemplos anteriores se ha utilizado una lista para facilitar la comprensión del funcionamiento de los bucles pero, si es posible hacerlo, se recomienda utilizar tipos range(), entre otros motivos porque durante la ejecución del programa ocupan menos memoria en el ordenador.

El siguiente programa es equivalente al programa del ejemplo anterior:

```
print("Comienzo")
for i in range(3):
    print("Hola )
print()
print("Final")

Comienzo
Hola Hola Hola
Final
```

Otra de las ventajas de utilizar tipos range() es que el argumento del tipo range() controla el número de veces que se ejecuta el bucle.

Esto permite que el número de iteraciones dependa del desarrollo del programa. En el ejemplo siguiente es el usuario quien decide cuántas veces se ejecuta el bucle:

```
veces = int(input("¿Cuántas veces quiere que le salude? "))
for i in range(veces):
    print("Hola ")
print()
print("Adios")

¿Cuántas veces quiere que le salude? 6
Hola Hola Hola Hola Hola Hola
Adios
```

2.3. Contadores y Acumuladores

En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que acumulen valores (acumuladores). Las situaciones pueden ser muy diversas, por lo que simplemente hay aquí un par de ejemplos para mostrar la idea.

Contador

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición.

El ejemplo siguiente es un ejemplo de programa con contador. En este caso la variable que hace de contador es la variable *cuenta*):

```
print("Comienzo")
cuenta = 0
for i in range(1, 6):
    if i % 2 == 0:
        cuenta = cuenta + 1
print("Desde 1 hasta 5 hay", cuenta, "múltiplos de 2")

Comienzo
Desde 1 hasta 5 hay 2 múltiplos de 2
```

Detalles importantes:

- En cada iteración, el programa comprueba si i es múltiplo de 2.
- El contador se modifica sólo si la variable de control i es múltiplo de 2.
- El contador va aumentando de uno en uno.
- Antes del bucle se debe dar un valor inicial al contador (en este caso, 0)

Acumulador

Se entiende por acumulador una variable que acumula el resultado de una operación.

El ejemplo siguiente es un ejemplo de programa con acumulador (en este caso, la variable que hace de acumulador es la variable suma):

```
print("Comienzo")
suma = 0
for i in [1, 2, 3, 4]:
    suma = suma + i
print(f"La suma de los números de 1 a 4 es {suma}")

Comienzo
La suma de los números de 1 a 4 es 10
```

Detalles importantes:

- El acumulador se modifica en cada iteración del bucle (en este caso, el valor de i se añade al acumulador suma).
- Antes del bucle se debe dar un valor inicial al acumulador (en este caso, 0)

2.4. El bucle WHILE

Un bucle while permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).

La sintaxis del bucle while es la siguiente:

```
while condicion:
    cuerpo del bucle
```

Cuando llega a un bucle while, Python evalúa la condición y, si es cierta, ejecuta el cuerpo del bucle. Una vez ejecutado el cuerpo del bucle, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del bucle) una y otra vez mientras la condición sea cierta.

Únicamente cuando la condición sea falsa, el cuerpo del bucle no se ejecutará y continuará la ejecución del resto del programa.

La variable o las variables que aparezcan en la condición se suelen llamar variables de control. Las variables de control deben definirse antes del bucle while y modificarse en el bucle while.

Por ejemplo, el siguiente programa escribe los números del 1 al 3:

```
i = 1
while i <= 3:
    print(i)
    i += 1
print("Programa terminado")

1
2
3
Programa terminado
```

El ejemplo anterior se podría haber programado con un bucle for. La ventaja de un bucle while es que la variable de control se puede modificar con mayor flexibilidad, como en el ejemplo siguiente:

```
i = 1
while i <= 50:
    print(i)
    i = 3*i + 1
print("Programa terminado")
```

```
1
4
13
40
Programa terminado
```

Otra ventaja del bucle while es que el número de iteraciones no están definidas antes de empezar el bucle porque los datos los proporciona el usuario.

Por ejemplo, el siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente:

```
numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")

Escriba un número positivo: -4
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: -8
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: 9
Gracias por su colaboración
```

2.5. Bucles infinitos

Si la condición del bucle se cumple siempre, el bucle no terminará nunca de ejecutarse y tendremos lo que se denomina un bucle infinito. Aunque a veces es necesario utilizar bucles infinitos en un programa, normalmente se deben a errores que se deben corregir.

Los bucles infinitos no intencionados deben evitarse pues significan perder el control del programa. Para interrumpir un bucle infinito, hay que pulsar la combinación de teclas Ctrl+F2.

Al interrumpir un programa se mostrará un mensaje de error similar a éste:

```
Traceback (most recent call last):
  File "ejemplo.py", line 3, in <module>
    print(i)
KeyboardInterrupt
```

Por desgracia, es fácil programar involuntariamente un bucle infinito, por lo que es inevitable hacerlo de vez en cuando, sobre todo cuando se está aprendiendo a programar.

Cuidado al ejecutarlos en vuestras máquinas porque tendréis seguramente que recurrir a matar los procesos.

Estos algunos ejemplos de bucles infinitos:

El programador ha olvidado modificar la variable de control dentro del bucle y el programa imprimirá números 1 indefinidamente:

```
i = 1
while i < 10:
   print(i,)

1 1 1 1 1 1 1 1 ...
```

El programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:

```
i = 1
while i > 0:
        print(i)
        i += 1

1 2 3 4 5 6 7 8 9 10 11 ...
```

Se aconseja expresar las condiciones como desigualdades en vez de comparar valores. En el ejemplo siguiente, el programador ha escrito una condición que se cumplirá siempre y el programa imprimirá números consecutivos indefinidamente:

```
i = 1
while i != 100:
    print(i)
    i += 2

1 3 5 7 9 11 ... 97 99 101
```