Estructuras de control

Las estructuras de control nos permiten crear programas con múltiples caminos posibles a seguir, dependiendo de ciertas condiciones a comprobar. Estas estructuras son también habituales en otros lenguajes de programación... hablamos de *if*, *while* o *for*.

1. Estructuras selectivas: *if*, *if..else*, *if..elif..else*

Si queremos ejecutar un conjunto de instrucciones si se cumple una determinada condición, debemos añadir esta condición en una cláusula `if`, y las instrucciones vinculadas a esa condición deben quedar **tabuladas** respecto a ese *if*. Por ejemplo:

```
numero = int(input("Escribe un número positivo:"))
if numero > 0:
    numero = numero + 1
    print ("El siguiente número es", numero)
print ("Fin del programa")
```

Podemos también distinguir entre dos posibles caminos con la estructura `if..else`. En este caso, debemos tabular el bloque de instrucciones que vaya asociado a cada parte (*if* o *else*):

```
numero = int(input("Escribe un número positivo:"))
if numero > 0:
    numero = numero + 1
    print ("El siguiente número es", numero)
else:
    print ("El número no es positivo")
print ("Fin del programa")
```

Si necesitamos tener más de dos caminos diferentes, podemos anidar estructuras `if..else` unas dentro de otras...

```
numero = int(input("Escribe un número positivo:"))
if numero > 0:
    numero = numero + 1
    print ("El siguiente número es", numero)
else:
    if numero < -10:
        print ("El número es demasiado bajo")
    else:
        print ("El número no es positivo")
print ("Fin del programa")
```

... pero, en lugar de anidar estas estructuras, también podemos utilizar la cláusula `if..elif` para especificar más de un bloque de condiciones. Podemos enlazar tantas clásulas `elif` como necesitemos, y también concluir con una cláusula `else` si queremos, para el último camino a distinguir:

```
numero = int(input("Escribe un número positivo:"))
if numero > 0:
    numero = numero + 1
    print ("El siguiente número es", numero)
elif numero < -10:
    print ("El número es demasiado bajo")
else:
    print ("El número no es positivo")
print ("Fin del programa")
```

No existe cláusula *switch/case* en Python, ya que normalmente lo que se puede hacer con este tipo de estructuras se puede hacer también con *if*.

1.1. Uso abreviado de *if..else*

La estructura `if..else` se puede usar de modo abreviado en una línea para asignar un valor u otro a una variable. Sería el equivalente al *operador ternario ?:* que existe en otros lenguajes, y que no está disponible en Python. Por ejemplo, esta instrucción asigna a la variable `par` un valor *True* si *n* es par, y *False* en caso contrario:

```
par = True if n % 2 == 0 else False
```

2. Estructuras repetitivas o bucles

2.1. El bucle *while*

La estructura `while` tiene una sintaxis similar a *if*. El bloque de instrucciones asociado se va a ejecutar repetidamente mientras se cumpla la condición indicada. Por ejemplo, este código le pide números al usuario repetidamente hasta que escribe un número negativo o cero:

```
numero = int(input())
while numero > 0:
    print ("Has escrito", numero)
    numero = int(input())
print ("Fin del programa")
```

No hay una estructura *do..while* en Python, que sí existe en otros lenguajes, ya que cualquier bucle *do..while* se puede representar como *while* con algunos pequeños cambios.

2.2. El bucle *for*

Sin embargo, como en otros muchos lenguajes, sí hay una cláusula `for` en Python, y se puede utilizar de varias formas:

- Podemos, por ejemplo, explorar una secuencia determinada de valores. En este ejemplo, la variable `valor` tomará los distintos valores indicados en la lista (2, 4 y 5):

```
for valor in [2, 4, 5]:
    print (valor)
```

- También podemos hacer un uso más "tradicional", e iterar desde un valor inicial hasta uno final. En este ejemplo, mostramos por pantalla los números del 1 al 4 (inclusive). Si sólo especificamos un número en el rango, entonces Python cuenta del 0 hasta ese número (sin incluir dicho número).

```
for valor in range(1, 5):
    print (valor)
```

- En el caso de que sólo queramos indicar un número de repeticiones sin intención de usar el contador, podemos especificar un único valor en `range`:

```
for i in range(4):
    ...
```

- Finalmente, podemos establecer un incremento distinto de 1, especificando un tercer parámetro dentro de la opción *range*. Por ejemplo, aquí contamos del 0 al 100 con un incremento de 10 en 10:

```
for valor in range(0, 101, 10):
    print (valor)
```

Ejercicio 1:

Crea un programa llamado `Notas.py` que le pida al usuario 3 notas, y calcule la nota final según estas reglas:

- Si ninguna nota es mayor que 4, la nota final es 0
- Si algunas notas son mayores que 4 (pero no todas), la nota final es 2
- Si todas las notas son mayores que 4, la nota final será el 30% de la primera más el 20% de la segunda más el 50% de la tercera

AYUDA: vídeo con la solución del ejercicio

Ejercicio 2:

Crea un programa llamado `Factura.py` que le pida al usuario precios para una factura, hasta que escriba 0. Entonces, el programa debe mostrar el total de la factura con 2 dígitos decimales.

AYUDA: vídeo con la solución del ejercicio

Ejercicio 3:

Crea un programa llamado `MayorMenor.py` que le pida al usuario que introduzca una secuencia de N números positivos (primero el usuario deberá indicar cuántos números va a introducir). Al final del proceso, el programa deberá mostrar por pantalla el valor del número mayor y el menor introducidos por el usuario. Por ejemplo:

```
Dime cuántos números vas a introducir:
3
Escribe 3 números:
3
7
2
El mayor es 7
El menor es 2
```

Ejercicio 4:

Crea un programa llamado `MCD.py` que le pida al usuario dos números *n1* y *n2* y utilice el algoritmo de Euclides para calcular su máximo común divisor (MCD). Este número es el divisor mayor que tienen en común los dos números. Aplicando el algoritmo de Euclides, se calcula de la siguiente forma:

1. Dividir el mayor de n1 y n2 entre el menor

2. Si la división es exacta (resto 0), el MCD es el número menor
3. Si no, se sustituye el número mayor por el resto de la división, y se vuelve al paso 1

Por ejemplo, para 20 y 12 haríamos algo así:

- Dividimos 20 / 12. No es exacta, y el resto es 8. Reemplazamos 20 por 8
- Dividimos 12 / 8. No es exacta, y el resto es 4. Reemplazamos 12 por 4
- Dividimos 8 / 4. Es exacta, con lo que el MCD es 4.

Ejercicio 5:

Crea un programa llamado `InvertirNumero.py` que le pida al usuario un número entero y construya otro en otra variable que sea el original dado la vuelta. Por ejemplo, si el número inicial es 2356, debe construir el 6532.

3. Gestión de errores mediante excepciones

Al igual que ocurre en otros lenguajes como Java, en Python podemos controlar los errores no deseados de un programa para que, en lugar de que éste termine de forma anómala y muestre un mensaje confuso al usuario, "atrapar" el error producido, mostrar un mensaje más conciso y seguir ejecutando el programa.

Para ello debemos incorporar el código que puede provocar el error dentro de un bloque `try`. Si dicho código se ejecuta con normalidad, al final del mismo se saldrá del bloque `try` y se continuará con la ejecución del programa. De producirse algún error, se acudirá al bloque `except` que pondremos justo a continuación. En este segundo bloque podemos mostrar el mensaje de error que queramos, y luego seguir con la ejecución del programa. También es habitual recibir un objeto en el bloque `except` que recoge el error producido, con lo que podemos mostrar directamente el mensaje de error almacenado en dicho objeto.

El siguiente ejemplo trata de dividir los dos números escritos por el usuario. Se producirá un error si el segundo número es 0, y lo mostraremos en el bloque `except`:

```
try:
    n1 = int(input("Escribe el dividendo:\n"))
    n2 = int(input("Escribe el divisor:\n"))
    resultado = n1 / n2
    print("El resultado es:", resultado)
except Exception as e:
    print("Error:", str(e))
```

También se pueden provocar o lanzar excepciones en un momento determinado con la instrucción `raise`. Esto permite, entre otras cosas, centralizar los errores producidos en un único punto. Por ejemplo, si no quisiéramos dividir números negativos en el ejemplo anterior podríamos provocar una excepción dentro del `try` para que se capture:

```
try:
    n1 = int(input("Escribe el dividendo:\n"))
    n2 = int(input("Escribe el divisor:\n"))
    if n1 < 0 or n2 < 0:
        raise Exception("No se admiten números negativos")
    resultado = n1 / n2
    print("El resultado es:", resultado)
except Exception as e:
    print("Error:", str(e))
```

Ejercicio 6:

Crea un programa llamado `Impares.py` que pida al usuario un número entero positivo y muestre por pantalla todos los números impares desde 1 hasta ese número, separados por comas. Si el número introducido no es un valor numérico entero, o no es positivo, se lo deberá volver a pedir las veces que sean necesarias antes de hacer el conteo, mostrando el mensaje de error correspondiente (por ejemplo, *Número no válido*).