1 Python - Control de flujo

Control de flujo

Python, como el resto de los lenguajes de programación, tenemos varios modos de controlar el flujo del programa.
En Python tenemos tres modos elementos para controlar el flujo:

- ◊ La declaración **if**, que ejecuta un bloque particular de comandos en función del resultado de un test.
- ◊ La declaración **while**, que ejecuta un bloque de comandos mientras que se cumpla un test determinado.
- ◊ La declaración **for loop**, que ejecuta un bloque de comandos un cierto número de veces.

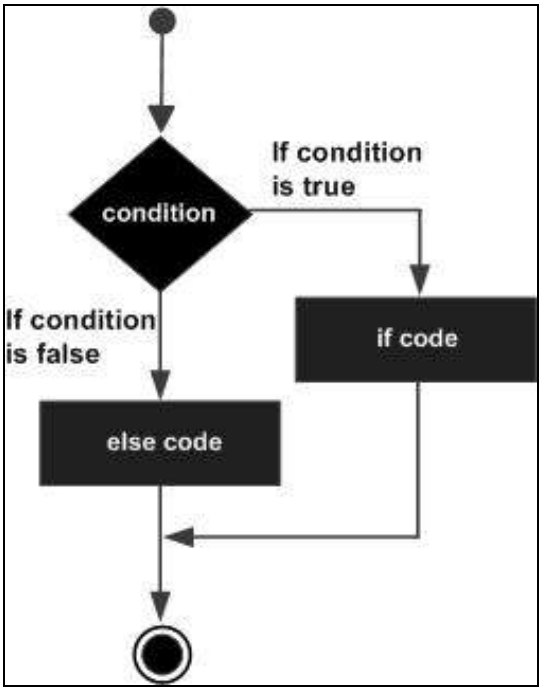
En Python es "obligatorio" definir una estructura de control del siguiente modo:

```
inicio de la estructura de control:
    expresiones
```

Es obligatoria la **IDENTACIÓN**, que no es otra cosa que realizar una sangría de **4 espacios en blanco**.
Recuerda siempre que **en Python la identación es obligatoria**.

1.1 Sumario

1.2 Declaración *if*



Se trata de una estructura de control condicional. Nos permite evaluar si una o más condiciones se cumplen, para decir qué acción vamos a ejecutar. Esta evaluación de condiciones sólo puede dar un resultado **Verdadero** o un resultado **Falso**.
Operadores condicionales: **==** , **!=** , **<** , **>** , **<=** , **>=**
Para evaluar más de una condición simultáneamente, se utilizan los **operadores lógicos**: **and** , **or** , **xor**

- Veamos un ejemplo donde se comprueba si un número es par o impar.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

num = 8
if num % 2 == 0:
    print('El número {} es par'.format(num))
```

```
else:
    print('El número {} es impar'.format(num))
```

- Ejemplo donde se comprueba si dos números son uno mayor que otro o, por el contrario, son iguales:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
numA = 8
numB = 8

if numA > numB:
    print('El número {} es mayor que {}'.format(numA, numB))
elif numA < numB:
    print('El número {} es mayor que {}'.format(numB, numA))
else:
    print('Los dos números son iguales a {}'.format(numA))
```

- Otro modo de hacer este ejercicio pero con "condiciones anidadas" sería el siguiente:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
numA = 8
numB = 10

if numA == numB:
    print('Los dos números son iguales a {}'.format(numA))
else:
    if numA > numB:
        print('El número {} es mayor que {}'.format(numA, numB))
    else:
        print('El número {} es mayor que {}'.format(numB, numA))
```

- Programa que te dice si un número dado está entre dos que también indicaremos en el código:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-
num = 15
numMenor = 10
numMayor = 30

if (num >= numMenor) and (num <= numMayor):
    print('El número {} está entre el {} y el {}'.format(num, numMenor, numMayor))
else:
    print('El número {} NO está entre el {} y el {}'.format(num, numMenor, numMayor))
```

1.3 Bucle *while*

Este bucle, se encarga de ejecutar una misma acción "mientras que" una determinada condición se siga cumpliendo.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

anio = 2008

while anio <= 2015:
    print('Informe del año {}'.format(anio))
    anio += 1
```

Salida:

```
Informe del año 2008
Informe del año 2009
Informe del año 2010
Informe del año 2011
Informe del año 2012
Informe del año 2013
Informe del año 2014
Informe del año 2015
```

También podemos hacer que salga del bucle utilizando la palabra clave reservada **break**.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

anio = 2008

while True:
    print('Informe del año {}'.format(anio))
    anio += 1
    if anio == 2010:
        break
```

Salida:

```
Informe del año 2008
Informe del año 2009
```

Ejemplos:

- Escribe un programa que calcule los números divisibles por 7 que se encuentren entre el 1500 y el 2700:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

divisor = 7
min = 1500
max = 2700

print('Los números del {} al {} divisibles por {} son:'.format(min, max, divisor))
num = min
while num <= 2700:
    if (num % divisor == 0):
        print('- {}'.format(num))
    num += 1
```

Salida:

```
Los números del 1500 al 2700 divisibles por 7 son:
- 1505
- 1512
- 1519
- 1526
- 1533
- 1540
- ...
```

- Escribir un programa que, dado un número **n**, mayor que 0, calcule la siguiente suma **1+2+3+4+...+n**.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

n = 10
x = 1
suma = 0
while x <= n:
    suma += x
    x += 1

print('La suma 1 + 2 + ... + n-1 + n con n = {} es : {}'.format(n, suma))
```

1.4 Bucle *for*

Un bucle *for* nos permite repetir un bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición iteración.

- La sintaxis de un bucle *for* es la siguiente:

```
for variable in elemento_recorrible (lista, cadena, range,...):
    cuerpo
```

- El cuerpo del bucle se ejecuta tantas veces como elementos tenga el "elemento_recorrible".

  Veamos un ejemplo:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

print('Inicio bucle for')
for i in [0, 1, 2]:
    print('Paso {}'.format(i))

print
print('Final del bucle for')
```

  En el ejemplo anterior, si la lista estuviese vacía, el bucle no se ejecutaría ninguna vez.

- La lista podría contener cualquier tipo de elementos:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

print "Inicio bucle for"
for i in ["Andrea", 1.45, 2]:
    print('Elemento de la lista: {}'.format(i))

print
print "Final bucle for"
```

- La variable de control, en nuestros ejemplos anteriores **i**, pudo haber sido utilizada con anterioridad, pero, al finalizar el bucle tendrá el valor con el que se terminó el bucle:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

var = 10
print('La variable de control ahora vale: {}'.format(var))
print
print('Ahora se inicia el bucle for')
for var in ["Andrea", 1.45, 2]:
    print('Elemento: {}'.format(var))

print
print('Final bucle for')
print
print('Ahora la variable de control vale: {}'.format(var))
```

- También es interesante poder recorrer una variable *string* del siguiente modo:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

palabra = "Clemente"
print('Descomponemos la palabra {} en sus letras:'.format(palabra))
for letra in palabra:
    print('- {}'.format(letra))
```

- Podemos anidar, sin problemas, varios bucles *for*. En el siguiente ejemplo vemos cómo se pueden generar las tablas de multiplicación del 2 al 9:

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

print('Tablas de multiplicar del 2 al 9:')
for numA in range(2, 10):
    print('Tabla del {}:'.format(numA))
    for numB in range(1, 11):
```

```
        sol = numA * numB
        print('\t{} * {} = {}'.format(numA, numB, sol))
```

En el ejemplo anterior utilizamos la función range(). También es interesante, en este momento, ver las características de la función xrange() y las diferencias entre ellas dos.

- Muchas veces nos encontraremos en un programa con la necesidad de salir del bucle *for* antes de que este se ejecute completamente. Es decir, puede existir alguna situación en la que nos interese que se pare el bucle por un momento o definitivamente. Para realizar estas tareas en Python tenemos las sentencias **break** y **continue**.

    ◊ La sentencia **break** finaliza el bucle en el que se encuentre y sigue con la ejecución del programa en la siguiente sentencia.
    El uso más común de *break* es cuando se activa una condición externa para producir una salida precipitada del bucle. Tener en cuenta que la sentencia *break* se puede utilizar tanto en los bucles *while* como en los *for*.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

#Primer ejemplo
print('Mostramos las letras de "Python", cuando llega a la "h" se para el bucle')
for letra in 'Python':
   if letra == 'h':
      break
   print('- {}'.format(letra))

#Segundo ejemplo
print('\nMostramos los números del 10 al 1, pero, cuando llega al 5 se para el bucle')
var = 10
while var > 0:
   print('- {}'.format(var))
   var = var - 1
   if var == 5:
      break
```

    ◊ La sentencia **continue** devuelve el control al principio del bucle, es decir, lo que hace es saltarse ese paso si se cumple cierta condición.
    La sentencia *continue* puede utliizarse en los bucles **for** y **while**.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

#Primer ejemplo
for letra in 'Python':
   if letra == 'h':
      continue
   print('- {}'.format(letra))

print

#Segundo ejemplo
var = 10
while var > 0:
   var = var - 1
   if var >= 4 and var <= 7:
      continue
   print('- {}'.format(var))
```

- La sentencia **else** utilizada con bucles.

Python puede emplear la sentencia *else* en los bucles.

- Si *else* se utiliza con *for*, la sentencia *else* es ejecutada cuando el bucle se ha terminado.
- Si *else* se utilza con *while*, la sentencia *else* es ejecutada cuando la condición se hace falsa.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

#Comprobamos qué números son primos desde el 10 al 20
for num in range(10,21):        # Iteración entre el 10 y el 20
```

```
    for i in range (2,num):     # Para iterar sobre los divisores del número
        if num%i == 0:          # Para determinar el primer divisor
            j = num/i           # Para calcular el segundo divisor
            print('{} = {} * {}'.format(num, i, j))
            break               # Para movernos al siguiente número (primer for)
    else:                       # Si no entró en el if... pues se termina el for
        print('{} es número primo.'.format(num))
```

- La sentencia **pass**.

  La sentencia *pass* en Python es utilizada cuando se requiere una sentencia pero no queremos, en ese momento, que se ejecute nada.

  > La sentencia *pass* es la operación **null**; no ocurre nada cuando se ejecuta. Es muy utilizada esta sentencia en sitios donde va a ir código más adelante pero, de momento, no está escrito.

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

for letra in "Python":
    if letra == "h":
        pass
    else:
        print('- {}'.format(letra))

print(\n"Bye!")
```

1.5 Problemas

1.- Ecuación primer grado. Diseñar un programa para resolver cualquier ecuación de primer grado de la forma: **a * x + b = c**

> El programa es muy sencillo, tendrá que pedir los valores de a, b y c, para luego calcular el valor de x = (c - b) / a
> Pero, si a = 0, el programa dará un error, pues se produce una división entre 0.
> Hay que evitar que, en la medida de lo posible, el programa se pare por un error, veamos cómo lo resolvemos:

2.- Par / **Impar.** Diseñar un programa que, dado un número entero, muestre por pantalla el mensaje "El número es par" cuando el número sea par y el mensaje "El número es impar" cuando sea impar. También debe preguntar si se desea seguir trabajando con más números. (Recuerda que un número es par cuando el resto de dividirlo entre 2 sea 0 y, en caso contrario, será impar).

3.- Después de ejecutar el siguiente fragmento de programa, ¿cuál será el valor final de la variable **x**?

```
# !/usr/bin/python3
# -*- coding: utf-8 -*-

x = 0
n = 16
while n % 2 == 0:
    x = x + n
    n = n / 2

print(x)
```

4.- Escribir un programa que calcule el mínimo, el máximo y la media de una lista de números enteros positivos introducidos por el usuario. La lista finalizará cuando se introduzca un número menor o igual a 0.

5.- Escribir un programa que solicite al usuario un número **n** y luego muestre por pantalla la siguiente salida:

> 1
> 1 2
> 1 2 3
> ...
> 1 2 3 ... N

6.- Escribir un programa que lea de forma repetida un número **n**. Para cada número leído el programa calculará la suma 1 + 2 + 3 + ... + n. Una vez mostrado el resultado, el programa preguntará al usuario si desea continuar; si se introduce **s** el programa continuará la ejecución, en caso contrario finalizará.

7.- Escribir un programa que lea un mes en número (1 para enero, 2 para febrero, etc.) y un año, e indique el número de días de ese mes. Para ver si un año es bisiesto seguir las siguientes reglas:

1. Si el año es divisible por 4, vaya al paso 2. Si no es así, vaya al paso 5.
2. Si el año es divisible por 100, vaya al paso 3. Si no es así, dirijase al paso 4.
3. Si el año es divisible por 400, vaya al paso 4. Si no es así, vaya al paso 5.
4. El año es un año bisiesto (tiene 366 días).
5. El año no es un año bisiesto (tiene 365 días).

8.- Escribir un programa que calcule el número Pi con cuatro cifras decimales. Para su cálculo se puede utilizar la serie de Leibniz

9.- Un número perfecto es un entero positivo igual a la suma de sus divisores propios. Un divisor propio es un entero positivo distinto que el número en sí mismo, que divide al número de forma exacta (es decir, sin resto). Por ejemplo, 6 es un número perfecto, porque la suma de sus divisores propios 1, 2 y 3 es igual a 6. 8 NO es un número perfecto, porque la suma de sus divisores propios 1, 2 y 4 es distinto de 8. Escribir un programa que acepte un entero positivo y determine si es un número perfecto. Igualmente, también debe mostrar todos los divisores propios del número.

10.- Escribir un programa que lea un número entero y lo descomponga en factores primos. Ejemplo:

18 = 2 * 3 * 3
11 = 11
35 = 5 * 7

11.- Escribir un juego de adivinanza. El programa pedirá al usuario dos números (el número inferior y el número superior), por ejemplo 1 y 100, y un número de intentos, por ejemplo, 4. El programa obtendrá, a continuación, un número aleatorio entre 1 y 100, y el usuario deberá adivinarlo utilizando como mucho 4 intentos. Cada vez que el usuario introduce un número, el programa le dice si es mayor o menor. Al final, el programa indica si se ha ganado o no.

12.- Escribir un programa que pida un número al usuario y muestre por pantalla el resultado de su raíz cuadrada. Para su resolución no se puede utilizar ninguna función **math**. Para realizar este ejercicio se puede utilizar un método aproximado que permita llegar a la solución con un error menor de 0,001.

13.- Escriba un programa que solicite del usuario tres números y los ordene de mayor a menor.

14.- Escribir un programa que, dada una cierta cantidad de dinero, calcule cuál es el número mínimo de monedas de curso legal que equivalen a dicha cantidad.

15.- Escribir un programa que muestre por pantalla si un número dado es o no primo.

16.- Escribir un programa que solicite un número de segundos y muestre por pantalla dicha cantidad de tiempo en horas, minutos y segundos.

-- Volver