P06 El bucle while

1.- El bucle while

`while` Repite una secuencia de acciones un numero indeterminado de veces mientras una condición se mantiene como cierta, La condición se da antes de que aparezca el bucle y se comprueba en cada repetición del bucle. En general se usa cuando no es posible determinar por adelantado cuantas repeticiones serán necesarias.

La sintaxis básica seria:

```
 while condicion:
    bloque de instrucciones
```

En primer lugar Python comprueba la condición. Si es falsa el bucle se termina y se pasa a ejecutar las sentencias escritas después del bloque. Si es cierta se ejecuta el bloque de instrucciones y se vuelve a chequear la condición. Esto se repite hasta que la condición sea falsa. Es relativamente fácil que nuestro programa acabe ejecutando un bucle infinito, ya que el intérprete no lo comprueba.

Por ejemplo, el siguiente programa presenta en pantalla los cuadrados de todos los enteros del 1 al 10. En este caso el programa se podría reescribir empleando la estructura `for ... in range(...)` :

```
 i = 1
while i <= 10:
    print(i ** 2)
    i += 1
```

En este ejemplo la variable i dentro del bucle recorre los valores del 1 al 10. Una variable así, tal que su valor cambia en cada iteración se denomina contador. Nótese que después de ejecutarse este trozo del programa el valor de la variable i está definido y es igual a 11, el valor que toma la primera vez que la condición se evaluó como falsa.

En el siguiente ejemplo se emplea `while` para determinar el número de dígitos de un entero

```
n = int(input())
length = 0
while n > 0:
    n //= 10  # esto es equivalente a  n = n // 10
    length += 1
print(length)
```

En cada iteración se corta el último dígito usando la división entera por 10, y en la variable length contamos cuantas veces se hace.

Como se vio en el tema anterior se podría hacer mucho más rápido como

`length = len(str(i))`.

2.- Métodos de control de bucles

2.1.- else

Si se escribe un bloque `else:` después de un bucle se ejecutará una sola vez cuando se finalice el bucle

```
i = 1
while i <= 10:
    print(i)
    i += 1
else:
    print('Loop ended, i =', i)
```

Puede parecer que no tiene mucho sentido emplear un bloque así, ya que ese bloque puesto sin más al final del bucle. La utilidad aparece cuando se emplea la instrucción `break`. Si durante la ejecución de un bucle el intérprete encuentra un `break`, se para la ejecución del bucle y se sale de el. En este caso el bloque `else:` no se ejecuta. Es importante recordar que `break` se emplea para finalizar un bucle.

Veamos un ejemplo con el Black Jack: El programa lee números y los suma hasta que el total sea igual o superior a 21. Si se introduce un 0 el programa para aunque la suma sea menor que 21.

Estudiemos como se comporta ante distintas entradas

Version 1. El bucle se termina normalmente después de comprobar la condición, así que se ejecuta el bloque else (para conseguir esto introducir varios números que sumen igual o menos que 21 y después un 0)

```
 total_sum = 0
a = int(input())
while a != 0:
    total_sum += a
    if total_sum >= 21:
        print('Total sum is', total_sum)
        break
    a = int(input())
else:
    print('Total sum is less than 21 and is equal to', total_sum
, '.')
```

Version 2. El bucle se termina con un `break`, así que no se ejecuta el bloque else (para conseguir esto introducir varios números que sumen mas de 21)

```
total_sum = 0
a = int(input())
while a != 0:
    total_sum += a
    if total_sum >= 21:
        print('Total sum is', total_sum)
        break
    a = int(input())
else:
    print('Total sum is less than 21 and is equal to', total_sum
, '.')
```

Los bloque else también se pueden emplear en bucles for, veamos como funciona con un ejemplo: usamos un for para leer cinco números, pero sale del bucle si encuentra un negativo.

Version 1: el bloque termina normalmente, por lo que el bloque else se ejecuta (para conseguir esto introducir cinco números no negativos)

```
 for i in range(5):
    a = int(input())
    if a < 0:
        print('Met a negative number', a)
        break
else:
    print('No negative numbers met')
```

Version 2: el bloque termina en el break, por lo que el bloque else nose ejecuta (para conseguir esto introducir un numero negativo)

```
for i in range(5):
    a = int(input())
    if a < 0:
        print('Met a negative number', a)
        break
else:
    print('No negative numbers met')
```

2.2.- continue

Otra instrucción muy útil en el control de los bucles es `continue`. Si el intérprete de Python encuentra un `continue` en mitad de una iteración la finaliza inmediatamente, y procede a la siguiente

```
for num in range(2, 10):
    if num % 2 == 0:
        print("Found an even number", num)
        continue
    print("Found a number", num)
```

Si cualquiera de estas dos instrucciones (`break` y `continue`) se emplean cuando hay varios bucles anidados su ejecución solo afecta al mas interno. Veamos esto en un ejemplo

```
for i in range(3):
    for j in range(5):
        if j > i:
            # breaks only the for on line 2
            break
        print(i, j)
```

Estas dos instrucciones (`break` y `continue`) son muy potentes, no obstante su uso dificulta mucho la lectura de los programas, por lo que solo deben emplearse si no se encuentra otra forma de implementar el comportamiento deseado.

Este es un ejemplo de mal uso de ellas

```
 n = int(input())
length = 0
while True:
    length += 1
    n //= 10
    if n == 0:
        break
print('Length is', length)
```

Que en la siguiente página se presenta de forma más simple

```
n = int(input())
length = 0
while n != 0:
    length += 1
    n //= 10
print('Length is', length)
```

3.- Asignación múltiple

En Python es posible cambiar el valor de varias variables en la misma instrucción

```
 a, b = 0, 1
```

Tiene el mismo efecto que

```
a = 0
b = 1
```

La diferencia entre las dos versiones del código es que cuando se emplea la asignación múltiple se cambia el valor de las dos variables.

La asignación múltiple es útil cuando se necesita intercambiar el valor de dos variables (recordemos los ejercicios con range). En lenguajes en que no se soporta la multiple asignación se realiza empleando una variable auxiliar

```
a = 1
b = 2
tmp = a
a = b
b = tmp
print(a, b)
# 2 1
```

Empleando la asignación múltiple se puede hacer de forma mas compacta

```
a = 1
b = 2
a, b = b, a
print(a, b)
# 2 1
```

A izquierda y derecha del igual tendremos listas de elementos separadas por comas, a los dos lados debe haber el mismo número de elementos

Ejercicios:

P1 Lista de cuadrados

Dado un entero (que se debe preguntar) N imprimir todos los cuadrados de números enteros tales que ese cuadrado es igual o menor que N.

P2 Menor divisor

Dado un entero (que se debe preguntar) N no menor que 2, imprimir el menor entero (mayor que 1) que lo divide

P3 Entrenamiento matutino

Supongamos que estas entrenando para una carrera. Dado que el primer día puedes correr x kilómetros y la carrera es de y km.
Calcular el número de días necesarios para entrenar hasta poder completar la carrera si cada día puedes correr un 10% mas que el dia anterior
Imprimir un entero representando el número de días de entrenamiento necesarios

P4 Longitud de una secuencia

Al ordenador se le debe dar una secuencia de enteros no negativos, escribiendo cada uno en una línea distinta. Determinar la longitud de la secuencia teniendo en cuenta que la secuencia termina cuando se introduzca un 0. Imprimir la longitud de la secuencia (sin contar el 0)

P5 Longitud de una secuencia

Determinar la suma de todos los elementos de una secuencia (igual a la del programa anterior)

P6 Media de una secuencia

Determinar la media de todos los elementos de una secuencia (igual a la del programa anterior)

P7 Máximo de una secuencia

Determinar el máximo de todos los elementos de una secuencia (igual a la del programa anterior)

P8 Índice del máximo de una secuencia

Determinar la posición máximo de todos los elementos de una secuencia (igual a la del programa anterior). Si varios elementos son el máximo dar la posición del primero