Capítulo 4

Bucles
Estructuras repetitivas

```
entrada='algo'
contador=0
while entrada!='':
    entrada=input('Escribe un texto: ')
    contador=contador+1
else:
    contador=contador-1
```

Los ordenadores se caracterizan por la rapidez en la realización de sus operaciones, de ahí que las estructuras repetitivas adquieran una gran importancia en todos los lenguajes de programación, sirven para poder realizar un gran número de operaciones escribiendo muy pco código, con lo cual nos ahorran muchísimo trabajo.

Las estructuras repetitivas reciben el nombre de bucles, y en Python, al igual que en otros muchos lenguajes de programación, encontramos 2 tipos de construcciones, las que podemos llevar a cabo con la palabra reservada **for** y las que podemos realizar con la palabra reservada **while**.

Estructura **for**

```
for variable in objeto-iterable:
    instruccion-1
    instruccion-2
    instruccion-3
    ...
```

Estructura **while**

```
While condicion:
    instruccion-1
else:
    instruccion-2
    ...
```

Estructura for

Un bucle for consta de una variable, un objeto iterable y un bloque de código.

El concepto nuevo en esta ocasión es el objeto iterable, se trata de un conjunto de valores que la variable irá tomando en cada una de las veces que ejecutará el bloque de código.

En Python tenemos muchos objetos iterables y además podemos construir los nuestros propios. Los más comunes son: listas, cadenas de textos, tuplas, …

La mejor forma de comprender como funciona un bucle for es haciendo algunos ejemplos:

Programa: objetoiterable1.py

```
1     for n in (1,2,3,4,5,6,7,8,9,10):
2     print('El valor de n es', n)
```

```
>>> %Run objetoiterable1.py

El valor de n es 1
El valor de n es 2
El valor de n es 3
El valor de n es 4
El valor de n es 5
El valor de n es 6
El valor de n es 7
El valor de n es 8
El valor de n es 9
El valor de n es 10

>>>
```

Lo que ha ocurrido ha sido lo siguiente:

La variable n ha tomado el valor n=1 y ha ejecutado el bloque de código que en esta ocasión es una única instrucción.

La variable n ha tomado el valor n=2 y ha vuelto a ejecutar el bloque de código.

Así sucesivamente hasta recorrer todos los valores del objeto iterable.

Veamos otro ejemplo simple de bucle for recorriendo otro iterable, en el siguiente caso el iterable será una lista de nombres.

Programa: objetoiterable2.py

```
1     nombres = ['Arturo', 'Julio', 'Sara', 'Juan', 'Roberto', 'Dani',
'Martina', 'Aurora']
2     for n in nombres:
3         print('El valor de n es',n)
4         print(n,'tiene',len(n),'caracteres\n')
```

En esta ocasión el programa genera la siguiente salida, que pasamos a explicar:

```
>>> %Run objetoiterable2.py

   El valor de n es Arturo
   Arturo tiene 6 caracteres

   El valor de n es Julio
   Julio tiene 5 caracteres

   El valor de n es Sara
   Sara tiene 4 caracteres

   El valor de n es Juan
   Juan tiene 4 caracteres

   El valor de n es Roberto
   Roberto tiene 7 caracteres

   El valor de n es Dani
   Dani tiene 4 caracteres

   El valor de n es Martina
   Martina tiene 7 caracteres

   El valor de n es Aurora
   Aurora tiene 6 caracteres

>>>
```

Como se puede observar el objeto iterable lo hemos definido en la línea 1 en donde hemos declarado una variable nombres que contiene una lista de nombres propios.

En la línea 2 definimos el bucle, donde la variable n recorre la lista de nombres.

En la línea 3 mostramos el valor que va tomando la variable n en cada una de las iteraciones del bucle.

En la línea 4, mostramos el valor de n, la longitud de la cadena de caracteres n, y además añadimos un salto de línea para que al ejecutar el programa visualmente se observe con más claridad cada una de las iteraciones del bucle for.

Como la lista nombres tiene 8 elementos, el programa ha ejecutado el bloque de instrucciones (líneas 3 y 4) 8 veces, una por cada valor de las lista de nombres.

Programa: objetoiterable3.py

```
1     print('Este programa deletrea una palabra,')
2     print('escribe cada uno de sus caracteres y el valor unicode del
carácter')
3     palabra=input('Escribe una palabra: ')
4     for c in palabra:
5         print(c,'valor unicode:',ord(c),'\n')
```

Este programa genera la salida siguiente:

```
>>> %Run objetoiterable3.py

   Este programa deletrea una palabra,
   escribe cada uno de sus caracteres y el valor unicode del carácter
   Escribe una palabra: Informática
   I valor unicode 73

   n valor unicode 110

   f valor unicode 102

   o valor unicode 111

   r valor unicode 114

   m valor unicode 109

   á valor unicode 225

   t valor unicode 116

   i valor unicode 105

   c valor unicode 99

   a valor unicode 97

>>>
```

En el bucle for la **variable es c** y va tomando cada uno de los caracteres que hayamos introducido a través de la instrucción input y almacenado en la variable **palabra**.

Como se puede observar en ejemplo, una cadena de texto también es un objeto iterable, siendo cada uno de los caracteres los elementos que se iteran en cada uno de los ciclos.

En la línea 5 hemos utilizado la función ord() que nos devuelve el valor unicode del carácter que se le pasa como parámetro.

Vamos a finalizar estos ejemplos de bucles for mostrando el uso de la función range(), la cual es muy común encontrarla en la línea de declaración del bucle.

La función **range** tiene la siguiente sintaxis:

range(valorinicial, valorfinal, incremento)

Esta función nos devuelve un objeto iterable que incluye valorinicial como primer elemento y llega hasta una unidad menos de valorfinal.

Incremento es un parámetro opcional que indica el incremento de la variable del bucle en cada iteración. Si se omite el incremento, por defecto se establece a 1.

Ejemplos:

range(1,10)	Obtendríamos como objeto iterable: 1, 2, 3, 4, 5, 6, 7, 8, 9
range(1,20,3)	Obtendríamos como iterable: 1, 4, 7, 10, 13, 16, 19
range(10, 1, -1)	Obtendríamos como iterable: 10, 9, 8, 7, 6, 5, 4, 3, 2

Programa: objetoiterable4.py

Vamos a realizar un programa que calcule las tablas de multiplicar. El programa preguntará qué tabla quieres calcular.

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Preguntará la tabla que quieres que se calcule.

3º Mostrará la tabla de multiplicar correspondiente.

```
1     print('Este programa calcula las tablas de multiplicar')
2     numero=input('Qué tabla de multiplicar quieres calcular: ')
3     numero=eval(numero)
4     for n in range(1,11):
5         print(n ,'x', numero, '=', n*numero)
```

La salida que genera el programa es:

```
>>> %Run objetoiterable4.py

  Este programa calcula las tablas de multiplicar
  Qué tabla de multiplicar quieres calcular: 8
  1 x 8 = 8
  2 x 8 = 16
  3 x 8 = 24
  4 x 8 = 32
  5 x 8 = 40
  6 x 8 = 48
  7 x 8 = 56
  8 x 8 = 64
  9 x 8 = 72
  10 x 8 = 80

>>>
```

Recuerda evaluar numero como un valor numérico, de lo contrario será interpretado como una cadena de caracteres.

Observa que en la definición del bucle hemos indicado range(1,11) para que estén incluidos desde el 1 hasta el 10 en el conjunto de valores iterables.

En la línea 5 visualizamos el signo 'x', aunque la operación se multiplicar se calcula con el operador multiplicación (*).

Estructura while

Un bucle while ejecuta un bloque de instrucciones hasta que la condición deja de ser True.

Los bucles while, al igual que los bucles for, son muy comunes y versátiles.

La cláusula **else** y el bloque de instrucciones contenidos en él, **es opcional** y se ejecutará cuando la condición deje de ser válida.

Programa: buclewhile1.py

Vamos a realizar un programa que permita introducir números de una cifra. Cuando el número escrito sea mayor o igual que 10, será equivalente a terminar.

> 1º El programa mostrará un mensaje indicando su funcionalidad.
>
> 2º Pedirá que se introduzca un número.
>
> 3º Debe estar pidiendo números **mientras** los números sean menores que 10.

```
1     print('Escribe números de una sola cifra')
2     print('Más de una cifra, será equivalente a terminar')
3     numero=0
4     while numero<10:
5         numero=input('Escribe un número: ')
6         numero=eval(numero)
```

Al ejecutar el programa podemos obtener una salida similar a esta:

```
>>> %Run buclewhile1.py

   Escribe números de una sola cifra
   Más de una cifra, será equivalente a terminar
   Escribe un número: 3
   Escribe un número: 5
   Escribe un número: 4
   Escribe un número: 3
   Escribe un número: 10

>>>
```

La construcción del bucle while incluye la condición numero<10, es decir, **el bucle estará repitiendo el bloque de instrucciones mientras que el valor de la variable número sea menor que 10**.

La línea 3 es necesaria para que la primera vez que se ejecute el bucle, número tenga algún valor y pueda compararlo con el número 10.

El mecanismo de **uso habitual de los bucles while**, siempre es el siguiente:

- En la condición del bucle while entran en juego una o más variables que han sido declaradas previamente.
- Dentro del bloque de instrucciones, algunas de las variables que forman parte de la condición son modificadas según el objetivo del programa.
- El bloque de instrucciones se repite mientras que las condición resulte verdadera.

Vamos a ver un nuevo ejemplo de bucle while en el que usaremos una técnica habitual en programación que consiste en:

1º Definir una variable de control del bucle con valor True. Esta variable será la condición del bucle.

2º Dentro del bucle realizaremos alguna comprobación con una estructura condicional que servirá para modificar el valor de la variable de control que hemos definido en el paso anterior.

3º El bloque de instrucciones del bucle se estará repitiendo mientras no cambie la variable de control.

Vamos a ver esta técnica con un sencillo ejemplo. Haremos un programa que estará pidiendo números pares indefinidamente hasta que se introduzca un número impar. El programa finalizará cuando se introduzca el primer número impar.

Programa: buclewhile2.py

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Definiremos una variable, seguir=True que será la que controlará la condición del bucle.

3º Pedirá que se introduzcan números pares.

4º Cuando se introduzca un número impar se cambiará el valor de seguir a False.

```
1     print('Escribe números pares')
2     print('Un número impar servirá para terminar')
3     seguir=True
4     while seguir:
5         numero=input('Escribe un número: ')
6         numero=eval(numero)
7         if numero%2==1:
8             seguir=False
```

Una posible ejecución de este programa sería:

```
>>> %Run buclewhile2.py

   Escribe números pares
   Un número impar servirá para terminar
   Escribe un número: 4
   Escribe un número: 6
   Escribe un número: 12
   Escribe un número: 334
   Escribe un número: 7

>>>
```

Bucles infinitos

Se llaman bucles infinitos a bucles que nunca finalizan y por tanto se están ejecutando indefinidamente. Se debe tener especial cuidado porque algunos bucles infinitos podrían agotar los recursos del equipo.

Cuando se está aprendiendo a programar, es habitual cometer errores de lógica en la elaboración de bucles while y generar bucles infinitos.

Técnicas comunes de programación: contadores y acumuladores

Dentro de un bucle es muy habitual encontrarnos una o varias instrucciones como las siguientes:

Contador	Acumulador
n = n + 1	**suma = suma + n**

Veamos algunos ejemplos donde podemos encontrar estas instrucciones:

Vamos a hacer un programa que nos pida introducir números hasta que una entrada se deje vacía. El programa debe ir contando cuántos números se introducen y la suma de esos números.

Para ello vamos a usar varias variables que nos permitan el control y guardar la información que en este caso nos interesa.

Programa: contador-acumulador1.py

1º El programa mostrará un mensaje indicando su funcionalidad.

2º Definiremos 3 variables, **seguir**=True que será la que controlará la condición del bucle, **n** que será el contador de las veces que se ejecuta el bucle y **suma** que será donde acumularemos la suma de los números que se introducen.

3º Pedirá que se introduzcan números.

4º Cuando se pulse Intro sin escribir ningún número, el programa finalizará (seguir=False).

5º Si se introduce un número, se aumenta en una unidad la variable **n** (contador) y el valor del número introducido se suma al valor que tenga en ese momento la variable **suma**, es decir, se acumula en la variable **suma**.

```
1     print('Programa que pedirá números hasta dejar una entrada vacía')
2     print('El programa cuenta cuántos números se han introducido')
3     print('y la suma de todos esos números introducidos')
4     print('(Entrada vacía = Terminar)')
5     n=0
6     suma=0
7     seguir=True
8     while seguir:
9         numero=input('Escribe un número: ')
10        if numero=='':
11            seguir=False
12        else:
13            numero=eval(numero)
14            n=n+1
15            suma=suma+numero
16
17    print('Se han introducido',n,'números')
18    print('Los números suman',suma)
```

Con este programa podríamos obtener una salida similar a la siguiente imagen:

```
>>> %Run contador-acumulador1.py

  Programa que pedirá números hasta dejar una entrada vacía
  El programa cuenta cuántos números se han introducido
  y la suma de todos esos números introducidos
  (Entrada vacía = Terminar)
  Escribe un número: 4
  Escribe un número: 15
  Escribe un número: 31
  Escribe un número: 20
  Escribe un número: 8
  Escribe un número:
  Se han introducido 5 números
  Los números suman 78

>>>
```

Ejercicios

Ejercicio 1:

Realizar un programa que muestre **sólo los números pares** comprendidos entre 1 y 1000.

Ejercicio 2:

Realizar un programa que muestre **sólo los números impares** comprendidos entre 1 y 1000.

Ejercicio 3:

Realizar un programa que muestre todos los divisores de un número (se entiende que el número será entero positivo).

Ejercicio 4:

Realizar una programa que solicite una palabra y sólo muestre las vocales de dicha palabra.

Ejercicio 5:

Realizar una programa que solicite una palabra y sólo muestre las consonantes de dicha palabra.

Ejercicio 6:

Realizar un programa que solicite entradas de texto hasta que se deje una de las entradas de texto vacía. Las palabras se irán incluyendo en una lista.

Ejercicio 7:

Realizar un programa que solicite un número y muestre todos los divisores de ese número y además cuente el número de divisores de dicho número.

Nota: Aplicar el concepto de contador, contador=contador+1 cada vez que se encuentre un divisor.

Ejercicio 8:

Realizar un programa que solicite una palabra, genere una lista a partir de dicha palabra y muestre la lista.

```
Programa que solicta una palabra
y la convierte en una lista
Escribe una palabra: Instituto
['I', 'n', 's', 't', 'i', 't', 'u', 't', 'o']
```

Ejercicio 9:

Realizar un programa que solicite una palabra, genere una lista a partir de dicha palabra y muestre la lista en orden inverso.

```
Programa que solicta una palabra
y la convierte en una lista en orden inverso
Escribe una palabra: Instituto
['o', 't', 'u', 't', 'i', 't', 's', 'n', 'I']
```

Ejercicio 10:

Realizar un programa que solicite una palabra y deletree dicha palabra en orden inverso. La salida del programa podría ser algo así.

```
Programa que solicta una palabra
y deletrea dicha palabra en orden inverso
Escribe una palabra: Insituto
o
t
u
t
i
s
n
I
```

Ejercicio 11:

Realizar un programa que solicite entradas de números hasta que se deje una de las entradas se deje vacía. El programa debe contar cuántos números se han introducido, calcular la suma de los números y la media.