Bucles

Se utilizan para repetir una serie de instrucciones un número determinado de veces o hasta que se cumpla una condición.

En esta clase, veremos los siguientes tipos de loops en Python:

- For loop
- While loop
- Nested loop
- Continue statement
- Break statement

For Loop

El for loop se utiliza para repetir una serie de instrucciones un número determinado de veces. El número de veces que se repetirá el loop se especifica en una variable de control.

```
for <variable de control> in <iterable>:
    # Instrucciones a repetir
```

Estos loops funcionan muy bien para recorrer arreglos como las listas usando sus indices. O usando los loops "inteligentes" para recorrer los elemntos directamente.

```
lista = [1,2,3,4,5,6,7,8,9,10]
for i in range(0,10):
    print(elemento:,lista[i])
```

Mientras que los "inteligentes" se ven asi

```
nombres = ["leonardo","donatelo","miguel angel","rafael"]
for nombre in nombres:
    print(nombre)
```

Estos loops inteligentes recorren un objeto iterable de manera nativa. Algunos otros lenguajes los tienen pero suelen nombrarlos como un foreach.

While loop

El while loop se utiliza para repetir una serie de instrucciones hasta que se cumpla una condición.

```
while <condición>:
    # Instrucciones a repetir
```

Por ejemplo tenemos este codigo:

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

NOTA Es importante notar que esta condicion debe cumplirse en algun momento, sino el procesador seguira corriendo este proceso eternamente. Obviamente esto no es bueno para nuestra computadora.

Nested loop

Este es un loop que se encuentra dentro de otro loop.

```
for <variable de control 1> in <iterable 1>:
    for <variable de control 2> in <iterable 2>:
        # Instrucciones a repetir
```

Un ejemplo de esto seria:

```
for i in range(1, 11):
    for j in range(1, 11):
        print(i, "x", j, "=", i * j)
```

Sentencia Break

Esta sentencia se utiliza para salir de un loop, (ya sea while como for). Es importante notar que en este caso vamos a usar dos estructuras juntas, un bucle y un condicional.

```
nombres = ["leonardo","donatelo","miguel angel","rafael"]
for nombre in nombres:
    if(nombre in "donatelo"):
        break
    print(nombre)
```

Continue statement

El continue statement se utiliza para omitir la siguiente iteración de un loop.

```
nombres = ["leonardo","donatelo","miguel angel","rafael"]
for nombre in nombres:
    if(nombre in "leonardo"):
        continue
    print(nombre)
```

Comprehension

```
[expresion(i) for i in list if condición]
```

Son una forma de crear listas/diccionarios de una manera elegante simplificando el código al máximo. Uniendo tanto un loop con un if. Supongamos que tenemos este codigo

```
numeros = [1, 2, 34, 86, 4, 5, 99, 890, 45]

pares = []
for num in numeros:
    if num % 2 == 0:
        pares.append(num)
print(pares)
```

El código anterior crea una lista de números pares a partir de una lista de números. Para este tipo de situaciones es ideal el uso de las list comprehensions. Modificando el codigo podemos hacer:

```
pares = [num for num in numeros if num % 2 == 0]
print(pares)
```

Nota: Este tipo de codigo no solo es mas "elegante" sino que ademas es un poco mas eficiente. De todos modos para el proposito de este curso es mas que suficiente que usen solo los loops normales. Pero es bueno que sepan que existe

En realidad las "comprehensions" se pueden hacer tambien dentro de un diccionario. Pero es importante notar que las listas son las mas comunes que usamos para esto.