Estructuras Repetitivas

Objetivo: Estudio y manejo de las estructuras repetitivas (Ciclos)

Ciclos

Estructuras Repetitivas o ciclos: permiten repetir automáticamente un grupo de instrucciones, ya sea un número determinado de veces o mientras que una condición particular se cumpla. Son aquellas que permiten que una operación o conjunto de ellas se repitan muchas veces. En muchos problemas se requiere que algunos cálculos o grupos de instrucciones se repitan una y otra vez, a continuación procederemos a examinar las diferentes estructuras que se utilizan para construir secciones de código repetitivas, como son: **for** y **while**

Definiciones Básicas:

Iteración: es la repetición de una secuencia de instrucciones.

Variable contadora: variable que se modifica así misma para llevar el computo de veces que ocurre un suceso, evento o situación, este tipo de variables su incremento es un valor constante: **A = A + valor_constante** o **A += valor_constante**.

Variable acumuladora: variable que se modifica así misma para acumular un valor variable: **B = B + valor_variable** o **B += valor_variable**.

Variable Bandera: variable que cambia su valor para indicar que algún suceso o evento ha tenido lugar, cambia de un valor a otro como de cero a uno o de falso a cierto.

Porcentaje: número de elementos de una lista que cumplen con una condición entre la cantidad de elementos a los cuales se les comprobó si cumplen o no con la condición por cien.

Promedio: sumatoria de un valor de una lista entre número de elementos sumados.

Ciclos For….in

Este tipo de estructura repetitiva se caracteriza porque las instrucciones del cuerpo del ciclo se ejecutan *un número determinado de veces* y de modo automático se controla el número de iteraciones o pasos a través del bloque de instrucciones del ciclo.

Sintaxis:

```
for variable_controladora_del_ciclo in range(argumentos):
        bloque_de_código
```

range genera una **lista inmutable de números enteros en sucesión aritmética**.

- Inmutable significa que no se pueden modificar.
- Una sucesión aritmética es una sucesión en la que la diferencia entre dos términos consecutivos es siempre la misma.

Para definir un **range** usaremos una de las siguientes alternativas como argumento:

- *range(*n,m,s*)* cumple: rango >= n y rango < m en incrementos de s.
- *range(*n,m*)* cumple: rango >= n y rango < m en incrementos de 1.
- *range(*m*)* cumple: rango >= 0 y rango < m en incrementos de 1.

Estructura de repetición : FOR

Este tipo de estructura permite implementar la repetición de un cierto conjunto de instrucciones (o bloque de código) un numero predeterminado de veces.

Para ello se utiliza una variable de control del bucle, llamada también índice, que va recorriendo un conjunto prefijado de valores en un orden determinado. Para cada valor del índice en dicho conjunto, se ejecuta una vez el mismo conjunto de instrucciones.

Condiciones de uso:

- **El valor de la *variable de control* puede ser utilizado o no dentro del conjunto de instrucciones que forman parte del cuerpo del FOR, pero no debe ser modificado.**
- **Las estructuras FOR pueden anidarse, es decir, incluir una dentro de la otra, con la restricción (de sentido común) de que la interior tiene que estar completamente contenida en uno de los bloques de instrucciones de la otra.**

Ejemplos:

```
for x in range(2,7,3):
    print(x)
```

```
2
5
```

En este caso el valor de la variable controladora del ciclo es: **2 <= X < 7** con un salto de **3** de tal manera que cuando el ciclo se ejecuta se imprime el primer valor **2**,a este se le suma 3 el siguiente valor a imprimir será **5**, a este se le suma 3 tememos **8** no se imprime y el ciclo termina.

```
for x in range(2,7):
    print(x)
```

```
2
3
4
5
6
```

En este caso el valor de la variable controladora del ciclo es: **2 <= X < 7** con un salto de 1 por defecto, de tal manera que cuando el ciclo se ejecuta se imprime el primer valor **2**,a este se le suma 1 el siguiente valor a imprimir será **3**, a este se le suma 1 tememos **4** y así sucesivamente hasta **6**.

```
for x in range(7):
    print(x)
```

```
0
1
2
3
4
5
6
```

En este caso el valor de la variable controladora del ciclo es: **0 <= X < 7** con un salto de 1 por defecto, de tal manera que cuando el ciclo se ejecuta se imprime el primer valor **0**,a este se le suma 1 el siguiente valor a imprimir será **1**, a este se le suma 1 tememos **2** y así sucesivamente hasta **6**.

```
for x in range(7,2,-1):
    print(x)
```

```
7
6
5
4
3
```

Ciclo decreciente:

En este caso el valor de la variable controladora del ciclo es: **7 >= X < 2** con un salto de -1, de tal manera que cuando el ciclo se ejecuta se imprime el primer valor **7**,a este se le resta 1 el siguiente valor a imprimir será **6**, a este se le resta 1 tememos **5** y así sucesivamente hasta **3**.

Ciclos While

El ciclo **while** permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras el resultado de evaluar la condición sea **True**).
La sintaxis del bucle **while** es la siguiente:

while condición:

cuerpo del ciclo

Cuando el flujo del programa alcanza la instrucción **while**, Python evalúa la condición y, si es cierta (True), ejecuta el grupo de instrucciones que forman parte del cuerpo del ciclo. Una vez ejecutado el cuerpo del ciclo, se repite el proceso (se evalúa de nuevo la condición y, si es cierta, se ejecuta de nuevo el cuerpo del ciclo) una y otra vez mientras la condición sea cierta. Únicamente cuando la condición sea falsa, el cuerpo del ciclo no se ejecutará y continuará la ejecución del resto del programa.
La variable o las variables que aparezcan en la condición se suelen llamar variables de control. Las variables de control deben definirse antes del ciclo **while** y modificarse en el cuerpo del ciclo **while**.

While controlados por contador.

Variable de control: cualquier variable que aparezca en la condición del while debe haber recibido un valor inicial antes de entrar en el ciclo, porque la variable ya debe tener un valor la primera vez que se pruebe la condición. Ejemplos:

```
x=2
while x <= 7:
    print(x)
    x += 3
```

```
2
5
```

```
x=2
while x <= 7:
    print(x)
    x +=1
```

```
2
3
4
5
6
7
```

```
x = 0
while x<=7:
    print(x)
    x +=1
```

```
0
1
2
3
4
5
6
7
```

While controlados con centinela o bandera

***Centinela*: un valor que, al ser leído, le indica al programa que el usuario desea salir del ciclo. El esquema del ciclo con centinela es el siguiente:**

Pedir datos.
Mientras el dato pedido no coincida con el centinela:
Realizar cálculos.
Pedir datos.

Ejemplo: se desea leer una lista de números enteros positivos que se desea sumar hasta que se introduzca el valor de -1.

```
1  suma=0
2  valor=0
3  valor=int(input("De un numero entero, -1 para terminar: "))
4  while(valor!=-1):
5      suma+=valor
6      valor = int(input("De un numero entero, -1 para terminar: "))
7  print("Suma = ",suma)
```

Pedir datos

verificamos dato no coincide com centinela

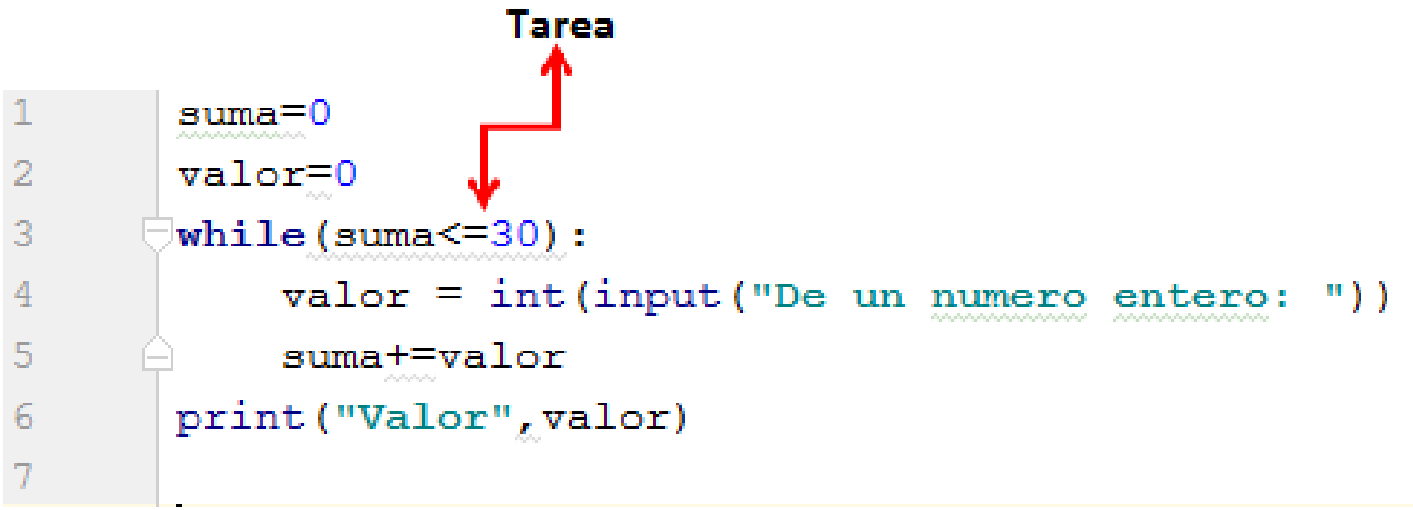
Pedir datos

Ejecución →

```
De un numero entero, -1 para terminar: 12
De un numero entero, -1 para terminar: 2
De un numero entero, -1 para terminar: 3
De un numero entero, -1 para terminar: -1
Suma =  17
```

While controlados por tarea.

El while en su condición especifica que el cuerpo del ciclo debe continuar ejecutándose mientras la tarea o se haya completado. Ejemplo: Escribir un programa para encontrar el primer números que hace que la suma sea mayor a 30.



Ejecución

```
De un numero entero: 2
De un numero entero: 8
De un numero entero: 15
De un numero entero: 3
De un numero entero: 6
Valor 6
```

Ciclos While infinitos

Son aquellos en los cuales la condición nunca se cumple. Ejemplo:

```
1   valor=0
2   while(valor!= 15):
3       valor += 2
4       print(valor)
5   print("Termino el programa")
6
```

Este programa nunca se detendrá ya el contenido de la variable valor siempre tomara valores pares por lo tanto diferentes de 15.

Funciones del modulo random

Para usar las funciones definidas en este modulo debemos importar random

import random

o

from random import randint, uniform, randrange

random.randint(a, b):

La función randint genera de forma aleatoria un número entero que estará en el rango a, b inclusive. Donde a es el límite inferior y b el límite superior del rango definido.

```
from random import randint

for i in range(1,10):
    print(" ", randint(1,4))
```

Siendo una posible salida:

```
 2
 2
 1
 2
 2
 3
 1
 2
 1
```

La función randint(1,4) genera aleatoriamente los valores enteros 1, 2, 3, 4. Los valores se pueden repetir en la medida que se use la función.

random.uniform(a,b): **genera un numero aleatorio real en el rango a y b inclusive.**
Donde a es el límite inferior y b el límite superior del rango definido.

```
from random import uniform
for i in range(1,10):
    print(" ", uniform(1,4))
```

Siendo una posible la salida:

```
3.0112539563420158
3.3218479426930285
2.697574780439341
1.5132430897812696
3.6542740565042164
3.533746775337613
1.8448478628582654
1.1369452062996301
3.4714464844658606
```

Random.randrange(a,b,s): **genera un numero aleatorio que debe estar en un rango establecido, con la variante, que podemos establecer un salto dentro del rango de valores. En donde a es el límite inferior, b el límite superior del rango, y s es salto en el rango.**

```
from ramdon import randrange
for i in range(1,10):
    print(randrange(0,10,2),"  ",end = "")
```

Siendo una posible salida:

```
0   0   8   4   8   0   8   4   2
```

El valor generado aleatoriamente es un entero, y dicho entero, será mayor o igual al límite inferior y menor que el límite superior. En el caso de que el límite inferior sea igual cero los valores generados serán múltiplos del salto, si el límite inferior es diferente de cero la serie aleatoria partirá desde este valor y a partir de ahí los siguientes valores se generaran de acuerdo al salto ejemplos:

```
from ramdon import randrange
for x in range(1,10):
    print("  ",randrange(1,10,2), end = "")
```

Siendo una posible salida:

```
  9   5   3   1   3   9   9   3   7
```

```
for x in range(1,10):
    print("  ",randrange(2,20,3), end = "")
```

Siendo una posible salida:

```
 17   14   17   11   11   2   14   17   2
```