Taller de Programación Web

Estructuras de Control Repetitivas

- contador, acumulador y bandera
-while
-for
-range
-breack
-continue

Fácil

Subsecretaría de Empleo
Chaco Gobierno de todos

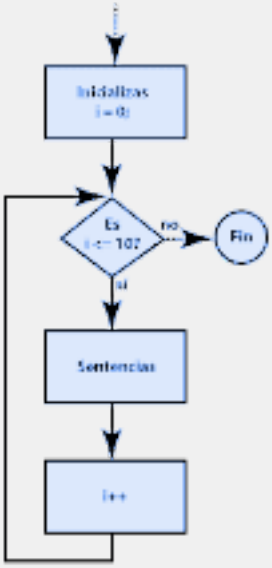
Ministerio de Producción, Industria y Empleo
Chaco Gobierno de todos

CHACO
Gobierno de todos

INFORMATORIO

INTRODUCCIÓN

Estructuras Repetitivas



Durante el proceso de creación de programas, es muy común, encontrarse con que una operación o conjunto de operaciones **deben repetirse muchas veces**.

Para ello es importante conocer las estructuras que permiten ***repetir una o varias acciones, un número determinado de veces***.

Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan estructuras repetitivas, iterativas o cíclicas.

Cada conjunto de instrucciones a ejecutar se denomina BUCLE. Y cada repetición del bucle se llama ITERACIÓN.

Todo bucle tiene que llevar asociada una condición, que es la que va a determinar **cuándo se repite el bucle** y **cuándo deja de repetirse**.

Antes de continuar veremos tipos de variables especiales que podríamos requerir para diseñar nuestros programas

Contadores, Acumuladores y Banderas

En muchos programas se necesitan variables que **cuenten cuántas veces ha ocurrido algo** (contadores), que indiquen si simplemente **ha ocurrido un evento** (banderas) o que **acumulen valores** (acumuladores).

Las situaciones pueden ser muy diversas, por lo que en este apartado simplemente se ofrecen unos ejemplos para entiendas cómo utilizar estos recursos:

>Contador<

Los procesos repetitivos requieren contar los sucesos y acciones internas, una forma de hacerlo es mediante un contador.

Un contador es una variable cuyo valor se incrementa o decrementa en una cantidad constante en cada repetición.

>Acumulador<

Un **acumulador** o totalizador es una variable cuya función es almacenar cantidades resultantes de operaciones sucesivas.

Realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante.

>Banderas<

Una **bandera**, es una variable que puede tomar uno de dos valores (verdadero o falso) a lo largo de la ejecución del programa y permite comunicar información de una parte a otra del mismo.

Sentencia while (mientras)

Un **bucle while** permite repetir la ejecución de un grupo de instrucciones mientras se cumpla una condición (es decir, mientras la condición tenga el valor True).

Queda en las manos del programador decidir el momento en que la **condición cambie a False** para hacer que el **ciclo While finalice**.

Características:

- No se conoce la cantidad de veces a iterar o repetir el conjunto de acciones
- El final del bucle está controlado con una condición.
- El conjunto de acciones se ejecutan mientras la evaluación de la condición devuelva un resultado verdadero
- El ciclo se puede ejecutar 0 o más veces.

Código

```
Dado un número entero N calcular la suma de todos los números entre 1 y N.

cont = 0
suma = 0
N = int(input('Ingrese tope màximo: '))
while cont <= N:
   suma = suma + cont
   cont = cont + 1
print('La suma total es: ', suma)
```

Resultado	
Ingresa N = 3 cont vale 0 cont vale 1 cont vale 2 cont vale 3	suma vale 0 suma vale 0 suma vale 1 suma vale 3 suma vale 6 La suma total es 6

Otra ventaja del bucle while es que el **número de iteraciones no está definida** antes de empezar el bucle, por ejemplo porque los datos los proporciona el usuario.

Por ejemplo, *el siguiente ejemplo pide un número positivo al usuario una y otra vez hasta que el usuario lo haga correctamente*:

Código

```
numero = int(input("Escriba un número positivo: "))
while numero < 0:
    print("¡Ha escrito un número negativo! Inténtelo de nuevo")
    numero = int(input("Escriba un número positivo: "))
print("Gracias por su colaboración")
```

Resultado

```
Escriba un número positivo: -4
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: -8
¡Ha escrito un número negativo! Inténtelo de nuevo
Escriba un número positivo: 9
Gracias por su colaboración
```

Uso de else en while

Se encadena al While para ejecutar un bloque de código una vez la condición ya no devuelve True (normalmente al final):

Código

```
c = 0
while c <= 5:
    c+=1
    print("c vale", c)
else:
    print("Se ha completado toda la iteración y c vale", c)
```

Resultado

```
c vale 1
c vale 2
c vale 3
c vale 4
c vale 5
c vale 6
Se ha completado toda la iteración y c vale 6
```

Sentencia for (para)

- Ahora **SI se conoce la cantidad de veces a iterar** o **repetir** el conjunto de acciones
- El **final de bucle** está controlado un contador (indica cantidad de iteraciones).
- La **variable contador** que maneja el bucle se incrementa automáticamente de acuerdo al incremento indicado.
- No necesita **inicialización**

La sintaxis de un bucle for es la siguiente:

Código

```
for variable in elemento iterable (lista, cadena, range, etc.):
    cuerpo del bucle
```

El cuerpo del bucle se ejecuta tantas veces como elementos tenga la estructura iterable(elementos de una lista o de un range(), caracteres de una cadena, etc.).

Utilizando tipos range()

Una las ventajas de utilizar tipos **range()** es que el argumento del tipo **range()** controla el número de veces que se ejecuta el bucle. Sirve para generar una lista de números que podemos recorrer fácilmente, pero no ocupa memoria porque se interpreta sobre la marcha:

Código

```
print("Comienzo")
for i in range(3):
    print("Hola ", end="")
print("Final")
```

Resultado

```
Comienzo
```

```
Hola Hola Hola
Final
```

Control de Bucles

Instrucción **break**

Se puede usar en bucles **for** y **while** y simplemente t**ermina el bucle actual** y sigue con la ejecución de la próxima instrucción, por ejemplo:

a) Controlar fin de bucle

```
while True:
op = input('Ingrese cualquier palabra, termina con FIN--> ')
if op == 'FIN':
break
else:
print(op)
print('Terminó la ejecución con FIN')
```

b) La sentencia break es usada también para terminar un ciclo aun cuando la evaluación de la condición no devuelva False,

```
# Primer ejemplo

for letra in "Python":

      if letra == "h":

      break

      print("Letra actual :", letra)
```

Si se fijan en el primer ejemplo al llegar a la letra "h" simplemente se termina (rompe) el ciclo (bucle) y se sigue con el segundo ejemplo. En el segundo ejemplo la variable va disminuyendo su valor hasta que llega a 5, en donde se termina (rompe) el ciclo (bucle)


```
# Segundo ejemplo

var = 10

while var > 0:

    var = var -1

    if var == 5:

       break

    print("Valor actual de la variable :", var)
```

Sentencia continue

Al aparecer un **continue** en Python, este regresa al comienzo del bucle, ignorando todos los estamentos que quedan en la iteración actual del bucle e inicia la siguiente iteración. Queda más claro con un ejemplo:

```
# Primer ejemplo
for letra in "Python":
     if letra == "h":
          continue
     print("Letra actual :", letra)

# Segundo ejemplo
var = 10
while var > 0:
     var = var -1
     if var == 5:
          continue
     print("Valor actual de la
     variable :", var)
```

En el primer ejemplo al llegar a la letra "h" simplemente termina esa iteración (ignorando al print que sigue en la linea 5) y continua con la siguientes iteraciones (letras o y n) hasta que se termina el ciclo (bucle).

En el segundo ejemplo la variable va disminuyendo su valor hasta que llega a 5, en donde se termina esa iteración del ciclo (bucle) y se continúa con las iteraciones que siguen hasta que se termina el bucle.