Fundamentos de Programación

TEMA: ESTRUCTURA DE ITERACIÓN 2
Semana 08

OBJETIVO DEL LABORATORIO

Utiliza los procesos repetitivos con el código FOR (Para – Hasta).

MARCO TEÓRICO

Completaremos los 3 tipos de algoritmos, viendo los de tipo repetitivo; donde su construcción incluye los secuenciales y condicionales; donde se repetirá una acción "n" veces en base a una situación en particular.

Como fuente de información, se tiene la misma plataforma del Netacad. Tanto para realizar los ejemplos propuestos en la interfaz integrada, así como para repasar lo visto en clase.

RECURSOS

a. Hardware
- Pc Pentium IV a superior
- Conexión de red

b. Software
- Sistema Operativo Windows XP a superior
- Navegador Chrome o Firefox
- Edube Sandbox de Python desde Netacad

PROCEDIMIENTO

1. EL BUCLE for

   En general, un bucle es una estructura de control que repite un bloque de instrucciones. Un bucle for es un bucle que repite el bloque de instrucciones un número predeterminado de veces. El bloque de instrucciones que se repite se suele llamar cuerpo del bucle y cada repetición se suele llamar iteración.

   La sintaxis de un bucle for es la siguiente:

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

   No es necesario definir la variable de control antes del bucle, aunque se puede utilizar como variable de control una variable ya definida en el programa.

   - El cuerpo del bucle se ejecuta tantas veces como elementos tenga el elemento recorrible (elementos de una lista o de un range(), caracteres de una cadena, etc.). Por ejemplo:

```
Ejemplo de bucle 1

print("Comienzo")
for i in [0, 1, 2]:
    print("Hola ", end="")
print()
```

```
print("Final")
```

RESULTADO:

```
Comienzo
Hola Hola Hola
Final
```

- En el ejemplo anterior, los valores que toma la variable no son importantes, lo que importa es que la lista tiene tres elementos y por tanto el bucle se ejecuta tres veces. El siguiente programa produciría el mismo resultado que el anterior:

Ejemplo de bucle 2

```
print("Comienzo")
for i in [1, 1, 1]:
    print("Hola ", end="")
print()
print("Final")
```

RESULTADO:

```
Comienzo
Hola Hola Hola
Final
```

- Si la lista está vacía, el bucle no se ejecuta ninguna vez. Por ejemplo:

Ejemplo de bucle 3

```
print("Comienzo")
for i in []:
    print("Hola ", end="")
print()
print("Final")
```

RESULTADO:

```
Comienzo


Final
```

- En los ejemplos anteriores, la variable de control "i" no se utilizaba en el bloque de instrucciones, pero en muchos casos sí que se utiliza. Cuando se utiliza, hay que tener en cuenta que la variable de control va tomando los valores del elemento recorrible. Por ejemplo:

Ejemplo de bucle 4

```
print("Comienzo")
for i in [3, 4, 5]:
    print("Hola. Ahora i vale ", i, " y su cuadrado", i ** 2)
print("Final")
```

RESULTADO:

```
Comienzo
Hola. Ahora i vale 3 y su cuadrado 9
Hola. Ahora i vale 4 y su cuadrado 16
Hola. Ahora i vale 5 y su cuadrado 25
Final
```

- La lista puede contener cualquier tipo de elementos, no sólo números. El bucle se repetirá siempre tantas veces como elementos tenga la lista y la variable irá tomando los valores de uno en uno. Por ejemplo:

Ejemplo de bucle 5

```
print("Comienzo")
for i in ["Alba", "Benito", 27]:
    print("Hola. Ahora i vale ", i)
print("Final")
```

RESULTADO:

```
Comienzo
Hola. Ahora i vale Alba
Hola. Ahora i vale Benito
Hola. Ahora i vale 27
Final
```

- La costumbre más extendida es utilizar la letra i como nombre de la variable de control, pero se puede utilizar cualquier otro nombre válido. Por ejemplo:

Ejemplo de bucle 6

```
print("Comienzo")
for numero in [0, 1, 2, 3]:
    print(numero, " * ", numero, " = ", numero ** 2)
print("Final")
```

RESULTADO:

```
Comienzo
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
Final
```

- La variable de control puede ser una variable empleada antes del bucle. El valor que tuviera la variable no afecta a la ejecución del bucle, pero cuando termina el bucle, la variable de control conserva el último valor asignado:

```
i = 10
print("El bucle no ha comenzado. Ahora i vale", i)

for i in [0, 1, 2, 3, 4]:
    print(i, "*", i, "=", i ** 2)

print("El bucle ha terminado. Ahora i vale", i)
```

RESULTADO:

```
El bucle no ha comenzado. Ahora i vale 10
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4
3 * 3 = 9
4 * 4 = 16
El bucle ha terminado. Ahora i vale 4
```

- Cuando se escriben dos o más bucles seguidos, la costumbre es utilizar el mismo nombre de variable puesto que cada bucle establece los valores de la variable sin importar los valores anteriores:

```
for i in [0, 1, 2]:
    print(i, "*", i, "=", i ** 2)

print()

for i in [0, 1, 2, 3]:
    print(i, "*", i, "*", i, "=", i ** 3)
```

RESULTADO:

```
0 * 0 = 0
1 * 1 = 1
2 * 2 = 4


0 * 0 * 0 = 0
1 * 1 * 1 = 1
2 * 2 * 2 = 8
3 * 3 * 3 = 27
```

- En vez de una lista se puede escribir una cadena, en cuyo caso la variable de control va tomando como valor cada uno de los caracteres:

```
for i in "AMIGO":
    print("Dame una ", i)
print("¡AMIGO!")
```

RESULTADO:

```
Dame una A
Dame una M
Dame una I
Dame una G
Dame una O
¡AMIGO!
```

- En los ejemplos anteriores se ha utilizado una lista para facilitar la comprensión del funcionamiento de los bucles pero, si es posible hacerlo, se recomienda utilizar tipos range(), entre otros motivos porque durante la ejecución del programa ocupan menos memoria en el ordenador.

  El siguiente programa es equivalente al programa del ejemplo anterior:

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

RESULTADO:

```
Comienzo
Hola Hola Hola
Final
```

- Otra de las ventajas de utilizar tipos range() es que el argumento del tipo range() controla el número de veces que se ejecuta el bucle.
- En el ejemplo anterior basta cambiar el argumento para que el programa salude muchas más veces.

```
print("Comienzo")
for i in range(10):
    print("Hola ", end="")
print()
print("Final")
```

RESULTADO:

```
Comienzo
Hola Hola Hola Hola Hola Hola Hola Hola Hola Hola
Final
```

- Esto permite que el número de iteraciones dependa del desarrollo del programa. En el ejemplo siguiente es el usuario quien decide cuántas veces se ejecuta el bucle:

```
veces = int(input("¿Cuántas veces quiere que le salude? "))
for i in range(veces):
```

```
        print("Hola ", end="")
    print()
    print("Adiós")
```

RESULTADO:

```
¿Cuántas veces quiere que le salude? 6
Hola Hola Hola Hola Hola Hola
Adiós
```

2. CONTADORES Y ACUMULADORES

En muchos programas se necesitan variables que cuenten cuántas veces ha ocurrido algo (contadores) o que acumulen valores (acumuladores). Las situaciones pueden ser muy diversas, por lo que simplemente hay aquí un par de ejemplos para mostrar la idea.

3. CONTADOR

Se entiende por contador una variable que lleva la cuenta del número de veces que se ha cumplido una condición. El ejemplo siguiente es un ejemplo de programa con contador (en este caso, la variable que hace de contador es la variable cuenta):

Ejemplo de contador

```
print("Comienzo")
cuenta = 0
for i in range(1, 6):
    if i % 2 == 0:
        cuenta = cuenta + 1
print("Desde 1 hasta 5 hay ", cuenta, " múltiplos de 2")
```

RESULTADO:

```
Comienzo
Desde 1 hasta 5 hay 2 múltiplos de 2
```

DETALLES IMPORTANTES:

- En cada iteración, el programa comprueba si i es múltiplo de 2.
- El contador se modifica sólo si la variable de control i es múltiplo de 2.
- El contador va aumentando de uno en uno.
- Antes del bucle se debe dar un valor inicial al contador (en este caso, 0)

4. ACUMULADOR

Se entiende por acumulador una variable que acumula el resultado de una operación. El ejemplo siguiente es un ejemplo de programa con acumulador (en este caso, la variable que hace de acumulador es la variable suma):

Ejemplo de acumulador

```
print("Comienzo")
suma = 0
for i in [1, 2, 3, 4]:
    suma = suma + i
print("La suma de los números de 1 a 4 es", suma)
```

RESULTADO:

```
Comienzo
La suma de los números de 1 a 4 es 10
```

DETALLES IMPORTANTES:

- El acumulador se modifica en cada iteración del bucle (en este caso, el valor de i se añade al acumulador suma).
- Antes del bucle se debe dar un valor inicial al acumulador (en este caso, 0)

5. BUCLES ANIDADOS

Se habla de bucles anidados cuando un bucle se encuentra en el bloque de instrucciones de otro bloque.
Al bucle que se encuentra dentro del otro se le puede denominar bucle interior o bucle interno. El otro bucle sería el bucle exterior o bucle externo.
Los bucles pueden tener cualquier nivel de anidamiento (un bucle dentro de otro bucle dentro de un tercero, etc.).
Aunque en Python no es necesario, se recomienda que los nombres de las variables de control de los bucles anidados no coincidan, para evitar ambigüedades.

6. BUCLES ANIDADOS (VARIABLES INDEPENDIENTES)

Se dice que las variables de los bucles son independientes cuando los valores que toma la variable de control del bucle interno no dependen del valor de la variable de control del bucle externo. Por ejemplo:
Ejemplo de bucle anidado (variables independientes)

```
for i in [0, 1, 2]:
    for j in [0, 1]:
        print("i vale ", i, " y j vale ", j)
```

RESULTADO:

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

En el ejemplo anterior, el bucle externo (el controlado por i) se ejecuta 3 veces y el bucle interno (el controlado por j) se ejecuta dos veces por cada valor de i. Por ello la instrucción print() se ejecuta en total 6 veces (3 veces que se ejecuta el bucle externo x 2 veces que se ejecuta cada vez el bucle interno = 6 veces).

En general, el número de veces que se ejecuta el bloque de instrucciones del bucle interno es el producto de las veces que se ejecuta cada bucle.

En el ejemplo anterior se han utilizado listas para facilitar la comprensión del funcionamiento del bucle pero, si es posible hacerlo, se recomienda utilizar tipos range(), entre otros motivos porque durante la ejecución del programa ocupan menos memoria en el ordenador y se pueden hacer depender del desarrollo del programa.

El siguiente programa es equivalente al ejemplo anterior:

```
for i in range(3):
    for j in range(2):
        print("i vale ", i, " y j vale ", j)
```

RESULTADO:

```
i vale 0 y j vale 0
i vale 0 y j vale 1
i vale 1 y j vale 0
i vale 1 y j vale 1
i vale 2 y j vale 0
i vale 2 y j vale 1
```

Al escribir bucles anidados, hay que prestar atención al sangrado de las instrucciones, ya que ese sangrado indica a Python si una instrucción forma parte de un bloque u otro. En los tres siguientes programas la única diferencia es el sangrado de la última instrucción:

```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(j, end=" ")
        print(i, end=" ")
```

RESULTADO:

```
11 1 12 1 11 2 12 2 11 3 12 3
```

En este caso, la última instrucción forma parte del cuerpo del bucle interno. Por tanto, el valor de i se escribe cada vez que se ejecuta el bucle interno.

Ejemplo de sangrado en bucle anidado (2)

```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(j, end=" ")
    print(i, end=" ")
```

RESULTADO:

```
11 12 1 11 12 2 11 12 3
```

En este caso, la última instrucción forma parte del cuerpo del bucle externo, pero no del interno. Por tanto, el valor de i se escribe cada vez que se ha terminado de ejecutar el bucle interno.

Ejemplo de sangrado en bucle anidado (3)

```
for i in [1, 2, 3]:
    for j in [11, 12]:
        print(j, end=" ")
print(i, end=" ")
```

RESULTADO:

```
11 12 11 12 11 12 3
```

En este caso, la última instrucción no forma parte de ningún bucle. Por tanto, el valor de i se escribe una sola vez, al terminarse de ejecutar el bucle externo.

Puede ver la ejecución paso a paso de este programa utilizando los iconos de avance y retroceso situados abajo a la derecha.

La costumbre más extendida es utilizar la letra "i" como nombre de la variable de control del bucle externo y la letra "j" como nombre de la variable de control del bucle interno (o "k" si hay un tercer nivel de anidamiento), pero se puede utilizar cualquier otro nombre válido.

En Python se puede incluso utilizar la misma variable en los dos bucles anidados porque Python las trata como si fueran dos variables distintas. Por ejemplo:

```
for i in range(3):
    print("i (externa) vale ", i)
    for i in range(2):
        print("i (interna) vale ", i)
```

RESULTADO:

```
i (externa) vale 0
i (interna) vale 0
i (interna) vale 1
i (externa) vale 1
i (interna) vale 0
i (interna) vale 1
i (externa) vale 2
i (interna) vale 0
i (interna) vale 1
```

De todas formas, este uso no se recomienda porque da lugar a programas difíciles de interpretar y además no es habitual en otros lenguajes de programación. Se aconseja utilizar siempre nombres de variables distintos.

7. BUCLES ANIDADOS (VARIABLES DEPENDIENTES)

Se dice que las variables de los bucles son dependientes cuando los valores que toma la variable de control del bucle interno dependen del valor de la variable de control del bucle externo. Por ejemplo:

Ejemplo de bucle anidado (variables dependientes) 1

```
for i in [1, 2, 3]:
    for j in range(i):
        print("i vale ", i, " y j vale ", j)
```

RESULTADO:

```
i vale 1 y j vale 0
i vale 2 y j vale 0
i vale 2 y j vale 1
i vale 3 y j vale 0
i vale 3 y j vale 1
i vale 3 y j vale 2
```

En el ejemplo anterior, el bucle externo (el controlado por i) se ejecuta 3 veces y el bucle interno (el controlado por j) se ejecuta 1, 2 y 3 veces. Por ello la instrucción print() se ejecuta en total 6 veces.
La variable i toma los valores de 1 a 3 y la variable j toma los valores de 0 a i, por lo que cada vez el bucle interno se ejecuta un número diferente de veces:

- Cuando i vale 1, range(i) devuelve la lista [0] y por tanto el bucle interno se ejecuta una sola vez y el programa escribe una sola línea en la que i vale 1 (y j vale 0).
- Cuando i vale 2, range(i) devuelve la lista [0, 1] y por tanto el bucle interno se ejecuta dos veces y el programa escribe dos líneas en la que i vale 2 (y j vale 0 o 1 en cada una de ellas).
- Cuando i vale 3, range(i) devuelve la lista [0, 1, 2] y por tanto el bucle interno se ejecuta tres veces y el programa escribe tres líneas en la que i vale 3 (y j vale 0, 1 o 2 en cada una de ellas).

CONCLUSIONES Y RECOMENDACIONES DE LA EXPERIENCIA

- Al momento de construir el bucle repetitivo, hay que verificar los parámetros de inicio, fin, y el criterio a incrementar o decrementar; si no, tendremos un cálculo diferente o incluso un error.
- Se recomienda usar el Edube SandBox para realizar nuestros ejercicios sin necesidad de instalar algún software adicional.
- Así como en algún momento se solicita cálculos con números enteros, sería factible buscar realizar lo mismo; pero con números reales.

ACTIVIDAD VIRTUAL

1. Observa y analiza el siguiente video: https://www.youtube.com/watch?v=DNEbf5ra0BI, y responde las siguientes preguntas:

- ¿En qué momento se detiene el bucle?
- Realiza ese algoritmo en según los datos proporcionados (Desarróllalo en Python)