Guía Práctica N.º 8: Estructuras Cíclicas (FOR) en Python

Resultado De Aprendizaje:

- Conocer las estructuras cíclicas en Python modo consola
- Desarrollar ejercicios propuestos con la estructura cíclica FOR para resolver situaciones cotidianas.

El ciclo for

Es una herramienta muy útil en Python que permite iterar sobre objetos iterables y realizar operaciones repetitivas de manera sencilla y eficiente. Entre sus principales ventajas se encuentra su facilidad de uso y su capacidad para recorrer cualquier objeto iterable.

Estructura básica

```
# Python
for elemento in iterable:
    # Hacer algo con el elemento
```

En este caso, en cada nueva iteración el "**elemento**" toma el valor del siguiente elemento del objeto "**iterable**". Además, como ocurre con las sentencias **if**, se requiere el uso de los **dos puntos** (**:**) al final de la sentencia y el cuerpo del bucle se tiene que **indentar**.

Indentación del ciclo for

El código por ejecutar para cada elemento debe estar indentado debajo de la declaración for. La indentación en Python es la forma en que se estructura el código en bloques, se utiliza la indentación para indicar el **inicio** y **final** de un bloque de código, en lugar de utilizar llaves o palabras clave como **begin** o **end**.

La indentación en Python se realiza mediante la inserción de espacios en blanco o tabulaciones antes de una línea de código. La cantidad de espacios o tabulaciones que se utilicen deben ser consistentes en todo el archivo para evitar errores. Por lo general, se utilizan cuatro espacios por nivel de indentación.

Es importante tener en cuenta que ***la indentación en Python es obligatoria*** y no es simplemente una cuestión de estilo de codificación. Si no se utiliza la indentación adecuada, el código generará errores de sintaxis y no se ejecutará correctamente.

Objetos iterables

Los objetos iterables o secuencias son aquellos que permiten la iteración o recorrido secuencial de sus elementos mediante un ciclo for. Algunos ejemplos de objetos iterables comunes en Python son: ***listas, tuplas, cadenas de texto (strings), conjuntos o colecciones (sets) y diccionarios.***

Iterable	Concepto	Ejemplo
Lista	Secuencia ordenada de elementos separados por comas y encerrados entre corchetes [ ].	`lista = [ "Uno" , "Dos" , "Tres" ]`
Tupla	Secuencia ordenada e inmutable de elementos separados por comas y encerrados entre paréntesis ( ).	`tupla = ( 1 , 2 , 3 , 4 , 5 )`
Cadena (String)	Secuencia inmutable de caracteres encerrados entre comillas simples ' ' o dobles " ".	`cadena = "Hola mundo"`
Conjunto (Set)	Colección desordenada de elementos únicos encerrados entre llaves { }.	`conjunto = { 8, 4, 6, 2 }`
Diccionarios	Colección de elementos que se almacenan como **clave-valor** y se encierran entre llaves { }.	`diccionario = { "nombre": "Juan", "edad": 21 }`

Ejemplos:

Código en Editor	Salida en Consola

`lista = [ "Uno" , "Dos" , "Tres" ]`
`for i in lista:`
`    print(i)`
	`Uno`
	`Dos`
	`Tres`

`tupla = ( 1 , 2 , 3 , 4 , 5 )`
`for j in tupla:`
`    print(j)`
	`1`
	`2`
	`3`
	`4`
	`5`

`cadena = "Python"`
`for letra in cadena:`
`    print(letra)`
	`P`
	`y`
	`t`
	`h`
	`o`
	`n`

`conjunto = { 0, 4, 6, 2, 2, 4 }`
`for elemento in conjunto:`
`    print(elemento)`
	`0`
	`2`
	`4`
	`6`

	Observe que el resultado es el conjunto sin repetir datos, y lo muestra ordenado a partir del segundo elemento.

Código en Editor	Salida en Consola

```python
diccionario = { "nombre": "Juan", "edad": 21 }
print("Claves:")
for clave in diccionario:
    print(clave)
#Otra forma
for clave in diccionario.keys():
    print(clave)

print("Valores:")
for clave in diccionario:
    print(diccionario[clave])
#Otra forma
for valor in diccionario.values():
    print(valor)

print("Claves con valores:")
for clave, valor in diccionario.items():
    print(clave,":",valor)
```

```
Claves:
nombre
edad
nombre
edad
Valores:
Juan
21
Juan
21
Claves con valores:
nombre : Juan
edad : 21
```

Ciclo for con función range()

La función **range()** es una función integrada en Python que crea una secuencia ordenada de enteros que podemos utilizar para crear iteraciones. Por ello la función **range()** acepta tres números enteros de los cuales dos son opcionales.

range(inicio, fin, paso)

- **Inicio:** es el valor inicial de la secuencia (opcional y será 0 si no se especifica).
- **Fin:** es el valor final de la secuencia, el cual no se incluye en el resultado.
- **Paso:** indica el incremento entre elementos de la secuencia (opcional y será 1 si no se especifica).

Como los objetos de tipo **range** son ***iterables***, también podemos usar esta función para iterar en un bucle **for**.

Ejemplos:

```python
for num in range(5):
    print(num)
```

Imprime los números del 0 al 4

```python
for num in range(6, 10, 2):
    print(num)
```

Imprime los números: 6 y 8, el final no se incluye.

```python
for num in range (3, 0, -1):
    print(num)
```

Imprime los números: 3, 2, 1 (es decir en orden inverso)

Sentencias break y continue

Las sentencias **break** y **continue** modifican el flujo normal de un bucle **for**. En concreto **break** interrumpe completamente el bucle, procediendo a la instrucción que viene después del bucle. Por su parte la sentencia **continue** termina la iteración actual del bucle y pasa a la siguiente.

Ejemplo:

```
for i in range(1 , 11):
    if i == 9:
        break
    elif i == 6:
        continue
    print(i)
```

El resultado sería:

```
1
2
3
4
5
7
8
```

Observe que el ciclo **for** esta programado para que recorra el rango de valores de 1 a 11 sin incluir el ultimo valor pero nunca llega al valor de 10 porque la condicion de **i==9** hace que se ejecute la instrucción **break** y finaliza el ciclo completo, pero en cambio la instrucción **continue** solo hace que termine la **iteracion en curso** por lo tanto cuando **i==6** continua a la siguiente iteracion sin pasar por la instrucción de **print(i)** haciendo que se omita el numero 6 en los resultados.

for … else

En relación con el apartado anterior, Python ofrece una estructura adicional de bucle for cuya estructura es la siguiente:

```
for e in iterable:
    # Tu código aquí
else:
    # Este código siempre se ejecuta si no
    # se ejecutó la sentencia break en el bloque for
```

Es decir, el código del bloque **else** se ejecutará siempre y cuando no se haya ejecutado la sentencia **break** dentro del bloque del **for**.

Ejemplo:

```
numeros = [1, 2, 4, 3, 5, 8, 6]
for n in numeros:
    if n == 3:
        break
else:
    print('No se encontró el número 3')
```

Como en el ejemplo anterior la secuencia números contiene al número 3, la instrucción **print** nunca se ejecutará.

GUIA DE EJERCICIOS.

1. **Imprimir los números del 1 al 10**: Escribe un programa que imprima los números del 1 al 10 utilizando un bucle for.
2. **Imprimir los números pares del 1 al 20**: Escribe un programa que imprima los números pares del 1 al 20 utilizando un bucle for.
3. **Imprimir los números impares del 1 al 20**: Escribe un programa que imprima los números impares del 1 al 20 utilizando un bucle for.
4. **Imprimir la tabla de multiplicar del 5**: Escribe un programa que imprima la tabla de multiplicar del 5 utilizando un bucle for.
5. **Sumar los números del 1 al 100**: Escribe un programa que sume los números del 1 al 100 y luego imprima el resultado.
6. **Calcular el factorial de un número**: Escribe un programa que calcule el factorial de un número dado.
7. **Imprimir los primeros 10 números de la serie de Fibonacci**: Escribe un programa que imprima los primeros 10 números de la serie de Fibonacci.
8. **Imprimir una pirámide de asteriscos**: Escribe un programa que imprima una pirámide de asteriscos.
9. **Imprimir los elementos de una lista**: Escribe un programa que imprima los elementos de una lista dada.
10. **Imprimir los índices y los elementos de una lista**: Escribe un programa que imprima los índices y los elementos de una lista dada.