El futuro digital es de todos

MinTIC

Bucle While

OPERADO POR:

Universidad Tecnológica de Pereira

Mision TIC 2022

RUTA DE APRENDIZAJE 1

Dar estilos a los textos con CSS

El ciclo While

Las computadoras se utilizan a menudo para realizar tareas repetitivas. Repetir tareas idénticas o similares sin cometer errores es algo que los ordenadores hacen bien y la gente lo hace mal. Debido a que la iteración es tan común, Python proporciona varias características de lenguaje para hacerlo más fácil.

En Python, así como en muchos otros lenguajes, la base de las instrucciones iterativas es una expresión como la siguiente: “mientras que X sea cierto, haga Y”. En inglés, esto se traduciría como “while X, do Y”. Por esto, la instrucción fundamental para expresar iteraciones en Python y en muchos lenguajes se llama while.

Así como un **if** define una estructura en la cual se tiene que tener como mínimo una condición y un cuerpo, en el caso del **while** ocurre algo similar: se requiere una condición (la X en la expresión anterior) y un cuerpo (la Y). La principal diferencia con un **if** es que la condición se va a evaluar muchas veces y que el cuerpo se va a ejecutar cada vez que la condición sea verdadera. Observemos esto en un ejemplo:

```
n = 5
while n > 0:
    print(n)
    n = n - 1
print('Despegue!')
```

Lo primero que encontramos en este código es que vamos a crear una nueva variable llamada n y la vamos a inicializar en 5.

A continuación inicia un bloque **while**, donde la condición es n > 0. Esto quiere decir que el cuerpo del **while** (todo lo que está indentado después de :) se va a ejecutar varias veces, hasta que la condición deje de ser verdadera.

Esta condición se evaluará antes de la primera vez que se ejecute el cuerpo y se volverá a evaluar después de cada ejecución del cuerpo. El cuerpo de este **while** tiene sólo dos instrucciones: la primer imprime en la consola el valor actual de la variable n mientras que la segunda reduce el valor de n en uno.

La última instrucción del ejemplo imprime la cadena '¡Despegue!' en la consola.

Más formalmente, aquí está el flujo de ejecución de la declaración **while**:

1.Evalúa la condición, dando Verdadero o Falso.
2.Si la condición es falsa, salga de la declaración while y continúe la ejecución en la siguiente declaración.
3.Si la condición es verdadera, ejecute el cuerpo y luego vuelva al paso 1.

Este tipo de flujo se llama **bucle** porque el tercer paso vuelve hacia la parte superior. Llamamos cada vez que ejecutamos el cuerpo del bucle una iteración. Para el bucle anterior, diríamos: "Tenía cinco iteraciones", lo que significa que el cuerpo del bucle se ejecutó cinco veces.

El cuerpo del **bucle** debe cambiar el valor de una o más variables para que eventualmente la condición se vuelva falsa y el **bucle** termine. Llamamos variable de iteración a la variable que cambia cada vez que el bucle se ejecuta y controla cuando el **bucle termina**.

Si no hay una variable de iteración, el bucle se repetirá para siempre, resultando en un bucle infinito.

Elementos de un while

Acabamos de mostrar con un ejemplo cómo es la ejecución de un **while**. Ahora estudiaremos con un poco más de cuidado los diferentes elementos que se tienen que considerar cuando se construya una instrucción iterativa. Para esto usaremos la siguiente función que sirve para calcular el factorial de un número recordando que:

```python
def factorial(n: int) -> int:
    resultado = 1
    numero_actual = 2
    while numero_actual <= n:
        resultado = resultado * numero_actual
        numero_actual += 1
    return resultado
```

Esta función calcula el factorial del número n multiplicando entre ellos todos los números anteriores. El punto importante es que esto lo vamos a hacer número por número, partiendo desde el número 1 y llegando hasta el número n.

Esta función calcula el factorial del número n multiplicando entre ellos todos los números anteriores. Lo mas importante es que esto lo vamos a hacer número por número, partiendo desde el número 1 y llegando hasta el número n.

Inicialización

Aunque no es parte explícita de un **while**, las instrucciones que se encuentran antes son importantísimas porque realizan la inicialización del ciclo. Es decir, dejan las variables que nos interesen en el estado necesario para que se pueda ejecutar el ciclo y se obtenga el resultado esperado.

En el caso de nuestra función, hay dos instrucciones que sirven para inicializar el ciclo:

```
resultado = 1
numero_actual = 2
```

La primera instrucción sirve para crear una variable donde dejaremos el resultado de nuestra función. En este caso, ese resultado será el factorial del parámetro n. La variable resultado la hemos inicializado en 1 por varios motivos que discutiremos más adelante.

La segunda instrucción sirve para crear una variable que nos permitirá saber cuál es el siguiente número que tenemos que multiplicar para seguir calculando el factorial. En este caso, la variable **numero_actual** la inicializamos en 2 porque el resultado ya estaba inicializado en 1. Si la variable la hubiéramos inicializado en 1 no habría cambiado nuestro programa, pero habría hecho una iteración más en el que habría multiplicado **1*1**.

Condición del ciclo

La siguiente parte del análisis de nuestra función se concentra en la condición del **while**. Debemos recordar que cuando la condición de un **while** sea verdadera, el cuerpo del ciclo deberá ejecutarse. Si lo vemos desde el punto de vista opuesto, el cuerpo del ciclo tendrá que ejecutarse hasta que la condición sea falsa.

Las dos perspectivas son equivalentes pero, dependiendo del problema, es posible que una de las dos perspectivas sea más fácil de entender.

La recomendación que podemos hacer es utilizar nombres de variables que sean muy claros, de tal forma que la condición sea fácil de leer.

Por **ejemplo**, en el caso de nuestra función, la condición puede leerse fácilmente como “Mientras que el número actual sea menor o igual que n, se debe hacer …”.

```
while numero_actual <= n:
```

La misma condición también se podría haber escrito de la siguiente forma, en la cual la lectura natural sería “Mientras que el número actual no sea mayor a n, se debe hacer …”.

```
while not numero_actual > n:
```

Cuerpo del ciclo

Después de evaluada la condición de un while, se ejecuta una o varias veces el cuerpo del ciclo. El caso de nuestra función, el cuerpo tiene dos instrucciones:

La primera instrucción es la que se encarga de ir acumulando en la variable resultado el valor del factorial del número. Para esto, la instrucción toma el valor que se había acumulado hasta el momento, lo multiplica por el número actual y vuelve a guardarlo en resultado. La segunda instrucción se encarga de ir incrementando de uno en uno el valor de **numero_actual**.

```
resultado = resultado * numero_actual
numero_actual = numero_actual + 1
```

La siguiente tabla muestra cómo va cambiando el valor de resultado y de numero_actual a medida que se ejecuta el ciclo:

resultado	nnumero_actual	Valor calculado en resultado
1	2	1!
2	3	2!
6	4	3!
24	5	4!
120	6	5!
720	7	6!

Antes de la primera ejecución, el valor de resultado es 1 y el valor de **numero_actual es 2**, como se determinó en la inicialización. Eso quiere decir que el valor de resultado es equivalente al valor de 1! y que el siguiente número por el que debería multiplicarse es 2.

Después de la primera ejecución del ciclo, el valor de resultado se modifica para que sea 2, y el valor de numero_actual se incrementa en uno. Esto quiere decir que ahora el valor de resultado es equivalente al valor de 2!.

En la siguiente iteración el valor de resultado se multiplica por 3 y y el valor de **numero_actual** llega a 4. Como ahora el valor de resultado es 6, quiere decir que es equivalente a 3!.

El proceso continua hasta que **numero_actual** es mayor a n. Por ejemplo, si n fuera 6, entonces en la última iteración resultado se multiplicaría por 6 quedando con un valor equivalente al de 6! y **numero_actual** llegaría a 7. La siguiente vez que se revisara la condición ya no sería verdadera y el ciclo terminaría.

Avance

La segunda instrucción del cuerpo, **numero_actual += 1**, tiene el rol de avanzar el ciclo hacia su terminación. Este rol es muy importante dentro de cualquier ciclo: siempre tiene que haber una o varias instrucciones que hagan que con cada iteración el ciclo esté más cerca de terminar. Si no se cumpliera esto, el ciclo nunca terminaría.

En nuestra función para el cálculo del factorial, con cada iteración aumentamos en uno el valor de **numero_actual**, con lo cual nos aseguramos que eventualmente este número sea mayor a n y el ciclo termine. En el caso del contador para el despegue, vamos reduciendo el valor de contador haciendo que eventualmente se vuelva falsa la condición del ciclo: contador > 0.

Problema con el avance

En los dos ejemplos que hemos estudiado identificar el avance fue relativamente fácil. Más adelante en esta sección estudiaremos algunos programas donde no es tan fácil ver que el ciclo se está acerca acercando a la terminación. Por ahora veamos unos ejemplos de programas con problemas y que resultarán en ciclos infinitos.

Alejarse de terminación:

Este primer programa nunca termina porque el valor de i siempre es mayor a 0.

```
i = 1
while i > 0:
    print(i)
    i = i+ 1
print("Terminé")
```

Brincarse la meta

Este programa tampoco termina porque i siempre va a ser diferente que 10. El problema acá es que estamos incrementando i de dos en dos, pero empezando en 1. Esto quiere decir que i sólo va a asumir valores impares. Una forma fácil de solucionar este problema habría sido cambiar la condición para que fuera i < 10.

```
# le toca oprimir el boton de pausa sino el sigue dandole

i = 1
while i != 10:
    print(i)
    i = i+ 2
print("Terminé")
```

Problemas de indentación:

Aunque este programa tiene todas las instrucciones que se esperarían, tampoco terminará nunca su ejecución. El problema acá es que el avance no está dentro del ciclo: la variable i siempre va a tener el valor 1 porque el único lugar donde se cambia es inmediatamente después del ciclo.

```
i = 1
while i < 10:
    print(i)
i = i + 1
print("Terminé") #aqui tambien le toca hacer lo mismo
```

Este es uno de los problemas mas comunes cuando estamos aprendiendo a programar con Python, por eso es muy importante practicar y construir varios códigos para mejorar.

Olvidar el avance:

Aunque es muy sencillo, este programa ilustra el problema que se presenta más frecuentemente cuando se trabaja con ciclos: olvidar el avance. Como en este caso **i** nunca cambia de valor, el ciclo no se acerca a terminación a medida que se ejecuta.

```
i = 1
while i < 10:
    print(i)
print("Terminé")
```

Bucle ‘while’ controlado por Evento:

A continuación, se presenta un ejemplo del uso del bucle **while** controlado por Evento:

```
promedio, total, contar = 0.0, 0, 0

print("Introduzca la nota de un estudiante (-1 para salir): ")
grado = int(input())
while grado != -1:
    total = total + grado
    contar = contar + 1
    print("Introduzca la nota de un estudiante (-1 para salir): ")
    grado = int(input())
promedio = total / contar
print("Promedio de notas del grado escolar es: " + str(promedio))
```

```
Introduzca la nota de un estudiante (-1 para salir):
```

Bucle ‘while’ con ‘else’

Al igual que la sentencia **if**, la estructura **while** también puede combinarse con una sentencia **else**).
El nombre de la sentencia **else** es equivocada, ya que el bloque **else** se ejecutará en todos los casos, es decir, cuando la expresión condicional del **while** sea **False**, (a comparación de la sentencia if).

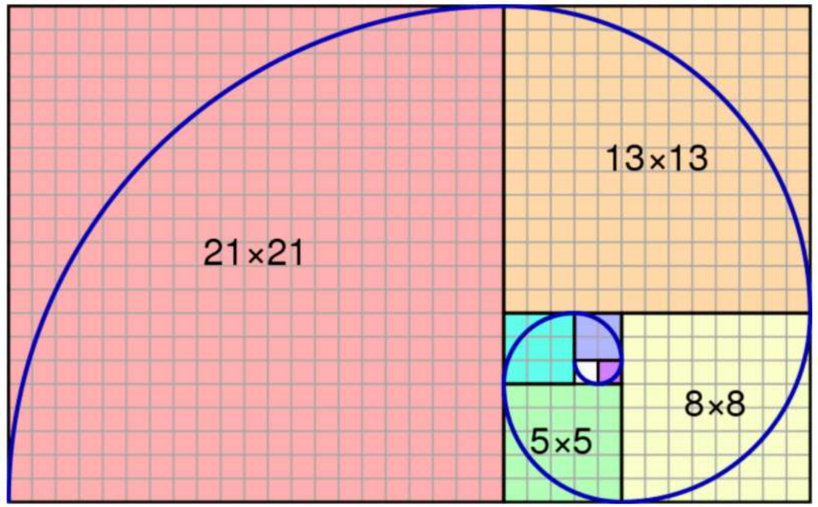
```
promedio, total, contar = 0.0, 0, 0
mensaje = "Introduzca la nota de un estudiante (-1 para salir): "

grado = int(input(mensaje))
while grado != -1:
    total = total + grado
    contar += 1
    grado = int(input(mensaje))
else:
    promedio = total / contar
    print("Promedio de notas del grado escolar: " + str(promedio))
```

La sentencia **else** tiene la ventaja de mantener el mismo nombre y la misma sintaxis que en las demás estructuras de control.

Ejemplo: Sucesión de Fibonacci:

Ejemplo de la Sucesión de Fibonacci con bucle **while,** vamos a generar una de las sucesiones mas conocidas e interesantes de la matemática,

```
a, b = 0, 1
while b < 100:
    print b,
    a, b = b, a + b
```



El futuro digital es de todos

MinTIC

GRACIAS

OPERADO POR:

Universidad Tecnológica de Pereira

MisiónTIC2022