Sentencias condicionales e iterativas

El Ciclo While

{desafío} latam_

Utilizar sentencias iterativas para la elaboración de un algoritmo que resuelve un problema acorde al lenguaje Python.

- Unidad 1:
  Introducción a Python
- Unidad 2:
  Sentencias condicionales e iterativas
- Unidad 3:
  Estructuras de datos y funciones

Te encuentras aquí

{desafío}
latam_

¿Qué aprenderás en esta sesión?

- *Utiliza sentencias FOR y WHILE para la elaboración de un algoritmo iterativo que resuelve un problema acorde al lenguaje Python.*
- *Utiliza ciclos anidados y condiciones de salida para resolver un problema acorde al lenguaje Python.*

{desafío} latam_

¿Qué entendemos por sentencias?

{desafío} latam_

/* Ciclos */

Introducción a ciclos

Los ciclos son sentencias que nos permiten repetir la ejecución de una o más instrucciones.

- Repetir instrucciones es la clave para crear programas avanzados, ya que, como programadores, nos interesa poder tener la mayor cantidad de funcionalidades sin tener que tener un código extremadamente largo.
- Optimizar no solo significa que utilice funciones más avanzadas, sino que también hacerlo más compacto, lo que ayudará a entenderlo de mejor manera.

```
Mientras se cumple una condición:

  Instrucción 1
  Instrucción 2
  Instrucción 3
```

Ciclo While

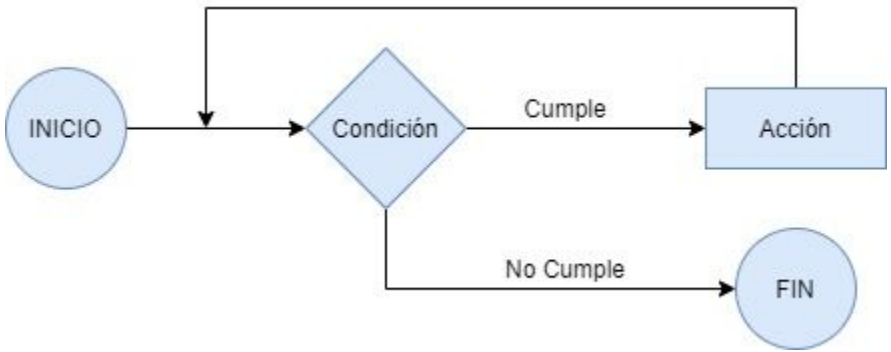
Permite ejecutar una o más operaciones mientras se cumpla una condición, la cual es idéntica a las utilizadas en nuestras sentencias if y la sintaxis es la siguiente:

```
while condición:
# código a implementar
```

Ciclo While

Ejemplo

1. Se evalúa la condición; si es True, ingresa al ciclo.
2. Se ejecutan secuencialmente las instrucciones definidas dentro del ciclo.
3. Una vez ejecutadas todas las instrucciones, se vuelve a evaluar la condición:
    - Si se evalúa como True: vuelve a repetir.
    - Si se evalúa como False: sale del ciclo.



Ciclo While

Salida del ciclo

Como vimos anteriormente, un algoritmo es una secuencia de pasos **FINITA** para resolver un problema.

En algún momento, algunas de las instrucciones dentro del bloque deben lograr que la condición no se cumpla, es decir, debe existir una **condición de salida o término** del ciclo.

Ejercicio guiado

Password

{desafío}
latam_

Password

Todo buen programador sabe que existen algunos elementos que son sensibles, ya que no todo puede ser público, por lo que algunas veces va a ser necesario la inclusión de contraseñas o passwords para proteger la información.

Entonces, ¿cómo podemos utilizar while para implementar un password?

Password

Solución

Paso 1

Creemos el archivo password.py

Paso 2

Solicitamos la clave. Para ello, utilizaremos la librería getpass.

```
import getpass
password = getpass.getpass("Ingrese la clave secreta: ")
```

Password

Solución

Paso 3

Si la clave es correcta queremos que nuestro programa inicie, de lo contrario, queremos que vuelva a solicitar la clave. Dado que queremos que vuelva a realizar una acción de solicitar la clave hasta que se ingrese la clave correcta, es necesario utilizar el ciclo while:

```
# En este caso definimos nuestro password como "hola mundo"
# En este caso, mientras la contraseña no sea hola mundo,
# seguirá solicitando la contraseña, pero esta vez con otro mensaje.

while password != "hola mundo":
   password = getpass.getpass("La clave secreta no es correcta. Intenta otra vez.")
```

Password

Solución

Paso 4

Finalmente, podemos incluir el código final de nuestro programa. En este caso, solo agregaremos un código genérico que da inicio a nuestro programa.

```
print("Clave Correcta. Puedes utilizar tu programa")
# Posterior a esto podríamos agregar el código de nuestro programa.
```

Password

Solución

```python
import getpass
password = getpass.getpass("Ingrese la clave secreta: ")

# En este caso definimos nuestro password como "hola mundo"
# En este caso, mientras la contraseña no sea hola mundo,
# seguirá solicitando la contraseña, pero esta vez con otro mensaje.

while password != "hola mundo":
    password = getpass.getpass("La clave secreta no es correcta. Intenta otra vez: ")

print("Clave Correcta. Puedes utilizar tu programa")
# Posterior a esto podríamos agregar el código de nuestro programa.
```

¿Por qué es necesario declarar dos veces el ingreso de datos por parte del usuario?

Hay varias razones por las cuales es necesario solicitar el ingreso de la clave dos veces:

- La línea 2 solicita la clave por primera vez. Si la clave es correcta, el ciclo while nunca comenzará, y el programa iniciará su funcionamiento de la manera esperada.
- Adicionalmente, si la línea 2 no existe, aparecerá un error, ya que la línea 8 realiza una prueba lógica con la variable password, que no estaría definida.
- Finalmente, queremos solicitar la clave de dos maneras distintas, una de manera inicial (línea 1) y una en caso de equivocarse línea 9.

Ahora bien, revisemos el funcionamiento de nuestro programa:

```
└─ python password.py
Ingrese la clave secreta:
La clave secreta no es correcta. Intenta otra vez:
```

Notamos que aparece el mensaje de error e inmediatamente nos solicita la clave nuevamente. La ventaja de usar una instrucción while, es que esto puede ocurrir cuantas veces sea necesario:

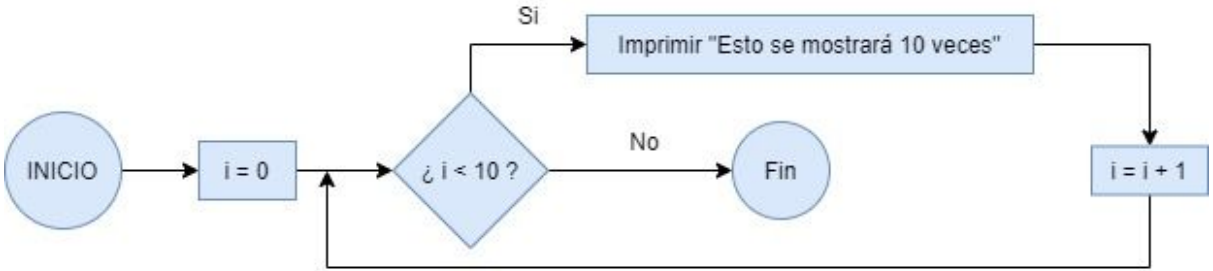
```
└─ python password.py
Ingrese la clave secreta:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
La clave secreta no es correcta. Intenta otra vez:
```

/* Iterar */

Iterar

Iterar es dar una vuelta al ciclo, y por diseño, el ciclo while es un ciclo infinito, donde la mayoría de las veces no se sabe cuántas iteraciones tendrá.

Por ejemplo, en el ejercicio de la contraseña, va a depender de cuantas veces el usuario ingrese el password de manera incorrecta; aún así, existen otros problemas en el que a priori sí se conoce cuántas veces es necesario iterar.



Contando con While

En el diagrama anterior, donde se buscaba imprimir un texto en pantalla 10 veces, la implementación del código se realiza de la siguiente forma:

```
i = 0
while i < 10:
    print("Esto se mostrará 10 veces") # está es la expresión a repetir
    i = i + 1 # IMPORTANTE
```

Contando con While

- La instrucción print("Esto se mostrará 10 veces") se repetirá hasta que la variable i alcance el valor 10. Una vez esto ocurra, la condición asociada al while se evaluará como False y terminará el ciclo.

- Considera que en programación, es una convención utilizar una variable llamada i como variable de iteración para operar en un ciclo.

- En este ciclo el iterador (i) en la primera vuelta valdrá 1, en la segunda iteración valdrá 2, en la tercera valdrá 3 y así sucesivamente hasta llegar a la condición declarada que sea menor que 10.

Si no se aumenta el valor de la variable i, entonces el ciclo nunca alcanzará su condición de salida, por ende, el loop será infinito.

/* Operadores de asignación */

Operadores de asignación

Los operadores de asignación permiten realizar una operación sobre una variable, pero a la vez sobreescribir esa misma variable, básicamente, son modificadores de esa variable.

```
# Este trozo de código incrementa el valor de i en 1.
Al valor actual de i le suma 1 y lo vuelve a asignar a i.
i = i + 1

# Mismo resultado, pero más compacto
i += 1 # Esto es un operador de asignación
```

Operadores de asignación

Comportamiento

Operador	Nombre	Ejemplo	Resultado
=	Asignación.	a = 2	a toma el valor 2.
+=	Incremento y asignación.	a += 2	a es incrementado en dos y asignado el valor resultante.
-=	Decremento y asignación.	a -= 2	a es reducido en dos y asignado el valor resultante.
*=	Multiplicación y asignación.	a *= 3	a es multiplicado por tres y asignado el valor resultante.
/=	División y asignación.	a /= 3	a es dividido por tres y asignado el valor resultante.

Ejercicio guiado

"Bomba de tiempo"

Bomba de tiempo

Para entender mejor el funcionamiento de los ciclos while, realizaremos un pequeño juego, el cual nos permitirá combinar ciclos y operadores de asignación.

Para ello, crearemos un programa en el que ingresamos un tiempo determinado hasta que explote una bomba.

Bomba de tiempo

Solución

Paso 1

Crear el archivo bomba.py

Paso 2

En este caso podemos utilizar sys.argv para ingresar el número de segundos antes de explotar.

```
import sys
i = int(sys.argv[1]) # fijar valor inicial
```

{desafío} latam_

Bomba de tiempo

Solución

Paso 3

Creamos un ciclo que nos permita generar decremento, desde el valor inicial hasta cero.

```
import sys
i = int(sys.argv[1]) # fijar valor inicial
# el ciclo terminará cuando i sea 0
while i > 0:
    print(i)
    i -= 1 # i irá decreciendo en 1 unidad.
```

Bomba de tiempo

Solución

Paso 4

Para que efectivamente **i** represente un segundo, podemos utilizar la librería time, la que nos permitirá esperar un segundo entre cada decremento. La idea es que al terminar el ciclo, la bomba explote, de manera que el código completo se vea así:

```
import time
import sys
i = int(sys.argv[1]) # fijar valor inicial

while i > 0:
    print(i)   # Muestra el valor de i
    time.sleep(1) # espera un segundo
    i -=1 # Descuenta 1 al valor de i.

print("BOOM!") # al salir del ciclo la bomba explota!!
```

Bomba de tiempo

Solución

Paso 5

Si bien este ejercicio puede parecer simple, no es tan intuitivo entender el código. A continuación, se muestra cómo sería en caso de no utilizar un ciclo, a través de un ejemplo que demuestra el caso de 5 segundos:

```
import time

i = 5
print(i)   # Muestra el valor 5
time.sleep(1) # espera un segundo

i = 4 # Descuenta 1 al valor de i.
print(i)   # Muestra el valor 4
time.sleep(1) # espera un segundo

i = 3 # Descuenta 1 al valor de i.
print(i)   # Muestra el valor 3
time.sleep(1) # espera un segundo

i = 2 # Descuenta 1 al valor de i.
print(i)   # Muestra el valor 2
time.sleep(1) # espera un segundo

i = 1 # Descuenta 1 al valor de i.
print(i)   # Muestra el valor de i
time.sleep(1) # espera un segundo

i = 0
# En este punto se evalúa el fin de ciclo ya que i es cero.

print("BOOM!") # al salir del ciclo la bomba explota!!
```

/* Contadores y acumuladores */

Elementos para trabajar con ciclos

Contador

Aumenta de 1 en 1.

- cont = cont + 1
- cont += 1

Acumulador

Acumula, el valor anterior más un valor adicional.

- acu = acu + valor
- acu += valor

También es posible utilizar el operador de asignación += para ir acumulando strings.

Esto porque, como ya hemos visto, el operador + funciona como un concatenador al ser aplicado este tipo de datos.

Elementos para trabajar con ciclos

Ejemplos

```
saludo = "hola"
saludo += " mundo"
print(saludo) # hola mundo
```

```
saludo += "chao"
print(saludo) # hola mundo chao
```

Ejemplo con While

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

Este es un caso muy típico de ciclo infinito, en donde no se agrega un contador a i por lo que este ciclo jamás terminará.

i siempre tendrá el valor 1, por lo que siempre será menor a 10, por lo tanto, este ciclo nunca alcanzará su valor de salida y, por ende, nunca terminará.

¡Continuemos practicando!

Sumando de 1 a 100

$$1 + 2 + 3 + ... + 100 = ¿?$$

Resolver esto es muy similar a contar cien veces, pero además de contar, se debe ir guardando la suma acumulada en cada iteración.

Tips:

- Itera 100 veces.
- En cada iteración, utiliza un contador para almacenar la iteración.
- En cada iteración, asegúrate de utilizar un acumulador para acumular tus contadores.
- El acumulador luego de terminado el ciclo, corresponderá al resultado de la suma.
- Asegúrate de iniciar el contador y el acumulador antes de entrar al ciclo.

Sumando de 1 a 100

Solución

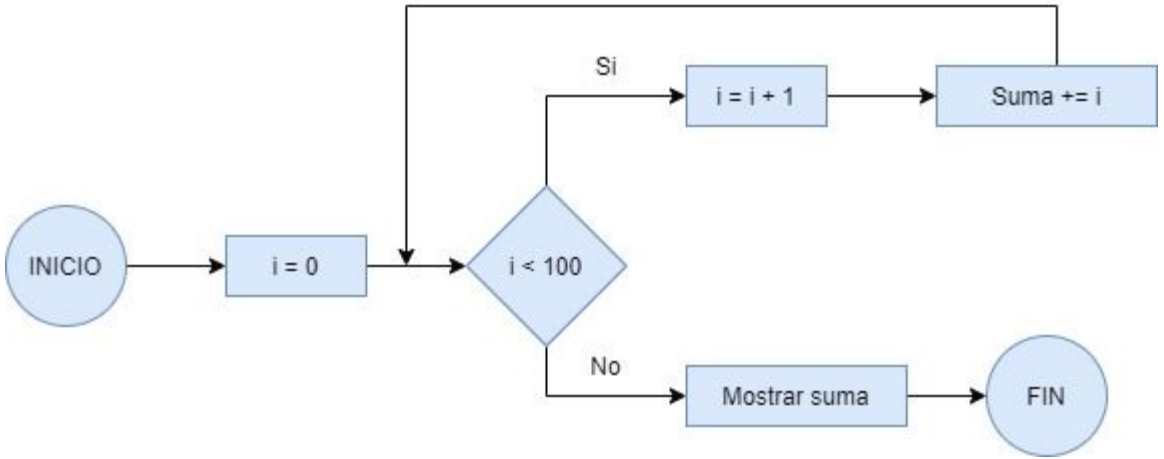
```
i = 1 # primer valor a sumar
suma = 0

while i <= 100:
    suma += i # acumulamos para la suma
    i += 1 # incrementamos para sumar el siguiente valor

print(f"El resultado final es {suma}")
```

Sumando de 1 a 100

Solución



¿Para qué nos sirve el ciclo while?

Próxima sesión...

- *Utiliza sentencias FOR y WHILE para la elaboración de un algoritmo iterativo que resuelve un problema acorde al lenguaje Python.*
- *Utiliza ciclos anidados y condiciones de salida para resolver un problema acorde al lenguaje Python.*

(continuación)

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales