Sentencias condicionales e iterativas

El Ciclo For

{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

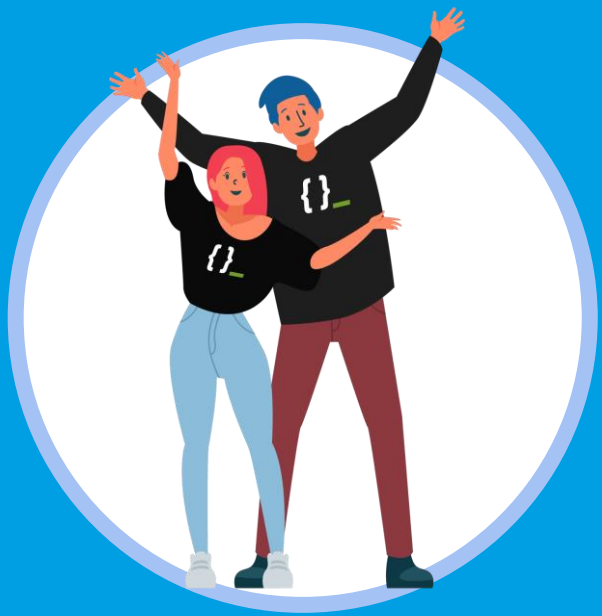
{desafío} latam_

¿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 otro ciclo conoces aparte del while?



FOR

En Python funciona como un “for each”, es decir, que el for iterará en cada elemento de un objeto. Este objeto tiene una característica que lo define: debe ser un iterable.

Su sintaxis es la siguiente:

```
for variable in iterable:
    # se ejecutará código para cada valor del iterable.
    # El código debe estar correctamente indentado.
```

Iterables

Un primer iterable y el más común inicialmente, es utilizar la función **range()**, la que permite generar un espacio con un rango de números. Existen 3 maneras de utilizar **range()**:

Opción 1

```
# Con un sólo valor
for i in range(10):
    print(i)
```

Opción 2

```
# Con dos valores
for i in range(4,10):
    print(i)
```

Opción 3

```
# Con tres valores
for i in range(4,10,2):
    print(i)
```

Iterables

Además, la función range() se verá siempre utilizada dentro de ciclo, ya que al utilizarla por sí sóla entrega lo siguiente:

```
print(range(4,10,2))
```

```
range(4,10,2)
```

De hecho, si se chequea el tipo de dato, nos arrojará que es de tipo range, por lo que inicialmente se utilizará solo como un generador de rangos en el ciclo for.

```
print(type(range(4,10,2)))
```

```
range
```

/* Utilizando estructuras de datos en un ciclo For */

Listas

Como primer ejemplo, podemos iterar por todos los elementos de una lista:

```
a = [1,5,8,3,4]
for elemento in a:
    print(elemento)
```

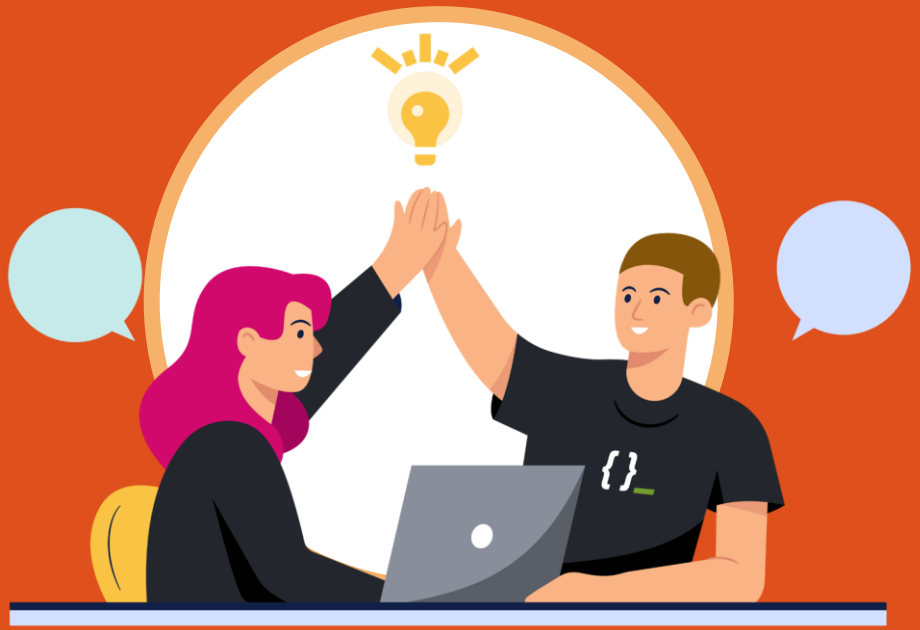
Entendamos el código, estamos asignando una lista de números a la variable a.
Esta lista la estamos recorriendo con el ciclo for donde queremos que por cada elemento dentro del iterable a (que es una lista de python) muestre en consola cada elemento.

Strings

Hemos mencionado en otras ocasiones que los Strings son muy similares a las listas, en este caso, los strings también son iterables. Por ejemplo, podemos deletrear una o más palabras:

```
texto = "hola mundo"
for caracter in texto:
    print(caracter)
```

El ciclo for permite nombrar la variable iteradora de cualquier manera, por lo tanto, es conveniente utilizar un nombre variable que sea representativo de lo que se está iterando.



Diccionarios

Como dijimos en la unidad anterior, un diccionario se compone de una clave y un valor, es por eso que la manera más común de iterar diccionarios es utilizando **.items()**.

Otra diferencia muy importante, es que en cada iteración se extraerán dos elementos, la clave y el valor:

```
diccionario = {"Nombre": "Carlos",
                   "Apellido": "Santana",
                   "Ocupación": "Guitarrista"}

for clave, valor in diccionario.items():
          print(f"Mi {clave} es {valor}")
```

/* Otras funciones */

enumerate()

- Permite agregar un contador a la iteración, por lo tanto extrae elemento y contador.
- Comienza su conteo en cero.

```
texto = "Esternocleidomastoideo"
for pos, letra in enumerate(texto):
    print(f"La letra en la posición {pos} es la {letra}")
```

zip()

- Permite unir varios iterables para utilizarlos dentro de la misma iteración.

```
prefijo = ['La','El','La','El']
frutas = ['manzana', 'platano','frutilla','tomate']
colores = ['verde','amarillo','roja','rojo']

for p, fruta, color in zip(prefijo, frutas, colores):
        print(f'{p} {fruta} es de color {color}')
```

break

Previamente, vimos que se puede terminar un ciclo while con una condición de salida. En el caso de un ciclo for, este termina cuando se recorre todo el iterable, pero ¿es posible terminar un ciclo for a propósito?

La respuesta es sí, pueden existir varios casos en que se requiera terminar el ciclo sin la necesidad de recorrer todos los elementos. Para hacerlo, se debe escribir la palabra **break**, y esta instrucción hace que el ciclo termine y se continúe con la ejecución del resto del programa.

Ejercicio guiado 1

"Búsqueda"

Búsqueda

Es muy común tener que buscar un elemento específico en la estructura contenedora. Uno podría almacenar el elemento buscado y terminar de recorrer los elementos sin uso de break, pero esto no tiene mucho sentido, ya que puede ser muy costoso el recorrer estructuras de datos muy grandes si es que ya se encontró lo que se busca.

En este caso buscaremos por un número y diremos en qué posición se encontraba.

Búsqueda

Solución

Paso 1

Creemos el archivo search.py

Paso 2

Ingresemos el número a buscar utilizando sys.argv

```
import sys
buscar = sys.argv[1] # número a buscar
```

Búsqueda

Solución

Paso 3

Busquemos en una lista de dígitos. Para poner un poco de dificultad podemos mezclar la lista, es importante considerar que acá se muestra un resultado, el cual puede variar según la mezcla que se realice.

```
import random
lista = [1,2,3,4,5,6,7,8,9,0]
# .shuffle de la librería random permite mezclar
# la lista de dígitos para aumentar un poco la dificultad.
random.shuffle(lista)
```

Búsqueda
Solución

Paso 4

Ahora, debemos implementar el algoritmo de búsqueda, el cual podemos conseguir mezclando instrucciones if y for:

```
# revisaremos cada elemento en la lista
# también llevamos registro de la posición en la que estamos
for position, elemento in enumerate(lista):

# Si el elemento es igual a lo que buscamos terminamos el ciclo
    if elemento == buscar:
        print("¡Elemento encontrado! Se terminará del ciclo")
              break
    else:
# Si es que no es el elemento buscado lo reportamos
        print("Elemento no encontrado")
```

Búsqueda

Solución

Paso 5

Finalmente, reportamos nuestros resultados:

```
print("Ha terminado el ciclo")
print(f'El elemento {buscar} se encontró en la posición {position}')
print(f'La lista mezclada es: {lista}')
```

/* Ciclos anidados */

Ciclo anidado

Es un ciclo dentro de otro ciclo, y no existe un límite explícito sobre cuántos ciclos pueden haber anidados dentro de un código, aunque por cada uno aumentará la complejidad.

Se debe tener en cuenta que la cantidad total de iteraciones será la multiplicación de cuántas iteraciones se haga en cada ciclo.

Dentro de las buenas prácticas de programación, se recomienda no usar más de 3 ciclos anidados.

Ejercicio guiado 2

"Escribiendo las tablas de multiplicar"

Escribiendo las tablas de multiplicar

Supongamos que queremos mostrar una tabla de multiplicar, por ejemplo la tabla del 5. Esto se puede escribir como:

```
for numero in range(10):
   print(f"5 * {numero} = {5 * numero}")
```

Ahora bien,
¿Cómo podríamos hacer para mostrar todas las tablas de multiplicar del 1 al 10?
Envolviendo el código anterior en otro ciclo que itere de 1 a 10.

Escribiendo las tablas de multiplicar

Solución

El ciclo más externo (numero1) será la tabla que nos interesa, mientras que el ciclo externo (numero2) serán todos los números que se irán multiplicando con el indicador de la tabla.

```
for numero1 in range(10):
    print(f'\nTabla del {numero1}:------------------------------\n')

    for numero2 in range(10):
        print(f"{numero1} * {numero2} = {numero1*numero2}")
```

¿Cuál es la diferencia entre un ciclo For un ciclo While?

¿Para qué sirve la función enumerate()?

{desafío} latam_

¿Para qué sirve la función zip()?

{desafío} latam_

¿Para qué sirve el operador break?

Próxima sesión...

- *Utiliza ciclos de instrucciones iterativas combinadas con sentencias if/else para resolver un problema acorde al lenguaje Python.*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales