Sentencias condicionales e iterativas

Python Comprehensions

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

{desafío} latam_

¿Qué son los ciclos?

{desafío}
latam_

Python Comprehensions

Un aspecto bastante único de Python son las llamadas **Python Comprehensions**, las que son una manera de reescribir un ciclo for en una sola línea.

Esta característica, única en este lenguaje, tiene la ventaja de reducir y compactar el código en el caso de operaciones cortas, lo cual es una característica propia de los programadores experimentados.

/* Transformando un ciclo for en un Comprehension */

Transformando un ciclo for en un Comprehension

Supongamos el siguiente problema: queremos generar los 10 primeros números pares y almacenarlos en una lista:

```
import sys
# solicitamos el número de pares a generar
n = 10

# generamos una lista vacía para almacenar los pares
lista_par = []

for i in range(n):
      # podemos hacer append de los valores generados
      # en este caso partimos desde el 2
      lista_par.append(2*i + 2)

# mostramos el resultado
print(lista_par)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Transformando un ciclo for en un Comprehension

Antes aplicar comprehension veamos la sintaxis a utilizar:

```
[fórmula for variable in iterable]
```

Esta sintaxis se lee de la siguiente manera:
"Empezamos por el ciclo, entonces, para cada variable en el iterable realiza la fórmula"

Transformando un ciclo for en un Comprehension

El código, en términos de comprehension se puede escribir de la siguiente manera:

```
lista_par = [2*i + 2 for i in range(n)]
print(lista_par)
```

```
[2, 4, 6, 8, 10, 12, 14, 16, 18, 20]
```

Se puede ver que en solo dos líneas se obtiene el mismo resultado (de hecho es posible dejarlo sólo en una línea); y lo que acabamos de ver es un List Comprehension, porque el resultado final se almacena en una lista.

Transformando un ciclo for en un Comprehension

Haciendo la lectura del código `[2*i + 2 for i in range(n)]` decimos entonces que:

- Para cada i en el rango de n (10) voy a guardar el resultado de la operación 2*i + 2
- En la primera iteración como i vale 0 se realizará la multiplicación 2*0 + 2
- En la siguiente iteración i vale 1 2*1 +2, en la siguiente i vale 2 2*2 + 2

/* Condicionales con List Comprehensions */

Condicionales con List Comprehensions

Existen otras versiones de un List Comprehension que permiten combinar Control de Flujo condicional con los ciclos for.

Por ejemplo, supongamos la siguiente lista de valores, donde queremos entregar una lista que nos diga si el número es divisible por dos o no.

```
valores = [0,4,5,6,7,8,9]
divisibles = []
for valor in valores:
     if valor % 2 == 0:
         divisibles.append('Divisible')
     else:
         divisibles.append('No Divisible')
```

Condicionales con List Comprehensions

List comprehensions tiene la siguiente sintaxis que equivale a lo mostrado anteriormente:

```
[expresión1 if condición1 else expresión2 for variable in iterable]
```

En este caso, si convertimos el código anterior en List comprehension queda de la siguiente manera:

```
['Divisible' if valor % 2 == 0 else 'No Divisible' for valor in valores ]
```

Condicionales con List Comprehensions

Ambos códigos devuelven el siguiente resultado:

```
['Divisible',
 'Divisible',
 'No Divisible',
 'Divisible',
 'No Divisible',
 'Divisible',
 'No Divisible']
```

Condicionales con List Comprehensions

¿Cómo se lee este código?

```
['Divisible' if valor % 2 == 0 else 'No Divisible' for valor in valores ]
```

Empezamos con el ciclo, entonces, para cada valor almacenado en nuestra lista de valores, vamos mostrar la palabra ‘Divisible’ si el valor módulo de 2 es igual a 0.

/* Operaciones de filtrado */

Operaciones de filtrado

Finalmente, los List Comprehensions pueden ser utilizados para filtrar datos de una lista. Por ejemplo, contemos la cantidad de datos de esta lista que son string:

```
lista = ['Lechugas', 'Tomates', 5, 10,
                   True, False, True, 'Papas',
                   5.1, 45.2, 1, 2, 0]
count_str = 0
for elemento in lista:
     if type(elemento) is str:
     count_str += 1

print(count_str)
```

En la prueba lógica se utiliza la palabra is, la que sirve para poder detectar si un cierto tipo de dato es el que se espera, en este caso si el tipo del elemento es un str (Un string)

Operaciones de filtrado

Es posible utilizar una tercera alternativa de sintaxis de los List Comprehensions:

```
[expresión for variable in iterable if condición ]
```

Esta expresión se aplicará sólo a los valores que cumplan la condición dada. Por ejemplo:

```
[elemento for elemento in lista if type(elemento) is str ]
```

Luego para contar el resultado podemos utilizar la función len(), de manera que el código completo se reduce a lo siguiente:

```
lst_str = [elemento for elemento in lista if type(elemento) is str ]
print(len(lst_str))
```

/* Dictionary Comprehensions */

Dictionary Comprehensions

Son la equivalencia en Diccionarios de los Python Comprehensions.

Recuerda que los diccionarios son estructuras clave-valor por lo que se debe considerar al momento de iterar.

```
claves = ['nombre','apellido','edad','altura']
valores = ['Juan','Pérez', 33, 1.75]
print({k:v for k,v in zip(claves, valores)})
```

```
{'nombre': 'Juan', 'apellido': 'Pérez', 'edad': 33, 'altura': 1.75}
```

Un dictionary comprehension seguirá exactamente las mismas reglas y estructuras que una List Comprehension pero utilizando llaves ({}) y : como separador entre la clave y el valor.

Normalmente, se utilizan las variables k para las claves (key) y v para los valores (value)

A continuación desarrollaremos en conjunto diversos ejercicios para poner en práctica lo aprendido

{desafío} latam_

Filtrado

A continuación, se muestra cómo crear un programa que filtre todos los números de una lista que sean menores a 1000. Esto es lo mismo que decir "seleccionar todos los elementos mayor o iguales a mil".

Transformarlo en un List Comprehension

```
a = [100, 200, 1000, 5000, 10000, 10, 5000]
n = len(a)
filtered_array = []

for i in range(n):
    if a[i] >= 1000:
        filtered_array.append(a[i])

print(filtered_array)
```

```
[1000, 5000, 10000, 5000]
```

Adicto a redes

Se tiene una lista con la cantidad de minutos usados en redes sociales de distintos usuarios.

```
[120, 50, 600, 30, 90, 10, 200, 0, 500]
```

Se debe retornar una lista que clasifique todos los tiempos menores a 90 minutos como 'bien' y todos los tiempos mayores o iguales a 90 como 'mal'.

El output debería ser algo similar a lo siguiente:

```
['mal', 'bien', 'mal', 'bien', 'bien', 'bien', 'mal', 'bien', 'mal']
```

Transformando segundos a minutos

Se tiene una lista con la cantidad de segundos que demoraron algunos procesos.

Se necesita una función para transformar todos los datos a minutos, (se requiere sólo la parte entera, la parte decimal se puede ignorar).

```
seconds = [100, 50, 1000, 5000, 1000, 500]
```

Países

Tenemos un listado de países y la cantidad de usuarios por cada país en la siguiente tabla:

País	Cantidad de usuarios
México	70
Chile	50
Argentina	55

1. Convertir la tabla mostrada en un diccionario.
2. Filtrar los países con menos de 60 usuarios.

Ventas

Dada la información de ventas de 3 meses:

Mes	Ventas
Octubre	65000
Noviembre	68000
Diciembre	72000

1. Convertir la tabla en un diccionario
2. Modificar el diccionario para incrementar las ventas en un 10%.
3. Hacer un nuevo diccionario pero con las ventas disminuidas un 20%.

¿Cuándo es posible utilizar Python Comprehensions?

Próxima sesión...

- *Desafío evaluado.*

{desafío}
latam_

{desafío}
latam_

Academia de
talentos digitales