DevCamp 2025 – Module 2 – Checkpoint 4: Questions
Date: 2025, Feb. 19th
Student: Alexandr Gomez.

1. ¿Cuál es la diferencia entre una lista y una tupla en Python?

Ambas son estructuras de datos, un tipo de dato primitivo de Python, designadas para contener una serie de items, incluso otras estructuras de datos anidadas, pero con unas claras diferencias:

- Sintáxis:

  La sintáxis típica de una lista usa corchetes [ ], conteniendo cada valor dentro de ellos, separados por coma.

  Ejemplo:

```
lista_1 = [
    'casa',
    'coche',
    'libro',
    'arbol'
]


lista_mixta_anidada_interpolada = [
    27,
    'Ordenador',
    f'{lista_1}',
    { 'colores_RGB' : ['rojo', 'verde', 'azul'] }
]
```

  Las tuplas, en cambio, utilizan paréntesis:

  Ejemplo:

```
tupla_ejemplo = (27,32,37.49,86)

tupla_multimixta_anidada_interpolada = (
    ["alfa", "beta", "gamma", "delta"],
    round(tupla_ejemplo[2]),
    "Indentación",
    { '+' : 'Sumar', '-' : 'Restar'},
    None,
    f'{tupla_ejemplo}',
    complex(0,2j),
    (5 * 22) % ( 3 - 22) // 3,
    lista_1.pop(),
    lista_mixta_anidada_interpolada.copy()[1]
)
```

- Inmutabilidad:

  Esta propiedad, en la que se basan las tuplas, impide poder editar el contenido de estas (no así reasignarlo a través de diversas técnicas, como la concatenación de nuevos items a una copia de la tupla original que después es renombrada).

  Las listas, en cambio, sí pueden modificadas directamente, por lo que su condición son de mutabilidad.

2. ¿Cuál es el orden de las operaciones?

En inglés, PEMDAS:

P - Paréntesis
E – Expontes / Raíces
M - Multiplicaciones
D - Divisiones
A - Sumas
S – Restas

Existe cierta controversia, en casi cualquier lenguaje de programación, respecto a si el orden de las operaciones debería anteponer las divisiones (PEDMAS) a las multiplicaciones (PEMDAS), incluso entre el orden de prioridad de restas y sumas, pero haciendo cálculos, no importaría qué orden seguir a ese respecto pues, los resultados son los mismos.

No obstante, se denominda/se utiliza PEMDAS.

3. ¿Qué es un diccionario Python?

Es otro tipo de dato primitivo, de estructura de datos, como también son Listas y Tuplas.

Su sintáxis, utiliza llaves { } y, a diferencia de las anteriores, contiene Claves y sus respectivos valores, separadas por un colon :.

Ejemplo:

```
logins = {
          'root' : 'mihiuhjkhbuih',
          'user1' : 'kasjdaokjdaosd'
}

print('The root pwd is: ' + list(logins.values())[0])


paletas_colores = {
          'dark' : 'black',
          'primavera' : ['verde', 'amarillo', 'cyan'],
          'ocaso' : ['naranja', 'ocre', 'teja']
}


print(list(paletas_colores.values())[1][2]) # cyan
print(list(paletas_colores.values())[0])   # black
```

4. ¿Cuál es la diferencia entre el método ordenado y la función de ordenación?

Función de Ordenación: .sort()
Método Ordenado: Sorted()

Ambas funciones pueden servir para ordenar una estructura de datos, una lista, alfabeticamente, o numericamente, pero tienen más usos y claras diferencias.

El método ordenado, si bien puede trabajar con tuplas además de listas, siempre devolverá una lista.

Puede ser usado para almacenar un valor de lista previo en una nueva variable, o ser directamente invocado.

Una característica curiosa es que, si pasamos una cadena como argumento, separará cada caracter de dicha como un nuevo item de la lista, ordenado.

```
lista = ['d', 'a', 'b', 'c']
print(sorted(lista))  #  ['a', 'b', 'c', 'd']

tupla = (3, 9, 8, 1)
sorted_tupla = sorted(tupla)
print(sorted_tupla)   # [1, 3, 8, 9]

new_lista = sorted('bad')
print(new_lista)      # [‘b’, ‘a’, ‘d’]
```

La Función de Ordenación, en cambio, sólo puede ser usada con listas, pues trabaja con un principio de inmutabilidad.

Su resultado no podrá ser almacenado en nuevas variables, sino únicamente ejecutado al invocarlo.

Además, como argumento podemos pedir que muestre el orden inverso.

```
lista = ['d', 'a', 'b', 'c']

lista.sort()
print(lista)    # ['a', 'b', 'c', 'd']

lista.sort(reverse=True)
print(lista)    # ['d', 'c', 'b', 'a']
```

5. ¿Qué es un operador de reasignación?

Son operadores preferentemente deseados para trabajar con tipos de datos inmutables, como pueden ser las cadenas o las tuplas.

Funcionan reasignando la operación , según el operador usado, sobre el operando indicado.

Su otra mejor función, en general, es hacer más fácil la mutación de valores sobre sí mismos, escribir código más claro, rápido y organizado.

Ejemplos:

```
number = 37
number += 1 # Igual que escribir number = number + 1
print(number) #  38



# En reasignación de elementos

tupla_reasignacion = (1,2,3,4)
tupla_reasignacion += (5,)
print(tupla_reasignacion)    # (1,2,3,4,5)
```