M2C4 Python Assignment

During this section of Module 2, you have learned more about Python. Python is a powerful programming language that serves a lot of purposes. To practice what you have learned in this section, you will complete some Python exercises. You may use Visual Studio Code, Repl.it, or another text editor/environment of your choice. Please complete the following assignment and reach out on the Support App to have a mentor review your work. If you have any questions or need any help, please reach out so we can help you! This assignment must be completed to pass this section of the coursework.

Exercise 1: Create a list, tuple, float, integer, decimal, and dictionary.

1. **List.** El ejemplo siguiente es una lista con distintos tipos de valores (integer, string, list, boolean, tupla anidada con lista). Es para mostrar que una lista **no** tiene por qué contener siempre el mismo tipo de valores, incluso puede tener tipos anidados. Aunque esto no suela ser lo más frecuente, es posible crear estructuras con muchos niveles de anidación, creando listas extraordinariamente complejas. En todo caso, salvo que sea imprescindible, no es recomendable crear estructuras de datos innecesariamente complejas, ya que esto va en contra de la legibilidad del código: debe tenerse en cuenta el principio de programación KISS (Keep It Simple, Stupid) Las listas son estructuras MUTABLES, es decir pueden ser modificadas y disponen de innumerables funciones y métodos que facilitan estas modificaciones.

```
lista_ejemplo = [360, 'Iñaki', ('lunes', 'martes', 'miércoles'), True, False, [14, 20, 35, ('Juan', 'Pedro')]]
```

2. **Tuple.** En las tuplas podemos definir estructuras tan complejas como en las listas, ya que son estructuralmente muy similares, aunque al ser una estructura INMUTABLE, debiera usarse con datos que no cambian a lo largo de la aplicación, o que varían muy poco. Debido a esto, no tiene asociadas tantas funciones y métodos como las listas, ya que no están diseñadas para ser modificadas.

```
tuple_ejemplo = ('alfa', 'beta', 'gamma')
```

muchos3. **Float.** Los float son tipos numéricos built-in de baja precisión, es decir no se contemplan en su diseño con muchos decimales. Si requerimos de cálculos con gran precisión (como en proyectos científicos, económicos,...) debemos usar el tipo 'Decimal' (Decimal(numero)) de la librería 'decimal', al no ser un tipo bulit-in como es float.

```
float_ejemplo = 12.45
```

4. **Integer.** Tipo built-in para números enteros.

```
integer_ejemplo = 24
```

5. **Decimal.** Los decimal son nºnúmeros de coma flotante de alta precisión, usados en cálculos matemáticos precisos.

```
from decimal import Decimal  # Al ser una librería no built-in, debemos importarla antes de usar

decimal_ejemplo = Decimal(236.56)
```

6. **Dictionary.** Los diccionarios son tipos de datos, que nos permiten definir pares clave-valor, pudiendo también crear estructuras anidadas tan complejas como queramos. Pero, como siempre, debemos aplicar el criterio KISS.

```
dictionary_ejemplo = { 'Nombre': 'Jon', 'Apellido': 'Ugartemendia', 'Edad': 26, 'Dirección': {'Calle': 'Plaza Otaegi', 'Número': 14, 'Piso': 1, 'Puerta': 'B' } }
```

Exercise 2: Round your float up.

```
import math  #Importamos la librería math para usar sus funciones matemáticas, en este caso ceil()

float_ejemplo = 45.15
redondeado = math.ceil(float_ejemplo)
print(redondeado)    # Salida: 46
```

Exercise 3: Get the square root of your float.

```
import math  #Importamos la librería math para usar sus funciones matemáticas, en este caso sqrt()

float_ejemplo = 45.15
print(math.sqrt(float_ejemplo))    # Salida: 6.7193749709329
```

Exercise 4: Select the first element from your dictionary.

```
dictionary_ejemplo = { 'Nombre': 'Jon', 'Apellido': 'Ugartemendia', 'Edad': 26, 'Dirección': {'Calle': 'Plaza Otaegi', 'Número': 14, 'Piso': 1, 'Puerta': 'B' } }

print(dictionary_ejemplo['Nombre'])  # Salida: Jon
```

Exercise 5: Select the second element from your tuple.

```
tuple_ejemplo = ('alfa', 'beta', 'gamma')
print(tuple_ejemplo[1])  # Salida: beta
```

Exercise 6: Add an element to the end of your list.

```
lista_ejemplo = [360, 'Iñaki', ('lunes', 'martes', 'miércoles'), True, False, [14, 20, 35, ('Juan', 'Pedro')], 'Pepe']
lista_ejemplo.append('Primavera')  # Añadimos un nuevo elemento a la lista
print(lista_ejemplo)  # Salida: [360, 'Iñaki', ('lunes', 'martes', 'miércoles'), True, False, [14, 20, 35, ('Juan', 'Pedro')], 'Pepe', 'Primavera']
```

Exercise 7: Replace the first element in your list.

```
lista_ejemplo = [360, 'Iñaki', ('lunes', 'martes', 'miércoles'), True, False, [14, 20, 35, ('Juan', 'Pedro')]]
lista_ejemplo[0] = 180
print(lista_ejemplo) # Salida: [180, 'Iñaki', ('lunes', 'martes', 'miércoles'), True, False, [14, 20, 35, ('Juan', 'Pedro')]]
```

Exercise 8: Sort your list alphabetically.

```
'''
He tenido que crear una nueva lista_ejemplo porque la original mezclaba diferentes tipos de valores y eso genera un error con sort()
'''

lista_ejemplo = ['lunes', 'martes', 'miércoles', 'jueves', 'viernes', 'sábado', 'domingo']
lista_ejemplo.sort()  # Ordena la lista
print(lista_ejemplo)  # Imprime: ['domingo', 'jueves', 'lunes', 'martes', 'miércoles', 'sábado', 'viernes']
```

Exercise 9: Use reassignment to add an element to your tuple.

```
tuple_ejemplo = ('alfa', 'beta', 'gamma')
tuple_ejemplo += ('delta',)
print(tuple_ejemplo)  # Imprime: ('alfa', 'beta', 'gamma', 'delta')
```