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.

Exercise 2: Round your float up.

Exercise 3: Get the square root of your float.

Exercise 4: Select the first element from your dictionary.

Exercise 5: Select the second element from your tuple.

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

Exercise 7: Replace the first element in your list.

Exercise 8: Sort your list alphabetically.

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

Primeramente debes realizar unos ejercicios en Python, y luego responder algunas preguntas teóricas, necesito que me las respondas con tus propias palabras, agregando ejemplos, buenas prácticas, y todo lo que has aprendido.

M2C4. 5 preguntas teóricas

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

List muy similar a una Array en Ruby y Javascript.

Puede contener number, string u otra list.

Buena práctica mantener los list data types lo más uniformes que se pueda y así poder tratar a cada tipo de la misma manera. Potencialmente muy peligroso porque puede romper tu programa y dar error al haber mezclas de data types.

Collection de values y esa Collection se puede añadir también. Puedes quitar items. Query elements dentro de ella.

Mutable puedes hacerles cualquier cosa que quieras a una de estas collections.

Sintaxis: usa corchetes **[ ]** para enmarcar su contenido**.**

users = ['Kristine', 'Lisa', 'Jordan'] ===>['Kristin
e', 'Lisa', 'Jordan']

lo que devuelve ya no son serie de strings sino un verdadero data structure, lista de string names qué es lo que se obtendría si hacemos una query o consulta a un database con users.

users**.insert**(0, 'Anthony') ===> [**'Anthony',** 'Kristine', 'Lisa', 'Jordan'**]**

añade 'Anthony' en el orden del **index** 0 en este caso.

users**.append**('Ian') ===> ['Kristine', 'Lisa', 'Jordan', **'Ian']**

añade 'Ian' y lo coloca al **final** de la lista. Muy común si no te importa o necesitas el orden y lugar de cada elemento de la lista.

print(users**[2]**) ===>**Lisa** devuelve string sin corchetes

print(**[**users**[2]]**) ===> [**Lisa**] si necesitas una lista con corchetes. List object y sólo puedes llamar las funciones propias disponibles de en la list data type.

Reasignar o editar uno de los valores. Sustituye Ian por Brayden, pero ojo, sin cambiar el string porque son immutable, solo **reasigna** el elemento

users **[4]** = **'Brayden'** ===> ['Kristine', 'Lisa', 'Jordan', **'Brayden']**

3 maneras de quitar elementos de una lista en python:

1) users**.remove**('Jordan')
2) Popped_user = users**.pop()**
3) **del** users**[0]**

3 maneras de quitar elementos de una lista en **tuple:**

1) post = post**[:-1]** desde el **final** del tuple con **rangos**
2) post = post**[1:]** desde el **principio** del tuple con **rangos**
3) Messy - **No recomendado** por si no vuelves a cambiarlo de **list a tuple otra vez ya que puede crear bugs:**

   Para cuando se necesita quitar de la zona del **medio:**

   post = **(** 'Python Basics', 'Intro guide to Python', 'Some cool python content', 'published'**)**

   post = list(post)

   post.remove('published')

   post = tuple(post)

Método orden .**sort()** en **lista** y **diferencia** en a **tuple:**

Lista:

post = **[** 'Python Basics', 'Intro guide to Python', 'Some cool python content'**]**

post.**sort()** ===> **['Intro guide to Python', 'Python Basics',** 'Some cool python content'**]**

title, sub_heading,content = post

lo ordena de forma **alfabética** así que **cambia el orden respecto a tuple**

Tuple:

post = ('Python Basics', 'Intro guide to Python', 'Some cool python content')

post.**sort()** ===>**('Python Basics', 'Intro guide to Python',** 'Some cool python content')

title, sub_heading,content = post

sort() cambia a **tuple** y devuelve que **no puede ordenar** tuple. **Atribute Error,** si realizas **Unpacking** siempre vas a tener title mapeado como el 1er elemento, sub_heading como el 2º y content el 3º.

Tuple muy similar a una **lista** con varias características diferencias.

Unpacking es una de las mejores razones por las que se usan los tuples, como vemos a continuación:

post = ('Python Basics', 'Intro guide to Python', 'Some cool python content')

title, sub_heading,content = post

que equivale a estas 3 líneas donde cada uno de los items hace de máquina de consulta (engine query):

title = post[0]

sub_heading= post[1]

content = post[2]

Tuples puedes consultar con index específico, unpack y acceder, pero necesitas y te permiten almacenar una serie de collection de data. Tuple no se pueden cambiar pero una list sí.

Accedemos a los elementos de la misma manera en este caso tuple a lo que conseguimos con una lista básica. **Unpacking en los 2 casos (list y tuple)**

El selector básico **post[0]** usa el index y almacena dentro de la variable.

Tuple es Immutable data structure que **no cambia y quiero mantener igual** y es la razón por la que elegir en contraposición a la **lista es mutable** que **sí cambia.**

Sintaxis: usa paréntesis **( )** para enmarcar su contenido.

2. ¿Cuál es el orden de las operaciones?

El orden de esta regla mnemotécnica 'PEMDAS': Paréntesis, Exponents, Multiplication, División, Addition (Suma) y Subtraction (Resta). Ejemplo del proceso según el orden:

8+2*5-(9+2)**2

8+2*5-11**2

8+2*5-121

8+10-121

= -103

3. ¿Qué es un diccionario Python?

Se llama **Key-value data store (Almacén de datos llave-valor)** que significa que podemos almacenarlo en una variable, podemos crear no solo elements como en las lists sino que podemos crear una key(llave) con su value(valor) correspondiente.

El nombre diccionario viene del mismo concepto con el que se usa el diccionario analógico, para buscar una palabras, primero usas alguna representación de una letra y de ahí vas a la palabra que necesitas. La letra sería la key y la palabra el value. Importante que estamos trabajando con palabras, no con indexes.

Sintaxis: usan llaves **{}** para enmarcar su contenido. Como es bastante largo el tipo de estructura normalmente se coloca en varias líneas**.** Ejemplo: ss es la llave y Correa el value

players = **{**
“ss” **:** “Correa”
“2b” **:** “Altuve”

“3b” **:** “Bregman”
}
print(players) ===> {‘ss’ **:** ‘Correa’, ‘2b’ **:** ‘Altuve’, ‘3b’ **:** ‘Bregman’ }

Para **consultas(query) los values**, similar a cuando lo hacemos en una list:

second_base = players[‘2b’]
print(second_base) ===> Altuve

Si no existe o es errónea la key introducida devuelve key Error indicando cuál es el problema.
second_base = players[‘asfg’] ===> key Error

third_base = players[‘3**B**’]
print(third_base) ===> key Error **CaseSensitive**

asegurarse que es idéntica a **3b** en minúsculas para que funcione ya que es sensible a la cajas

Value puede ser sencilla o una **colección de values,** puede contener **cualquier tipo de estructura de datos (list, diccionary, number, tuple, string…)**

teams = **{**

“astros” **: [** “Altuve”, “Correa”, “Bregman”**]**
}
print(astros) ===> **[** ‘Altuve’, ‘Correa’, ‘Bregman’**]**

Podemos **almacenarlos** y conseguir que funcione igual, en una **variable:**

astros = teams[‘astros’]
print(astros) ===> **[** ‘Altuve’, ‘Correa’, ‘Bregman’**]**

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

sort() y **sorted()**

Método orden en **lista:**

Sintaxis: sort()

post **= [** ‘**P**ython Basics’, ‘**I**ntro guide to Python’, ‘**S**ome cool python content’**]**

post.**sort() ===> [‘I**ntro guide to Python’, **‘P**ython Basics’, ‘**S**ome cool python content’**]**

title, sub_heading,content = post

lo ordena de forma **alfabética** así que **cambia el orden** respecto a por **defecto que es el del index.**

post.**sort(reverse=True)** ===> **[‘S**ome cool python content’, **‘P**ython Basics’, **‘I**ntro guide to Python’ **]**

Lo ordena de la **Z a la A al contrario de alfabético**

De forma **análoga** ocurre con los **números (integers)** donde **el orden alfabético** es de **menor a mayor número** y el **contrario al alfabético de mayor a menor número.**

Función de **ordenación** en **lista:**

sale_prices = [100, 83, 220, 40, 100, 400, 10,1,3]
sale_prices.sort()
print(sale_prices)
nueva variable sorted_list
sorted_list = sale_prices.**sort()**
print(sorted_list)**===> None.** Ya que **sort() no devuelve un valor en nueva variable.**

Qué hacer si queremos que nos devuelva un valor?

usamos sorted() te deja guardarlo dentro de la una variable diferente.
El método sort() parecido comportamiento pero te deja almacenarlo ese valor.

Sintaxis: sorted() y se llama por delante:

sorted(sale_prices)

Se usa cuando **solo** quieres **cambiar el orden** de una lista **sin cambiar la lista original.**

sorted_list = sorted(sale_prices)

print(sorted_list)

print(sales_prices) no se altera está intacta la original

Devuelve de **menor a mayor número**

sorted_list = sorted(sale_prices, **reverse=True)**

print(sorted_list)

Devuelve de **mayor a menor número**

5. ¿Qué es un operador de reasignación?

Como el tuple es immutable y no se puede cambiar directamente se usa el operador de reasignación para poder crear un nuevo tuple.

Sintaxis:

+=
Equivale a: **post +**

Si queremos añadir, a posteriori, 'status' de manera dinámica con 'published' en un Blog Post:

post = ('Python Basics', 'Intro guide to Python', 'Some cool python content')

title, sub_heading, content, status = post

La coma final si no se pone, python asumiría que estamos tratando de sobreescribir el orden de operaciones matemáticas y necesita que sean 2 tuples, no vale string y tuple**:**
post **+=** ('published'**,**)
Equivale a:
post = **post +** ('published'**,**)