M2C4 PAYTHON Task

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

Una lista y una tupla se diferencian principalmente en la posibilidad de modificarlas, y en su sintaxis.
Los elementos de las listas se recogen entre corchetes [], y se separan por comas. Los elementos de una tupla se encierran entre paréntesis (), y se separan por comas.
Las listas son conjuntos de datos asignados a una variable que se pueden mutar, es decir, que podemos actuar sobre ellas y las podemos modificar, por ejemplo, en un listado de asistencia, dónde podemos modificar a los asistentes:

- Añadir elementos:

    - append(): Agrega un elemento al final de la lista.

```
colors = ['red', 'blue', 'White']
colors.append('Brown')

print(colors)
#La salida es:    ['red','blue','white','Brown']
```

    - insert(): Inserta un elemento en la posición seleccionada de la lista.

```
colors = ['red', 'blue', 'white']
colors.insert(1, "brown")

print(colors)
#La salida es:
['red', 'brown', 'blue', 'white']
```

- extend(): añade elementos de otra lista al final de nuestra lista.

```
colors = ['red', 'White']
colors.extend(['blue', 'Brown'])

print(colors)
#La salida es:
[“red”, “White”, “blue”, “Brown”]
```

- Eliminar elementos:

  - remove(): Elimina el elemento seleccionado de la lista.

```
colors = ["red", "blue", "white"]
colors.remove("blue")

print(colors)
#La salida es: [‘red’,’white]
```

  - pop(): Elimina el elemento de la lista seleccionado con un índice; si no existe, Python nos devolverá un error .Si no se indica nada (), recoge el último elemento que podemos asignar a una nueva variable. También se puede utilizar en un bucle, por ejemplo, para una lista de envío.

```
colors = ["red", "blue", "white"]
colors.pop(1)

print(colors)
#La salida es: [‘red’, ’white’]

new_colors = colors.pop()

print(new_colors)
#La salida es: White
```

  - del: (delete) Borra el elemento de la lista con el índice indicado.

```
colors = [‘red’, ‘blue’, ‘white’]
del colors[0]

print(colors)
#La salida es: [‘blue’, ’white’]
```

- Reorganizar los elementos:

    - sort(): Organiza los elementos de la lista de la A a la Z, o de mayor a menor. Sort(reverse=true) opera de la Z a la A, y de menor a mayor.

```
colors = ["red", "blue", "white"]
colors.sort()

print(colors)
#La salida es: ['blue', 'red', 'white']
```

    - reverse(): Invierte los elementos de la lista; no los organiza, solo invierte su orden inicial.

```
colors = [“red”, “blue”, “White”]
colors.reverse()

print(colors)
#La salida es: [’white’, ‘blue’, ‘red’]
```

- Modificar los elementos:

    - Puede añadir un elemento a una lista indicando el índice a sustituir y el nuevo elemento.

```
colors = ['red', 'blue', 'white']
colors[-1] = "green"

print(colors)
#La salida es: [’red’, ‘blue’, ‘green’]
```

    - Puede añadir un elemento y guardarlo en una nueva lista.

```
colors = ['red', 'blue', 'White']
new_colors = colors + ['black']

print(new_colors)
#La salida es: ['red', 'blue', 'White', 'black']
```

    - replace(): Reemplazar letras específicas.

```
colors = ['red', 'blue', 'white']
new_colors = [letter.replace('e', 'o') for   letter in colors]

print(new_colors)
# La salida es: ['rod', 'bluo', 'whito']
```

Las tuplas se pueden usar para elementos únicos que no necesitan ser actualizados o modificados. Al ser inmutables, una vez que se creada, sus elementos no se pueden modificar directamente.

- Agregar elementos por reasignación:

  Consiste en unir dos tuplas en una nueva; Unir dos objetos en una nueva variable. Se debe escribir entre paréntesis () y con una coma para que Python la interprete como una tupla.

```
colors = ('red', 'blue', 'white')
colors += ('green',)

print(colors)
 #La salida es: ('red', 'blue', 'white', 'green')
```

- Convertir a lista:

  Podemos convertir la tupla en un alista, hacer las modificaciones, y volver a convertirla en tupla.

```
colors = ('red', 'blue', 'white')
colors = list(colors)

print(colors)
#La salida es una lista:['red', 'blue', 'white']

colors[1] = 'green'
 colors.append("Brown")
colors = tuple(colors)

print(colors)
#La salida vuelve a ser una tupla: ('red','blue', 'white', 'green')
```

- Tuplas anidadas:

  Podemos trabajar con tuplas que tienen anidados elementos que se pueden modificar (listas)

```
colors = ('red', ['brow', 'black'], 'blue',       'white')
colors[1][0] = 'green'

 print(colors)
#La salida es: ('red', ['green', 'black'],      'blue', 'white')
```

- Eliminar la tupla:

  Podemos eliminar la tupla al completo, no por partes:

```
colors = ('red', 'blue', 'white')
del colors

print(colors)
#La salida es: "NameError: name 'colors' is not defined"
```

¿Cuál es el orden de las operaciones?

En Python, el orden de las operaciones sigue las reglas matemáticas de operaciones. Para recordar este orden, se utiliza el acrónimo PEMDAS. Esto es:

- **P:** Paréntesis ()
- **E:** Exponentes ^
- **M**: Multiplicación **
- **D**: División /
- **A**: Adición +
- **S**: Sustracción –

```
operation = ((22*5-4)+3**2)-15
    #((22*5-4)+9)))
    #((110-4)+9))
    #(106)+9))
    #(115)-15

print(operation)
    # La salida es : 100
```

¿Qué es un diccionario Python?

Un diccionario en Python es un almacén de datos estructurados en pares de clave-valor. A cada clave le corresponde un valor, y se representan entre llaves, y cada par separado por coma.

```
colors_list ={
  "fence" : "red",
  "floor" : "bronw",
  "ceiling" : "white",
  "wall" : "blue"
}
```

No trabaja con índices como las listas, si no que trabaja con las palabras clave, con las que llamas a su valor. Si esta palabra clave no está en el diccionario, el sistema nos devolverá un error indicando la línea del código en el que está y la palabra clave a la intentábamos acceder.

```
colors_list ={
  "fence" : "red",
  "floor" : "brown",
  "ceiling" : "white",
  "wall" : "blue"
}

print(colors_list)
# La salida es: {'fence': 'red', 'floor': 'brown', 'ceiling': 'white', 'wall': 'blue'}

print(colors_list["fence"])
# La salida es: red

print(colors_list["window"])
 # La salida es un error porque no existe esta clave en el diccionario: Traceback
(most recent call last):
  File "/home/runner/workspace/main.py", line 134, in <module>
    print(colors_list["window"])
```

Las claves son elementos inmutables (no se pueden cambiar), son sensibles a las mayúsculas/minúsculas y pueden estar formados por varias palabras. Pueden ser todo tipo de números, cadenas y tuplas. El valor puede ser de cualquier tipo. Si el valor es una lista, se puede operar con ella como una lista normal.

```
colors_list = {
  ("floor", "fence") : 1,
  2 : "bronw",
  "ceiling" : ["white", "red"],
  2.5 : ("blue", "green")
 }

print(colors_list)
    # La salida es: {('floor', 'fence'): 1, 2: 'bronw', 'ceiling': ['white', 'red'], 2.5: ('blue',
    'green')}

print(colors_list["ceiling"][0])
    # La salida es: White
```

Los diccionarios también pueden estar anidados en otros diccionarios.

```
colors_list = {
  "house" : {
    "floor" : "brown",
    "ceiling" : "white",
    "wall" : "blue",
  },
  "garage" : {
    "fence" : "red",
    "floor" : "black",
    "wall" : "brown",
  }
}

print(colors_list)
# La salida es: {'house': {'floor': 'brown', 'ceiling': 'white', 'wall': 'blue'}, 'garage':
{'fence': 'Red', 'floor': 'black', 'wall': 'brown'}}
print(colors_list["house"])
# La salida es: {'floor': 'brown', 'ceiling': 'white', 'wall': 'blue'}
print(colors_list["garage"]["fence"])
# La salida es: red
```

OBJETOS DE VISTA DE DICCIONARIO:

A un diccionario de Python se le puede añadir nuevos pares (clave-valor); recoger todas las claves del diccionario (*.keys() -> dick-keys*); recoger los valores en una lista (*.values() ->dick-values);* recoger los pares clave-valor como tuplas (*.items() ->dick-items*); podemos generar un alista de los valores, y que se puede trabajar con ella como una lista real (*list(diccionario.values())*)

```
colors_list = {
    "floor" : "brown",
    "ceiling" : "white",
    "wall" : "blue",
}

colors_list["window"] = "green"

print(colors_list)
# La salida añade una nueva clave-valor al diccionario: {'floor': 'brown', 'ceiling': 'white',
'wall': 'blue', 'window': 'green'}

print(colors_list.keys())
# la salida es: dict_keys(['floor', 'ceiling', 'wall', 'window'])

print(colors_list.values())
# La salida es: dict_values(['brown', 'white', 'blue', 'green'])

print(colors_list.items())
# La salida es: dict_items([('floor', 'brown'), ('ceiling', 'white'), ('wall', 'blue'), ('window',
'green')])

colors_house = list(colors_list.values())
colored_house = colors_house[0]

print(colored_house)
# La salida es: brown
```

También podemos eliminar elementos de un diccionario conociendo la clave(*del clave[valor]*); si no la clave no está en el diccionario, el sistema nos devolverá un error. Podemos guardar el valor que queremos quitar en una variable nueva (*.pop()*)

```
colors_list = {
    "floor" : "brown",
    "ceiling" : "white",
    "wall" : "blue",
}

del colors_list["floor"]

print(colors_list)
# Borra el elemento del diccionario selecionado junto a su valor: {'ceiling': 'white', 'wall':
'blue'}

color_ceiling = colors_list.pop("ceiling")
```

```
print(color_ceiling)
# Borra el elemento del diccionario selecionado, y recogemos su valor en una nueva variable: white

print(colors_list)
# En el dicionario solo queda: {'wall': 'blue'}
```

Cuando intentamos buscar un valor en un diccionario y no existe, nos devuelve un error. Este error lo podemos configurar para que nos retorne el aviso que queramos. Esto se realiza con la función .get()

```
colors_list = {
    "floor" : "brown",
    "ceiling" : "white",
    "wall" : "blue",
}

color_fence = colors_list.pop("fence")

print(color_fence)
# ERROR:  Traceback (most recent call last): File "/home/runner/workspace/main.py", line 222, in <module> color_ceiling = colors_list.pop("fence")

color_fence = colors_list.get("fence", "No fence found")

print(color_fence)
# Como "fence" no está en el diccionario, la función get() devuelve el valor: "No fence found"
```

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

Ambas se utilizan para ordenar elementos. El método ordenado se refiere al método *.sort()* que se usa únicamente con listas, y las modifica creando una nueva lista en su lugar con sus valores ordenados.

```
color_list = ["red", "white", "blue", "black"]
new_color_list = color_list.sort()
# Aquí, el método ordena la lista, pero retorna nada

print(new_color_list)
# Si imprimimos esta lista, el resultado es: None

print(color_list)
# Si imprimos la lista original, se ha modificado, y nos retorna los valores ordenados:
['black', 'blue', 'red', 'white']
```

Mientras que la función ordenada se refiere a la función *sorted()* que se puede utilizar con listas, diccionarios, tuplas, y lo que hace es retornar una nueva lista ordenada, sin modificar el original.

```
color_list = ["red", "white", "blue", "black"]

color_list_sorted = sorted(color_list)

# Ordena la lista y retorna una nueva lista


print(color_list_sorted)

# la lista ordenada: ['black', 'blue', 'red', 'white']

print(color_list)

# la lista original: ['red', 'white', 'blue', 'black']
```

¿Qué es un operador de reasignación?

Un operador de reasignación realiza una operación sobre una variable y reasigna el resultado a la misma variable, actualizando su valor después de la operación. Realiza la operación matemática o lógica con la misma variable. También permite que la operación se refleje en el código de una manera más sencilla.

```
number = 3
number += 2
# Es igual que escribir: number = number + 2

print(number)
# La salida es: 5

number = 10
number *= 2
#Es igual que escribir: number = number * 2

print(number)
# La salida es: 20

number = 10
number **= 2
#Es igual que escribir: number = number * number

print(number)
# La salida es: 100

colors_house = ["red", "white"]
colors_house += ["blue", "black"]
# Es igual que escribir: colors_house = colors_house + ["blue", "black"]


print(colors_house)
# La salida es: ['red', 'white', 'blue', 'black']
```