Tuple methods

Tuples:-

- Tuple is similar to list (It is ordered collection of objects) but main difference is immutable.

⇒ creating a tuple:-

```
tup1 = (2, "Hello", "Python")
print (tup1)
```

⇒ Tuple characteristics:-

- ordered
- immutable
- Allow duplicates.

⇒ Access Tuple Items:-

- Index value starts with zero (0)

Ex:-

```
Lang = ("Python", 'C', "C++")
Print (lang[0])  #Python
Print (lang[2])  #C++
```

⇒ Tuples cannot be modified:-

It will get an error.

```
Lang[2] = "Java";  # error
```

⇒ Iterate through a tuple:-

```
fruits = ("apple", "banana", "orange")
for fruit in fruits:
    print(fruit).
```

Python Tuple Methods :-

1. count() :- it returns the number of times the specified element appears in the tuple

Ex :-

```
vowels = ('a', 'e', 'i', 'o', 'i', 'u')
count = vowels.count('i')
print(count)  #2
```

2. Index() :- returns the index of specified element in the tuple.

Ex :-

```
index = vowels.index('e')
print(index)  #1
```

Nested Tuples :-

Nested tuples are tuples that contain other tuples as their element. They can be thought as a multidimensional tuples.

Characteristics of Nested Tuples :-

- Immutability
- Accessing Elements
- Fixed structure.

Ex :-

```
nested = (
        (1, 2, 3),
        (4, 5, 6),
        (7, 8, 9)
    )

#Accessing Nested Tuples.
print(nested[0][0])  #1
```

Find a num in Matrix :-

```
matrix = [(1,2,3),(4,5,6),(2,8,9)]
num = 7
exists = False
for i in matrix:
    for j in i:
        if (j == num):
            exists = True

if (exists):
    print("True")
else:
    print("False")
```

Dictionary Methods

Dictionagi

- Stores the element in key value pairs.
- they are enclosed in {} curly braces / Hauge braces.
- key should be always string
- we can store multiple data types in dict.

Example :-

```
thisdict = {
        "brand": "Ford",
        "model": "Mustang",
        "year": 1964
    }
print(thisdict)
```

=> Dictionary Item :-

- dictionary items are ordered, changable, and do not allow duplicates.

Ex:-

```
thisdict = {
        "brand" : "Ford",
        "Model" : "Mustang",
        "year" : 1964
    }
print ( thisdict ["brand"])
```

O/P :- Ford.

=> changable

```
thisdict ["year"] = 1965
```

=> Duplicates not allowed

It automatically removes past element.

=> Dictionary length:-

```
print (len (thisdict))   #3
```

=> Accessing Items :-

```
thisdict = {
        "brand" : "Ford",
        "Model" : "Mustang",
        "year" : "1964"
    }
x = thisdict ["Model"]
(or)
x = thisdict.get ("Model")
# Mustang
```

=> Get keys :-

it returns list of keys in the dictionary

Ex:- x = thisdict.keys()

dict_keys (['brand', 'Model', 'year'])

=> Get values :-

It returns a list of all the values in the dictionary

Ex:-

x = thisdict.values()

dict_values (['Ford', 'Mustang', 1964])

=> Get Items :-

It returns each item in a dictionary, as tuples in a list.

Ex:-

x = thisdict.items()

dict_items ([('Brand', 'Ford'), ('Model', 'Mustang'), ('year', 1964)])

=> Update Ditionary :-

The update() method will update the dictionary with the items from the given argument.

Ex:-

```
thisdict = {
        "brand": "Ford",
        "Model": "Mustang",
        "year": 1964
    }
this dict.update ({"year": 2020})
```

Adding Items:-

Ex:- thisdict["color"] = "Red"

Removing Items:-

They are several methods to remove items from a dictionary.

Ex-1:- thisdict.pop("Model")

Ex-2:- thisdict.popItem("Model")

Ex-3:- del thisdict["Model"]

Ex-4:- del thisdict #delete the dictionary completely.

clear():- method empties the dictionary

Ex:- thisdict.clear()

{}

Loop through a dictionary:-

- you can loop through a dictionary by using a for loop.
- when looping through a dictionary, the default value are the keys of the dictionary, but there are methods to return values as well.

Ex:-

```
for x in thisdict:
    print(x)
    # Brand Model Year
```

Ex-2:-

```
for x in thisdict:
    print(thisdict[x])
# Ford Mustang 1964
```