Collections

There are four types of collections in Python.

1. List

List is a collection which is ordered and can be changed. Lists are specified in square brackets.

Example

```
mylist=["iPhone","Pixel","Samsung"]
print(mylist) # prints ['iPhone', 'Pixel', 'Samsung']
```

How to access List items?

You can access the list items similar to arrays using Indexes. Indexes starts with 0.

```
mylist=["iPhone","Pixel","Samsung"]
print(mylist[0]) # prints iPhone
print(mylist[7]) # throws IndexError : list index out of range
print(mylist[-1]) # prints Samsung
```

Note:

You can specify negative indexes if you want to start the search from the end of the list.

How to print range of Indexes

Just like accessing a single element in the list, you can also specify the range of indexes.

```
mylist = ["iPhone", "Pixel","Samsung", "Oppo", "Vivo", "Redmi", "Nokia" ]

print(mylist[1:5]) # Prints ['Pixel', 'Samsung', 'Oppo', 'Vivo']
```

How to change an item in the list

You can change the specific item by it's index number like below:

```
mylist = ["iPhone", "Pixel", "Samsung", "Nokia" ]
mylist[1] = "OnePlus"

print(mylist) # prints ['iPhone', 'OnePlus', 'Samsung', 'Nokia']
```

2. Tuple

Tuple is a collection which is ordered and can not be changed. Tuples are specified in round brackets.

Example

```
myTuple=("iPhone", "Pixel", "Samsung")
print(myTuple) # Prints ('iPhone', 'Pixel', 'Samsung')
```

How to access Tuple items

You can access the Tuple items similar to Lists or Arrays using Indexes. Indexes starts with 0.

```
myTuple = ["iPhone","Pixel","Samsung"]
print(myTuple[0]) # prints iPhone
print(myTuple[7]) # throws IndexError: tuple index out of range
print(myTuple[-1]) # prints Samsung
```

Note:

You can specify negative indexes if you want to start the search from the end of the list.

How to print range of Indexes

Just like accessing a single element in the tuple, you can also specify the range of indexes.

```
myTuple = ("iPhone", "Pixel", "Samsung", "Oppo", "Vivo", "Redmi", "Nokia" )

print(myTuple[1:5]) # Prints ('Pixel', 'Samsung', 'Oppo', 'Vivo')
```

How to change an item in the tuple

You can't change the item in a tuple directly. Below throws an error if you assign another value to tuple again.

```
myTuple = ("iPhone", "Pixel", "Samsung")
myTuple[1] = "onePlus" # throws error as TypeError: 'tuple' object does not support item assi
print(myTuple)
```

If you want to change a specific item then you need to convert it into List first. Here is how you can do it.

```
myTuple = ("iPhone", "Pixel", "Samsung")
myList = list(myTuple)
myList[1] = "onePlus"
myTuple = tuple(myList)
print(myTuple)
```

3. Set

Set is a collection which is unordered and unindexed. Sets are specified in curly brackets.

Example

```
myset{"iPhone","Pixel","Samsung"}
print{myset}
```

How to access Set items

You can't access the set items using indexes because they are unordered and unindexed. Hence you can use for loop in order to loop through set items.

```
mySet = {"iPhone", "Pixel", "Samsung"}

for mbl in mySet:
  print(mbl)
```

How to add an item in a set

You can't change the item in a set once created. But, you can add items to a set using `add` method.

```
mySet = {"iPhone", "Pixel", "Samsung"}
mySet.add('OnePlus')
print(mySet) # prints {'iPhone', 'Samsung', 'OnePlus', 'Pixel'}
```

Python provides in-built methods to use on sets like add method. Check below:

Method	Description	Usage
add()	to add an element to the set	mySet.add('value')
clear()	to remove all the elements from the set	mySet.clear()
pop()	to remove last element from the set	mySet.pop()
remove()	to remove a specified element from the set	mySet.remove("value")
del()	to delete a set	del myset
copy()	to return a copy of the set	copySet = mySet.copy()
difference()	to return a set containing the difference between two or more sets	mySet3 = mySet1.difference(mySet2)
difference_update()	to remove the items present in a set that are also included in another set	mySet1.difference_update(mySet 2)
discard()	to remove a specified item	mySet.discard("value")

Method	Description	Usage
intersection()	to return a set which is the intersection of two other sets	mySet3 = mySet1.intersection(mySet2)
intersection_update()	to remove the items in this set that are not present in other set	mySet1.intersection_update(mySet2)
isdisjoint()	to return whether two sets have a intersection or not	mySet3 = mySet1.isdisjoint(mySet2)
issubset()	to return whether another set contains this set or not	mySet3 = mySet1.issubset(mySet2)
issuperset()	to return whether this set contains another set or not	mySet3 = mySet1.issuperset(mySet2)
symmetric_difference()	to return a set with the symmetric differences of two sets	mySet3 = mySet1.symmetric_difference(mySet2)
symmetric_difference_update()	to insert the symmetric differences from this set and another	mySet1.symmetric_difference_update(mySet2)
union()	to return a set containing the union of sets	mySet3 = mySet1.union(mySet2)
update()	to update the set with the union of this set and others	mySet1.update(mySet2)

4. Dictionary

Dictionary is a collection of key value pairs which is unordered, can be changed, and indexed. They are written in curly brackets with key - value pairs.

Example

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11"
}
print(mydict)
```

How to access an item present in a dictionary

You can access the items present in a dictionary using key name present in a square brackets.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11"
}
val = mydict["brand"]
print(val) # prints iPhone
```

You can also use get() method which gives the same result.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11"
}
val = mydict.get("brand")
print(val) # prints iPhone
```

How to change a value present in a dictionary

You can change the value of a specific item by referring it's key name.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000"
}

mydict["cost"] ="$999"
print(mydict) # prints {'brand': 'iPhone', 'model': 'iPhone 11', 'cost': '$999'}
```

How to loop through items in a dictionary

`for` can be used to loop through the items present in a dictionary.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000"

}
for mbl in mydict:
  print(mydict)
```

How to add items in a dictionary

You can add items to a dictionary using a new index key and assign value to it.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000"
}
mydict["color"] = "Black"
print(mydict) # prints {'brand': 'iPhone', 'model': 'iPhone 11', 'cost': '$1000', 'color': 'B
```

How to remove items from a dictionary

There are several methods to remove an item from a dictionary.

1. pop()

`pop()` method is used to remove an item from a dictionary.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000",
    "color" : "Black"
}
mydict.pop("color")
print(mydict) # prints {'brand': 'iPhone', 'model': 'iPhone 11', 'cost': '$1000'}
```

2. del()

- `del()` method is used to remove an item from a dictionary.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000",
    "color" : "Black"
}
del mydict["color"]
print(mydict)
```

- `del` keyword can also delete whole dictionary.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000",
    "color" : "Black"
}
del mydict
print(mydict) # throws an error that 'mydict' is not defined
```

- `clear()` is used to empty the dictionary items.

```
mydict = {
    "brand" :"iPhone",
    "model": "iPhone 11",
    "cost" : "$1000",
    "color" : "Black"
}
mydict.clear()
print(mydict) # prints {}
```