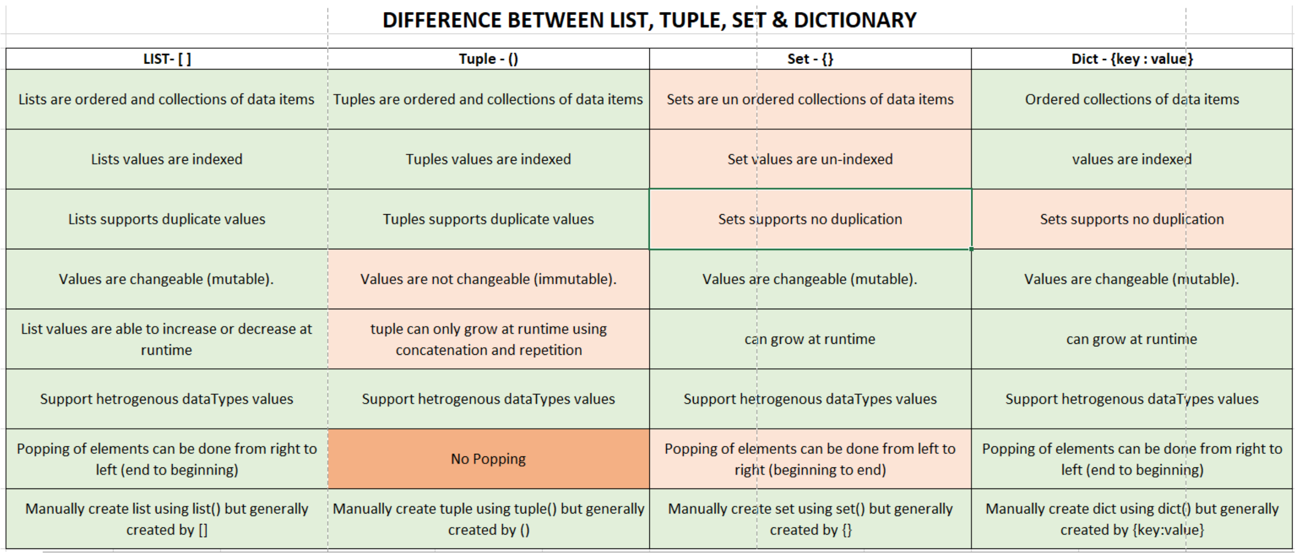
List, Tuples, Set, Dictionary Data Types



DIFFERENCE BETWEEN LIST, TUPLE, SET & DICTIONARY

LIST- [ ]	Tuple - ()	Set - {}	Dict - {key : value}
Lists are ordered and collections of data items	Tuples are ordered and collections of data items	Sets are un ordered collections of data items	Ordered collections of data items
Lists values are indexed	Tuples values are indexed	Set values are un-indexed	values are indexed
Lists supports duplicate values	Tuples supports duplicate values	Sets supports no duplication	Sets supports no duplication
Values are changeable (mutable).	Values are not changeable (immutable).	Values are changeable (mutable).	Values are changeable (mutable).
List values are able to increase or decrease at runtime	tuple can only grow at runtime using concatenation and repetition	can grow at runtime	can grow at runtime
Support hetrogenous dataTypes values	Support hetrogenous dataTypes values	Support hetrogenous dataTypes values	Support hetrogenous dataTypes values
Popping of elements can be done from right to left (end to beginning)	No Popping	Popping of elements can be done from left to right (beginning to end)	Popping of elements can be done from right to left (end to beginning)
Manually create list using list() but generally created by []	Manually create tuple using tuple() but generally created by ()	Manually create set using set() but generally created by {}	Manually create dict using dict() but generally created by {key:value}

List Comprehension

Python is famous for allowing you to write code that's elegant, easy to write, and almost as easy to read as plain English.

One of the language's most distinctive features is the **list comprehension**, which you can use to create powerful functionality within a single line of code.

List comprehension is a concise and elegant way to create a new list in Python by applying an expression to each element of an existing iterable and filtering the results based on a condition.

Syntax

newList **= [** *expression(element)* ***for*** *element* ***in*** *oldList* ***if*** *condition* ***]***

Advantages of List Comprehension

- More time-efficient and space-efficient than loops.
- Require fewer lines of code.
- Transforms iterative statement into a formula.

example:

list1 = [i for i in range(20) if i % 2 == 0]

output: list1 = [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]

Creation and Iteration or list

Creation:

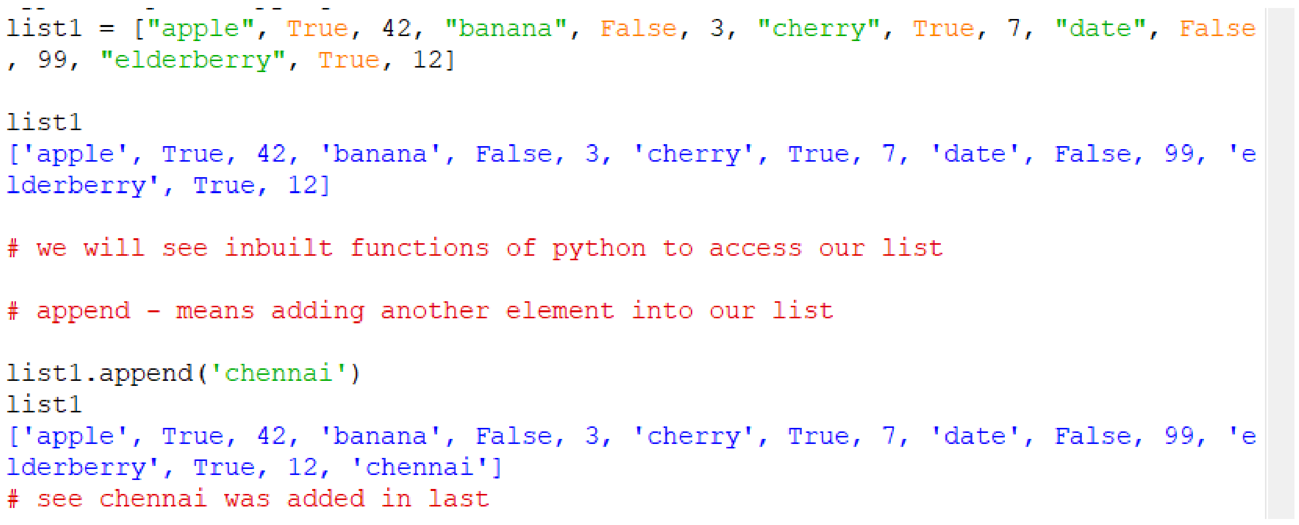
```
# Creating lists

list1 = ["apple", "banana", "cherry", "apple", "date"]
list1
['apple', 'banana', 'cherry', 'apple', 'date']
list2 = [1, "apple", True, 3.14]
list2
[1, 'apple', True, 3.14]
tuple1 = (2, 5, 3, 9, 7, 1, 8, 4, 6, 0)
tuple1
(2, 5, 3, 9, 7, 1, 8, 4, 6, 0)
list3 = list(tuple1)
list3
[2, 5, 3, 9, 7, 1, 8, 4, 6, 0]

# the above tuple1 is not a list actually, its a tuple. we just convert it into
a list with the use of keyword 'list(tuple)' like this. then the list3 is now a
list.


type(tuple1)
<class 'tuple'>
type(list3)
<class 'list'>
```

Iteration:

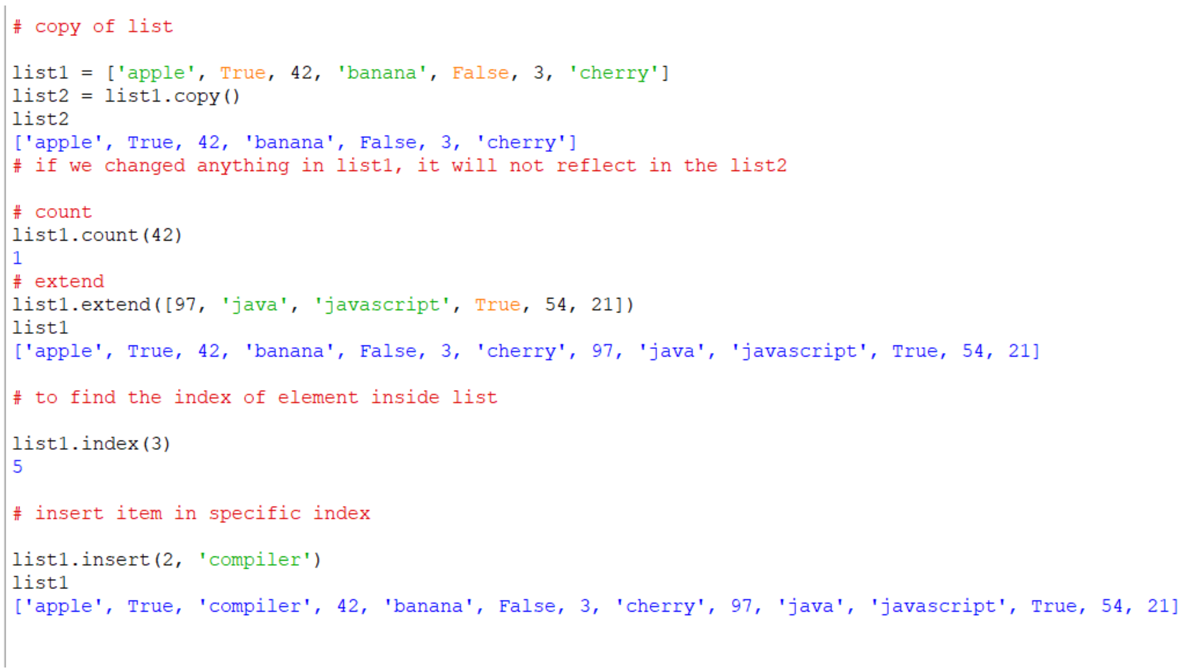
```
# Iterations of lists

list1 = ["apple", "banana", "cherry", "apple", "date"]

# we have iter function and magical method __next__(). with the help of these tw
o func we can iterate lists.

iterTest = iter(list1)
iterTest.__next__()
'apple'
iterTest.__next__()
'banana'
iterTest.__next__()
'cherry'
iterTest.__next__()
'apple'

# like this we can iterate list. but we need to call next method each time...ins
tead we can use looping.

for i in list1:
    print("item of list" ,i)


item of list apple
item of list banana
item of list cherry
```

```
item of list apple
item of list date
```

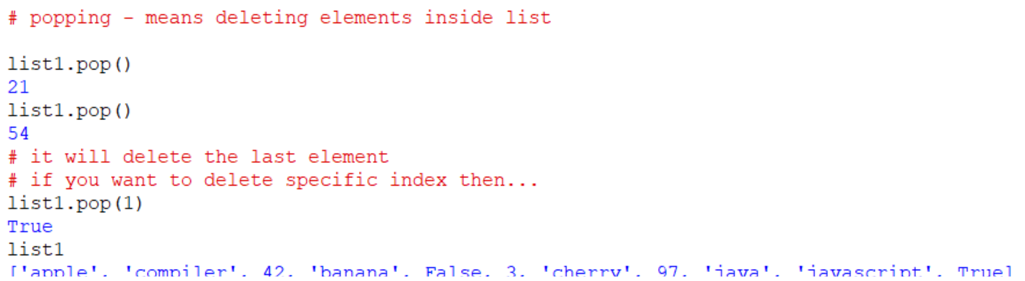
Built-in Functions of Python



```
list1 = ["apple", True, 42, "banana", False, 3, "cherry", True, 7, "date", False
, 99, "elderberry", True, 12]

list1
['apple', True, 42, 'banana', False, 3, 'cherry', True, 7, 'date', False, 99, 'e
lderberry', True, 12]

# we will see inbuilt functions of python to access our list

# append - means adding another element into our list

list1.append('chennai')
list1
['apple', True, 42, 'banana', False, 3, 'cherry', True, 7, 'date', False, 99, 'e
lderberry', True, 12, 'chennai']
# see chennai was added in last
```



```
# copy of list

list1 = ['apple', True, 42, 'banana', False, 3, 'cherry']
list2 = list1.copy()
list2
['apple', True, 42, 'banana', False, 3, 'cherry']
# if we changed anything in list1, it will not reflect in the list2

# count
list1.count(42)
1
# extend
list1.extend([97, 'java', 'javascript', True, 54, 21])
list1
['apple', True, 42, 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True, 54, 21]

# to find the index of element inside list

list1.index(3)
5

# insert item in specific index

list1.insert(2, 'compiler')
list1
['apple', True, 'compiler', 42, 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True, 54, 21]
```



```
# popping - means deleting elements inside list

list1.pop()
21
list1.pop()
54
# it will delete the last element
# if you want to delete specific index then...
list1.pop(1)
True
list1
['apple', 'compiler', 42, 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]
```

```
['apple', 'compiler', 42, 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]
list1.pop(2)
42
list1
['apple', 'compiler', 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]



# remove - diff bwt pop and remove is in pop we will give index, in remove we will give name of the element.
list1
['apple', 'compiler', 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]
list1.remove('apple')
list1
['compiler', 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]


# reverse the list

list1
['compiler', 'banana', False, 3, 'cherry', 97, 'java', 'javascript', True]
list1.reverse()
list1
[True, 'javascript', 'java', 97, 'cherry', 3, False, 'banana', 'compiler']
```

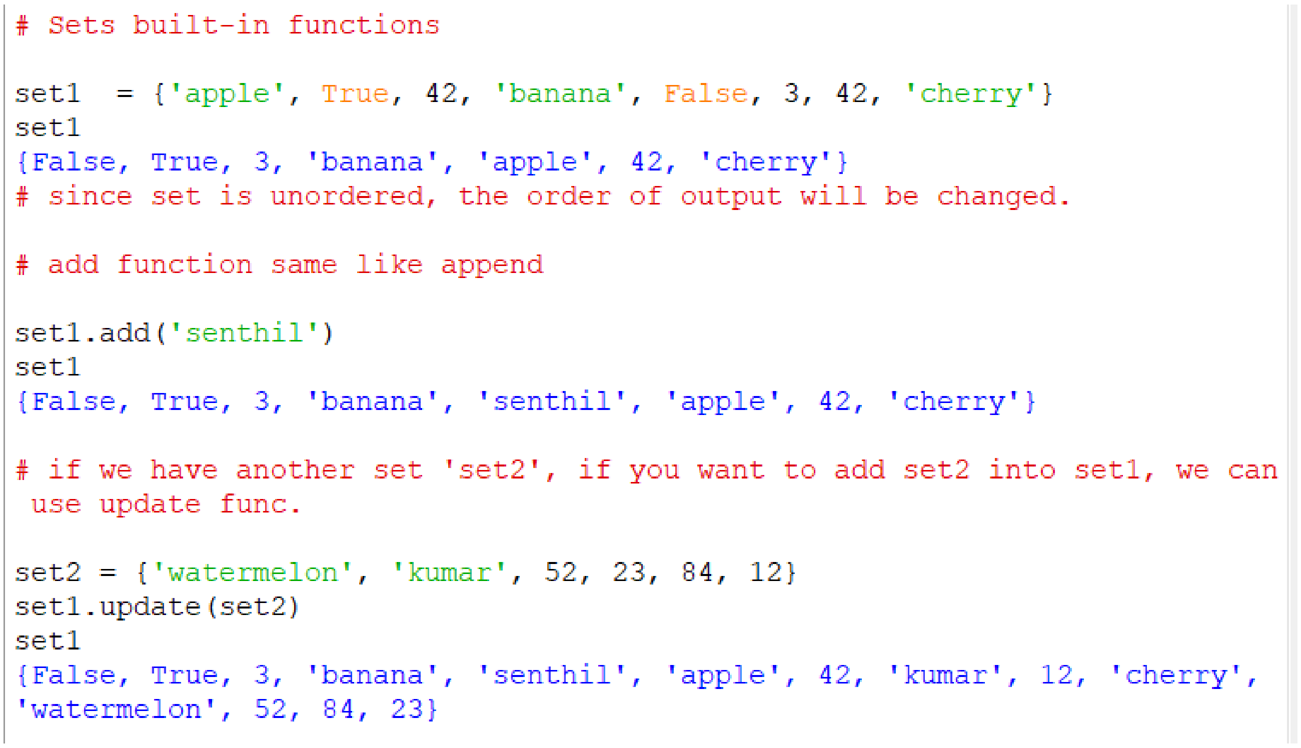
Tuples built-in Functions

```
# Tuples
tup = ('apple', True, 42, 'banana', False, 3, 42, 'cherry')
tup
('apple', True, 42, 'banana', False, 3, 42, 'cherry')
type(tup)
<class 'tuple'>

# tuples has only two built in functions

tup.index(42)
2
tup.count(42)
2
tup.count('cherry')
1

# and we can concatenate tuple

tup2 = ('india', 'australia', 'america')

tup3 = tup + tup2
tup3
('apple', True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america')

# and do tuple repetation

tup2*2
('india', 'australia', 'america', 'india', 'australia', 'america')
```

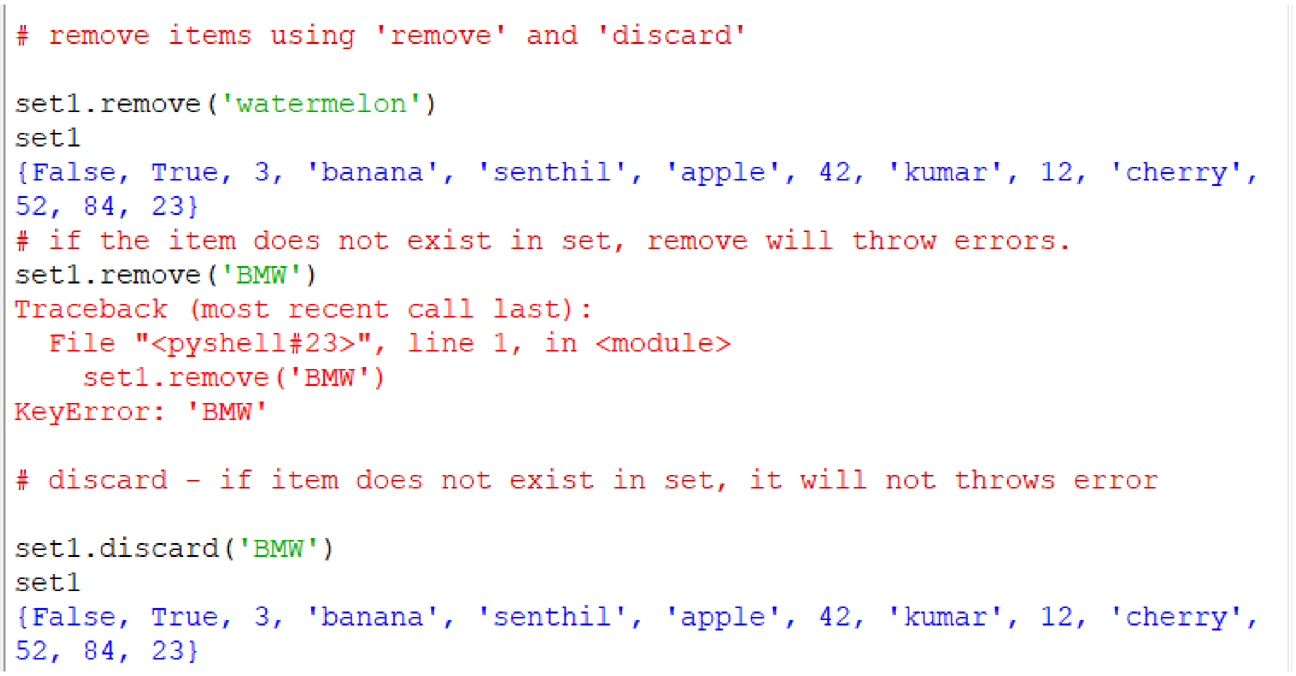
```
# if you want to change or append or pop or do something in tuple, you should convert it into list and
then change whatever and the again convert it into tuple.

tup3
('apple', True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america')
tupList = list(tup3)
tupList
['apple', True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america']
tupList.pop(0)
'apple'
tupList
[True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america']
tupList.insert(0, "Grapes")
tupList
['Grapes', True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america']
tup3 = tuple(tupList)
tup3
('Grapes', True, 42, 'banana', False, 3, 42, 'cherry', 'india', 'australia', 'america')
```

```
# like this we can modify tuples
```

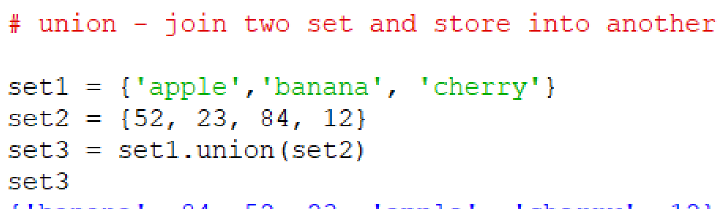
Sets Built-in Functions



```
# Sets built-in functions

set1  = {'apple', True, 42, 'banana', False, 3, 42, 'cherry'}
set1
{False, True, 3, 'banana', 'apple', 42, 'cherry'}
# since set is unordered, the order of output will be changed.

# add function same like append

set1.add('senthil')
set1
{False, True, 3, 'banana', 'senthil', 'apple', 42, 'cherry'}

# if we have another set 'set2', if you want to add set2 into set1, we can
 use update func.

set2 = {'watermelon', 'kumar', 52, 23, 84, 12}
set1.update(set2)
set1
{False, True, 3, 'banana', 'senthil', 'apple', 42, 'kumar', 12, 'cherry',
'watermelon', 52, 84, 23}
```



```
# remove items using 'remove' and 'discard'

set1.remove('watermelon')
set1
{False, True, 3, 'banana', 'senthil', 'apple', 42, 'kumar', 12, 'cherry',
52, 84, 23}
# if the item does not exist in set, remove will throw errors.
set1.remove('BMW')
Traceback (most recent call last):
  File "<pyshell#23>", line 1, in <module>
    set1.remove('BMW')
KeyError: 'BMW'

# discard - if item does not exist in set, it will not throws error

set1.discard('BMW')
set1
{False, True, 3, 'banana', 'senthil', 'apple', 42, 'kumar', 12, 'cherry',
52, 84, 23}
```



```
# union - join two set and store into another

set1 = {'apple','banana', 'cherry'}
set2 = {52, 23, 84, 12}
set3 = set1.union(set2)
set3
{'banana', 84, 52, 23, 'apple', 'cherry', 12}
```

```
{'banana', 84, 52, 23, 'apple', 'cherry', 12}

#intersection_update
set1 = {'apple','banana', 'cherry'}
set2 = {52, 23, 84, 12, 'apple'}
set1.intersection_update(set2)
set1
{'apple'}
# why only 'apple' because apple is only in set1 and set2 so remaining are remove
d with this func

# intersection ()

set1 = {'apple','banana', 'cherry'}
set2 = {52, 23, 84, 12, 'apple'}
set3 = set1.intersection(set2)
set3
{'apple'}
# here same concept but the value will be stored in set3 using this intersection
function
```

```
# symmetric_difference_update()
set1 = {'apple','banana', 'cherry'}
set2 = {52, 23, 84, 12, 'apple'}
set1.symmetric_difference_update(set2)
set1
{'banana', 12, 52, 84, 23, 'cherry'}

# here what will happen is, set1 will store unique elements only on both set1 and
set2


# symmetric_differenct()



# symmetric_difference()

set1 = {'apple','banana', 'cherry'}
set2 = {52, 23, 84, 12, 'apple'}
set3 = set1.symmetric_difference(set2)
set3
{'banana', 84, 52, 23, 12, 'cherry'}
# same function but the value will be stored in set3
```

Dictionary Built-in Functions

```
# Dictionary
my_dict = {
    "name": "Alice",
    "age": 25,
    "gender": "Female",
    "country": "USA",
    "list" : [1,5, {"name": "ram"}],
    "set" : {"asdfbajks"}
}
my_dict
```

```
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}],
'set': {'asdfbajks'}}

my_dict["name"]
'Alice'

# get function

my_dict.get("name")
'Alice'

# getting keys only from a dictionary
keyss = my_dict.keys()
keyss
dict_keys(['name', 'age', 'gender', 'country', 'list', 'set'])
# getting values only from a dict
valuess = my_dict.values()
valuess
dict_values(['Alice', 25, 'Female', 'USA', [1, 5, {'name': 'ram'}], {'asdfbajks'}])

#copy
copied = my_dict.copy()
copied
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}],
'set': {'asdfbajks'}}
```

```
# form keys
cart = ["added1", "added2", "added3", "added4"]
product = ({"FoodName": "Veg Meals", "Price": 450})

total_cartItems = dict.fromkeys(cart, product)
total_cartItems
{'added1': {'FoodName': 'Veg Meals', 'Price': 450}, 'added2': {'FoodName': 'Veg Meals', 'Price': 450
}, 'added3': {'FoodName': 'Veg Meals', 'Price': 450}, 'added4': {'FoodName': 'Veg Meals', 'Price': 4
50}}


total_cartItems.items()
dict_items([('added1', {'FoodName': 'Veg Meals', 'Price': 450}), ('added2', {'FoodName': 'Veg Meals'
, 'Price': 450}), ('added3', {'FoodName': 'Veg Meals', 'Price': 450}), ('added4', {'FoodName': 'Veg
Meals', 'Price': 450})])


my_dict
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}],
'set': {'asdfbajks'}}
# pop in dict

my_dict.pop('name')
'Alice'
my_dict
{'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}], 'set': {'asdfbajk
s'}}
# here what was happened is pop func removed the name key value pair.
```

```
# popItems() function
# it will remove the last key value pair in a dictionary
my_dict
{'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}], 'set': {'asdfbajk
s'}}
my_dict.popitem()
('set', {'asdfbajks'})
my_dict
{'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': 'ram'}]}
# here the last element set was removed.
```

```
my_dict = {
    "name": "Alice",
    "age": 25,
    "gender": "Female",
    "country": "USA",
    "list" : [1,5, {"name": "ram"}],
    "set" : {"asdfbajks"}
}
my_dict
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': '
ram'}], 'set': {'asdfbajks'}}
# setdefault function

# it will fetch the value of given key
val = my_dict.setdefault("age")
val
25
# if the key doesn't exist, it will create that key with given value
val2 = my_dict.setdefault("email", "ram@gmail.com")
val2
'ram@gmail.com'
my_dict
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': '
ram'}], 'set': {'asdfbajks'}, 'email': 'ram@gmail.com'}
# email was added
# if you haven't given any value then it will take "None" automatically...


# update functions
my_dict
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': '
ram'}], 'set': {'asdfbajks'}, 'email': 'ram@gmail.com'}
my_dict.update({"phone_number": 9080716503})
my_dict
{'name': 'Alice', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name': '
ram'}], 'set': {'asdfbajks'}, 'email': 'ram@gmail.com', 'phone_number': 9080716503}
```

```
my_dict = {
    "name": "Alice",
    "age": 25,
    "gender": "Female",
    "country": "USA",
    "list" : [1,5, {"name": "ram"}],
    "set" : {"asdfbajks"}
}
# accessing items inside the dictionary

my_dict['name']
'Alice'
my_dict.get('name')
'Alice'

# change some value inside dict

my_dict['name'] = "ram"
my_dict
{'name': 'ram', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name':
'ram'}], 'set': {'asdfbajks'}}
# add a new key value pair without any function

my_dict["newKey"] = "newly added value"
my_dict
{'name': 'ram', 'age': 25, 'gender': 'Female', 'country': 'USA', 'list': [1, 5, {'name':
'ram'}], 'set': {'asdfbajks'}, 'newKey': 'newly added value'}
# looping the dictionary

for i in my_dict:
    print(my_dict[i])


ram
25
Female
USA
[1, 5, {'name': 'ram'}]
{'asdfbajks'}
```

```
newly added value
```