Dictionary and Sets

(Retrieved from https://www.geeksforgeeks.org/add-a-keyvalue-pair-to-dictionary-in-python/)

A dictionary is a sequence of items. Each item is a pair made of a key and a value. Dictionaries are not sorted. You can access to the list of keys or values independently.

```
# Creating an empty Dictionary
Dict = {}
print("Empty Dictionary: ")
print(Dict)

# Creating a Dictionary with Integer Keys
Dict = {1: 'Apple', 2: 'Strawberry', 3: 'Orange'}
print("\nDictionary with the use of Integer Keys: ")
print(Dict)

# Creating a Dictionary with Mixed keys
Dict = {'Name': 'Tracey', 1: [1, 2, 3, 4]}
print("\nDictionary with the use of Mixed Keys: ")
print(Dict)

# Creating a Dictionary with dict() method
Dict = dict({1: 'Apple', 2: 'Strawberry', 3:'Orange'})
print("\nDictionary with the use of dict(): ")
print(Dict)

# Creating a Dictionary with each item as a Pair
Dict = dict([(1, 'Banana'), (2, 'Kiwi')])
print("\nDictionary with each item as a pair: ")
print(Dict)
```

Test-run

```
Empty Dictionary:
{}

Dictionary with the use of Integer Keys:
{1: 'Apple', 2: 'Strawberry', 3: 'Orange'}

Dictionary with the use of Mixed Keys:
{'Name': 'Tracey', 1: [1, 2, 3, 4]}

Dictionary with the use of dict():
{1: 'Apple', 2: 'Strawberry', 3: 'Orange'}
```

Dictionary with each item as a pair:
{1: 'Banana', 2: 'Kiwi'}
==========

Accessing an element from a Dictionary

```
# Creating a Dictionary
Dict = {1: 'Apple', 'name': 'Mary', 3: 'Kiwi'}

# accessing a element using key
print("Acessing a element using key:")
print(Dict['name'])

# accessing a element using key
print("Acessing an element using key:")
print(Dict[1])

# accessing a element using get() method
print("Acessing a element using get:")
print(Dict.get(3))
```

Test-run
Acessing a element using key:
Mary
Acessing an element using key:
Apple
Acessing a element using get:
Kiwi

==========
Removing key value

```
# Initial Dictionary
Dict = {5 : 'Welcome', 6 : 'To', 7 : 'Geeks',
        'A' : {1 : 'Geeks', 2 : 'For', 3 : 'Geeks'},
        'B' : {1 : 'Geeks', 2 : 'Life'}}
print("Initial Dictionary: ")
print(Dict)

# Deleting a Key value
del Dict[6]
print("\nDeleting a specific key: ")
print(Dict)

# Deleting a Key from
# Nested Dictionary
```

```
del Dict['A'][2]
print("\nDeleting a key from Nested Dictionary: ")
print(Dict)

# Deleting a Key
# using pop()
Dict.pop(5)
print("\nPopping specific element: ")
print(Dict)

# Deleting a Key
# using popitem()
Dict.popitem()
print("\nPops first element: ")
print(Dict)

# Deleting entire Dictionary
Dict.clear()
print("\nDeleting Entire Dictionary: ")
print(Dict)
```

==========

Adding key value using subscript notation

```
# Python program to add a key:value pair to dictionary

dict = {'key1':'geeks', 'key2':'for'}
print("Current Dict is: ", dict)

# using the subscript notation
# Dictionary_Name[New_Key_Name] = New_Key_Value

dict['key3'] = 'Geeks'
dict['key4'] = 'is'
dict['key5'] = 'portal'
dict['key6'] = 'Computer'
print("Updated Dict is: ", dict)
```

Current Dict is: {‘key2’: ‘for’, ‘key1’: ‘geeks’}
Updated Dict is: {‘key3’: ‘Geeks’, ‘key5’: ‘portal’, ‘key6’: ‘Computer’, ‘key4’: ‘is’, ‘key1’: ‘geeks’, ‘key2’: ‘for’}

Adding key value using update method

```
dict = {'key1':'geeks', 'key2':'for'}
print("Current Dict is: ", dict)

# adding dict1 (key3, key4 and key5) to dict
dict1 = {'key3':'geeks', 'key4':'is', 'key5':'fabulous'}
dict.update(dict1)

# by assigning
dict.update(newkey1 ='portal')
print(dict)
```

Current Dict is: {‘key2’: ‘for’, ‘key1’: ‘geeks’}
{‘newkey1’: ‘portal’, ‘key4’: ‘is’, ‘key2’: ‘for’, ‘key1’: ‘geeks’, ‘key5’: ‘fabulous’, ‘key3’: ‘geeks’}

```
stocks = {
        'IBM': 146.48,
        'MSFT':44.11,
        'CSCO':25.54
}

#print out all the keys
for c in stocks:
        print(c)

#print key n values
for k, v in stocks.items():
        print("Code : {0}, Value : {1}".format(k, v))

main()
```

MSFT
IBM
CSCO
Code : MSFT, Value : 44.11
Code : IBM, Value : 146.48
Code : CSCO, Value : 25.54

A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements.

```
# Python program to demonstrate differences
# between normal and frozen set

# Same as {"a", "b","c"}
normal_set = set(["a", "b","c"])

# Adding an element to normal set is fine
normal_set.add("d")

print("Normal Set")
print(normal_set)

# A frozen set
frozen_set = frozenset(["e", "f", "g"])

print("Frozen Set")
print(frozen_set)

# Uncommenting below line would cause error as
# we are trying to add element to a frozen set
# frozen_set.add("h")
```

```
Normal Set
set(['a', 'c', 'b', 'd'])
Frozen Set
frozenset(['e', 'g', 'f'])
```

Methods for Sets

(Retrieved from https://www.geeksforgeeks.org/sets-in-python/)

1. add(x) Method: Adds the item x to set if it is not already present in the set.

```
people = {"Jay", "Idrish", "Archil"}

people.add("Daxit")
```

-> This will add Daxit in people set.

2. **union(s) Method**: Returns a union of two set. Using the '|' operator between 2 sets is the same as writing set1.union(set2)

```
people = {"Jay", "Idrish", "Archil"}
vampires = {"Karan", "Arjun"}
population = people.union(vampires)
```

OR

```
population = people|vampires
```

-> Set population set will have components of both people and vampire

3. **intersect(s) Method:** Returns an intersection of two sets.The '&' operator comes can also be used in this case.

```
victims = people.intersection(vampires)
```

-> Set victims will contain the common element of people and vampire

4. difference(s) Method: Returns a set containing all the elements of invoking set but not of the second set. We can use '-' operator here.

```
safe = people.difference(vampires)
```

OR

```
safe = people – vampires
```

-> Set safe will have all the elements that are in people but not vampire

5. clear() Method: Empties the whole set.

```
victims.clear()
```

-> Clears victim set

However, there are two major pitfalls in Python sets:

1. The set doesn't maintain elements in any particular order.
2. Only instances of immutable types can be added to a Python set.