Topic: Dictionary and Set in python.

**Sets**

* A Set is an unordered collection data type that is iterable, mutable, and has no duplicate elements.
* The set itself is mutable. We can add or remove items from it.
* Sets can be used to perform mathematical operations like union, intersection, symmetric difference, etc. So we can say that python’s set class represents the mathematical notion of a set.

Set Creation

* Syntax

Example: We create a set of integers

| Input  s = {1, 2, 3}  print(s) |
| --- |

| Output  {1,2,3} |
| --- |

* Set doesn't allow duplicates. They store only one instance.

| Input  s = {1, 2, 3, 1, 4}  print(s) |
| --- |

| Output  {1, 2, 3, 4} |
| --- |

* We can make a set from a list and tuple.

| Input  s = set([1, 2, 3, 1])  print(s) |
| --- |

| Output  {1, 2, 3} |
| --- |

* We can initialize a set with set() method

| Input  s = set()  ​print(type(s)) #it will output the data structure of s |
| --- |

| Output  <class 'set'> |
| --- |

How to Add an element to a Set?

* we can add a single element using add() method and add multiple elements using the update() method

| Input  s = {1, 3}  #add element  s.add(2)  print(s) |
| --- |

| Output  {1, 2, 3} |
| --- |

| Input  #add multiple elements  s.update([5, 6, 1])  print(s) |
| --- |

| Output  {1, 2, 3, 5, 6} |
| --- |

| Input  #add list and set  s.update([8, 9], {10, 2, 3})  print(s) |
| --- |

| Output  {1, 2, 3, 5, 6, 8, 9, 10} |
| --- |

​

* Like list, Set object doesn't support indexing.

| Input  print(s[1]) #will get TypeError |
| --- |

| Output  ---------------------------------------------------------------------------  TypeError Traceback (most recent call last)  <ipython-input-4-c52fc339e293> in <module>()  TypeError: 'set' object does not support indexing |
| --- |

* If you want to remove elements from a Set.

→ A particular item can be removed from the set using methods, discard() and remove().

| Input  s = {1, 2, 3, 5, 4}  print(s)  ​s.discard(4) #4 is removed from set s  ​print(s) |
| --- |

| Output  {1, 2, 3, 4, 5}  {1, 2, 3, 5} |
| --- |

​ → You can use the remove method to remove an element

| Input  s.remove(2)  ​print(s) |
| --- |

| Output  {1, 3, 5} |
| --- |

→ If you remove an element not present in a set s you will get KeyError.

| Input  s.remove(7) # will get KeyError |
| --- |

| Output  ---------------------------------------------------------------------------  KeyError Traceback (most recent call last)  <ipython-input-14-f37cc9806699> in <module>()  KeyError: 7 |
| --- |

→ discard an element not present in a set s

| Input  s.discard(7)  print(s) |
| --- |

| Output  {1, 3, 5} |
| --- |

* We can remove the item using the pop() method also but it removes the random element.

| Input  s = {1, 2, 3, 5, 4}  ​s.pop()  ​print(s) |
| --- |

| Output  {2, 3, 4, 5} |
| --- |

| Input  s.pop()  print(s) |
| --- |

| Output  {3, 4, 5} |
| --- |

→ We can remove all items in the set using the clear() method

​ s.clear()

* Python Set Operations

→ Union of 2 sets using | operator

| Input  set1 = {1, 2, 3, 4, 5}  set2 = {3, 4, 5, 6, 7}  ​​print(set1 | set2) |
| --- |

| Output  {1, 2, 3, 4, 5, 6, 7} |
| --- |

→ Another way of getting a union of 2 sets

| Input  print(set1.union(set2)) |
| --- |

| Output  {1, 2, 3, 4, 5, 6, 7} |
| --- |

→ The intersection of 2 sets using & operator

| Input  print(set1 & set2) |
| --- |

| Output  {3, 4, 5} |
| --- |

→ Using intersection function

| Input  print(set1.intersection(set2)) |
| --- |

| Output  {3, 4, 5} |
| --- |

→ We can find the set Difference: set of elements that are only in set1 but not in set2

| Input  print(set1 - set2) |
| --- |

​

| Output  {1, 2} |
| --- |

→ We can use differnce function

| Input  print(set1.difference(set2)) |
| --- |

| Output  {1, 2} |
| --- |

→ Symmetric difference: The set of elements in both set1 and set2

except those that are common in both.

​ Use ^ operator

| Input  print(set1^set2) |
| --- |

| Output  {1, 2, 6, 7} |
| --- |

​ → Use symmetric\_difference function

| Input  print(set1.symmetric\_difference(set2)) |
| --- |

| Output  {1, 2, 6, 7} |
| --- |

For finding that set s1 is the subset of another function we can use issubset() method.

| Input  x = {"a","b","c","d","e"}  y = {"c","d"}  ​print("set 'x' is subset of 'y' ?", x.issubset(y)) #check x is subset of y  print("set 'y' is subset of 'x' ?", y.issubset(x)) #check y is subset of x |
| --- |

| Output  set 'x' is subset of 'y' ? False  set 'y' is subset of 'x' ? True |
| --- |

​

**Dictionary**

* Python dictionary is an unordered collection of items. While other compound data types have only value as an element, a dictionary has a key: value pair. Key-value is provided in the dictionary to make it more optimized.
* Dict Creation

| Input  #empty dictionary  my\_dict = {}  ​  #dictionary with integer keys  my\_dict = {1: 'abc', 2: 'xyz'}  print(my\_dict)  ​  #dictionary with mixed keys  my\_dict = {'name': 'satish', 1: ['abc', 'xyz']}  print(my\_dict)  ​  #create empty dictionary using dict()  my\_dict = dict()  ​  my\_dict = dict([(1, 'abc'), (2, 'xyz')]) #create a dict with list of tuples  print(my\_dict) |
| --- |

| Output  {1: 'abc', 2: 'xyz'}  {'name': 'satish', 1: ['abc', 'xyz']}  {1: 'abc', 2: 'xyz'}  Dict Access  my\_dict = {'name': 'satish', 'age': 27, 'address': 'guntur'}​ |
| --- |

| Input  #get name  print(my\_dict['name']) |
| --- |

| Output  satish |
| --- |

→ If the key is not present it gives KeyError

| Input  print(my\_dict['degree']) |
| --- |

| Output  ---------------------------------------------------------------------------  KeyError Traceback (most recent call last)  <ipython-input-5-c5aba24e1656> in <module>()  1 #if key is not present it gives KeyError  ----> 2 print(my\_dict['degree'])  KeyError: 'degree' |
| --- |

→ Another way of accessing key

| Input  print(my\_dict.get('address')) |
| --- |

| Output  guntur |
| --- |

→ If the key is not present it will give None using the get method

| Input  print(my\_dict.get('degree')) |
| --- |

| Output  None |
| --- |

* Dict Add or Modify Elements

| Input  my\_dict = {'name': 'satish', 'age': 27, 'address': 'guntur'}  #update name  my\_dict['name'] = 'raju'  print(my\_dict) |
| --- |

| Output  {'name': 'raju', 'age': 27, 'address': 'guntur'} |
| --- |

→ add new key

| Input  my\_dict['degree'] = 'M.Tech'  print(my\_dict) |
| --- |

| Output  {'name': 'raju', 'age': 27, 'address': 'guntur', 'degree': 'M.Tech'} |
| --- |

→ Dict Delete or Remove Element

| Input  #create a dictionary  my\_dict = {'name': 'satish', 'age': 27, 'address': 'guntur'}  ​#remove a particular item  print(my\_dict.pop('age'))  ​print(my\_dict) |
| --- |

| Output  27  {'name': 'satish', 'address': 'guntur'} |
| --- |

| Input  my\_dict = {'name': 'satish', 'age': 27, 'address': 'guntur'}  #remove an arbitarty key  my\_dict.popitem()  ​print(my\_dict) |
| --- |

| Output  {'name': 'satish', 'age': 27} |
| --- |

​

| Input  squares = {2: 4, 3: 9, 4: 16, 5: 25}  ​#delete particular key  del squares[2]  ​print(squares) |
| --- |

| Output  {3: 9, 4: 16, 5: 25} |
| --- |

| Input  #remove all items  squares.clear()  ​print(squares) |
| --- |

| Output  {} |
| --- |

| Input  squares = {2: 4, 3: 9, 4: 16, 5: 25}  ​#delete dictionary itself  del squares  ​print(squares) #NameError because dict is deleted |
| --- |

| Output  ---------------------------------------------------------------------------  NameError Traceback (most recent call last)  <ipython-input-16-355e8277492b> in <module>()  4 del squares  5  ----> 6 print(squares) #NameError because dict is deleted  NameError: name 'squares' is not defined |
| --- |

→ Dictionary Methods

| Input  squares = {2: 4, 3: 9, 4: 16, 5: 25}  ​my\_dict = squares.copy()  print(my\_dict) |
| --- |

| Output  {2: 4, 3: 9, 4: 16, 5: 25} |
| --- |

→ fromkeys[seq[, v]] -> Return a new dictionary with keys from seq and value equal to v (defaults to None).

| Input  subjects = {}.fromkeys(['Math', 'English', 'Hindi'], 0)  print(subjects) |
| --- |

| Output  {'Math': 0, 'English': 0, 'Hindi': 0} |
| --- |

| Input  subjects = {2:4, 3:9, 4:16, 5:25}  print(subjects.items()) #return a new view of the dictionary items (key, value) |
| --- |

| Output  dict\_items([(2, 4), (3, 9), (4, 16), (5, 25)]) |
| --- |

| Input  subjects = {2:4, 3:9, 4:16, 5:25}  print(subjects.keys()) #return a new view of the dictionary keys |
| --- |

| Output  dict\_keys([2, 3, 4, 5]) |
| --- |

| Input  subjects = {2:4, 3:9, 4:16, 5:25}  print(subjects.values()) #return a new view of the dictionary values |
| --- |

| Output  dict\_values([4, 9, 16, 25]) |
| --- |

| Input  #get list of all available methods and attributes of dictionary  d = {}  print(dir(d)) |
| --- |

| Output  ['\_\_class\_\_', '\_\_contains\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_len\_\_', '\_\_lt\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', 'clear', 'copy', 'fromkeys', 'get', 'items', 'keys', 'pop', 'popitem', 'setdefault', 'update', 'values'] |
| --- |

* Dict Comprehension

Dict comprehensions are just like list comprehensions but for dictionaries.

​

| Input  d = {'a': 1, 'b': 2, 'c': 3}  for pair in d.items():  print(pair) |
| --- |

| Output  ('a', 1)  ('c', 3)  ('b', 2) |
| --- |

| Input  #Creating a new dictionary with only pairs where the value is larger than 2  d = {'a': 1, 'b': 2, 'c': 3, 'd': 4}  new\_dict = {k:v for k, v in d.items() if v > 2}  print(new\_dict) |
| --- |

| Output  {'c': 3, 'd': 4} |
| --- |

| Input  #We can also perform operations on the key value pairs  d = {'a':1,'b':2,'c':3,'d':4,'e':5}  d = {k + 'c':v \* 2 for k, v in d.items() if v > 2}  print(d) |
| --- |

| Output  {'cc': 6, 'dc': 8, 'ec': 10} |
| --- |