DATE : 8 june 2024

DAY : Saturday

TOPICS : Sets, Operations on sets, Dictionary, Rules for creating a dictionary, Built-in Dictionary Methods

SET

A Python set is a well-defined collection of distinct objects, called elements or members.

Sets are unordered. Set elements are unique–if you try to add an element to a set a second time, it has no effect.

A set itself may be modified, but the elements contained in a set must be immutable (actually hashable, but for the types you are familiar with so far, immutable and hashable are effectively the same thing).

Sets are unordered collections of unique objects, there are two types of set:

1. Sets - They are mutable and new elements can be added once sets are defined

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
print(basket) # duplicates will be removed
```

```
{'pear', 'orange', 'banana', 'apple'}
```

```
a = set('abracadabra')
print(a)
```

```
{'d', 'a', 'b', 'r', 'c'}
```

2. Frozen Sets - They are immutable and new elements cannot added after its defined.

```
b = frozenset('asdfagsa')
print(b)
```

```
frozenset({'g', 'd', 'a', 's', 'f'})
```

```
cities = frozenset(["Frankfurt", "Basel","Freiburg"])
print(cities)
```

```
frozenset({'Basel', 'Frankfurt', 'Freiburg'})
```

Operations on sets

```
#intersection
a = {1,2,3,4,5,6,7}
b = {2,3,5,7}
print(a.intersection(b))
print(a&b)
```

```
{2, 3, 5, 7}
{2, 3, 5, 7}
```

```
#union
print(a.union(b))
print(a|b)
```

```
{1, 2, 3, 4, 5, 6, 7}
{1, 2, 3, 4, 5, 6, 7}
```

```
# difference
print(a.difference(b))
print(a-b)
```

```
{1, 4, 6}
```

```
#symmetric difference
print(a.symmetric_difference(b))
print(a^b)
```

```
{1, 4, 6}
{1, 4, 6}
```

```
#subset
print(b.issubset(a))
print(b<=a)
```

```
True
True
```

```
#superset
print(a.issuperset(b))
print(a>=b)
```

```
True
True
```

```
#propersubset
print(a<b)
print(set()<b)
```

```
False
True
```

```
#disjoint
print(a.isdisjoint(b))
```

```
False
```

With single elements

Existence check

```
#in
print(2 in a)
print(8 in a)
```

```
True
False
```

```
#not in
2 not in a
```

```
False
```

Add and Remove

```
#add
a.add(8)
print(a)
```

```
{1, 2, 3, 4, 5, 6, 7, 8}
```

```
#discard/remove
c = {1,2,3,5,8}
c.discard(3)
print(c)
c.remove(8)
print(c)
c.remove(2)
print(c)
```

```
{1, 2, 5, 8}
{1, 2, 5}
{1, 5}
```

Get the unique elements of a list

Let's say you've got a list of restaurants -- maybe you read it from a file. You care about the unique restaurants in the list. The best way to get the unique elements from a list is to turn it into a set:

```
restaurants = ["McDonald's", "Burger King", "McDonald's", "Chicken Chicken"]
unique_restaurants = set(restaurants)
print(unique_restaurants)
```

```
{'Burger King', "McDonald's", 'Chicken Chicken'}
```

```
restaurants = ["McDonald's", "Burger King", "Burger king","McDonald's", "Chicken Chicken"]
restaurants.remove("McDonald's")
restaurants
```

```
['Burger King', 'Burger king', "McDonald's", 'Chicken Chicken']
```

Note that the set is not in the same order as the original list; that is because sets are unordered, just like dicts.

```
# Removes all duplicates and returns another list
list(set(restaurants))
```

```
['Burger King', "McDonald's", 'Chicken Chicken', 'Burger king']
```

Practice Questions(Sets)

1. Create a set with the first five prime numbers.
2. Add the number 7 to the set.
3. Remove the number 3 from the set.
4. Check if the set is a subset of {2, 3, 5, 7, 11}.
5. Find the union of the set with {7, 11, 13}.
6. Create a frozen set from the original set.
7. Check if the set has any common elements with {1, 4, 9}.
8. Remove all elements from the set.

```
prime_numbers = {2,3,5,7,11}
prime_numbers.add(7)
print(prime_numbers)
prime_numbers.remove(3)
print(prime_numbers)
#check subset
if {2,3,5,7,11} in prime_numbers:
  print("True")
else:
  print("False")
#union
print(prime_numbers.union({7,11,13}))
#frozen
f_set = frozenset({2,3,5,7,11})
print(f_set)
#check for common elements
if {1,4,9} in prime_numbers:
  print("True")
else:
  print("False")
#remove all the elements
prime_numbers.remove(2)
prime_numbers.remove(5)
prime_numbers.remove(7)
prime_numbers.remove(11)
print(prime_numbers)
```

```
{2, 3, 5, 7, 11}
{2, 5, 7, 11}
False
```

```
{2, 5, 7, 11, 13}
frozenset({2, 3, 5, 7, 11})
False
set()
```

9. Create a set of your favorite fruits and find the intersection with { 'apple', 'banana', 'orange' }.
10. Use set comprehension to create a set of squares for numbers 1 to 10

```
fruits = {"apple","mango","banana","lichi"}
fruits.intersection({'apple','banana','orange'})
```

```
{'apple', 'banana'}
```

You have two sets representing students enrolled in Math and Science classes. Find the set of students who are enrolled in both classes.

```
maths_class = {"heena", "arsh","nishu","neha","mansi"}
science_class = {"arsh","davil","nishu","mansi","riya"}
maths_class.intersection(science_class)
```

```
{'arsh', 'mansi', 'nishu'}
```

Given a set of unique employee IDs, add a new employee ID to the set

```
employee_ID = {1342, 1986, 1232, 1416}
employee_ID.add(1562)
print(employee_ID)
```

```
{1986, 1416, 1232, 1562, 1342}
```

You are managing two different projects and have two sets representing the team members for each project. Find the set of team members who are working on either project or both projects.

```
project_1 = {"alisha","devil","bob","carol","heli"}
project_2 = {"carol","helen","devil","bob"}
#working on either of projects
print(project_1.symmetric_difference(project_2))
#working on both projects
print(project_1.intersection(project_2))
```

```
{'alisha', 'heli', 'helen'}
{'carol', 'bob', 'devil'}
```

You have a set of customer IDs who have made purchases this month. Remove a customer ID from the set who has canceled their order.

```
customer_id = {1235, 1984, 1461, 1342,1324, 1476}
#if customer cancel their order then id is removed (eg-1461)
cancel_order = "1461"
if "cancel_order" in customer_id:
  customer_id.remove(cancel_order)
print(customer_id)
```

```
{1984, 1235, 1476, 1461, 1324, 1342}
```

Given two sets of product IDs, one representing products in stock and the other representing products that have been sold, find the set of products that are still in stock.

```
productid_in_stock = {1256, 1367, 1463, 1289, 2346, 1187, 3214}
productid_sold = {1256, 1463, 1289, 1187}
#product still in stock
print(productid_in_stock-productid_sold)
```

```
{2346, 3214, 1367}
```

You have a set of registered conference attendees. Check if a particular person is registered

```
registered_attendees = {"bob","helen","david","heln","carol"}
if "helen" in registered_attendees:
  print("yes")
else:
  print("no")
```

```
yes
```

Given two sets of book titles, one representing books available in the library and the other representing books borrowed by students, find the set of books that are not currently available in the library.

```
available_books = {"Wuthering Heights","Pride and Prejudice","To kill a Mockingbird","Gulliver's Travels","Frankenstein"}
borrowed_books = {"Pride and Prejudice","Gulliver's Travels"}
#Books that are not currently available
print(available_books-borrowed_books)
```

```
{'Frankenstein', 'To kill a Mockingbird', 'Wuthering Heights'}
```

You have a set of unique tags assigned to a blog post. Add multiple new tags to the set at once.

```
tags = {"beauty","fashion","health","environment","food"}
tags.update({"internet","world","water","happy"})
print(tags)
```

```
{'internet', 'beauty', 'water', 'world', 'health', 'food', 'environment', 'happy', 'fashion'}
```

Given a set of employee skills and another set of skills required for a new project, find the set of skills that need to be acquired or improved.

```
skills ={"leadership","risk management","communication","time management","problem solving"}
required_skills = {"problem solving","critical think","adaptibility","project planning","decision making"}
#set of skills to be acquired
print(f"Skills need to be acquired and improved are {required_skills.difference(skills)}")
```

```
Skills need to be acquired and improved are {'project planning', 'critical think', 'decision making', 'adaptibility'}
```

you have two sets of favorite movies from two different friends. Find the set of movies that are favorite to at least one of them but not both.

```
friend_1 = {"jawan","gadar","stupid7","golmaal"}
friend_2 = {"bahubali","777 charlie","gadar","pathaan"}
friend_1.symmetric_difference(friend_2)
```

```
{'777 charlie', 'bahubali', 'golmaal', 'jawan', 'pathaan', 'stupid7'}
```

Given a set of emails collected from a signup form, ensure that the set contains only unique emails by checking the length before and after adding more emails.

```
mails = {'abc@gmail.com','xyz@gmail.com','pqr@gmail.com'}
print(f" The length of set is {len(mails)}")
mails.add("xyz@gmail.com")
print(mails)
print(f" The length of set is {len(mails)}")
```

```
 The length of set is 3
{'xyz@gmail.com', 'abc@gmail.com', 'pqr@gmail.com'}
 The length of set is 3
```

You have a set of courses you are interested in and another set of courses you have completed. Find the set of courses you are still interested in but have not yet completed.

```
interested_courses = {'web development','cyber security','data science','machine learning','artificial intelligence'}
completed_courses = {'digital marketing','web development','data science'}
#courses interested in but havenot completed yet
interested_courses-completed_courses
```

```
{'artificial intelligence', 'cyber security', 'machine learning'}
```

Given a set of cities visited last year and another set of cities visited this year, find the set of cities that were visited in either year but not both

```
visited_lastyear = {"delhi","agra","mumbai","lucknow","chandigarh"}
visited_thisyear = {"himachal","chandigarh","kolkata","lucknow","punjab"}
visited_thisyear.symmetric_difference(visited_lastyear)
```

```
{'agra', 'delhi', 'himachal', 'kolkata', 'mumbai', 'punjab'}
```

You have a set of parts required to build a machine. Remove a part from the set if it is faulty.

```
machine_parts = {'needle','spoon pin','boobin','balance wheel','thread guide','needle plate'}
faulty_part = 'balance wheel'
#remove the part if it is faulty
if faulty_part in machine_parts:
  machine_parts.remove(faulty_part)
  print(machine_parts)
else:
  print("All parts are fine")
```

```
{'spoon pin', 'boobin', 'needle plate', 'needle', 'thread guide'}
```

You have a set of ingredients required for a recipe. Check if a specific ingredient is missing.

```
ingredients = {"salt", "milk", "water", "baking powder"}
#check if a particular ingredient is missing or not
if "sugar" in ingredients:
  print("It is present in ingredient list")
else:
  print("It is missing in ingredient list")
```

```
It is missing in ingredient list
```

Given a set of friends on social media and another set of mutual friends with a colleague, find the set of mutual friends.

```
#social media friends set
socialmedia_friends = {"bob", "frank", "calin", "david", "john"}
#set of mutual friends with colleague
friends_colleague = {"calin", "ken", "david", "helen"}
#find set of mutual friends
socialmedia_friends.intersection(friends_colleague)
```

```
{'calin', 'david'}
```

You have a set of items in your inventory and another set representing items that need restocking. Find the set of items that are currently in stock and need restocking.

```
items_inventory = {'biscuits','chocolate','cake','toffees','sweets','cookies','namkeens'}
items_restocking = {'chocolate','cake','biscuits','cookies'}
#items that are in stock and need restocking
items_inventory.intersection(items_restocking)
```

```
{'biscuits', 'cake', 'chocolate', 'cookies'}
```

Given a set of languages spoken by employees in a company and another set of languages required for a new project, find the set of languages that are both spoken by employees and required for the project.

```
spoken_languages = {"english", "hindi", "punjabi", "gujrati"}
required_languages = {"marathi", "himachali"}
#both spoken and required languages
spoken_languages.union(required_languages)
```

```
{'english', 'gujrati', 'himachali', 'hindi', 'marathi', 'punjabi'}
```

You have a set of completed tasks and another set of tasks for a project. Find the set of tasks that are yet to be completed.

```
tasks = {'assign team','monitor progress','identify resources','prioritize work','define deadline','manage timelines'}
completed_tasks = {'assign team','identify resources','define deadlines'}
#task yet to be completed
print(f" Tasks yet to be completed are {tasks-completed_tasks}")
```

```
 Tasks yet to be completed are {'prioritize work', 'manage timelines', 'monitor progress', 'define deadline'}
```

Given two sets of preferred travel destinations from two friends, find the set of destinations that both friends prefer.

```
friend_1 = {'leh','manali','gokarna','goa','srinagar'}
friend_2 = {'manali','goa','amritsar','masuri','ooty'}
#preffered by both friends
friend_1.intersection(friend_2)
```

```
{'goa', 'manali'}
```

You have a set of students who have submitted an assignment and another set of students who have attended a class. Find the set of students who have either submitted the assignment or attended the class.

```
submit_assignment = {'helen','david','bob','heena','ishaan'}
attend_class = {'helen','frank','bob','dave','carol'}
#students who has done either of two
submit_assignment.symmetric_difference(attend_class)
```

```
{'carol', 'dave', 'david', 'frank', 'heena', 'ishaan'}
```

Given a set of current employees and another set of new hires, find the set of all employees after the new hires join.

```
current_employees = {"carol","david","helen","dave","bob"}
new_hired = {"ishaan","heena","frank"}
#all employees
print(f" list of all employees {current_employees.union(new_hired)}")
```

```
 list of all employees {'helen', 'bob', 'david', 'heena', 'frank', 'ishaan', 'carol', 'dave'}
```

You have a set of countries visited in your lifetime and another set of countries on your travel wish list. Find the set of countries that you have already visited and are also on your wish list.

```
visited = {'china','russia','france','japan'}
wish_list = {'yemen','algeria','albania'}
#both visited and wish list
visited.union(wish_list)
```

```
{'albania', 'algeria', 'china', 'france', 'japan', 'russia', 'yemen'}
```

Given two sets representing the collection of books in two different libraries, find the set of books that are available in both libraries.

```
library_1 = {"David Copperfield","The Great Gatsby","Don Quixote","Lord of the Rings"}
library_2 = {"Pride and Prejudice","David Copperfield","War and Peace","Don Quixote"}
#present in both libraries
library_1.intersection(library_2)
```

```
{'David Copperfield', 'Don Quixote'}
```

DICTIONARY

Python dictionary is a container of the unordered set of objects like lists. The objects are surrounded by curly braces { }. The items in a dictionary are a comma-separated list of key:value pairs where keys and values are Python data type.

Each object or value accessed by key and keys are unique in the dictionary. As keys are used for indexing, they must be the immutable type (string, number, or tuple). You can create an empty dictionary using empty curly braces. values can be assigned and accessed using square brackets[].

```
dis={'name':'red','age':10}
print(dis)
```

```
{'name': 'red', 'age': 10}
```

```
print(dis['name'])
```

```
print(list(dis.values()))
```

```
print(dis.keys())
```

It is also known as Mapping in Python. As they are primarily used for referencing items by key, they are not sorted.

Rules for creating a dictionary:

1. Every key must be unique (otherwise it will be overridden)
2. Every key must be hashable (can use the hash function to hash it; otherwise TypeError will be thrown).
3. There is no particular order for the keys.

Creating a dict

Dictionaries can be initiated in many ways:

1. literal syntax

```
d = {} # empty
d = {'key': 'value'} # dict with initial values
```

2. dict comprehension

```
d = {k:v for k,v in [('key', 'value',)]}
d = { 1: "Mukta" for k,v in[('key', 'value')]}
d
```

3. built-in class: dict()

```
d = dict() # empty dict
d = dict(key='value') # explicit keyword arguments
d = dict([('key', 'value')]) # passing in a list of key/value pairs
di = dict(Name = 'Diksha')
di
```

```
{'Name': 'Diksha'}
```

```
d = dict([('Name', 'Diksha'),('Age', '21')])
d
```

Built-in Dictionary Methods

In Python Dictionary we have various built-in functions that provide a wide range of operations for working with dictionaries. These techniques enable efficient manipulation, access, and transformation of dictionary data. ￼ Built-in Dictionary Methods

1. Dictionary clear() Method

The clear() method in Python is a built-in method that is used to remove all the elements (key-value pairs) from a dictionary. It essentially empties the dictionary, leaving it with no key-value pairs.

```
my_dict = {'1': 'MUKTA', '2': 'DIKSHA', '3': 'NISHTHA'}
my_dict.clear()
print(my_dict)
```

2. Dictionary get() Method

In Python, the get() method is a pre-built dictionary function that enables you to obtain the value linked to a particular key in a dictionary. It is a secure method to access dictionary values without causing a KeyError if the key isn't present.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(d.get('Name'))
print(d.get('Gender'))
print(d.get('Country'))
```

3. Dictionary items() Method

In Python, the items() method is a built-in dictionary function that retrieves a view object containing a list of tuples. Each tuple represents a key-value pair from the dictionary. This method is a convenient way to access both the keys and values of a dictionary simultaneously, and it is highly efficient.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print((d.items()))
print(list(d.items())[1][1])
print(list(d.items())[1])
```

4. Dictionary keys() Method

The keys() method in Python returns a view object with dictionary keys, allowing efficient access and iteration.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(d.keys())
print(list(d.keys()))
```

5. Dictionary values() Method

The values() method in Python returns a view object containing all dictionary values, which can be accessed and iterated through efficiently.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
print(list(d.values()))
```

6. Dictionary update() Method

Python's update() method is a built-in dictionary function that updates the key-value pairs of a dictionary using elements from another dictionary or an iterable of key-value pairs. With this method, you can include new data or merge it with existing dictionary entries.

```
d1 = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
d2 = {'Name': 'Neha', 'Age': '22'}
d1.update(d2)
print(d1)
```

7. Dictionary pop() Method

In Python, the pop() method is a pre-existing dictionary method that removes and retrieves the value linked with a given key from a dictionary. If the key is not present in the dictionary, you can set an optional default value to be returned

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India'}
d.pop('Age')
print(d)
```

8. Dictionary popitem() Method

In Python, the popitem() method is a dictionary function that eliminates and returns a random (key, value) pair from the dictionary. As opposed to the pop() method which gets rid of a particular key-value pair based on a given key, popitem() takes out and gives back a pair without requiring a key to be specified.

```
d = {'Name': 'Ram', 'Age': '19', 'Country': 'India', 'Gender':'Male'}
d.popitem()
print(d)
```

```
{'Name': 'Ram', 'Age': '19', 'Country': 'India'}
```

```
d.popitem()
print(d)
d.popitem()
print(d)
```

```
{'Name': 'Ram', 'Age': '19'}
{'Name': 'Ram'}
```

Practice Questions(Dictionary)

Create a dictionary named student with keys "name," "age," and "grade," and assign appropriate values.

Print the value associated with the "age" key. Update the "grade" to a new value.

Add a new key-value pair for "subject" and its corresponding value.

```
dict ={"name":"Heer","age":"20","grade":"A"}
print(dict)
print(dict.get("age"))
dict.update(grade="O")
print(dict)
dict.update(Subject="English")
print(dict)
```

```
{'name': 'Heer', 'age': '20', 'grade': 'A'}
20
{'name': 'Heer', 'age': '20', 'grade': 'O'}
{'name': 'Heer', 'age': '20', 'grade': 'O', 'Subject': 'English'}
```

You have a dictionary representing the inventory of a store with item names as keys and quantities as values. Add a new item to the inventory with a specific quantity.

```
inventory_story = {'Apple':'50','Banana':'60','Mango':'40'}
print(inventory_story)
inventory_story.update(Guava='45')
print(inventory_story)
```

```
{'Apple': '50', 'Banana': '60', 'Mango': '40'}
{'Apple': '50', 'Banana': '60', 'Mango': '40', 'Guava': '45'}
```

Given a dictionary of student names and their scores, find the score of a particular student.

```
students_list = {'Helen':'89%','Bob':'86%','Rishab':'95%','Nishu':'92%'}
print(f"The marks is {students_list.get('Bob')}")
```

```
The marks is 86%
```

You have a dictionary of country names as keys and their capitals as values. Update the capital of a specific country.

```
dict ={'Afghanistan':'Kabul','Australia':'Canberra','Bangladesh':'Dhaka'}
print(dict)
dict.update(Bangladesh='DHAKA')
print(dict)
```

```
{'Afghanistan': 'Kabul', 'Australia': 'Canberra', 'Bangladesh': 'Dhaka'}
{'Afghanistan': 'Kabul', 'Australia': 'Canberra', 'Bangladesh': 'DHAKA'}
```

Given a dictionary of product prices with product names as keys, remove a product from the dictionary.

```
dict = {'shampoo':'180rs','soap':'50rs','hair oil':'100rs'}
print(dict)
dict.popitem()
print(dict)
```

```
{'shampoo': '180rs', 'soap': '50rs', 'hair oil': '100rs'}
{'shampoo': '180rs', 'soap': '50rs'}
```

You have two dictionaries representing two different departments with employee names as keys and their salaries as values. Merge these two dictionaries into one.

```
dep_1 ={'Helen':'15000','Bob':'14000'}
dep_2 ={'Nishu':'17000','Ridhi':'13000'}
dep_1.update(dep_2)
dep_1
```

```
{'Helen': '15000', 'Bob': '14000', 'Nishu': '17000', 'Ridhi': '13000'}
```

Start coding or generate with AI.