Name: Abhishek Kanoujia

DATA ENGINEERING BATCH 1

DAY 10 ASSIGNMENT

SET

In Python, a set is a collection of unique and unordered elements that supports mutable operations. The language provides several functions to manipulate sets, allowing users to add, remove, and perform various set operations. Here is a summary of key set functions along with an example:

1.add(): Adds a given element to the set

Python

```
s = {'g', 'e', 'k', 's'}
s.add('f')
print('Set after updating:', s)
```

Output:

```
Set after updating: {'g', 'k', 's', 'e', 'f'}
```

2.discard(): Removes a specified element from the set.

python

```
s.discard('g')
print('\nSet after updating:', s)
```

Output:

```
Set after updating: {'k', 's', 'e', 'f'}
```

3.remove(): Removes a specified element from the set.

python

```
s.remove('e')
print('\nSet after updating:', s)
```

Output:

```
Set after updating: {'k', 's', 'f'}
```

4.pop(): Returns and removes a random element from the set.

python

```
print('\nPopped element', s.pop())
print('Set after updating:', s)
```

Output:

```
Popped element k
Set after updating: {'s', 'f'}
```

5.clear(): **Removes all elements from the set.**

python

```
s.clear()
print('\nSet after updating:', s)
```

Output:

```
Set after updating: set()
```

Additionally, the table provides a comprehensive list of other set methods like difference(), intersection(), union(), and more. These methods enable users to perform set operations efficiently, making sets a powerful tool for handling unique collections in Python.

Unique Values from a List Using Set Method

In Python, there are several methods to obtain unique values from a list. Here are various approaches along with examples:

1.Using Set Method:

- Convert the list to a set using set().
- Convert the set back to a list to print unique values.

python

```
def unique(list1):
    list_set = set(list1)
    unique_list = list(list_set)
    for x in unique_list:
        print(x)

list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

Unique values from 1st list:

40 10 20 30

Unique values from 2nd list:

1 2 3 4 5

2.Using reduce() function:

- Utilize reduce() from functools to iterate over the list and filter out duplicates.

python

```
from functools import reduce


def unique(list1):
    ans = reduce(lambda re, x: re + [x] if x not in re else re, list1, [])
    print(ans)


list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
unique(list1)


list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

Unique values from 1st list:

[10, 20, 30, 40]

Unique values from 2nd list:

[1, 2, 3, 4, 5]

3.Using Operator.countOf() method:

- Employ the op.countOf() method to check for duplicates.

python

```
import operator as op

def unique(list1):
    unique_list = []
    for x in list1:
        if op.countOf(unique_list, x) == 0:
            unique_list.append(x)
    for x in unique_list:
        print(x)

list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
unique(list1)
```

```
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

Unique values from 1st list:

10 20 30 40

Unique values from 2nd list:

1 2 3 4 5

4.Using pandas module:

- Utilize the pandas library to create a Series and then use drop_duplicates().

python

```
import pandas as pd


def unique(list1):
    unique_list = pd.Series(list1).drop_duplicates().tolist()
    for x in unique_list:
        print(x)

list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
```

```
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

```
Unique values from 1st list:
10 20 30 40
Unique values from 2nd list:
1 2 3 4 5
```

5.Using numpy.unique():

- Use numpy to obtain unique elements from the list.

python

```
import numpy as np

def unique(list1):
    x = np.array(list1)
    print(np.unique(x))

list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
unique(list1)
```

```
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

```
Unique values from 1st list:
[10 20 30 40]
Unique values from 2nd list:
[1 2 3 4 5]
```

6.Using collections.Counter():

- Import Counter() from collections to directly print unique keys.

python

```
from collections import Counter

def unique(list1):
    print(*Counter(list1))

list1 = [10, 20, 10, 30, 40, 40]
print("Unique values from 1st list:")
unique(list1)

list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]
```

```
print("\nUnique values from 2nd list:")
unique(list2)
```

Output:

```
Unique values from 1st list:
10 20 30 40
Unique values from 2nd list:
1 2 3 4 5
```

7.Using dict.fromkeys():

- Use fromkeys() method to fetch unique elements from the list.

python

```
list1 = [10, 20, 10, 30, 40, 40]
list2 = [1, 2, 1, 1, 3, 4, 3, 3, 5]

unique_list_1 = list(dict.fromkeys(list1))
unique_list_2 = list(dict.fromkeys(list2))

print("Unique values from 1st list:", unique_list_1)
print("Unique values from 2nd list:", unique_list_2)
```

Output:

```
Unique values from 1st list: [10, 20, 30, 40]
Unique values from 2nd list: [1, 2, 3, 4, 5]
```

These methods provide flexibility in choosing the approach that best suits the requirements of the specific use case. Each method has its own time and space complexity considerations, so the choice depends on the characteristics of the data and the desired outcome.

JSON

JSON (JavaScript Object Notation) is a widely used data interchange format, replacing XML for its simplicity and efficiency. It facilitates easy data structuring with support for arrays and objects. As a language-independent format derived from JavaScript, JSON is employed by various web applications, allowing rapid execution on servers. Its official media type is application/json, and files are typically saved with a .json extension.

Features:

- **Ease of Understanding**: JSON is easy to read and write.
- **Format:** It is a text-based interchange format supporting diverse data types.
- **Wide Support:** JSON is lightweight, supported by various languages, and compatible with different operating systems and browsers.
- **Independence:** As a text-based, independent language, JSON offers faster execution compared to other structured data formats.

JSON Syntax Rules:

- Data is organized in name/value pairs, separated by commas.
- Curly brackets hold objects, and square brackets hold arrays.

Example:

json

```
{
  "Courses": [
    {
      "Name": "Java Foundation",
      "Created by": "Hexa",
      "Content": ["Java Core", "JSP", "Servlets", "Collections"]
    },
    {
      "Name": "Data Structures",
      "also known as": "Interview Preparation Course",
      "Topics": ["Trees", "Graphs", "Maps"]
    }
  ]
}
```

Advantages:

- JSON simplifies data transfer, supporting various data types.
- Its syntax is concise and lightweight, ensuring fast execution and response.
- Wide compatibility across browsers and operating systems.

Disadvantages:

- Lack of error handling in JSON.
- Potential security risks when used with unauthorized browsers.
- Limited supported tools during JSON development.
- manipulated within the Python environment.

JSON in Python:

In Python, the JSON module facilitates working with JSON data. The json.load() function parses JSON from a file, while json.loads() parses JSON from a string. JSON data can be converted to Python dictionaries or vice versa.

Example:

python

```
import json


# JSON string
employee = '{"id":"09", "name": "Nitin", "department":"Finance"}'
```

 # Convert string to Python dict

employee_dict = json.loads(employee)

 # Print Dictionary

print(employee_dict)

 # Print values using keys

print(employee_dict['name'])

JSON File Handling in Python:

Reading JSON file using json.load().

Writing JSON to a file using json.dump().

Nested JSON:

JSON supports nested structures, allowing for complex data representation. Nested JSON objects can be accessed and manipulated in Python.

Pretty Print JSON:

The json.dumps() function supports pretty printing by adding indentation and sorting keys, making the JSON format more readable.