Kmit

SONET

Data Science

(Level-I)

Ajeet K. Jain, M. Narsimlu

(ML TEAM)- SONET, KMIT, Hyderabad

Session - 15

This session deals with

- Sets
- Operators Vs Methods
- Sets Methods
- Tuple Vs Sets
- Dictionary
- Exercises on Dictionary
- Numpy

Sets

A set is an unordered collection of items.

Every element is unique (no duplicates) and must be immutable (which cannot be changed)..

We can add or remove items from it.

Sets can be used to perform mathematical set operations like union, intersection, symmetric difference etc.

A set is created by placing all the items (elements) inside curly braces {}, separated by comma or by using the built-in function set()

```
courses = set(['java', 'Data', 'science', 'java', 'Data'])
print(courses)
```

Output

{'science', 'Data', 'java'}

Sets

```
courses = set(('java', 'Data', 'science', 'java', 'Data'))
print(courses)
```

Output

{'science', 'Data', 'java'}

```
course="Data Science"
print(set(course))
```

Output

{' ', 'c', 'n', 'D', 'i', 't', 'S', 'a', 'e'}

```
courses={"java","python","python","java","Data Science"}
print(courses)
```

{'python', 'java', 'Data Science'}

Operators Vs methods

set operations in Python can be performed in two different ways: by operator or by method.

```
course1 = {'java', 'Data', 'science', 'java', 'Data',"python"}
course2={"Data","science","java","Data","c"}
print(course1|course2)
Print(course1.union(course2))
```

Output

```
{'Data', 'python', 'science', 'c', 'java'}
{'Data', 'python', 'science', 'c', 'java'}
```

Intersection

set operations in Python can be performed in two different ways: by operator or by method.

```
course1 = {'java', 'Data', 'science', 'java', 'Data',"python"}
course2={"Data","science","java","Data","c"}
print(course1&course2)
Print(course1.intersection(course2))
```

Output

```
{'java', 'Data', 'science'}
{'java', 'Data', 'science'}
```

Difference

set operations in Python can be performed in two different ways: by operator or by method.

```
course1 = {'java', 'Data', 'science', 'java', 'Data',"python"}
course2={"Data","science","java","Data","c"}
print(course1-course2)
Print(course1.difference(course2))
```

Output

```
{'python'}
{'python'}
```

Sets-Methods

Add()-	adding the elements to the list
Update()-	updating existing set
Remove()-	To remove particular element from the set
Clear()-	To clear the set
Pop()-	To remove top most element from the set

Sets-Methods


```
course1={30}
course1.add(10)
course1.add(20)
print(course1)
course1.update((45,90,125))
print(course1)
course1.remove(90)
print(course1)
course1.pop()
print(course1)
course1.clear()
print(course1)
```

Output

```
{10, 20, 30}
{10, 45, 20, 90, 125, 30}
{10, 45, 20, 125, 30}
{45, 20, 125, 30}
set()
```

Tuple Vs Sets

1. **Tuple** -
A tuple is basically an immutable list, it means you can't add, remove, or replace any elements, once you declare it ,tuples are hashable.
slicing and dicing in tuples works exactly like list

Set -
A set is like a array of unique elements and its has no order . but has the advantage over a list and tuples that it is much faster, almost regardless of the size of the set. It also has some handy operations such as union,difference and intersection.
The elements of a set is not hashable.

Dictionaries

- A dictionary is a data structure.
- A dictionary is a collection which is
  - unordered,
  - indexed.
  - Changeable or updated and
- In Python dictionaries are written with curly brackets { },
- and every element is a
  - key and value pair.
    - 1: “First”
    - “B”:200

Dictionaries

- In dictionary the values are accessed using key rather than index
- 1. Create a dictionary:
- myDict ={'a':"apple",'b':'boy',3:'third class','d':400}
- A dictionary is created.
- This dictionary contains three elements.
- Each element constitutes of a **key (A) value (Apple)** pair.

This dictionary can be accessed using the dictionary identifier myDict.

```
 print(myDict)
{'a': 'apple', 'b': 'boy', 3: 'third class', 'd': 400}
```

Dictionaries

- 2. Access Dictionary Elements
- Once a dictionary is created, you can access each value using the key to which it is assigned during creation.

>>> myDict[‘a’]

‘apple'

>>> myDict[“b"]

‘boy'

Dictionaries

- To access all **key** : **values** the variable myDict can be used to access the dictionary elements.
- >>> myDict
- {'a':"apple",'b':'boy',3:'third class','d':400}
- Only dictionary keys can be used as indexes.
- This means that myDict[‘a’] would produce ‘apple’ in output but myDict[‘apple’] cannot produce ‘a’ in the output.

>>> myDict["Apple"]

Traceback (most recent call last): File "<stdin>", line 1, in <module>

KeyError: 'Apple'

Dictionaries

3. Update Dictionary Elements

- Just the way dictionary values are accessed using keys, the values can also be modified using the dictionary keys:
- >>> myDict[‘a’] = "Application"
- >>> myDict
- {‘a': 'Application', 'b':'boy',3:'third class','d':400}

- **Note:-** In a dictionary two keys cannot be same.
- >>> mydict = {**'a'**:‘Application',**'a'** : ‘App'}
- >>> mydict
- {'a': 'app'}

Dictionaries

4. Delete Dictionary Elements

- Individual elements can be deleted easily from a dictionary.
- We need to use **del** and the key of value to be deleted:-
- >>> myDict
- {‘a': 'Application', 'b':'boy',3:'third class','d':400}
- >>> del myDict[“a"]
- >>> myDict
- {'b': 'boy', 3: 'third class', 'd': 400}

Dictionaries

- If you want to delete all the elements in the dictionary then it can be done using the clear() function. Here is an example :
- >>> myDict
- {'C': 'Cat', 'B': 'Boy'}
- >>> myDict.clear()
- >>> myDict
- {}
- All the elements were deleted making the dictionary empty.

Dictionaries

- Even if you try to add other elements to python dictionary:
- >>> myDict["D"] = "Dog"
- >>> myDict
- {'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'D': 'Dog'}
- >>> myDict["E"] = "Elephant"
- >>> myDict
- **{'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog'}**
- You'll observe that it's not necessary that elements will be stored in the same order in which they were created.

Dictionaries

2. Dictionary Keys are Case Sensitive

- Same key name but with different case are treated as different keys in python dictionaries.
- Here is an example :
- >>> myDict["F"] = "Fan"
- >>> myDict["f"] = "freeze"
- >>> myDict
- **{'A': 'Apple', 'C': 'Cat', 'B': 'Boy', 'E': 'Elephant', 'D': 'Dog', 'F': 'Fan', 'f': 'freeze'}**

Dictionary and for iterator

```
'''
for iterator and dictionary
Using for iterator print all the key and value elements
d = {'x': 1, 'y': 2, 'z': 3}
'''
d = {'x': 1, 'y': 2, 'z': 3}
for key in d:
    print (key, 'corresponds to', d[key])
```

x corresponds to 1
y corresponds to 2
z corresponds to 3

Dictionaries

- In Python "None" is a special value like null or nil.
- get() returns None as default value if no value is found in a dictionary.
- We can use the get() method with one or two arguments. This does not cause any annoying errors. It returns None.
- **Argument 1:**The first argument to get() is the key you are testing. This argument is required.
- **Argument 2:**The second, optional argument to get() is the default value. This is returned if the key is not found.

Dictionaries


```
#Declare plants as dictionary
plants = {}
# Add three key-value tuples to the dictionary.
plants["radish"] = 2
plants[“onion"] = 4
plants["carrot"] = 7
# get syntax 1.
print(plants["radish"])
# Get syntax 2.
print(plants.get(“beans"))
print(plants.get(" beans ", "no beans found"))
print(plants.get("carrot", "Yes carrot found"))
```

```
2
None
no beans found
7
```

Dictionary-Methods

items()-	returns view of dictionary's (key, value) pair
Update()-	updating existing set
Get() -	Returns Value of The Key
Pop()-	To remove top most element from the set
Keys()-	Returns View Object of All Keys
Values()-	returns view of all values in dictionary
Dict()-	Creates a Dictionary
Clear()-	To clear the set
Zip()-	Returns an Iterator of Tuples

Exercise-1


```
released={"iphone":2007,
          "iphone 3G":2008,
          "iphone 3GS":2009,
          "iphone 4":2010
              }
for k,v in released.items():
    print(k,"->",v)
released.update({"iphone4G":2011,
"iphone4S":2015})
print(released)
print(released.get("iphone4G"))
released.pop("iphone4S")
print(released)
print(released.keys())
released.clear()
print(released)
```

Output

```
iphone -> 2007
iphone 3G -> 2008
iphone 3GS -> 2009
iphone 4 -> 2010
{'iphone': 2007, 'iphone 3G': 2008, 'iphone
3GS': 2009, 'iphone 4': 2010, 'iphone4G':
2011, 'iphone4S': 2015}
2011
{'iphone': 2007, 'iphone 3G': 2008, 'iphone
3GS': 2009, 'iphone 4': 2010, 'iphone4G':
2011}
dict_keys(['iphone', 'iphone 3G', 'iphone
3GS', 'iphone 4', 'iphone4G'])
{}
```

ZIP function-Exercise-2

The zip() function take iterables (can be zero or more), makes iterator that aggregates elements based on the iterables passed, and returns an iterator of tuples.

```
names = [“sree”, “Ram”, “Sri”]
pwds = [[“sree123”,”sree321”],[“Ram12”,”Ram21”], [“Sri34”,”Sri43”]]

# Two iterables are passed
result = zip(names, pwds)
print(dict(result))
```

Output:

```
{'sree': ['sree123', 'sree321'], 'Ram': ['Ram12', 'Ram21'], 'Sri':
['Sri34', 'Sri43']}
```

ZIP function


```
names = ["sree", "Ram", "Sri"]
pwds = [["sree123","sree321"],["Ram12","Ram21"], ["Sri34","Sri43"]]
# Two iterables are passed
result = zip(names, pwds)
print(list(result))
```

Output:

```
[('sree', ['sree123', 'sree321']), ('Ram', ['Ram12', 'Ram21']),
('Sri', ['Sri34', 'Sri43'])]
```

ZIP function


```
names = ["sree", "Ram", "Sri"]
pwds = [["sree123","sree321"],["Ram12","Ram21"], ["Sri34","Sri43"]]
# Two iterables are passed
result = zip(names, pwds)
print(tuple(result))
```

Output:

```
(('sree', ['sree123', 'sree321']), ('Ram', ['Ram12', 'Ram21']),
('Sri', ['Sri34', 'Sri43']))
```

Exercise-3

- Given a string,write a python script to that counts the frequency of words and digits. Use a python dictionary to store the words and the digits, and then display their frequency in ascending order.
- wordstring = 'it was the best of times it was the worst of times '
- wordstring += 'it was the age of wisdom it was the age of foolishness‘

Solution

- wordstring = 'it was the best of times it was the worst of times '
- wordstring += 'it was the age of wisdom it was the age of foolishness‘
- wordlist = wordstring.split()
- wordfreq = []
- print("String\n" + wordstring +"\n")
- print("List\n" + str(wordlist) + "\n")
- wordfreq = [wordlist.count(p) for p in wordlist]
- freqdict=dict(zip(wordlist,wordfreq))
- aux = [(key,freqdict[key]) for key in freqdict]
- aux.sort()
- print(aux)
- aux.reverse()
- print(aux)

Exercise-4

•**Given the following dictionary:**

```
inventory = {
    'gold' : 500,
    'pouch' : ['flint', 'twine', 'gemstone'],
    'backpack' : ['xylophone','dagger', 'bedroll','bread loaf‘] }
```

Try to do the followings:

- Add a key to inventory called 'pocket'.
- Set the value of 'pocket' to be a list consisting of the strings 'seashell', 'strange berry', and 'lint'.
- .sort()the items in the list stored under the 'backpack' key.
- Then .remove('dagger') from the list of items stored under the 'backpack' key.
- Add 50 to the number stored under the 'gold' key.

Solution


```
inventory = {
    'gold' : 500,
    'pouch' : ['flint', 'twine', 'gemstone'],
    'backpack' : ['xylophone','dagger', 'bedroll','bread loaf']}
inventory.update({"packet":["seashell","strangeberry","lint"]})
sort_back=sorted(inventory.get("backpack"))
print(sort_back)
sort_back.remove("dagger")
print(sort_back)
#Add 50 to the number stored under the 'gold' key.
inventory["gold"]=inventory["gold"]+50
print(inventory["gold"])
```

Conclusion

You are aware of

List Data Structures

Tuple

We will proceed with

Numpy

Kmit

SONET

DATA SCIENCE

THANK YOU