Department of Computer Science and Engineering
PES University, Bangalore, India

Lecture Notes
Python for Computational Problem Solving
UE23CS151A

Lecture #36

Set and it's operations

By,
Prof. Sindhu R Pai,
Anchor, PCPS - 2023
Assistant Professor
Dept. of CSE, PESU

Verified by,
PCPS Team - 2023

Many Thanks to
Dr. Shylaja S S (Director, CCBD and CDSAML Research Centers, Former Chairperson, CSE, PES University)
Prof. Chitra G M(Asst. Prof, Dept. of CSE, PCPS Anchor – 2022)

Sets in Python

Introduction

Set is a **Non-primitive Non-Linear (Non-sequence type) Data Structure**. It is an unordered **collection of unique elements**. Let us try to understand why set is required through an example.

The requirement is to find the set of students who are actively participating in all the hackathons conducted by the department. Rather than storing these details in the list which allows duplicate entries, we can store the details of students for each hackathon in different sets. Thus mathematical set for each hackathon has the details of students within it. Thus performing any of these operations like intersection, union, difference etc. we can extract the required set of students details. Thus, sets are usually helpful in removing duplicate elements; in turn find the unique elements. Also used for comparing two iterable for common elements or differences.

Characteristics of Sets:

- It has **0 or more elements.**
- **All elements are of type Immutable and hence Hashable.**
- It does not allow duplicates.
- There is no **name for each element** in a set separately.
- **Elements** in the set can be **heterogeneous** in nature.
- Set is **non indexable** - Elements **cannot be accessed using indexing operation.**
- Set is **mutable** - Can grow or shrink as its size is not fixed.
- Set is **iterable –** Can get one element at a time. It is eager and not lazy.
- It has number of **built-in functions to manipulate itself.**

Syntactically, elements of the set must be inside **flower brackets(curly braces) – {** and **}** and **all elements must be separated by a comma** between each of them .

FYR: Hashing is a mechanism to convert the given element in to an integer. An object is hashable if it has a hash value which never changes during its lifetime.

- **Hashable objects:** int, float, complex, bool, string, tuple, range, frozenset, bytes, decimal .
- **Unhashable objects:** list, dict, set, bytearray

A. Creation of set variables

Example 1:

```
set1 = {(1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True}
                    #can have heterogeneous type of elements in it.
print(type(set1))   #  <class 'set'>
```

Example 2:

```
fruits = { “apple”, “orange”, “kiwi”, “banana”, “cherry”}
print(type(set1))   #  <class 'set'>
```

```
>>> s1 = {23,12,[33,66,11],3+6j}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> s1 = {23,12,(33,"pes",11),{34.8, "sindhu"}}
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
>>> s3 = {(1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True}
>>> s3
{True, 2, (3+5j), (1, 'a', 'hi'), 5.56, 'hello'}
>>> s4 = set()
>>> type(s4)
<class 'set'>
>>> type(s3)
<class 'set'>
>>>
```

Note: Empty set can be created using set() # constructor function

Observe that when you print the set variable, the order is not same as how you have given in the creation of set variable.

B. Accessing the elements of the set

The set variable is not indexable but iterable. To access each element of the set individually, we can use the for loop.

```
>>> s1 = {23,11,90,"pes", "cse",(23.6,33)}
>>> s1
{'pes', 23, 'cse', (23.6, 33), 90, 11}
>>> for i in s1:
...     print(i)
...
pes
23
cse
(23.6, 33)
90
11
>>>
```

```
s1 = {23,11,90,"pes", "cse",(23.6,33)}
print(s1)
for i in s1:
    print(i)
```

```
{23, 'cse', (23.6, 33), 90, 11, 'pes'}
23
cse
(23.6, 33)
90
11
pes
```

Note: Index should never be used to access the elements of the set. Results in Error

```
>>> s1 = {23,11,90,"pes", "cse",(23.6,33)}
>>> s1[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>> s1[5]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
>>>
```

Common Operations on Sets:

- len(): Returns the number of items in a container set.
- min(): Returns the smallest item in a container set.
- max(): Returns the largest item in a container set.
- sum(): Displays the sum of all the items from a container set.
- …

```
>>> s1 = {6,5,7,8,9,10}
>>> s1
{5, 6, 7, 8, 9, 10}
>>> len(s1)
6
>>> min(s1)
5
>>> max(s1)
10
>>> sum(s1)
45
>>>
```

```
>>> s2 = {6,5,7,8,9,'10'}
>>> len(s2)
6
>>> min(s2)
Traceback (most recent call
    last):
  File "<stdin>", line 1, in
    <module>
TypeError: '<' not supported
    between instances of 'int'
    and 'str'
>>> sum(s2)
```

C. Common operators on sets

- | operator: This union operator is used to combine two sets together, results in a set that contains all items from the given two sets.
- & operator: This Intersection operator is used to combine the common elements which are there in both the sets.
- - operator: Difference Operator(Elements which are there only in first set)
- ^ operator: Symmetric Difference Operator(Elements which are there only in set1 union elements which are there only in set2)

```
>>> s1 = {2,4,6,"pes"}
>>> s2 = {2,8,10,"python"}
>>> s1
{2, 4, 6, 'pes'}
>>> s2
{8, 2, 10, 'python'}
>>> s1 | s2
{2, 4, 6, 8, 10, 'python', 'pes'}
>>> s1 & s2
{2}
>>> s1 - s2     #elements which are there only in s1
{4, 6, 'pes'}
>>> s1 ^ s2     # (s1 – s2) | (s2-s1)
{4, 6, 8, 10, 'python', 'pes'}
>>>
```

Note: Usage of + operator and * doesn't make any sense on sets.

Relational Operators on sets: ==, !=, >, >=, <=

== checks if both the set operands have the same elements

!= Checks if both the set operands have different elements

<, <=, >, >= compares two sets to determine their superset or subset relationship

<= operator: Subset test operation. S1 is a subset of S2 if every element of S1 is in S2.

< operator: Proper subset test operation. S1 is a proper subset of S2 if every element of S1 is in S2 and S2 has at least one additional element, not in S1.

Same for super set and proper super set operators

```
>>> s1 = {31,44,"pes"}
>>> s2 = {"pes",31,44}
>>> s3 = {"pes",44}
>>> s1
{'pes', 44, 31}
>>> s2
{'pes', 44, 31}
>>> s3
{'pes', 44}
>>> s1 == s2
True
>>> s1 == s3
False
>>> s1 != s3
True
```

```
>>> s1
{'pes', 44, 31}
>>> s2
{'pes', 44, 31}
>>> s3
{'pes', 44}
>>> s1 <s2
False
>>> s1 <= s2
True
>>> s2 >= s2
True
>>> s2 > s2
False
>>> s2 >= s2
True
>>>
```

Membership Operators – in, not in

```
>>> s1
{'pes', 44, 31}
>>> s2
{'pes', 44, 31}
>>> "pes" in s1
True
>>> "peS" in s1
False
>>> "pesu" not in s2
True
>>>
```

D. Modifying the elements of the set

All specific functions of set can be listed using dir() function.

```
C:\Users\Dell>python
Python 3.11.3 (tags/v3.11.3:f3909b8, Apr  4 2023, 23:49:59) [MSC v.1934 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> dir(set)
['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__
eq__', '__format__', '__ge__', '__getattribute__', '__getstate__', '__gt__', '__hash__', '__iand__', '
__init__', '__init_subclass__', '__ior__', '__isub__', '__iter__', '__ixor__', '__le__', '__len__', '_
_lt__', '__ne__', '__new__', '__or__', '__rand__', '__reduce__', '__reduce_ex__', '__repr__', '__ror__
', '__rsub__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__x
or__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersec
tion_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmet
ric_difference_update', 'union', 'update']
>>>
```

add()- Inserts an element to the container set.

```
>>> s1 = set()
>>> type(s1)
<class 'set'>
>>> s1.add(12)
>>> s1.add("pes")
>>> s1.add((23,55,77))
>>> s1
{(23, 55, 77), 12, 'pes'}
>>> len(s1)
3
>>> s1.add(12)
>>> len(s1)
3
>>> s1.add({22,55})                    # mutable types cannot be the elements of a set
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

update()- Updates the current set by adding items from any iterable.

```
>>> s1
{(23, 55, 77), 12, 'pes'}
>>> s1.update({2,6,1})
>>> s1
{1, 2, 6, (23, 55, 77), 12, 'pes'}
>>> s1.update(99)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
>>> s1.update("pes")
>>> s1
{1, 2, 6, 12, 'pes', 'p', (23, 55, 77), 's', 'e'}
```

```
>>> s1.update([12,99])
>>> s1
{1, 2, 6, 12, 'pes', 'p', (23, 55, 77), 99, 's', 'e'}
>>> s1.update({"a":"apple"})                # only key added to set from this dictionary
>>> s1
{1, 2, 6, 12, 'pes', 'p', (23, 55, 77), 'a', 99, 's', 'e'}
>>>
```

remove()- Removes the specified item in a set. If the item to remove does not exist, remove() will raise an error.

```
>>> s1
{1, 99, 's', 'e'}
>>> s1.remove()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set.remove() takes exactly one argument (0 given)
>>> s1.remove(99)
>>> s1.remove("99")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: '99'
>>> s1
{1, 's', 'e'}
>>> s1.remove(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 2
>>>
```

discard()- Removes the specified item in a set. If the item to remove does not exist, discard() will NOT raise an error.

```
>>> s1
{1, 'pes', 's', 'e', 'python'}
>>> s1.discard('s')
>>> s1
{1, 'pes', 'e', 'python'}
>>> s1.discard('s')
>>> s1
{1, 'pes', 'e', 'python'}
>>> s1.discard(90)
>>> s1
```

```
{1, 'pes', 'e', 'python'}
>>>
```

pop() - Removes an item from a set and returns that item. Sets are unordered, so when pop() function is used, we cannot decide which element is removed.

```
>>> s1
{ 2, 6, 12, 'pes', (23, 55, 77), 'a', 99, 's', 'e'}
>>> s1.pop()
2
>>> s1.pop()
6
>>> s1.pop()
12
>>> s1.pop()
'pes'
>>> s1.add(1)
>>> s1.pop()
(23, 55, 77)
>>> s1
{1, 'a', 99, 's', 'e'}
>>> s1.pop()
'a'
>>> s1.pop('s')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: set.pop() takes no arguments (1 given)
>>>
```

clear() - Removes all the elements in a set.

```
>>> s1 = {2,4,6,1}
>>> s1.add("pes")
>>> s1
{1, 2, 4, 6, 'pes'}
>>> id(s1)
1944174013888
>>> s1.clear()
>>> id(s1)
1944174013888
>>> s1
set()
s>>>
```

Note: Above functions prove the mutable property of sets

The operators |, &, ^ and – does different operations same as below functions. These functions return a new set.

```
>>> s1 = {11,44,66,12}
>>> s2 = {12,"pes", 67,11}
>>> s1
{66, 11, 44, 12}
>>> s1.union(s2)
{66, 67, 11, 44, 12, 'pes'}
>>> s1
{66, 11, 44, 12}
>>> s1.intersection(s2)
{11, 12}
>>> s1
{66, 11, 44, 12}
```

```
>>> s1.difference(s2)
{66, 44}
>>> s1
{66, 11, 44, 12}
>>> s1.symmetric_difference(s2)
{66, 67, 44, 'pes'}
>>> s1
{66, 11, 44, 12}
>>>
```

There are a few functions which does operation and updates the set object calling the function

```
>>> s1
{66, 11, 44, 12}
>>> s2
{67, 11, 'pes', 12}
>>> s1.intersection_update(s2)
>>> s1
{11, 12}
>>> s1 = {66, 11, 44, 12}
>>> s1.difference_update(s2)
>>> s1
{66, 44}
>>>
```

Few functions for practice are here: isdisjoint(), issubset(), issuperset() and copy()

Points to think!

- **Can a set contain set as an element?**
- **What is the error if we try to access the set elements using the index?**
- **Is set an iterable? If yes, how to find the length of it?**
- **If you keep on adding the duplicate elements to the set using add function, will that throw any error?**

- END -