



PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Sets in Python

Prof. Sindhu R Pai

PCPS Theory Anchor - 2024

Department of Computer Science and Engineering

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Introduction



Definition:- A set is an unordered collection of Unique elements with zero or more elements present with the following attributes.

- In Python **sets are written with curly braces.**
- **Constructor set()** is used to create an empty set.

Examples :

- `Empty_set=set()` # Use the **constructor set()**.
- `fruitset = { "apple", "orange", "kiwi", "banana", "cherry" }`
- `set1 = {(1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True}`

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING



Characteristics

- **Mutable:** Modifiable
- **Unordered** – Order of elements in a set need not be same as the order in which we add or delete elements to/from the set.
- **Iterable** – A container object capable of returning its members one at a time. Set is eager and not lazy.
- **Not indexable** - Cannot access items in a set by referring to an index, since sets are unordered in nature.
- Check for **membership** using the **in operator is faster** in case of a set compared to a list, a tuple or a string.
- **Sets are used to :**
 - o Remove the duplicate elements, inturn allowing us to find the unique elements.
 - o Compare two iterables for common elements or difference.

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING



Characteristics

- Elements are **unique** – does not support repeated elements.
- Elements should be **hashable**.
- Hashing is a mechanism to convert the given element into an integer.
- An object is hashable if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method).

Examples of hashable objects:

- int, float, complex, bool, string, tuple, range, frozenset, bytes, decimal.

Examples of Unhashable objects:

- list, dict, set, bytearray

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Examples



- `#set1={ [1, 'a', 'hi'], 2, 'hello', {3, 4.5, 'how r u' } }`
#TypeError: unhashable type: 'list'
- `#set2={ (1, 'a', 'hi'), 2, 'hello', {1:'hi', 2:'hello', 3:'how r u'} }`
#TypeError: unhashable type: 'dict'
- `set3={ (1, 'a', 'hi'), 2, 'hello', 5.56, 3+5j, True }`
`print(set3)`
#{True, (1, 'a', 'hi'), 2, 5.56, (3+5j), 'hello'}
- `my_dict = {'name': 'John', tuple([1,2,3]):'values'}`
`print(my_dict)`
#{'name': 'John', (1, 2, 3): 'values'}

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Common methods of Set



There are number of **functions built into Python** that takes **set** as the arguments.

- **len()**
- **sum()**
- **sorted()**
- **max()**
- **min()**

Note: Check reversed()

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific methods of Set



The **dir()** function returns all properties and methods of the specified object.

- `>>> dir(set)`
- `['__and__', '__class__', '__class_getitem__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__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__', '__xor__', 'add', 'clear', 'copy', 'difference', 'difference_update', 'discard', 'intersection', 'intersection_update', 'isdisjoint', 'issubset', 'issuperset', 'pop', 'remove', 'symmetric_difference', 'symmetric_difference_update', 'union', 'update']`

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Methods in detail



len()	Returns the number of elements in a container set.	>>> set1={22,33,11,55,44,120} >>> len(set1) 6
max()	Returns the element from Set with maximum value	>>> set1={42,73,25,75,64,145} >>> max(set1) 25
min()	Returns the element from Set with maximum value	>>> set1={25,43,21,66,45,120} >>> min(set1) 21
sum()	Returns the sum of elements of the Set.	>>> set1={4,10,2,7,25} >>> sum(set1) 48

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Common methods of Set



sorted()	Returns the sorted sequence or sorted collection in the form of list.	<pre>>>> set1={22,33,11,55,44,120} >>> sorted(set1) [11, 22, 33, 44, 55, 120] >>> s4={"pes",0,1} >>> sorted(s4) Traceback (most recent call last): File "<stdin>", line 1, in <module> TypeError: '<' not supported between instances of 'str' and 'int'</pre>
----------	-----------------------------------------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Common operators applied on set



Concatenation (|=) operator can be used to combine two sets together, results in a set that contains items of the two sets.

```
>>> s1={1,2.34,(34,22),"python"} >>> s2={10,3.22,"programming"}
>>> s1|=s2
>>> s1
{1, 2.34, 'python', (34, 22), 3.22, 'programming', 10}
```

Repetition operator(*) – Throws an error as set has unique elements.

```
>>> s1={1,2,3,4,5,6}
>>> print(s1*3)
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for *: 'set' and 'int'

•

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Common operators applied on set



Something to find out:

Why are we using the `|=` operator here?

We used the `+` operator for lists and tuples, so why not use that here?

What's the difference between `+`, `|`, and `|=`?

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Common operators applied on set



Membership operator(in) – returns True if a sequence with the specified value is present in the string otherwise False.

```
>>> s1={1,2,3,4,5,6}
```

```
>>> print(1 in s1)
```

```
True
```

```
>>> print('a' in s1)
```

```
False
```

Membership(not in) operator- Returns **True** if the element is not present in the set, else return **False**.

```
>>> s1={1,2.34,(34,22),"python"}
```

```
>>> "PYTHON" not in s1
```

```
True
```

```
>>> (34,22) not in s1
```

```
False
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



union()- Return the union of sets as a new set. (i.e. all elements that are in either set.) .

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s2={6,5,7,8,9,10}
```

```
>>> print(s1.union(s2))
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

or

```
>>> print(s1|s2)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

intersection()- Return the intersection of sets as a new set. (i.e. all elements that are in both sets.) .

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s2={6,5,7,8,9,10}
```

```
>>> print(s1.intersection(s2))
```

```
{5, 6}
```

or

```
>>> print(s1&s2)
```

```
{5, 6}
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



difference()- Return the difference of two or more sets as a new set.

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s2={6,5,7,8,9,10}
```

```
>>> print(s1.difference(s2))
```

```
{1, 2, 3, 4}
```

or

```
>>> print(s1-s2)
```

```
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
```

add()- adds an element in a container set.

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s1.add(7)
```

```
>>> s1
```

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

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s1.add("pes")
```

```
>>> s1
```

```
{1, 2, 3, 4, 5, 6, "pes"}
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



symmetric_difference()- Return a set with the symmetric differences of two sets.

```
>> s1={1,2,3,4,5,6}
>>> s2={6,5,7,8,9,10}
>>> print(s1.symmetric_difference(s2))    or    >>> print(s1^s2)
{1, 2, 3, 4, 7, 8, 9, 10}                  {1, 2, 3, 4, 7, 8, 9, 10}
```

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

```
>>> s1={1,2,3,4,5,6}
>>> s1.remove(5)           or    >>> s1.remove(7)
>>> print(s1)              Displays an error
{1, 2, 3, 4, 6}
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



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

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s1.remove(6)
```

or

```
>>> s1.discard(7)
```

Doesn't display an Error.

```
>>> print(s1)
```

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

pop()- Removes an item in a set. Sets are unordered, so when using the pop() method, doesn't know which item that gets removed.

```
>>> s1={1,2,3,4,5,6}
```

```
>>> s1.pop()
```

```
1
```

```
>>> print(s1)
```

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


PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



update()- updates the current set, by adding items from another set (or any other iterable). If the element is present in both the sets, only one appearance of element present in updated set

```
>>> s1={120,12,23}
>>> s1.update([22,15,67,"pes"])
>>> s1
{67, 12, 15, 22, 23, 120, 'pes'}
```

intersection_update()- Removes the items that is not present in both sets.

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.intersection_update(set2)
>>> set1
{'apple'}
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



difference_update()- Removes the items that exists in both sets.

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.difference_update(set2)
>>> set1
{'cherry', 'banana'}
```

Symmetric_difference_update()- Updates the original set by removing items that are present in both sets, and inserting the other items.

```
>>> set1={"apple", "banana", "cherry"}
>>> set2={"google", "microsoft", "apple"}
>>> set1.symmetric_difference_update(set2)
>>> set1
{'microsoft', 'apple', 'cherry', 'google', 'banana'}
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set



issubset()- Returns True if all items in set 'x' are present in set 'y'.

```
>>> x = {"a", "b", "c"}
```

```
>>> y = {"f", "e", "d", "c", "b", "a"}
```

```
>>> x.issubset(y)
```

```
True
```

```
>>> y.issubset(x)
```

```
False
```

issuperset()- Returns True if all items in set 'y' are present in set 'x'.

```
>>> x = {"f", "e", "d", "c", "b", "a"}
```

```
>>> y = {"a", "b", "c"}
```

```
>>> x.issuperset(y)
```

```
True
```

```
>>> y.issuperset(x)
```

```
False
```

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Specific functions of set

isdisjoint()- Returns True if no items in set s1 is present in set s2.

```
>>> s1={1,2,5}
>>> s2={11,22,55}
>>> s1.isdisjoint(s2)
True
```

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

```
>>> x = {"f", "e", "d", "c", "b", "a"}
>>> x.clear()
>>> x
set()
```



PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Example code 1:



Display the elements of a set one by one in separated by a space between each of them.

```
a = { 10, 30, 10, 40, 20, 50, 30 }
```

```
for i in a :
```

```
    print(i, end = " ")
```

Output: 40 10 50 20 30

PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Example code 2:

Create a set of numbers from 2 to n.

Steps: create an empty set and add the elements to it.

```
s = set()
```

```
n=int(input("enter the value of n")) #10
```

```
for i in range(2,n+1):
```

```
    s.add(i)
```

```
print(s)
```

Output: {2, 3, 4, 5, 6, 7, 8, 9, 10}



PYTHON FOR COMPUTATIONAL PROBLEM SOLVING

Example code 3:



Program to check whether a set is empty or not.

Version 1:

```
s1=set()
if len(s1)==0:
    print("set is empty")
else:
    print("not empty")
```

Version 2:

```
s1=set()
if(s1):
    print("set is empty")
else:
    print("not empty")
```

Output: set is empty

Note: Empty data structure is False



THANK YOU

Department of Computer Science and Engineering

Dr. Shylaja S S, Director, CDSAML & CCBD, PESU

Prof. Sindhu R Pai – sindhurpai@pes.edu

Dr. Chethana Srinivas