Unit 4 - Lesson 10 - Introduction to Sets, Basic Set Operations

About this unit

Introduction to Sets, Basic Set Operations

Introduction to Sets

Unit • 100% completed

Basic Set Operations

Unit • 100% completed

Introduction to Sets

About this unit

Introduction to Sets

Introduction to Sets
Question

Introduction to Frozenset
Question

Understanding Set Creation
Question

39.1.1. Introduction to Sets

Set is an unordered collection that contains unique elements.

- The elements within a set are **immutable**, which means they cannot be changed after being added. As a result, a set cannot include mutable elements such as **lists**, **sets** or **dictionaries**.
- The set is **mutable** which means elements can be added or removed from it.

The main operations that can be performed on a set are:

- Membership test
- Eliminating duplicate entries.
- Mathematical set operations like union, intersection, difference and symmetric difference.

Creating a set : A set can be created in two ways.

1. Placing all the items/elements inside curly braces `{}` each separated by a comma (`,`).

```
numset = {1, 2, 3, 4, 5, 3, 2}
print(numset)          # Result: {1, 2, 3, 4, 5}
```

2. Using `set()` built-in function.

```
numset2 = set([1, 2, 3, 2, 4, 5])
print(numset2)         # Result: {1, 2, 3, 4, 5}

emptyset = set()
print(type(emptyset))  # Result: <class 'set'>
```

Note: A set can contain any number of items as well as any types of items - **integer**, **float**, **tuple**, **string** etc.

What unique quality makes a set a powerful tool in Python programming?

- [ ] It allows duplicate elements, ensuring comprehensive data representation.

Incorrect! Set doesn't allow duplicate elements,ensuring that each element appears only once

- [x] It stores only unique elements and offers various set operations.

Correct!

- [ ] It can include mutable elements like lists and dictionaries, enhancing flexibility.

Incorrect! As sets are immutable,cannot include mutable elements

- [ ] It maintains the order of elements, enabling efficient indexing.

Incorrect! Set is disordered and do not support indexing.

39.1.2. Introduction to Frozenset

00:05

As we have seen that a set is mutable which can allow addition/deletion of elements from it.

The **elements in a set** should be immutable i.e., they cannot be modified/changed.

Hence, Lists cannot be used as elements of a set.

```
set1 = set((("C", "C++"), ("Java", "OOPS")))
print(type(set1))
# Result: <class 'set'>
set2 = set((["C", "C++"], ["Java", "OOPS", "Scala"]))
# Result: TypeError: unhashable type: 'list'
```

Another type of set exists called the frozenset, which is an **immutable** set.

```
cities = frozenset(["Hyderabad", "Bengaluru", "Pune", "Kochi"])
print(type(cities))
# Result: <class 'frozenset'>
print(cities.add("London"))
# Result: AttributeError: 'frozenset' object has no attribute 'add'
```

Select the correct statements from the given options.

- [x] Set does not maintain the order of insertion

Correct!

- [ ] Elements cannot be added to a Frozenset, but can be deleted.

Incorrect! Frozenset is frozen

- [ ] {1, 2, 3, 4}.add({90, 80}) will result in {1, 2, 3, 4, 90, 80}.

Incorrect! will give error unhashable type 'set'.

- [ ] s1 = {1, 2, 3, 4} s2 = frozenset([90, 80]) s1.add(s2) will give an error.

Incorrect! The result will be: {1, 2, 3, 4, frozenset({90, 80})}.

39.1.3. Understanding Set Creation

00:45

Let's discuss the creation of a set using the user-given elements.

Follow the given instructions and write the code.

Steps to be followed:

1. Take an input from user using `input()` function.(with comma separated).
2. Use `split()` function to convert the given **input** into **list** using the ,(comma) separator.
3. Convert the **list** into **set** using `set()` method
4. Print the obtained set in sorted order using `sorted()` function, which converts a set into sorted order. When we try to convert a set into sorted order, it returns a sorted list

Sample Input and Output:

```
data1: 1,5,6,9,6,78,96,2,4,3
sorted set: ['1', '2', '3', '4', '5', '6', '78', '9', '96']
```

Let's consider a simple example:

```
set1 = {45, 89, 65, 3, 47, 400, 2, 963, 1, 963}
print(set1)          # Result {65, 2, 3, 963, 1, 45, 47, 400, 89}
print(sorted(set1))  # Result [1, 2, 3, 45, 47, 65, 89, 400, 963]
```

Here, when we convert the **set** into sorted order using `sorted()` it returns a **sorted list**.

Note: Please print the set in sorted order (sorted(setname)), otherwise the test cases will fail, because, sets will not follow a proper order and can be printed randomly.

Sample Test Cases

Explorer

SetTest1.py

```
#Take the elements of the list with comma separated.
a = input("data1: ")
#Use split() to convert the given input to list.
b = a.split(",")
#convert the list to set.
c = set(b)

result = sorted(c)
print("sorted set:", result)
```

Average time: 0.009 s (9.25 ms)

Maximum time: 0.012 s (12.00 ms)

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 12 ms

Debug

Expected output

data1: 1,5,6,9,6,78,96,2,4,3

sorted set: ['1', '2', '3', '4', '5', '6', '78', '9', '96']

Actual output

data1: 1,5,6,9,6,78,96,2,4,3

sorted set: ['1', '2', '3', '4', '5', '9', '96']

Test case 2 8 ms

Terminal

Test cases

Basic Set Operations

About this unit

Basic Set Operations

- **Add Elements to a Set**
  Question
- **Remove Elements of a Set**
  Question
- **Membership Test**
  Question
- **Mathematical Set Union**
  Question
- **Mathematical Set Intersection**
  Question
- **Mathematical Set Difference**
  Question

39.2.1. Add Elements to a Set

01:05

Create a `set` with the user given inputs. Take an `element` from the user and add that element to the set. Similarly, create a `list` by taking the inputs from the user and update the `set` with the `list`, print the result as shown in the example.

Sample Input and Output:

```
data1: 1,2,3,4,5
element: 456
sorted set after adding: ['1', '2', '3', '4', '456', '5']
data2: 77,88,99
sorted set after updating: ['1', '2', '3', '4', '456', '5', '77', '88', '99']
```

Note: Please print the set in sorted order (sorted(setname)), otherwise the test cases will fail because sets are not ordered and can be printed in random order.

Sample Test Cases +

Question Hints +

Explorer

SetAdd.py

```
1  data1 = input("data1: ")
2  list1 = data1.split(",")
3  #create a set1 from list1
4  set1 = set(list1)
5
6  element = input("element: ")
7  set1.add(element)
8  #add the element to the set1
9  print("sorted set after adding:", sorted(set1))
10 data2 = input("data2: ")
11 list2 = data2.split(",")
12 set1.update(list2)
13 #update the set1 with the list2
14 print("sorted set after updating:", sorted(set1))
```

Average time **0.017 s** 16.50 ms

Maximum time **0.018 s** 18.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 18 ms

Debug

Expected output

```
data1: 1,2,3,4,5
element: 456
sorted set after adding: ['1', '2', '3', '4', '456', '5']
data2: 77,88,99
sorted set after updating: ['1', '2', '3', '4', '456', '5', '77', '88', '99']
```

Actual output

```
data1: 1,2,3,4,5
element: 456
sorted set after adding: ['1', '2', '3', '4', '456', '5']
data2: 77,88,99
sorted set after updating: ['1', '2', '3', '4', '456', '5', '77', '88', '99']
```

Terminal Test cases

39.2.2. Remove Elements of a Set

01:52

Write a program to remove the elements from the set using the methods **discard()** and **remove()**, print the result as shown in the example. If the element is not present in the set print the error message as shown in the example.

Sample Input and Output 1:

```
data1: 10,20,30,40,50
element to discard: 30
sorted set after discarding: ['10', '20', '40', '50']
element to remove: 40
sorted set after removing: ['10', '20', '50']
```

Sample Input and Output 2:

```
data1: Blue,Green,Red
element to discard: Orange
not in set
```

Note: We have not included `pop()` in this program to remove an element as it randomly removes an element from the set.

Note: Please print the set in sorted order **(sorted(setname))**, otherwise the test cases will fail because sets are not ordered and can be printed in random order. The output

Sample Test Cases +

Question Hints +

Explorer

Setremde...

```
1  data1 = input("data1: ")
2  list1 = data1.split(",")
3  set1 = set(list1)
4  element = input("element to discard: ")
5
6  #Read the Question text carefully and complete the code...
7  if element in set1:
8      set1.discard(element)
9      print("sorted set after discarding:", sorted(set1))
10     a = input("element to remove: ")
11     set1.remove(a)
12     print("sorted set after removing:", sorted(set1))
13
14 else:
15     print("not in set")
```

Average time **0.016 s** 15.80 ms

Maximum time **0.020 s** 20.00 ms

3 out of 3 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 20 ms

Debug

Expected output

```
data1: 10,20,30,40,50
element to discard: 30
sorted set after discarding: ['10', '20', '40', '50']
element to remove: 40
sorted set after removing: ['10', '20', '50']
```

Actual output

```
data1: 10,20,30,40,50
element to discard: 30
sorted set after discarding: ['10', '20', '40', '50']
element to remove: 40
sorted set after removing: ['10', '20', '50']
```

Terminal Test cases

Prev Reset Next >

39.2.3. Membership Test

01:09

Write a program to check if the user-given element is `existing` in the set `or not`, and print the result as shown in the examples.

Sample Input and Output 1:

```
data1: 1,2,3,4,5,6,7
sorted set: ['1', '2', '3', '4', '5', '6', '7']
element: 7
is 7 in set: True
```

Sample Input and Output 2:

```
data1: 11,22,33,44,55,66
sorted set: ['11', '22', '33', '44', '55', '66']
element: 77
is 77 in set: False
```

Note: Please print the set in sorted order (sorted(setname)), otherwise the test cases will fail because sets are not ordered and can be printed in random order. The output will be printed as a list as sorted(set) method returns a list.

Sample Test Cases +

Question Hints +

Explorer

SetMemb...

```
1   data1 = input("data1: ")
2   list1 = data1.split(",")
3   my_set = set(list1)
4   print("sorted set:", sorted(my_set))
5   element = input("element: ")
6
7
8   if element in my_set:
9       print("is",element,"in set:",element in my_set)
10  else:
11      print("is",element,"in set:",element in my_set)
```

Average time 0.013 s 12.75 ms

Maximum time 0.015 s 15.00 ms

2 out of 2 shown test case(s) passed

2 out of 2 hidden test case(s) passed

Test case 1 15 ms

Debug

Expected output

```
data1: 1,2,3,4,5,6,7
sorted set: ['1', '2', '3', '4', '5', '6', '7']
element: 7
is 7 in set: True
```

Actual output

```
data1: 1,2,3,4,5,6,7
sorted set: ['1', '2', '3', '4', '5', '6'
element: 7
is 7 in set: True
```

Terminal Test cases

39.2.4. Mathematical Set Union

03:26

Create three sets, **set1**, **set2** and **set3** by taking the inputs from the user. Find the **Union** of three sets using the method **union()** and also using the operator |, print the result as shown in the sample test cases.

Note: Please print the set in sorted order **(sorted(setname))**, otherwise the test cases will fail because sets are not ordered and can be printed in random order. The output will be printed as a list as **sorted(set)** method returns a list.

Sample Test Cases +

Question Hints +

Explorer

SetMathU...

```
1   data1 = input("data1: ")
2   list1 = data1.split(",")
3   set1 = set(list1)
4
5   data2 = input("data2: ")
6   list2 = data2.split(",")
7   set2 = set(list2)
8
9   data3 = input("data3: ")
10  list3 = data3.split(",")
11  set3 = set(list3)
12
13  print("set1 sorted:", sorted(set1))
14  print("set2 sorted:", sorted(set2))
15  print("set3 sorted:", sorted(set3))
16
17
```

Average time: **0.013 s** (12.50 ms)

Maximum time: **0.014 s** (14.00 ms)

2 out of 2 shown test case(s

2 out of 2 hidden test case(s

Test case 1 11 ms

Expected output	Actual output
data1: 1,2,3	data1: 1,2,3
data2: 4,5,6	data2: 4,5,6
data3: 7,8,9	data3: 7,8,9
set1 sorted: ['1', '2', '3']	set1 sorted: ['1', '2
set2 sorted: ['4', '5', '6']	set2 sorted: ['4', '5
set3 sorted: ['7', '8', '9']	set3 sorted: ['7', '8

Terminal Test cases