Python Set

Summary: in this tutorial, you'll learn about Python `Set` type and how to use it effectively.

Introduction to the Python Set type

A Python set is an unordered list of immutable elements. It means:

- Elements in a set are unordered.
- Elements in a set are unique. A set doesn't allow duplicate elements.
- Elements in a set cannot be changed. For example, they can be numbers (https://www.pythontutorial.net/python-basics/python-numbers/) , strings (https://www.pythontutorial.net/python-basics/python-string/) , and tuples (https://www.pythontutorial.net/python-basics/python-tuples/) , but cannot be lists (https://www.pythontutorial.net/python-basics/python-tuples/) or dictionaries (https://www.pythontutorial.net/python-basics/python-dictionary/) .

To define a set in Python, you use the curly brace `{}` . For example:

```
skills = {'Python programming','Databases', 'Software design'}
```

Note a dictionary also uses curly braces, but its elements are key-value pairs.

To define an empty set, you cannot use the curly braces like this:

```
empty_set = {}
```

...because it defines an empty dictionary.

Therefore, you need to use the built-in `set()` function:

```
empty_set  = set()
```

An empty set evaluates to False in Boolean (https://www.pythontutorial.net/python-basics/python-boolean/) context. For example:

```
skills = set()

if not skills:
    print('Empty sets are falsy')
```

Output:

```
Empty sets are falsy
```

In fact, you can pass an iterable (https://www.pythontutorial.net/python-basics/python-iterables/) to the `set()` function to create a set. For example, you can pass a list, which is an iterable, to the `set()` function like this:

```
skills = set(['Problem solving','Critical Thinking'])
print(skills)
```

Output:

```
{'Critical Thinking', 'Problem solving'}
```

Note that the original order of elements may not be preserved.

If an iterable has duplicate elements, the `set()` function will remove them. For example:

```
characters = set('letter')
print(characters)
```

Output:

```
{'r', 'l', 't', 'e'}
```

In this example, the string `'letter'` has two e and t characters and the set() removes each of them.

Getting sizes of a set

To get the number of elements in a set, you use the built-in `len()` function.

```
len(set)
```

For example:

```
ratings = {1, 2, 3, 4, 5}
size = len(ratings)

print(size)
```

Output:

```
5
```

Checking if an element is in a set

To check if a set contains an element, you use the `in` operator:

```
element in set
```

The `in` operator returns `True` if the set contains the element. Otherwise, it returns `False`. For example:

```
ratings = {1, 2, 3, 4, 5}
rating = 1

if rating in ratings:
    print(f'The set contains {rating}')
```

Output:

```
The set contains 1
```

To negate the `in` operator, you use the `not` operator. For example:

```
ratings = {1, 2, 3, 4, 5}
rating = 6

if rating not in ratings:
    print(f'The set does not contain {rating}')
```

Output:

```
The set does not contain 6
```

Adding elements to a set

To add an element to a set, you use the `add()` method:

```
set.add(element)
```

For example:

```
skills = {'Python programming', 'Software design'}
skills.add('Problem solving')

print(skills)
```

Output:

```
{'Problem solving', 'Software design', 'Python programming'}
```

Removing an element from a set

To remove an element from a set, you use the `remove()` method:

```
set.remove(element)
```

For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skills.remove('Software design')

print(skills)
```

Output:

```
{'Problem solving', 'Python programming'}
```

If you remove an element that doesn't exist in a set, you'll get an error (`KeyError`). For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skills.remove('Java')
```

Error:

```
KeyError: 'Java'
```

To avoid the error, you should use the `in` operator to check if an element is in the set before removing it:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
if 'Java' in skills:
    skills.remove('Java')
```

To make it more convenient, the set has the `discard()` method that allows you to remove an element. And it doesn't raise an error if the element is not in the list:

```
set.discard(element)
```

For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skills.discard('Java')
```

Returning an element from a set

To remove and return an element from a set, you use the `pop()` method.

Since the elements in a set have no specific order, the `pop()` method removes an unspecified element from a set.

If you execute the following code multiple times, it'll show a different value each time:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skill = skills.pop()

print(skill)
```

Removing all elements from a set

To remove all elements from a set, you use the `clear()` method:

```
set.clear()
```

For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skills.clear()

print(skills)
```

Output:

```
set()
```

Frozen a set

To make a set immutable, you use the built-in function called `frozenset()`. The `frozenset()` returns a new immutable set from an existing one. For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}
skills = frozenset(skills)
```

After that, if you attempt to modify elements of the set, you'll get an error:

```
skills.add('Django')
```

Error:

```
AttributeError: 'frozenset' object has no attribute 'add'
```

Looping through set elements

Since a set is an iterable, you can use a for loop to iterate over its elements. For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}

for skill in skills:
    print(skill)
```

Output:

```
Software design
Python programming
Problem solving
```

To access the index of the current element inside the loop, you can use the built-in `enumerate()` function:

```
skills = {'Problem solving', 'Software design', 'Python programming'}

for index, skill in enumerate(skills):
    print(f"{index}.{skill}")
```

Output:

```
0.Software design
1.Python programming
2.Problem solving
```

By default, the index starts at zero. To change this, you pass the starting value to the second argument of the `enumerate()` function. For example:

```
skills = {'Problem solving', 'Software design', 'Python programming'}

for index, skill in enumerate(skills, 1):
    print(f"{index}.{skill}")
```

Output:

```
1.Python programming
2.Problem solving
3.Software design
```

Notice that every time you run the code, you'll get the set elements in a different order.

In the next tutorial, you'll learn how to perform useful operations on sets such as:

- Union of Sets (https://www.pythontutorial.net/python-basics/python-set-union/)
- Intersection of Sets (https://www.pythontutorial.net/python-basics/python-set-intersection/)
- Difference between Sets (https://www.pythontutorial.net/python-basics/python-set-difference/)
- Symmetric Difference of Sets (https://www.pythontutorial.net/python-basics/python-symmetric-difference/)

Summary

- A set is an **unordered** collection of **immutable elements**.