Sets

This lesson highlights the key features of the set data structure.

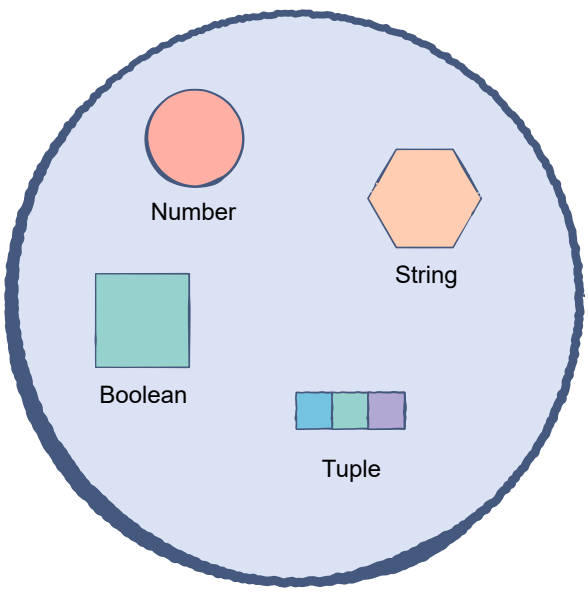
We'll cover the following

Structure

> A set is an unordered collection of data items.

The data is not indexed, so we can't access elements using indices or `get()`.

This is perhaps the simplest data structure in Python. We can think of it as a bag containing random items.



Mutable data structures like lists or dictionaries can't be added to a set. However, adding a tuple is perfectly fine.

One might wonder, "Why would I need a set?"

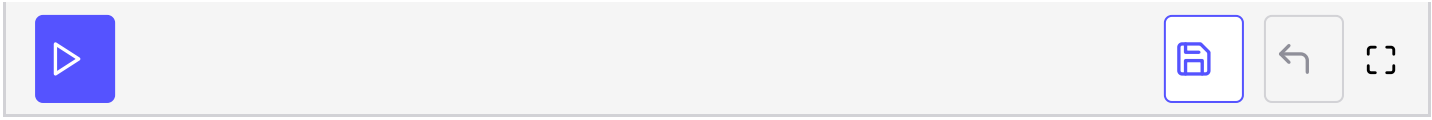
Well, a set is perfect when we simply need to keep track of the existence of items.

It doesn't allow duplicates, which means that we can convert another data structure to a set to remove any duplicates.

Creating a Set

The contents of a set are encapsulated in curly brackets, `{}`. Like all data structures, the length of a set can be calculated using `len()`:

```
random_set = {"Educative", 1408, 3.142,
              (True, False)}
print(random_set)
print(len(random_set))  # Length of the set
```

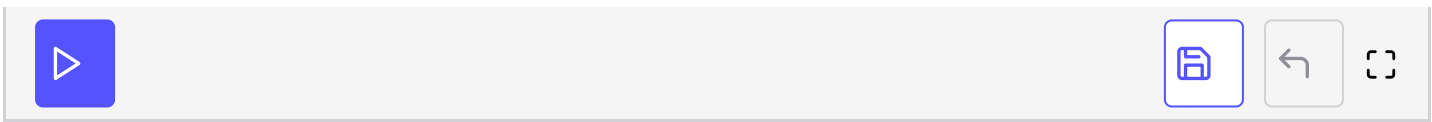


The `set()` Constructor

The `set()` constructor is an alternate way of creating sets. The advantage it presents is that it allows us to make an empty set:

```
empty_set = set()
print(empty_set)

random_set = set({"Educative", 1408, 3.142, (True, False)})
print(random_set)
```



Adding Elements

To add a single item, we can use the `add()` method. To add multiple items, we'd have to use `update()`.

The input for `update()` must be another set, list, tuple, or string.

Let's add elements to an empty set:

```
empty_set = set()
print(empty_set)

empty_set.add(1)
print(empty_set)

empty_set.update([2, 3, 4, 5, 6])
print(empty_set)
```

Deleting Elements

The `discard()` or `remove()` operations can be used to delete a particular item from a set:

```
random_set = set({"Educative", 1408, 3.142, (True, False)})
print(random_set)

random_set.discard(1408)
print(random_set)

random_set.remove((True, False))
print(random_set)
```

The `remove()` method generates an error if the item is not found, unlike the `discard()` method.

Iterating a Set

The `for` loop can be used on unordered data structures like sets. However, we wouldn't know the order in which the iterator moves meaning elements will be picked randomly.

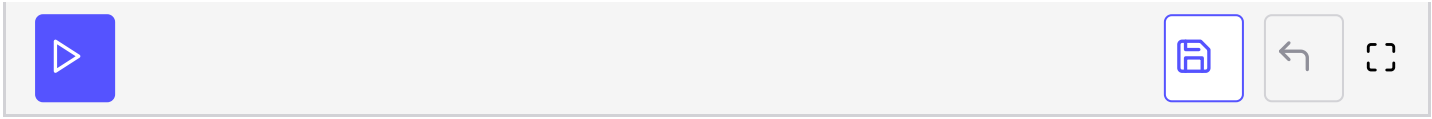
In the example below, we'll take the elements of a set and append them to a list if they are odd:

```
odd_list = [1, 3, 5, 7]
unordered_set = {9, 10, 11, 12, 13, 14, 15, 16, 17}

print(unordered_set)
```

```
for num in unordered_set:
    if(not num % 2 == 0):
        odd_list.append(num)


print(odd_list)
```



These are all the basic operations that can be performed on a set. However, there are several more which you can check out in the official documentation.

In the next lesson, we'll look at how set theory can be used in Python through sets.