Sets

Lists are extremely versatile tools that suit most container object applications. But they are not useful when we want to ensure objects in the list are unique. For example, a song library may contain many songs by the same artist. If we want to sort through the library and create a list of all the artists, we would have to check
the list to see if we've added the artist already before we add them again.

This is where sets come in. Sets come from mathematics, where they represent an unordered group of (usually) unique numbers. We can add a number to a set five times, but it will show up in the set only once.

In Python, sets can hold any hashable object, not just numbers. Hashable objects are the same objects that can be used as keys in dictionaries, so again, lists and dictionaries are out. Like mathematical sets, they can store only one copy of each object. So if we're trying to create a list of song artists, we can create a set of string names and simply add them to the set. This example starts with a list of (song, artist) tuples and creates a set of the artists:

```
song_library = [("Phantom Of The Opera", "Sarah Brightman"),
("Knocking On Heaven's Door", "Guns N' Roses"),
("Captain Nemo", "Sarah Brightman"),
("Patterns In The Ivy", "Opeth"),
("November Rain", "Guns N' Roses"),
("Beautiful", "Sarah Brightman"),
("Mal's Song", "Vixy and Tony")]
artists = set()
for song, artist in song_library:
artists.add(artist)
print(artists)
```

There is no built-in syntax for an empty set as there is for lists and dictionaries; we create a set using the set() constructor. However, we can use the curly braces of dictionary syntax to create a set, so long as the set contains values. If we use colons to separate pairs of values, it's a dictionary, as in `{'key': 'value', 'key2': 'value2'}`. If we just separate values with commas, it's a set, as in `{'value', 'value2'}`. Items can be added individually to the set using its add method. If we run this script, we see that the set works as advertised:

{'Sarah Brightman', "Guns N' Roses", 'Vixy and Tony', 'Opeth'}

If you're paying attention to the output, you'll notice that the items are not printed in the order they were added to the sets. Sets, like dictionaries, are unordered. They both use an underlying hash-based data structure for efficiency. Because they are unordered, sets cannot have items looked up by index. The primary purpose of a set is to divide the world into two groups: "things that are in the set", and, "things that are not in the set". It is easy to check if an item is in the set or to loop over the items.

While the primary feature of a set is uniqueness, that is not its primary purpose. Sets are most useful when two or more of them are used in combination. Most of the methods on the set type operate on other sets, allowing us to efficiently combine or compare the items in two or more sets. These methods have strange names if you're not familiar with

mathematical sets, since they use the same terminology used in mathematics. We'll start with three methods that return the same result regardless of which is the calling set and which is the called set.