Python Sets

Python comes equipped with several built-in data types to help us organize our data. These structures include lists, dictionaries, tuples and **sets**.

Initializing a set

There are two ways to create sets: using curly braces `{}` and the built-in function `set()`

```
>>> s = {1, 2, 3}
>>> s = set([1, 2, 3])

>>> s = {}  # this will create a dictionary instead of a set
>>> type(s)
# <class 'dict'>
```

Unordered collections of unique elements

A set automatically remove all the duplicate values.

```
>>> s = {1, 2, 3, 2, 3, 4}
>>> s
# {1, 2, 3, 4}
```

And as an unordered data type, they can't be indexed.

```
>>> s = {1, 2, 3}
>>> s[0]
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# TypeError: 'set' object does not support indexing
```

set add() and update()

Using the `add()` method we can add a single element to the set.

```
>>> s = {1, 2, 3}
>>> s.add(4)
>>> s
# {1, 2, 3, 4}
```

And with `update()`, multiple ones:

```
>>> s = {1, 2, 3}
>>> s.update([2, 3, 4, 5, 6])
>>> s
# {1, 2, 3, 4, 5, 6}
```

set remove() and discard()

Both methods will remove an element from the set, but `remove()` will raise a `key error` if the value doesn't exist.

```
>>> s = {1, 2, 3}
>>> s.remove(3)
>>> s
# {1, 2}

>>> s.remove(3)
# Traceback (most recent call last):
#   File "<stdin>", line 1, in <module>
# KeyError: 3
```

`discard()` won't raise any errors.

```
>>> s = {1, 2, 3}
>>> s.discard(3)
>>> s
# {1, 2}
>>> s.discard(3)
```

set union()

`union()` or `|` will create a new set that with all the elements from the sets provided.

```
>>> s1 = {1, 2, 3}
>>> s2 = {3, 4, 5}
>>> s1.union(s2)  # or 's1 | s2'
# {1, 2, 3, 4, 5}
```

set intersection

`intersection` or `&` will return a set with only the elements that are common to all of them.

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s3 = {3, 4, 5}
>>> s1.intersection(s2, s3)  # or 's1 & s2 & s3'
# {3}
```

set difference

`difference` or `-` will return only the elements that are unique to the first set (invoked set).

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}

>>> s1.difference(s2)  # or 's1 - s2'
# {1}
```

```
>>> s2.difference(s1) # or 's2 - s1'
# {4}
```

set symetric_difference

`symetric_difference` or `^` will return all the elements that are not common between them.

```
>>> s1 = {1, 2, 3}
>>> s2 = {2, 3, 4}
>>> s1.symmetric_difference(s2)  # or 's1 ^ s2'
# {1, 4}
```