Advanced Sets

In this lecture we will learn about the various methods for sets that you may not have seen yet. We'll go over the basic ones you already know and then dive a little deeper.

In [1]:

```
s = set()
```

add ¶

add elements to a set. Remember, a set won't duplicate elements; it will only present them once (that's why it's called a set!)

In [2]:

```
s.add(1)
```

In [3]:

```
s.add(2)
```

In [4]:

```
s
```

Out[4]:

```
{1, 2}
```

clear

removes all elements from the set

In [5]:

```
s.clear()
```

In [6]:

```
s
```

Out[6]:

```
set()
```

copy

returns a copy of the set. Note it is a copy, so changes to the original don't effect the copy.

In [7]:

```
s = {1,2,3}
sc = s.copy()
```

In [8]:

```
sc
```

Out[8]:

```
{1, 2, 3}
```

In [9]:

```
s
```

Out[9]:

```
{1, 2, 3}
```

In [10]:

```
s.add(4)
```

In [11]:

```
s
```

Out[11]:

```
{1, 2, 3, 4}
```

In [12]:

```
sc
```

Out[12]:

```
{1, 2, 3}
```

difference

difference returns the difference of two or more sets. The syntax is:

```
set1.difference(set2)
```

For example:

In [13]:

```
s.difference(sc)
```

Out[13]:

```
{4}
```

difference_update

difference_update syntax is:

```
set1.difference_update(set2)
```

the method returns set1 after removing elements found in set2

In [14]:

```
s1 = {1,2,3}
```

In [15]:

```
s2 = {1,4,5}
```

In [16]:

```
s1.difference_update(s2)
```

In [17]:

```
s1
```

Out[17]:

```
{2, 3}
```

discard

Removes an element from a set if it is a member. If the element is not a member, do nothing.

In [18]:

```
s
```

Out[18]:

```
{1, 2, 3, 4}
```

In [19]:

```
s.discard(2)
```

In [20]:

```
s
```

Out[20]:

```
{1, 3, 4}
```

intersection and intersection_update

Returns the intersection of two or more sets as a new set.(i.e. elements that are common to all of the sets.)

In [21]:

```
s1 = {1,2,3}
```

In [22]:

```
s2 = {1,2,4}
```

In [23]:

```
s1.intersection(s2)
```

Out[23]:

```
{1, 2}
```

In [24]:

```
s1
```

Out[24]:

```
{1, 2, 3}
```

intersection_update will update a set with the intersection of itself and another.

In [25]:

```
s1.intersection_update(s2)
```

In [26]:

```
s1
```

Out[26]:

```
{1, 2}
```

isdisjoint

This method will return True if two sets have a null intersection.

In [27]:

```
s1 = {1,2}
s2 = {1,2,4}
s3 = {5}
```

In [28]:

```
s1.isdisjoint(s2)
```

Out[28]:

```
False
```

In [29]:

```
s1.isdisjoint(s3)
```

Out[29]:

```
True
```

issubset

This method reports whether another set contains this set.

In [30]:

```
s1
```

Out[30]:

```
{1, 2}
```

In [31]:

```
s2
```

Out[31]:

```
{1, 2, 4}
```

In [32]:

```
s1.issubset(s2)
```

Out[32]:

```
True
```

issuperset

This method will report whether this set contains another set.

In [33]:

```
s2.issuperset(s1)
```

Out[33]:

```
True
```

In [34]:

```
s1.issuperset(s2)
```

Out[34]:

```
False
```

symmetric_difference and symmetric_update

Return the symmetric difference of two sets as a new set.(i.e. all elements that are in exactly one of the sets.)

In [35]:

```
s1
```

Out[35]:

```
{1, 2}
```

In [36]:

```
s2
```

Out[36]:

```
{1, 2, 4}
```

In [37]:

```
s1.symmetric_difference(s2)
```

Out[37]:

```
{4}
```

union

Returns the union of two sets (i.e. all elements that are in either set.)

In [38]:

```
s1.union(s2)
```

Out[38]:

```
{1, 2, 4}
```

update

Update a set with the union of itself and others.

In [39]:

```
s1.update(s2)
```

In [40]:

```
s1
```

Out[40]:

```
{1, 2, 4}
```

Great! You should now have a complete awareness of all the methods available to you for a set object type. This data structure is extremely useful and is underutilized by beginners, so try to keep it in mind!

Good Job!