SECTION 20: ADVANCED PYTHON PROJECTS AND DATA STRUCTURES - 41 minutes, 7 parts

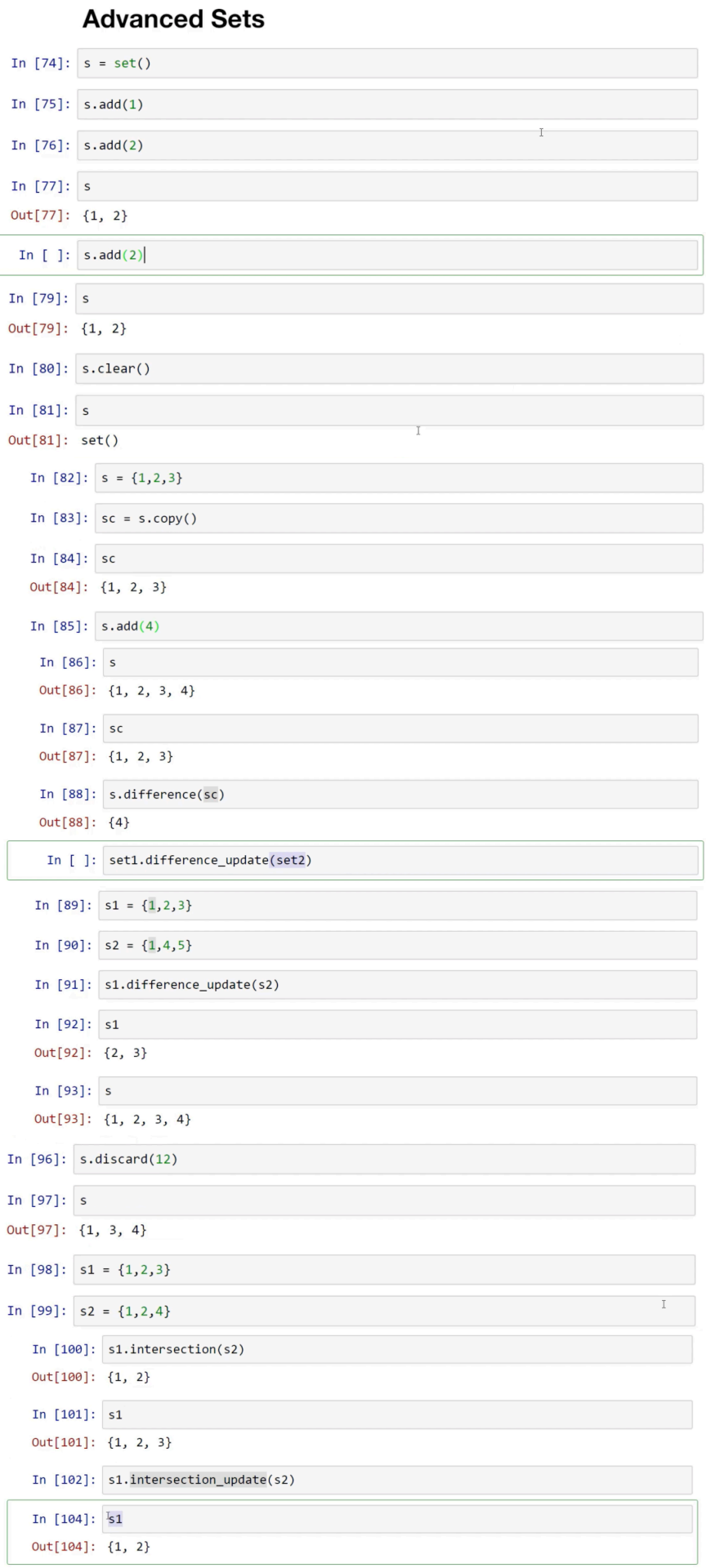
3/7 Advanced Sets

- **advanced sets**
  - -> methods for sets
  - -> built in ones and then more complex ones
  - -> he biulds a set and then adds one to it
  - -> then adds two
  - **-> a set contains no duplicate items**
  - -> you can add the same number and then the set won't change
  - -> .add() <- to add elements to the set
  - -> .clear() <- to empty the set
  - -> .copy() <- to copy the set
  - -> he sets this equal to another variable
  - -> changes to the original set won't change a copy of it

  - **-> the .difference() method**
    - -> this returns the difference of two sets
    - -> it returns the elements which are in one set, but not in the other
    - -> set1.difference_update(set2) <- this returns the difference between these two sets
    - -> we are returning the first set after removing all of the elements found in the second

  - **-> the .discard() method**
    - -> this removes an element from a set if it is a member
    - -> s1.discard(s2) <- this gets rid of the elements of s2 which are in set s1

  - **-> the .intersection() method**
    - -> this returns the intersection of two or more elements
    - -> these elements are common to them both
    - -> s1.intersection_update (s2) <- this updates the intersection of set1 with s2
      - -> the intersection is the elements which the two sets have in common
      - -> and the performing an intersection on this

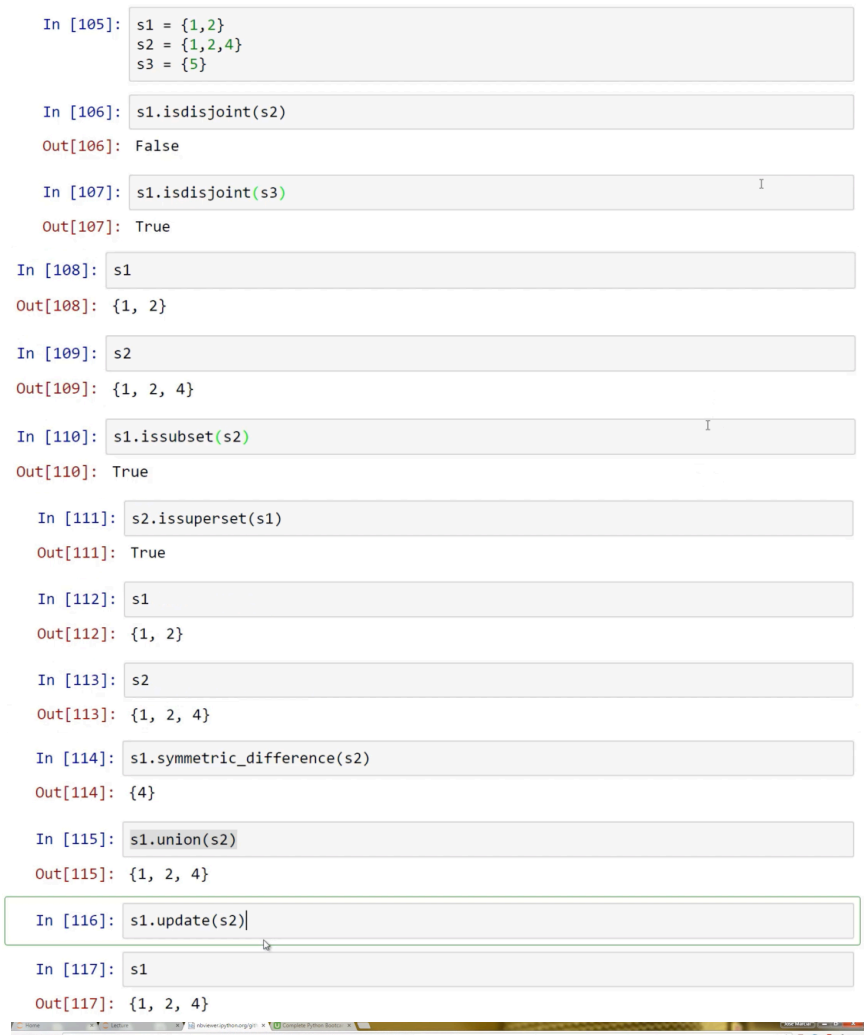


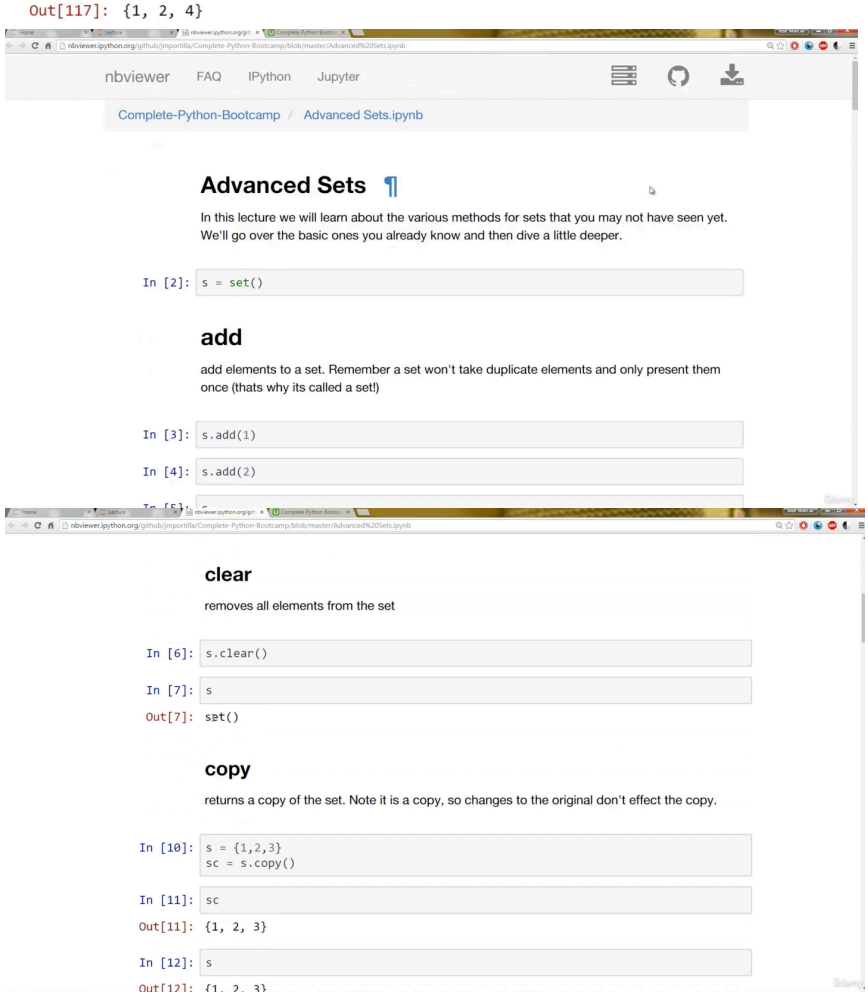
- -> this makes the first set equal to the intersection of the two

- **-> other methods which can be used**
  - -> he defines three different sets and stores them in variables
  - -> the .isdisjoint() method <- this returns True if the intersection is null
  - -> s1.issubset(s2) <- this is a boolean which returns True if s1 is a subset of s2
  - -> s2.issuperset(s1) <- this returns True is s1 is a superset of s2

- **-> symmetric difference**
  - -> the symmetric difference of two sets is all of the elements which are exactly in one of them
  - -> this can be thought of as the opposite of .intersection()
  - -> s1.symmetric_difference(s2) <- this returns the element(s) which are exactly in one of the sets

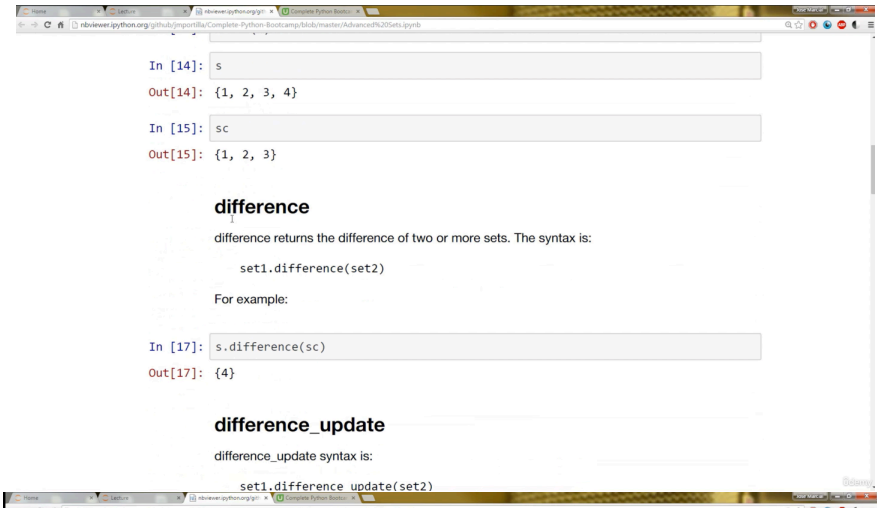
- **-> the .union() method**
  - -> s1.union(s2) <- this retrieves the union of these two sets
  - -> s1.update(s2) <- this is the update method for this





- **review**
  - -> these are all of the methods available for a set object type
  - -> this is for comparison / unique values
  - -> the .add() method adds elements to a set
  - -> the .clear() method removes all elements from a set
  - -> the .copy() method copies sets
  - -> the .difference() method returns the difference between two or more sets
  - -> the .difference_update() method returns set 1 after removing elements found in set 2
  - -> the .discard() method removes an element from a set if it is a member

- -> if the element is not a member, this does nothing
- -> the .intersection() and .intersection_update() methods return the intersection of two or more sets as new sets
  - -> this also has an update method (.update())
- -> the .isdisjoint() method will return True if two sets have a null intersection
- -> then the .issubject() method reports whether this set is contained by the other set
- -> then the .issuperset() method will report whether this set contains another set
- -> then the .symmetric_difference() and .symmetric_update() methods return the symmetric difference of two sets as the new sets
  - -> this is for all elements that are in exactly one of the sets
- -> the .union() method returns the union of two sets (this is all elements that are in either set)
- -> then the .update() method will update a set with the union of itself and others
  - -> these are passed in as its arguments



```
In [14]: s
Out[14]: {1, 2, 3, 4}

In [15]: sc
Out[15]: {1, 2, 3}
```

difference

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

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

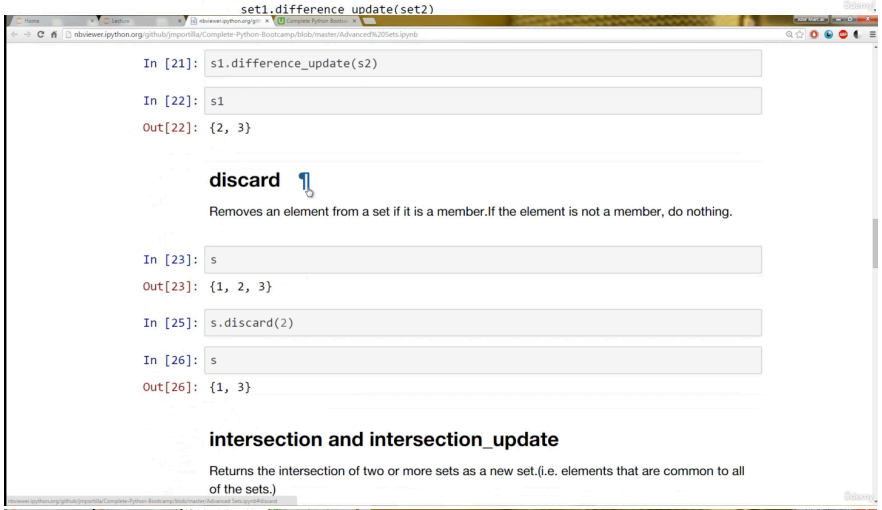
For example:

```
In [17]: s.difference(sc)
Out[17]: {4}
```

difference_update

difference_update syntax is:

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



```
In [21]: s1.difference_update(s2)

In [22]: s1
Out[22]: {2, 3}
```

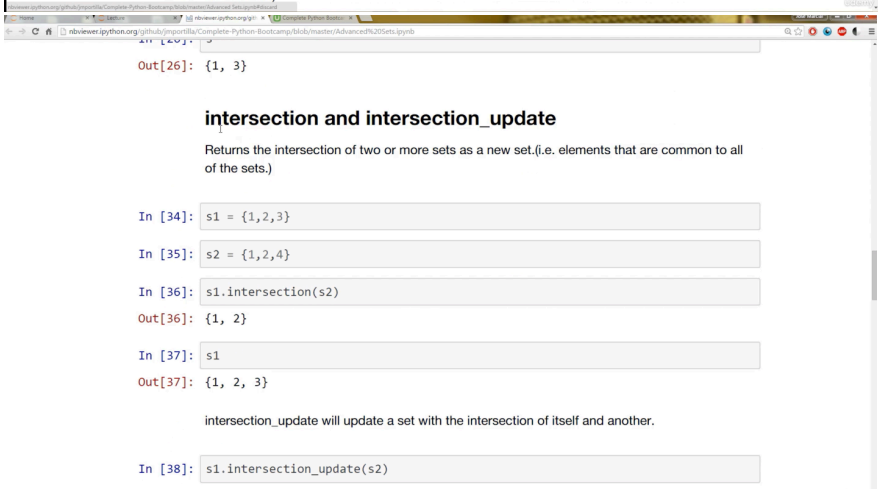
discard

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

```
In [23]: s
Out[23]: {1, 2, 3}

In [25]: s.discard(2)

In [26]: s
Out[26]: {1, 3}
```

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.)



```
Out[26]: {1, 3}
```

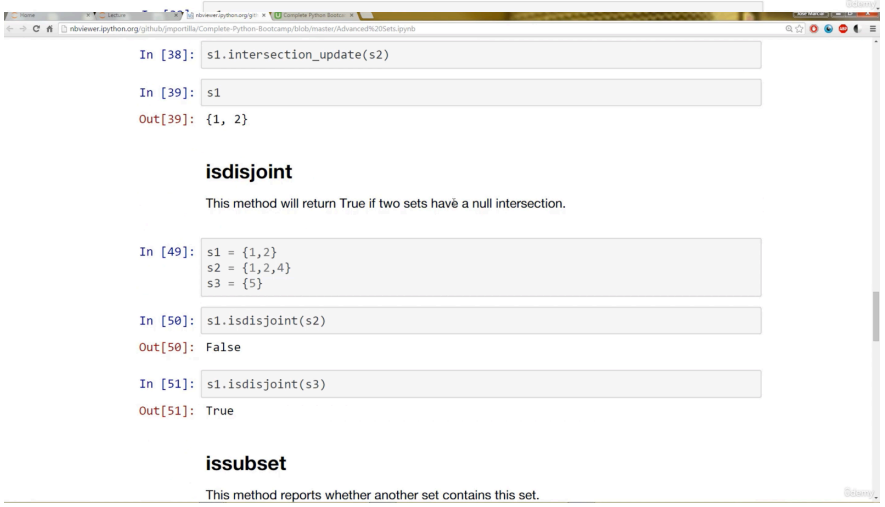
intersection and intersection_update

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

```
In [34]: s1 = {1,2,3}

In [35]: s2 = {1,2,4}

In [36]: s1.intersection(s2)
Out[36]: {1, 2}

In [37]: s1
Out[37]: {1, 2, 3}
```

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

```
In [38]: s1.intersection_update(s2)
```



```
In [38]: s1.intersection_update(s2)

In [39]: s1
Out[39]: {1, 2}
```

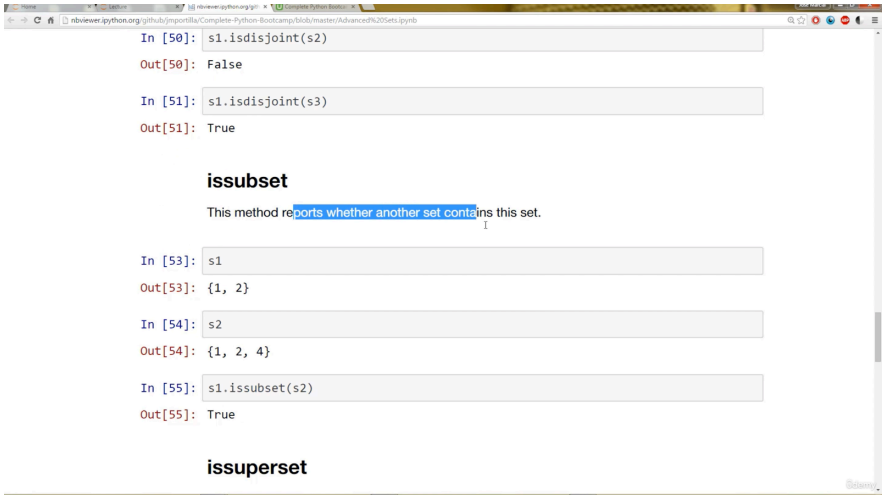
isdisjoint

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

```
In [49]: s1 = {1,2}
         s2 = {1,2,4}
         s3 = {5}

In [50]: s1.isdisjoint(s2)
Out[50]: False

In [51]: s1.isdisjoint(s3)
Out[51]: True
```

issubset

This method reports whether another set contains this set.



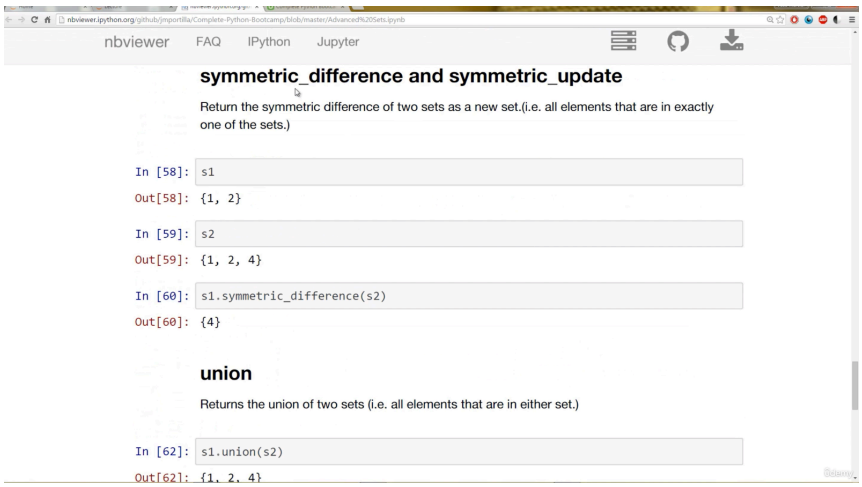
```
In [50]: s1.isdisjoint(s2)
Out[50]: False

In [51]: s1.isdisjoint(s3)
Out[51]: True
```

issubset

This method reports whether another set contains this set.

```
In [53]: s1
Out[53]: {1, 2}

In [54]: s2
Out[54]: {1, 2, 4}

In [55]: s1.issubset(s2)
Out[55]: True
```

issuperset



nbviewer FAQ IPython Jupyter

symmetric_difference and symmetric_update

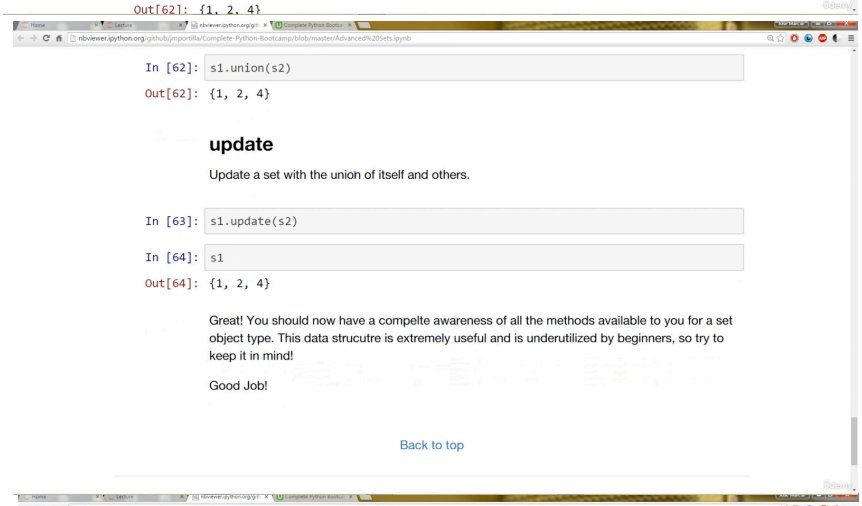
Return the symmetric difference of two sets as a new set.(i.e. all elements that are in exactly one of the sets.)

```
In [58]: s1
Out[58]: {1, 2}

In [59]: s2
Out[59]: {1, 2, 4}

In [60]: s1.symmetric_difference(s2)
Out[60]: {4}
```

union

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

```
In [62]: s1.union(s2)
Out[62]: {1, 2, 4}
```



```
In [62]: s1.union(s2)
Out[62]: {1, 2, 4}
```

update

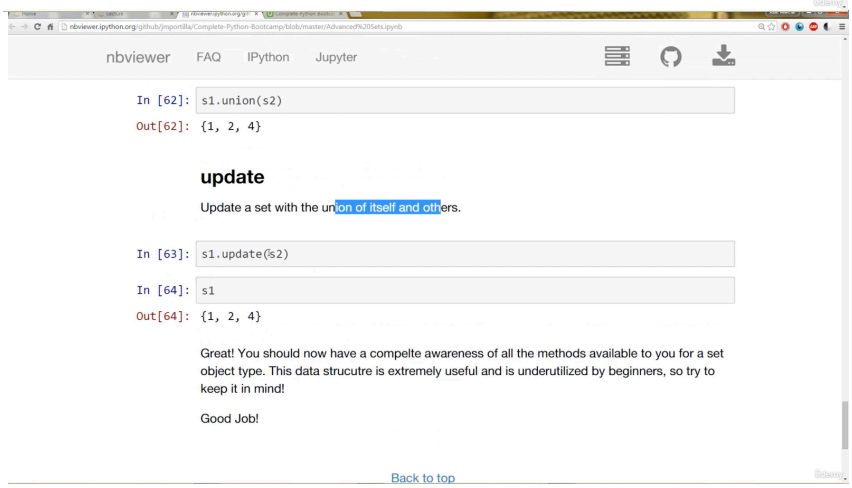
Update a set with the union of itself and others.

```
In [63]: s1.update(s2)

In [64]: s1
Out[64]: {1, 2, 4}
```

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

Good Job!

Back to top



nbviewer FAQ IPython Jupyter

```
In [62]: s1.union(s2)
Out[62]: {1, 2, 4}
```

update

Update a set with the union of itself and others.

```
In [63]: s1.update(s2)

In [64]: s1
Out[64]: {1, 2, 4}
```

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

Good Job!

Back to top