**Sets**

Sets are an unordered collection of *unique* elements. We can construct them by using the set() function. Let's go ahead and make a set to see how it works

st **=** {1,2,3,3,5,6,6}

print(st)

{1, 2, 3, 5, 6}

*#any data structure can be converted to set*

set2 **=** set((2,3,4)) *#As single object you can create , but can not instert*

print("Tuple",set2)

​

​

set3 **=** set({1:3,2:5,3:6})

print("Dict ",set3)

​

​

set4 **=** set([1,2,3,4,5,6,6])

print("List ",set4)

​

​

​

​

​

​

​

​

​

Tuple {2, 3, 4}

Dict {1, 2, 3}

List {1, 2, 3, 4, 5, 6}

x **=** {(1,2,3)}

print(x)

{(1, 2, 3)}

x **=**{1,2,3}

x.add(4)

x

Out[4]:

{1, 2, 3, 4}

*# We add to sets with the add() method*

x **=** {1,2,3}

x.add(1)

​

*#Show*

x

Out[10]:

{1, 2, 3}

Note the curly brackets. This does not indicate a dictionary! Although you can draw analogies as a set being a dictionary with only keys.

We know that a set has only unique entries. So what happens when we try to add something that is already in a set?

*# Add a different element*

x.add(2)

*#Show*

x

Out[12]:

{1, 2, 3}

*# Try to add the same element*

x.add(1)

​

*#Show*

x

Out[8]:

{(1, 2, 3), 1, 2}

Notice how it won't place another 1 there. That's because a set is only concerned with unique elements! We can cast a list with multiple repeat elements to a set to get the unique elements. For example:

*# Create a list with repeats*

list1 **=** [1,1,2,2,3,4,5,6,1,1]

*# Cast as set to get unique values*

z**=** set(list1)

z

Out[8]:

{1, 2, 3, 4, 5, 6}

**Accessing Elements in a Set**

Unlike Lists it is not possible to access elements from a Set via an index; this is because they are unordered containers and thus there are no indexes available. However, they are Iterable containers. Elements of a Set can be iterated over using the for statement:

z[0] **--** **not** work

**for** index,x **in** enumerate(z):

​

print(index,x)

0 1

1 2

2 3

3 4

4 5

5 6

*# for x in z:*

*# print(x)*

​

*#or*

​

​

my\_iter **=** iter(z) *#make the z as iterator object and then traverse it , it will give error if we iterate more than the*

*#exisitng values*

​

print(next((my\_iter))) *#1*

print(next((my\_iter))) *#2*

print(next((my\_iter))) *#3*

print(next((my\_iter))) *#4*

print(next((my\_iter))) *#5*

print(next((my\_iter))) *#6*

print(next((my\_iter))) *#7 it will be error*

​

​

​

​

1

2

3

4

5

6

**---------------------------------------------------------------------------**

**StopIteration** Traceback (most recent call last)

**<ipython-input-39-3eafc977c782>** in <module>

15 print**(**next**((**my\_iter**)))**

16 print**(**next**((**my\_iter**)))**

**---> 17** print**(**next**((**my\_iter**)))**

18

19

**StopIteration**:

*# to convert sets to immutable using frozen*

*# if we want to make sets also immutable then make it as frozenset*

​

s**=**{1,2,3,4}

s.add(10)

print(s)

f **=** frozenset(s)

print(type(f))

f.add(11) *# error*

​

​

​

{1, 2, 3, 4, 10}

<class 'frozenset'>

**---------------------------------------------------------------------------**

**AttributeError** Traceback (most recent call last)

**<ipython-input-2-aa707f15c6c4>** in <module>

7 f **=** frozenset**(**s**)**

8 print**(**type**(**f**))**

**----> 9** f**.**add**(11)** **# error**

10

11

**AttributeError**: 'frozenset' object has no attribute 'add'

f

Out[13]:

frozenset({1, 2, 3, 4, 10})

*#To unfreez the set which was earlier freezed using frozen*

s2 **=** {1,2,3}

s2.add(4)

print(type(s2)) *# normal set*

s3 **=** frozenset(s2) *# frozen set*

print(type(s3))

*# s3.add(5)*

s1 **=** set(s3) *#unfreez back*

print(type(s1))

s1.add(5)

print(s1)

<class 'set'>

<class 'frozenset'>

<class 'set'>

{1, 2, 3, 4, 5}

​

Checking for Presence of an Element You can check for the presence of an element in a set using the in keyword, for eg.

list1 **=**[1,2,3]

​

a **=** set([2,3,4])

a

Out[1]:

{2, 3, 4}

print(3 **in** a)

​

**if**(3 **in** a):

print("Yes")

True

Yes

*# Adding items*

a.add(40)

a

Out[2]:

{2, 3, 4, 40}

*#to add more than one value*

a.update([80,90]) *#the argument for update is either an list,set or dictionary, what ever it is it will convert it as set and will add*

a

a.update({100,200}) *#it should be only iterable object*

a

​

a.update({9,12})

a

​

a.update((5,6))

a

​

a.update([7])

a

*#a.add([10,2]) #indiviaul value only else error*

a.add(10)

a

Out[7]:

{10, 100, 110, 12, 2, 200, 3, 30, 4, 40, 5, 6, 7, 80, 9, 90, 'age', 'name1'}

a.update({'name1':120,'age':100})

print(a)

a.update((90,30))

print(a)

a.update({110,200})

print(a)

​

{2, 3, 4, 100, 5, 6, 40, 200, 9, 7, 12, 10, 'name1', 'age', 80, 90}

{2, 3, 4, 100, 5, 6, 40, 200, 9, 7, 12, 10, 'name1', 'age', 80, 90, 30}

{2, 3, 4, 5, 6, 7, 200, 9, 10, 12, 'age', 80, 90, 30, 100, 'name1', 40, 110}

*#you can not alter the values through indexing of the set because you can not index it as they are not ordered*

b **=** set([5,6,7,8])

*# print(b[0]) #error*

c **=** set(['good','bad','new','old'])

print(len(b))

print(min(b))

print(max(b))

print(min(c))

4

5

8

bad

*#use remove method to remove the element and use discard method to remove an element*

*#remove will generate error if element is not there where as discard does not*

​

b.remove(6)

b

b.remove(10)

b

​

**---------------------------------------------------------------------------**

**KeyError** Traceback (most recent call last)

**<ipython-input-27-04bf48c347ea>** in <module>

2 **#remove will generate error if element is not there where as discard does not**

3

**----> 4** b**.**remove**(6)**

5 b

6 b**.**remove**(10)**

**KeyError**: 6

b.discard(10)

b

Out[29]:

{5, 7, 8}

b.pop()

Out[30]:

8

b.update([10,20,304,506,60])

b

b.pop() *#since we will not know the order it will pop out the last item only*

​

*#clear will clear entire set*

b.clear()

b

Out[33]:

set()

*#question to be asked*

set1 **=** set([1,1.0,2,2.0]) *#no duplicates*

​

print(set1)

set1 **=** {1,1.0,2,2.0}

set1

{1, 2}

Out[3]:

{1, 2}

**Nesting Sets**

It is possible to hold any immutable object within a set. This means that a set can contain a reference to a Tuple (as that is immutable). We can thus write:

Set Methods Python has a set of built-in methods that you can use on sets.

add() Adds an element to the set

clear() Removes all the elements from the set

copy() Returns a copy of the set

difference() Returns a set containing the difference between two or more sets

difference\_update() Removes the items in this set that are also included in another,specified set

discard() Remove the specified item

intersection() Returns a set, that is the intersection of two other sets

intersection\_update() Removes the items in this set that are not present in other,specified set(s)

isdisjoint() Returns whether two sets have a intersection or not

issubset() Returns whether another set contains this set or not

issuperset() Returns whether this set contains another set or not

pop() Removes an element from the set

remove() Removes the specified element

symmetric\_difference() Returns a set with the symmetric differences of two sets

symmetric\_difference\_update() inserts the symmetric differences from this set and another

union() Return a set containing the union of sets

update() Update the set with the union of this set and others

**Booleans**

Python comes with Booleans (with predefined True and False displays that are basically just the integers 1 and 0). It also has a placeholder object called None.

*# Set object to be a boolean*

a **=** **True**

*#Show*

a

Out[11]:

True

We can also use comparison operators to create booleans. We will go over all the comparison operators later on in the course.

*# Output is boolean*

1 **>** 2

Out[12]:

False

We can use None as a placeholder for an object that we don't want to reassign yet:

*# None placeholder*

b **=** **None**

*# Show*

print(b)

None

**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.

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!)

s.add(1)

s.add(2)

s

Out[8]:

{1, 2}

**clear**

removes all elements from the set

s.clear()

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.

s **=** {1,2,3}

sc **=** s.copy()

​

sc

Out[2]:

{1, 2, 3}

s

Out[3]:

{1, 2, 3}

s.add(4)

s

Out[5]:

{1, 2, 3, 4}

sc

Out[6]:

{1, 2, 3}

**difference**

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

set1.difference(set2)

For example:

print(s.difference(sc))

print(s)

print(sc)

{4}

{1, 2, 3, 4}

{1, 2, 3}

**difference\_update**

difference\_update syntax is:

set1.difference\_update(set2)

the method returns set1 after removing elements found in set2

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

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

​

print(s2.difference(s1))

print(s1.difference(s2))

{4, 5}

{2, 3}

print(s2.difference\_update(s1))

None

print(s1)

print(s2)

{1, 2, 3}

{4, 5}

**discard**

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

s

Out[18]:

{1, 2, 3, 4}

s.discard(2)

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.)

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

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

s1.intersection(s2)

Out[15]:

{1, 2}

print(s1)

print(s2)

{1, 2, 3}

{1, 2, 4}

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

s1.intersection\_update(s2)

print(s1)

print(s2)

{1, 2}

{1, 2, 4}

**isdisjoint**

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

s1 **=** {1,2}

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

s3 **=** {5}

s1.isdisjoint(s2)

Out[28]:

False

s1.isdisjoint(s3)

Out[29]:

True

**issubset**

This method reports whether another set contains this set.

s1

Out[30]:

{1, 2}

s2

Out[31]:

{1, 2, 4}

s1.issubset(s2)

Out[32]:

True

**issuperset**

This method will report whether this set contains another set.

s2.issuperset(s1)

Out[33]:

True

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.)

s1 **=** {'a','c','d'}

s2 **=** {'c','d','e'}

s1.symmetric\_difference(s2)

Out[21]:

{'a', 'e'}

print(s1)

print(s2)

{'c', 'a', 'd'}

{'e', 'c', 'd'}

s1.symmetric\_difference\_update(s2)

print(s1)

print(s2)

{'e', 'a'}

{'e', 'c', 'd'}

**union**

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

s1.union(s2)

Out[24]:

{'a', 'c', 'd', 'e'}

**update**

Update a set with the union of itself and others.

s1.update(s2)

print(s1)

print(s2)

{'d', 'e', 'c', 'a'}

{'e', 'c', 'd'}