Chapter 32
Sets

32.1 Introduction

In the last chapter we looked at Tuples and Lists; in this chapter we will look at a further container (or collection) types; the Set type. A Set is an unordered (un indexed) collection of *immutable* objects that does not allow duplicates.

32.2 Creating a Set

A `Set` is defined using curly brackets (e.g. '`{}`').

For example,

```
basket = {'apple', 'orange', 'apple', 'pear',
          'orange', 'banana'}
print(basket)
```

When run this code will show that *apple* is only added once to the set:

```
{'banana', 'orange', 'pear', 'apple'}
```

Note that because a `Set` is unordered it is not possible to refer to elements of the set using an index.

J. Hunt, *A Beginners Guide to Python 3 Programming*,
Undergraduate Topics in Computer Science,
https://doi.org/10.1007/978-3-030-20290-3_32

32.3 The Set() Constructor Function

As with tuples and lists Python provides a predefined function that can convert any iterable type into a Set. The function signature is:

```
set(iterable)
```

Given an iterable object, this function returns a new `Set` based on the values obtained from the iterable. This means that a `Set` can be easily created from a `List`, `Tuple` or `Dictionary` as well as any other data type that implements the iterable protocol. For example, the following code snippet converts a Tuple into a Set:

```
set1 = set((1, 2, 3)
print(set1)
```

which prints out

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

32.4 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:

```
for item in basket:
    print(item)
```

This applies the `print` function to each item in the list in turn.

32.5 Working with Sets

32.5.1 Checking for Presence of an Element

You can check for the presence of an element in a set using the `in` keyword, for example:

```
print('apple' in basket)
```

This will print `True` if '`apple`' is a member of the set `basket`.

32.5.2 Adding Items to a Set

It is possible to add items to a set using the `add()` method:

```
basket = {'apple', 'orange', 'banana'}
basket.add('apricot')
print(basket)
```

This generates:

```
{'orange', 'apple', 'banana', 'apricot', 'pear'}
```

If you want to add more than one item to a `Set` you can use the `update()` method:

```
basket = {'apple', 'orange', 'banana'}
basket.update(['apricot', 'mango', 'grapefruit'])
print(basket)
```

Generating

```
{'orange', 'apple', 'mango', 'banana', 'apricot', 'grapefruit'}
```

The argument to update can be a set, a list, a tuple or a dictionary. The method automatically converts the parameter into a set if it is not a set already and then adds the value to the original set.

32.5.3 Changing Items in a Set

It is not possible to change the items already in a Set.

32.5.4 Obtaining the Length of a Set

As with other collection/container classes; you can find out the length of a `Set` using the `len()` function.

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange',
'banana'}
print(len(basket)) # generates 4
```

32.5.5 Obtaining the Max and Min Values in a Set

You can also obtain the maximum or minimum values in a set using the `max()` and `min()` functions:

```
print(max(a_set))
print(min(a_set))
```

32.5.6 Removing an Item

To remove an item from a set, use the `remove()` or `discard()` functions. The `remove()` function removes a single item from a `Set` but generates an error if that item was not initialling the set. The `remove()` function also removes a single item from a set but does not throw an error if it was not initially present in the set.

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange',
'banana'}
print(basket)
basket.remove('apple')
basket.discard('apricot')
print(basket)
```

This generates:

```
{'pear', 'banana', 'orange', 'apple'}
{'pear', 'banana', 'orange'}
```

There is also a method `pop()` that can be used to remove an item (and return that item as a result of running the method); however it removes the last item in the `Set` (although as a set is unordered you will not know which item that will be).

The method `clear()` is used to remove all elements from a `Set`:

```
basket = {'apple', 'orange', 'banana'}
basket.clear()
print(basket)
```

which prints out

```
set()
```

Which is used to represent an empty set.

32.5.7 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:

```
s1 = { (1, 2, 3)}
print(s1)
```

This prints out:

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

However, we cannot nest Lists or other Sets within a `Set` as these are not immutable types. The following would both generate a runtime error in Python:

```
# Can't have the following
s2 = { {1, 2, 3} }
print(s2)

s3 = { [1, 2, 3] }
print(s3)
```

However we can use `Frozensets` and nest these within sets. A `Frozenset` is exactly like a `Set` except that it is immutable (it cannot be modified) and thus it can be nested within a `Set`. For example:

```
# Need to convert sets and lists into frozensets
s2 = { frozenset({1, 2, 3}) }
print(s2)

s3 = { frozenset([1, 2, 3]) }
print(s3)
```

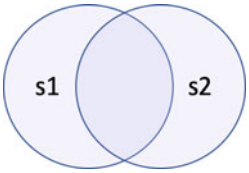
This generates:

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

32.6 Set Operations

The `Set` container also supports *set like* operations such as (`|`), intersection (`&`), difference (−) and symmetric difference (`^`). These are based on simple Set theory.

Given the two sets:




```
print('Union:', s1 | s2)
```

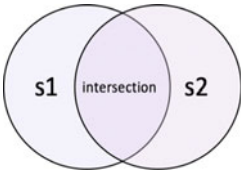
For example, the Union of two sets represents the combination of all the values in the two sets:

```
s1 = {'apple', 'orange', 'banana'}
s2 = {'grapefruit', 'lime', 'banana'}
```

This would print out:

```
Union: {'apple', 'lime', 'banana', 'grapefruit', 'orange'}
```

The intersection of two sets represents the common values between two sets:




```
print('Intersection:', s1 & s2)
```

This generates

```
Intersection: {'banana'}
```

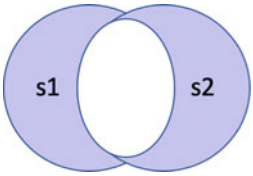
The difference between two sets is the set of values in the first set that are not in the second set:


```
print('Difference:', s1 - s2)
```

which produces the output

```
Difference: {'apple', 'orange'}
```

The symmetric difference represents all the unique values in the two sets (that is it is the inverse of the intersection:




```
print('Symmetric Difference:', s1 ^ s2)
```

The output from this final operation is

```
Symmetric Difference: {'orange', 'apple', 'lime', 'grapefruit'}
```

In addition to the operators there are also method versions:

- `s1.union(s2)` is the equivalent of `s1 | s2`
- `s1.interaction(s2)` is the equivalent of `s1 & s2`
- `s1.difference(s2)` is the equivalent of `s1 - s2`
- `s1.symmetric_difference(s2)` is the equivalent of `s1 ^ s2`

32.7 Set Methods

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

Method	Description
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

32.8 Online Resources

Online resources on sets are listed below:

- https://docs.python.org/3/tutorial/datastructures.html Python Tutorial on data structures.
- https://docs.python.org/3/tutorial/datastructures.html#sets the online Set tutorial.

- https://www.python-course.eu/python3_sets_frozensets.php A tutorial on sets and frozen sets.
- https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset for sets

32.9 Exercises

The aim of this exercise is to use a `Set`.

Create two sets of students, one for those who took an exam and one for those that submitted a project. You can use simple strings to represent the students, for example:

```
# Set up sets
exam = {'Andrew', 'Kirsty', 'Beth', 'Emily', 'Sue'}
project = {'Kirsty', 'Emily', 'Ian', 'Stuart'}

# Output the basic sets
print('exam:', exam)
print('project:', project)
```

Using these sets answer the following questions:

- Which students took both the exam and submitted a project?
- Which students only took the exam?
- Which students only submitted the project?
- List all students who took either (or both) of the exam and the project.
- List all students who took either (but *not* both) of the exam and the project.