SET DATA TYPES

DR IAN CORNELIUS

HELLO

- Learning Objectives:
    1. Understand the set data types that are built-in to Python
    2. Demonstrate the ability to use these set data types

INTRODUCTION TO SETS

- Sets are used to store multiple items into a single variable
- They are considered to be:
  - **unordered**: the items do not have a defined order and they can appear in a different order each time they are used
  - **changeable**: the items of a set are mutable, meaning that items can be added or removed
  - **no duplicates allowed**: duplicates are not allowed as a set is unordered
- The size of a set (or the number of items stored in a set) can be determined using the `len()` function

CREATING A SET

- Sets are created by using a set of curly braces (`{}`)
- Other data types can be type-casted as a set by using its constructor

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
setExample2 = set(["4061CEM", "Programming", "Algorithms"])
```

```
setExample1 = {'Algorithms', 'Programming', '4061CEM'}
setExample2 = {'Algorithms', 'Programming', '4061CEM'}
```

ITEMS OF A SET

- The items of a set can be any data type
    - i.e. it can be a mixture of data types such as booleans, strings or integers
- However, the `list`, `tuple` and `set` data types cannot be hashed or used inside of a set

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
setExample2 = {4061, "Programming and Algorithms", True}
```

```
setExample1 = {'Algorithms', 'Programming', '4061CEM'}
setExample2 = {'Programming and Algorithms', 4061, True}
```

ACCESSING SET ITEMS

- Items **cannot** be accessed in a set by referring to it via an index
- Instead, items can be accessed using a `for` loop

MODIFYING A SET (1)

- Sets are considered to be mutable, and as such we are able to add and remove items to them

ADDING AN ITEM

- Items can be added into the set using the `add()` function

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1 = {'Algorithms', 'Programming', 'Dr Ian Cornelius', '4061CEM'}
```

MODIFYING A SET (2)

REMOVING ITEMS FROM A SET I

- Items can be removed from a set using the `remove()` function
    - this will search the set for a specific value and then remove it
- If the item does not exist, an error will be thrown

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1.remove("Programming")
```

```
[Before] setExample1 = {'Algorithms', 'Programming', '4061CEM'}
[After] setExample1 = {'Algorithms', '4061CEM'}
```

MODIFYING A SET (3)

REMOVING ITEMS FROM A SET II

- Items can also be removed from a set using the `pop()` function
- This will only remove the last item from the set
    - but you will not know which item will be removed, as the items in a set can constantly change index

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1.pop()
```

```
[Before] setExample1 = {‘Algorithms’, ‘Programming’, ‘4061CEM’}
[After] setExample1 = {‘Programming’, ‘4061CEM’}
```

MODIFYING A SET (4)

REMOVING ITEMS FROM A SET III

- Items can also be removed from a set using the `discard()` function
  - this will search the set for a specific value and then remove it
- If the item does not exist an error *will not** be thrown

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1.discard("Programming")
```

```
[Before] setExample1 = {'Algorithms', 'Programming', '4061CEM'}
[After] setExample1 = {'Algorithms', '4061CEM'}
```

MODIFYING A SET (5)

CLEARING A SET

- A set can be cleared of all its items, but still reserve its memory location by using the `clear()` function
  - this will empty the contents of a set and leave it empty, symbolised by the curly brackets (`{}`) or the `set()` constructor

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
setExample1.clear()
```

```
[Before] setExample1 = {'Algorithms', 'Programming', '4061CEM'}
[After] setExample1 = set()
```

MODIFYING A SET (6)

DELETING A SET

- The entire set can be deleted and removed from the memory using the `del` keyword

```
setExample1 = {"4061CEM", "Programming", "Algorithms"}
```

```
del setExample1
```

MERGING A SET (1)

- There are various methods of merging two sets together
    - `update()` and `union()`

UPDATING A SET

- The items of a set can be updated with the items of another set using the `update()` function

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards"}
```

```
setExample1.update(setExample2)
```

```
[Before] setExample1 = {'Programming and Algorithms 1', '4061CEM', 'Dr Ian Cornelius'}
[After] setExample1 = {'Legal and Ethical Foundations', 'Dr Ian Cornelius', 'Programming and Algorithms 1', '4061CEM',
'Mr Terry Richards', '4059CEM'}
```

MERGING A SET (2)

UNIONISING A SET

- The items of a set can be unionised with the items of another set using the `union()` function
  - this method will return a new set with both sets merged

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards"}
```

```
mergedSetExample1 = setExample1.union(setExample2)
```

```
mergedSetExample1 = {'Legal and Ethical Foundations', 'Dr Ian Cornelius', 'Programming and Algorithms 1', '4061CEM', 'Mr
Terry Richards', '4059CEM'}
```

MERGING A SET (3)

INTERSECTION OF A SET I

- The items that only exist in both sets can be kept using the `intersection_update()` function
    - this will update the set it is called upon with only the duplicate values, removing any unique values

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards", "Dr Ian Cornelius"}
```

```
setExample1.intersection_update(setExample2)
```

```
[Before] setExample1 = {'Programming and Algorithms 1', '4061CEM', 'Dr Ian Cornelius'}
[After] setExample1 = {'Dr Ian Cornelius'}
```

MERGING A SET (4)

INTERSECTION OF A SET II

- The items that only exist in both sets can be kept using the `intersection()` function
  - this method will create a new set with only the duplicated values

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards", "Dr Ian Cornelius"}
```

```
mergedSetExample1 = setExample1.intersection(setExample2)
```

```
mergedSetExample1 = {'Dr Ian Cornelius'}
```

MERGING A SET (5)

SYMMETRIC DIFFERENCE OF A SET I

- The items that do not exist in both sets can be kept using the `symmetric_difference_update()` function
  - this will update the set it has been called upon with only the unique values, removing any duplicate values

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards", "Dr Ian Cornelius"}
```

```
setExample1.symmetric_difference_update(setExample2)
```

```
[Before] setExample1 = {'Programming and Algorithms 1', '4061CEM', 'Dr Ian Cornelius'}
[After] setExample1 = {'Legal and Ethical Foundations', 'Programming and Algorithms 1', '4059CEM', '4061CEM', 'Mr Terry
Richards'}
```

MERGING A SET (6)

SYMMETRIC DIFFERENCE OF A SET (II)

- The items that do not exist in both sets can be kept using the `symmetric_difference()` function
    - this method will create a new set with only the unique values

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
setExample2 = {"4059CEM", "Legal and Ethical Foundations", "Mr Terry Richards", "Dr Ian Cornelius"}
```

```
mergedSetExample1 = setExample1.symmetric_difference(setExample2)
```

```
mergedSetExample1 = {'Legal and Ethical Foundations', 'Programming and Algorithms 1', '4059CEM', '4061CEM', 'Mr Terry Richards'}
```

COPYING A SET

- The method of copying a set by using `set2 = set1` is incorrect
  - this method creates a reference to `set1` and not an actual copy; therefore, any changes made in `set1` will occur in `set2`
- The correct process of copying a set can be achieved by the `copy()` function or the `set()` constructor itself

```
setExample1 = {"4061CEM", "Programming and Algorithms 1", "Dr Ian Cornelius"}
```

```
copySetExample1 = setExample1.copy()
copySetExample2 = set(setExample1)
```

GOODBYE

- Questions?
    - Post them in the **Community Page** on Aula
- Contact Details:
    - Dr Ian Cornelius, ab6459@coventry.ac.uk