# Sets

## Introduction to Programming

Rae Harbird, r.harbird@ucl.ac.uk

# Sets - definition

A set is a collection of unique items.

*A set of Toby Jugs, Birmingham Museum via Wikimedia*

# Creating a Set

- A set is created using a set literal or the set function.

```
studentNamesSet = { "Bogdan Ionita", "Xiang Li", "Asim Ali" }
```

- You cannot use {} to make an empty set in Python.
- Instead, use the `set()` function with no arguments or you can use the set function to convert any sequence into a set:

```
studentNamesList = ["Bogdan Ionita", "Xiang Li", "Asim Ali"]

studentNamesSet = set(studentNamesList)
```

# Set length and testing membership

- As with other python containers, you can use the `len()` function to obtain the number of elements in a set:

```
numberOfStudents = len(studentNamesSet)
```

- The `in` operator is used to test whether an element is a member of a set.

```
if "Xiang Li" in studentNamesSet :
    print("Xiang is taking COMP0015.")
else :
    print("Xiang is not registered on COMP0015.")
```

- To determine whether an element is not contained in the set, use the `not in` operator.

# Traversing a set

- Sets are unordered, you cannot access the elements of a set by position as you can with a list. Instead, use a `for` loop:

```
print("COMP0015 in term 1:")
for student in studentNamesSet :
    print(student)
```

- To use the elements in sorted order use the `sorted` function, which returns a list (not a set):

```
for student in sorted(studentNamesSet) :
    print(student)
```

# Adding items to a set

Just like other Python containers, you can use `add` to add items:

```
studentNamesSet.add("Ella White")
```

- **Remember** that a set cannot contain duplicate elements. If you attempt to add an element that is already in the set, there is no effect and the set is not changed.
- Take a look in the examples folder at the program `course_manager.py`. Run it and look at the code.

# Removing set elements

- There are two methods that can be used to remove individual elements from a set. The `discard` method removes an element if the element exists:

```
studentNamesSet.discard("Ella White")
```

- Again, it has no effect if the given element is not a member of the set.
- The `remove` method, on the other hand, removes an element if it exists, but raises an exception if the given element is not a member of the set:

```
studentNamesSet.remove("Ella White")   # Raises an exception
```

- We have not learned about exceptions yet so we will come back to this later.

# Emptying a set

- Finally, the `clear` method removes all elements of a set, leaving the empty set:

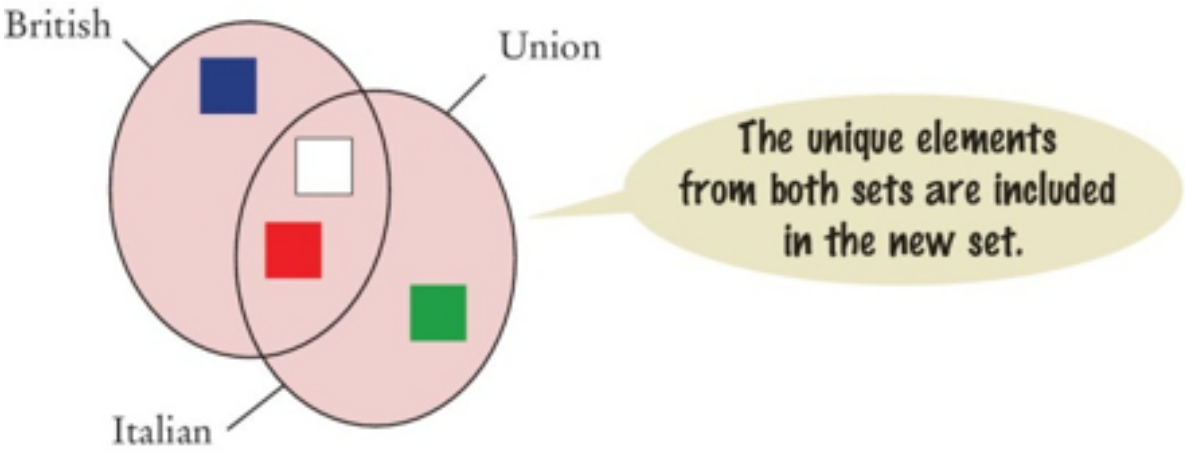
```
studentNamesSet.clear()   # set now has size 0
```

# Subset, union, intersection and difference

- Set operations such as subset, union, intersection and difference can be best illustrated with an example. Imagine that a country can be represented by the set of the colours in its flag:
  - Britain is represented by the set `{"red", "white", "blue"}`
  - Canada is `{"red", "white"}`
  - Italy is `{"red", "white", "green"}`.

# Union

- The `union` of two sets is the elements that appear in both sets without including duplicates.



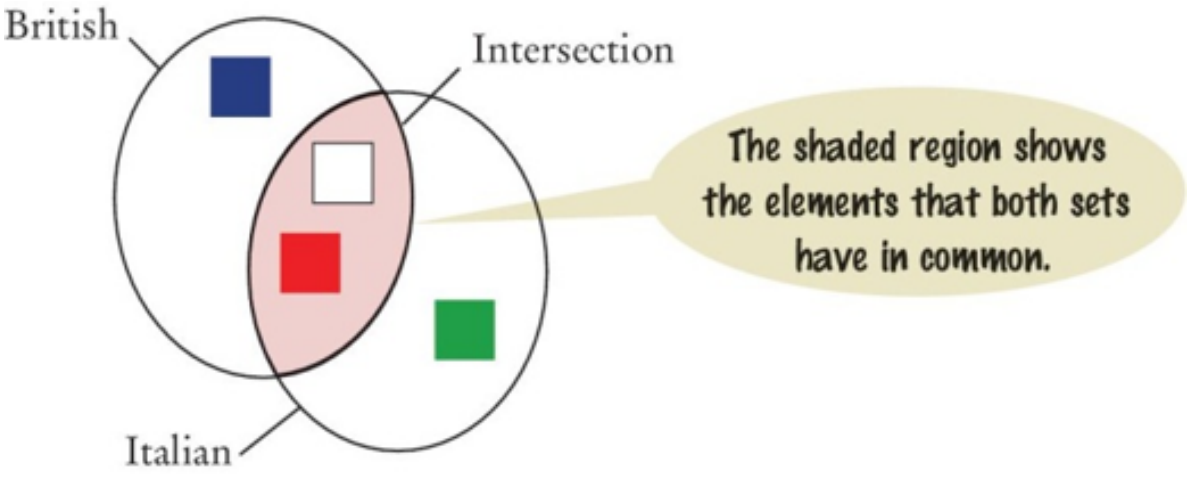

- In Python we would write:

```
britain.union(italy)
```

and the result is: `{"white", "red", "blue", "green"}`

# Intersection

- The intersection of two sets is the elements which appear in both sets, discarding duplicates.



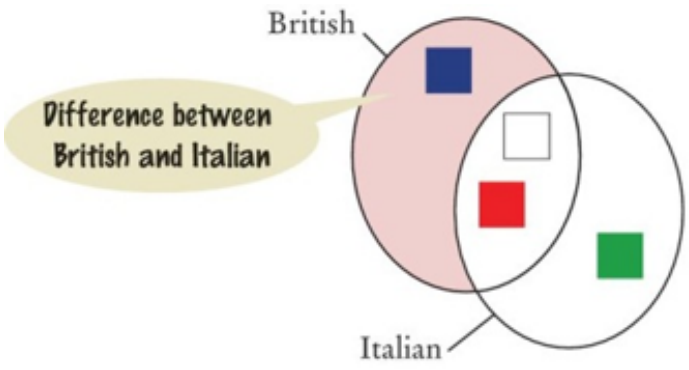

- In Python we write:

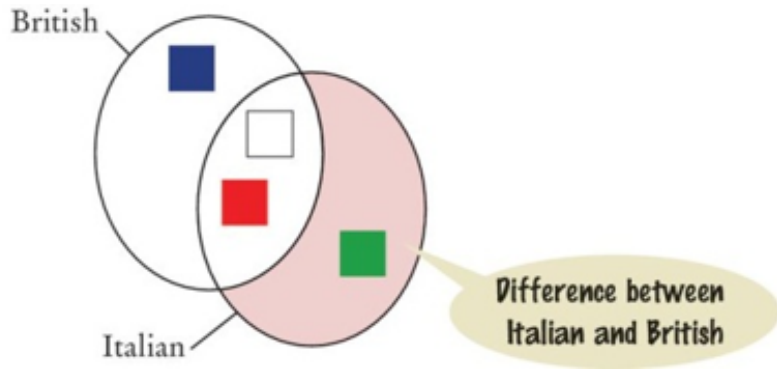
```
britain.intersection(italy)
```

and the result is: `{"white", "red"}`

# Difference

- The `difference` between two sets depends on the order of the operands.




- In python:

`britain.difference(italy)` is: `{'blue'}` and

`italy.difference(britain)` is `{'green'}`

*Note: the example and diagrams are from Python For Everyone, 2nd Ed., C. Horstmann and R. Necaise*

# Set operations

A complete list of set operations are **here**

| Operation | Description |
|---|---|
| `value in set` | returns `True` if the given value is found in the set |
| `value not in set` | returns `True` if the given value is *not* found |
| `len(set)` | number of elements in the set |
| `str(set)` | returns string representation such as `"{'a', 'b', 'c'}"` |
| `set.add(value)` | adds the given value to the set, if not already present |
| `set.clear()` | removes all elements |
| `set.isdisjoint(set2)` | returns `True` if there are no elements in common between `set` and `set2` |
| `set.pop()` | removes and returns one element from the set |
| `set.remove(value)` | removes the given value from the set, if present |
| `set.update(sequence)` | adds all values from the sequence to the set, if not already present |

# Question 1

- Consider the following code segment. What is true about these sets?

```
names = set(["Jane", "Joe", "Amy", "Lisa"])
names1 = set(["Joe", "Amy", "Lisa"])
names2 = set(["Jane", "Joe"])
```

(i) `names2` is a subset of `names`.
(ii) `names2` is not a subset of `names`.
(iii) `names2` is an intersection of the set `names` and `names1`.
(iv) `names2` is a union of `names` and `names1`.

# Question 2

- Given the following set definitions, which statement correctly determines whether `colours` is a subset of `rainbow`?

```
rainbow = {"red", "orange", "yellow", "green", "blue", "indigo", "viole
colours = {"red", "orange", "blue", "green"}
```

(i)
```
if colours in rainbow :
    print("rainbow colours")
```

(ii)
```
if colours.issubset(rainbow) :
    print("rainbow colours")
```

(iii)
```
if rainbow.issubset(colours) :
print("rainbow colours")
```

(iv)
```
if colours is rainbow :
    print("rainbow colours")
```

# Answer 1

- Consider the following code segment. What is true about these sets?

```
names = set(["Jane", "Joe", "Amy", "Lisa"])
names1 = set(["Joe", "Amy", "Lisa"])
names2 = set(["Jane", "Joe"])
```

(i) `names2` is a subset of `names`. **True**
(ii) `names2` is not a subset of `names`. **False**
(iii) `names2` is an intersection of the set `names` and `names1`. **False**
(iv) `names2` is a union of `names` and `names1`. **False**

# Answer 2

- Given the following set definitions, which statement correctly determines whether colours is a subset of rainbow?

```
rainbow = {"red", "orange", "yellow", "green", "blue", "indigo", "v
colours = {"red", "orange", "blue", "green"}
```

(ii)
```
if colours.issubset(rainbow) :
    print("rainbow colours")
```