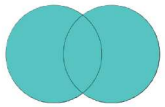
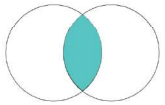
# SET OPERATIONS

- List Limitations
- Dictionary Basics
- Modifying Dictionaries
- Dictionary Methods
- Nested Dictionaries
- Sets

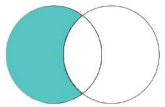
Python has useful **operations** that can be performed between sets



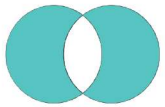
**union** — *Returns all unique values in both sets* — **set1.union(set2)**



**intersection** — *Returns values present in both sets* — **set1.intersection(set2)**



**difference** — *Returns values present in set 1, but not set 2* — **set1.difference(set2)**

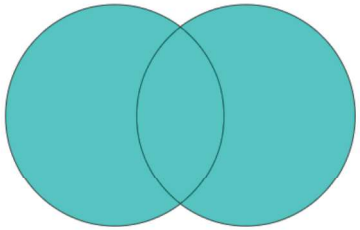


**symmetric difference** — *Returns values not shared between sets (opposite of intersection)* — **set1.symmetric_difference(set2)**

**PRO TIP:** Chain set operations to capture the relationship between three or more sets, for example – set1.union(set2).union(set3)

# UNION

- List Limitations
- Dictionary Basics
- Modifying Dictionaries
- Dictionary Methods
- Nested Dictionaries
- Sets



**Union** returns all unique values in both sets

```
friday_items = {'snowboard', 'snowboard', 'skis', 'snowboard', 'sled'}

saturday_items = {'goggles', 'helmet', 'snowboard', 'skis', 'goggles'}

friday_items.union(saturday_items)
```

```
{'goggles', 'helmet', 'skis', 'sled', 'snowboard'}
```

*All values from both sets are returned, without duplicates*

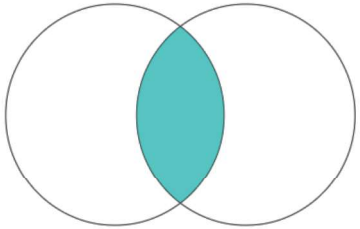
```
sunday_items = {'coffee'}

friday_items.union(saturday_items).union(sunday_items)
```

```
{'coffee', 'goggles', 'helmet', 'skis', 'sled', 'snowboard'}
```

*All values from the three sets are returned by chaining two union operations*

# INTERSECTION





**Intersection** returns the values present in both sets

```
friday_items = {'snowboard', 'snowboard', 'skis', 'snowboard', 'sled'}

saturday_items = {'goggles', 'helmet', 'snowboard', 'skis', 'goggles'}

friday_items.intersection(saturday_items)
```

```
{'skis', 'snowboard'}
```

*Only the values in both sets are returned, without duplicates*

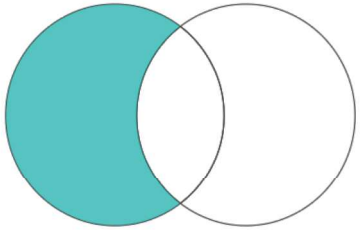
```
sunday_items = {'coffee'}

friday_items.intersection(saturday_items).intersection(sunday_items)
```

```
set()
```

*Since no value is present in all three sets, an empty set is returned*

# DIFFERENCE





**Difference** returns the values present set 1, but not set 2 *(the order matters)*

```
friday_items = {'snowboard', 'snowboard', 'skis', 'snowboard', 'sled'}

saturday_items = {'goggles', 'helmet', 'snowboard', 'skis', 'goggles'}

friday_items.difference(saturday_items)
```

```
{'sled'}
```

*'sled' is the only value in friday_items that is NOT in saturday_items*

```
saturday_items - friday_items
```

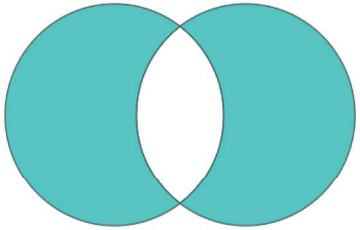
```
{'goggles', 'helmet'}
```

***If you reverse the order, the output changes*** *– 'goggles' and 'helmet' are in saturday_items but NOT in friday_items*

*Note that the subtraction sign can be used instead of difference*

# SYMMETRICAL DIFFERENCE





**Symmetrical difference** returns all values not shared between sets

```
friday_items = {'snowboard', 'snowboard', 'skis', 'snowboard', 'sled'}

saturday_items = {'goggles', 'helmet', 'snowboard', 'skis', 'goggles'}

friday_items.symmetric_difference(saturday_items)
```

```
{'goggles', 'helmet', 'sled'}
```

*'sled' is only in set 1, and 'goggles' and 'helmet' are only in set 2*

# SET USE CASES



1. Sets are more efficient than lists for performing **membership tests**

```
time_list = list(range(1000000))
time_set = set(range(1000000))
```

*Using* ***Lists***

```
%%time
100000 in time_list
```

```
CPU times: user 5.07 ms, sys: 454 µs, total: 5.53 ms
```

*Using* ***Sets***

```
%%time
100000 in time_set
```

```
CPU times: user 5 µs, sys: 1e+03 ns, total: 6 µs
```

Sets are implemented as **hash tables**, which makes looking up values extremely fast; the downside is that they cannot preserve order (lists rely on dynamic arrays that preserve order but have slower performance)

# SET USE CASES

- List Limitations
- Dictionary Basics
- Modifying Dictionaries
- Dictionary Methods
- Nested Dictionaries
- Sets

2. Sets can **gather unique values** efficiently without looping

*Using* ***Lists***

```
%%time
unique_items = []

for item in shipments_today:
    if item not in unique_items:
        unique_items.append(item)

unique_items
```

```
CPU times: user 27 µs, sys: 1 µs, total: 28 µs
['ski', 'snowboard', 'helmet', 'hat', 'goggles']
```

*Using* ***Sets***

```
%%time
list(set(shipments_today))
```

```
CPU times: user 9 µs, sys: 0 ns, total: 9 µs
['snowboard', 'ski', 'helmet', 'hat', 'goggles']
```

# SET USE CASES



3. Set operations can find the **data shared, or not shared, between items** without looping

*Using* ***Lists***

```
shipment_today = ['ski', 'snowboard', 'ski', 'ski', 'helmet', 'hat', 'goggles']
shipment_yesterday = ['hat', 'goggles', 'snowboard', 'hat', 'bindings']

unique_today = []
for item_t in shipment_today:
    if item_t not in shipment_yesterday:
        if item_t not in unique_today:
            unique_today.append(item_t)

unique_today
```

```
['ski', 'helmet']
```

*Using* ***Sets***

```
set(shipment_today).difference(set(shipment_yesterday))
```

```
{'helmet', 'ski'}
```