Extra practice – W6S1

The set type

The set type

- **Sets** are another **collections of variables** type in Python.
- Very similar to **lists** and **tuples**, their values are listed between **curly brackets {}.**

- **Sets** are used to implement **concepts from mathematical set theory** (intersection of sets, union of sets, etc.)

```
1  # Sets are another collection type
2  # Does not allow for duplicates
3  a_set = {1, 2, 3, 4, 4, 1}
4  print(a_set)
5  print(type(a_set))
6  print(len(a_set))
7  # Conversions lists <-> sets
8  another_set = set([1, 2, 3, 4, 4, 1])
9  print(another_set)
```

```
{1, 2, 3, 4}
<class 'set'>
4
{1, 2, 3, 4}
```

The set type

- **Sets** are another **collections of variables** type in Python.
- Very similar to **lists** and **tuples**, their values are listed between **curly brackets {}.**

- **Key difference #1:** elements in sets **have no duplicates.** If duplicates exist on creation, they will be removed automatically.

```
1  # Sets are another collection type
2  # Does not allow for duplicates
3  a_set = {1, 2, 3, 4, 4, 1}
4  print(a_set)
5  print(type(a_set))
6  print(len(a_set))
7  # Conversions lists <-> sets
8  another_set = set([1, 2, 3, 4, 4, 1])
9  print(another_set)
```

```
{1, 2, 3, 4}
<class 'set'>
4
{1, 2, 3, 4}
```

Sets are very limited

- **Membership** (**in**) works on sets.
- **Traversing** a set with a **for** loop works.

However,

- Most methods from lists/tuples do not work.
- No indexing, slicing, updating.
- Very limited range of applications.

```
1 a_set = {1, 2, 3, 4}
2 print(a_set)
3 # Membership check works
4 print(1 in a_set)
5 print(5 in a_set)
6 # Not subscriptable!
7 # (No indexing, slicing, updating, etc.)
8 print(a_set[0])
```

```
{1, 2, 3, 4}
True
False

---------------------------------------------
TypeError                                 Tra
<ipython-input-2-8ee8c7232731> in <module>
      6 # Not subscriptable!
      7 # (No indexing, slicing, updating, et
----> 8 print(a_set[0])

TypeError: 'set' object is not subscriptable
```

```
1 a_set = {1, 2, 3, 4}
2 for val in a_set:
3     print(val)
```

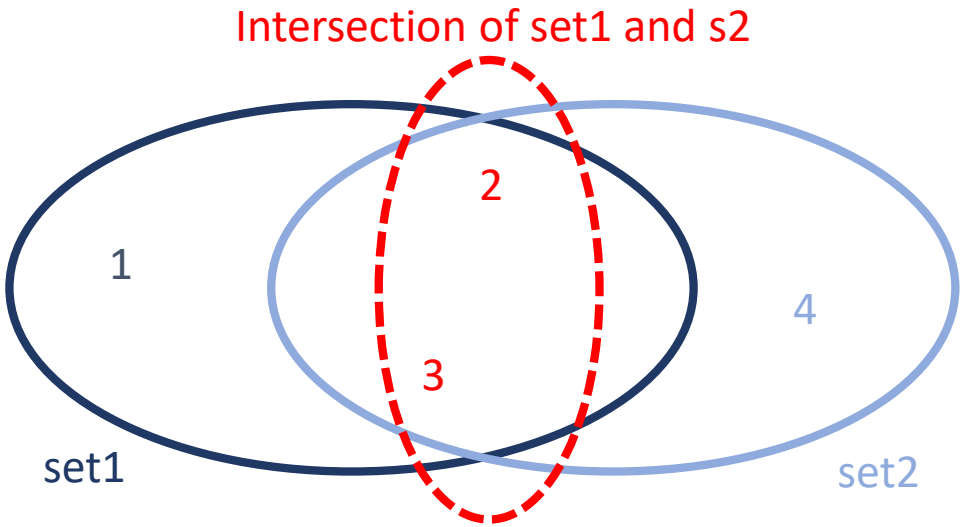
```
1
2
3
4
```

Sets: mathematical concepts

- However, sets have implemented **mathematical concepts from set theory**.
- **Intersection:** elements that appear in two given sets.
- Can be computed using the **intersection()** method.

```
1  set1 = {1, 2, 3}
2  set2 = {2, 3, 4}
3  # Intersection: elements in both sets
4  intersect_set = set1.intersection(set2)
5  print(intersect_set)
```

```
{2, 3}
```



Sets: mathematical concepts

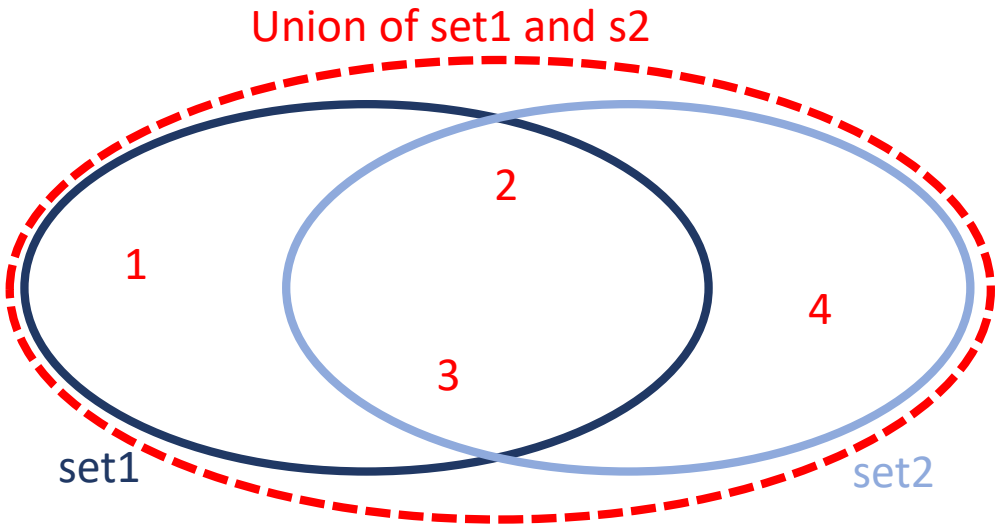
- However, sets have implemented **mathematical concepts from set theory**.

```python
set1 = {1, 2, 3}
set2 = {2, 3, 4}
# Union: elements in at least one of the sets
union_set = set1.union(set2)
print(union_set)
```

```
{1, 2, 3, 4}
```

- **Union:** elements that appear in either of two given sets.

- Can be computed using the **union()** method.

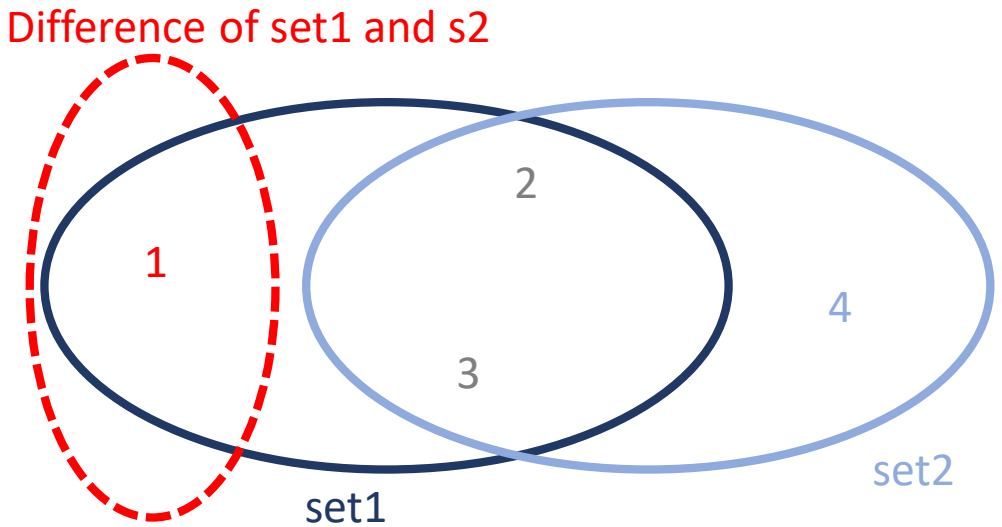


Sets: math. concepts

- However, sets have implemented **mathematical concepts from set theory**.
- **Difference:** elements that appear in one set but not the other.
- Can be computed using the **difference()** method.

```
1  set1 = {1, 2, 3}
2  set2 = {2, 3, 4}
3  set3 = {1, 2, 3, 4, 5}
4  # Difference: elements in set1 but not set2?
5  print(set1.difference(set2))
6  print(set1 - set2)
7  # Difference: elements in set3 but not set2?
8  print(set3.difference(set2))
9  print(set3 - set2)
```

```
{1}
{1}
{1, 5}
{1, 5}
```

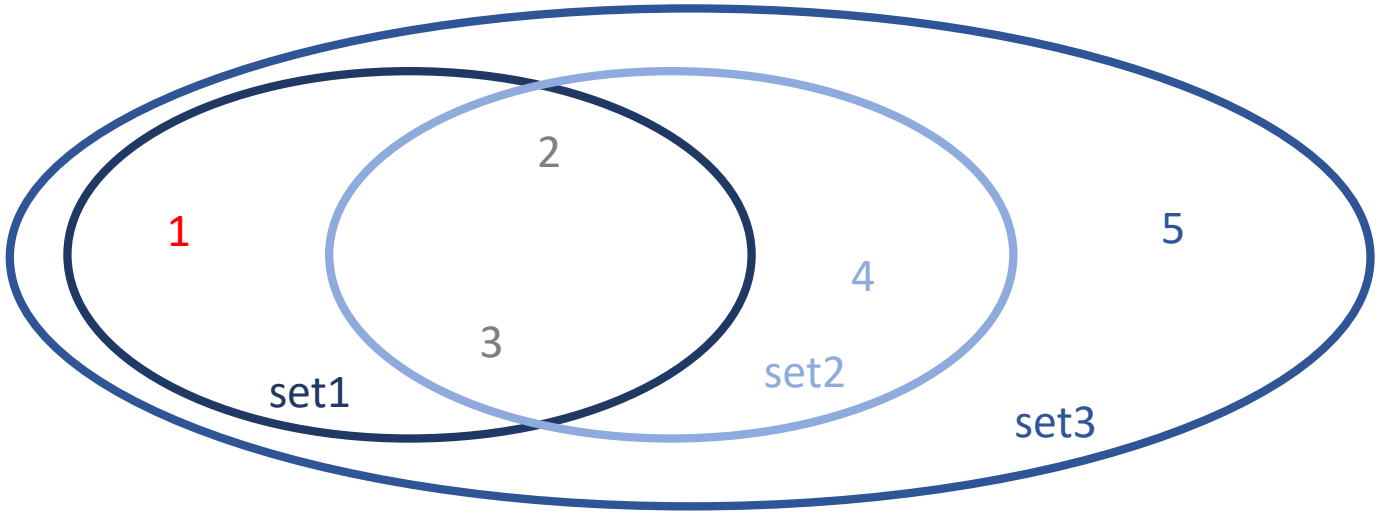


Sets: math. concepts

- However, sets have implemented **mathematical concepts from set theory**.
- **Subset:** are all elements from one set also in another?
- Can be computed using the **issubset()** method, and even comparison operators (**<=**, **>=**).

```
1 set1 = {1, 2, 3}
2 set2 = {2, 3, 4}
3 set3 = {1, 2, 3, 4, 5}
4 # Are all elements in set1 also in set2?
5 print(set1.issubset(set2))
6 print(set1 <= set2)
7 # Are all elements in set1 also in set3?
8 print(set1.issubset(set3))
9 print(set1 <= set3)
```

```
False
False
True
True
```



Matt's Great advice #10

Matt's Great Advice #10: Lists are more versatile, sets/tuples are more efficient.

What should we prefer between lists, tuples and sets? Well, it depends.

- Overall, **lists** are **more versatile** (more functions and methods).
- **Tuples** are **more efficient** for basic operations.
- **Sets** offer very **specific methods** covering mathematical concepts from **set theory**.

Use all three, based on your needs, and use **types conversion** if needed!

Activity 1 - Friends list suggestion

Let us consider three persons, and their respective lists of friends, defined as sets below.

```
matt_friends_set = {"Oka", "Sergey", "Chris"}
sergey_friends_set = {"Chris", "Sergey", "Norman", "Tony"}
chris_friends_set = {"Norman", "Sergey", "Natalie"}
```

Your objective is to write a function, **suggest_friends()**, which will receive all three lists as its parameters, and will suggest friends for the first person (here, Matt).

Activity 1 - Friends list suggestion

Let us consider three persons, and their respective lists of friends, defined as sets below.

matt_friends_set = {"Oka", "Sergey", "Chris"}

sergey_friends_set = {"Chris", "Sergey", "Norman", "Tony"}

chris_friends_set = {"Norman", "Sergey", "Natalie"}

The function should suggest a friend **suggest_friends()**, if and only if:

- Matt is not already friend with this person,
- Both Sergey and Chris are friends with this person.
- Ideally, your function should be a single line function.