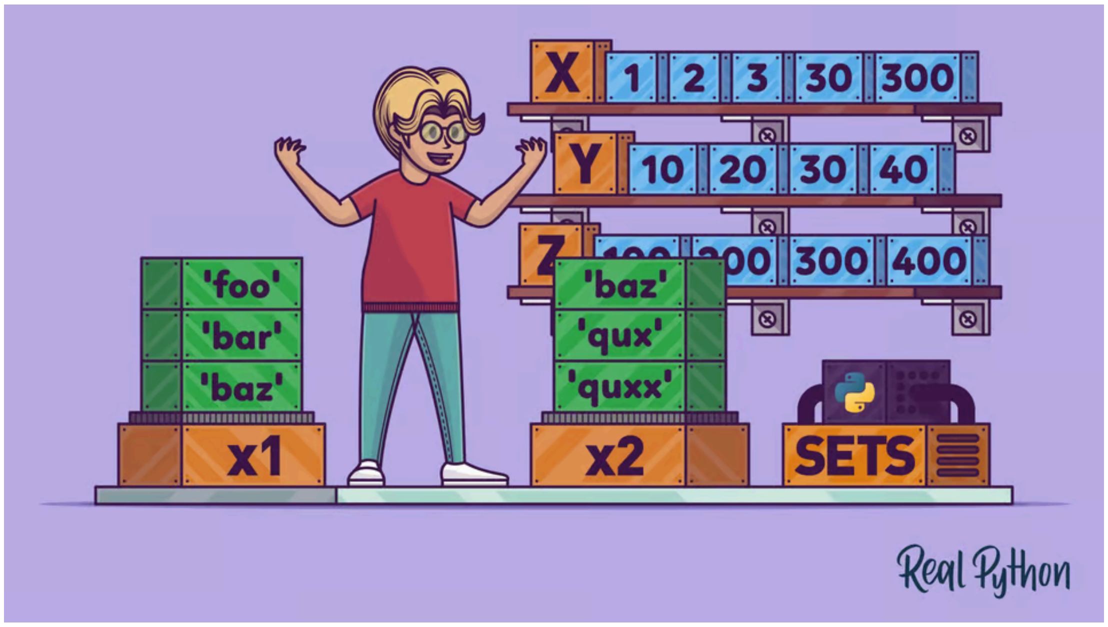
7. Set

Sets are unordered collections of unique elements in Python. They are similar to lists but with the key difference that they do not allow duplicate elements. Sets are often used for membership testing, removing duplicates from a list, and performing set operations like union, intersection, and difference.

We'll learn about the following topics:

- 7.1. Creating Sets
- 7.2. Set Properties
- 7.3. Set Operators
- 7.4. Built-in Set Methods
- 7.5. Frozen Sets



Name	Type in Python	Description	Example
Sets	set	unordered collection of unique elements.	{10, 'hello'}

7.1. Creating Sets:

Sets in Python are created using curly braces `{}` . You can include elements within the braces, separated by commas.

```
In [1]: set1 = {10, "hello", 3.14, True}
```

```
In [2]: type(set1)
```

```
Out[2]: set
```

Also a set can be created with the built-in `set()` function.

```
In [3]: a = set()
```

```
In [4]: type(a)
```

```
Out[4]: set
```

```
In [5]: # We add to sets with the add() method
        a.add(89)
        a.add('hello')
        a.add(2.0)
```

```
In [6]: a
```

```
Out[6]: {2.0, 89, 'hello'}
```

7.2. Set Properties:

- **Unique Elements**

```
In [7]: #Create a list with repeated items
        lst = [1,1,2,2,3,4,5,6,1,1]
```

```
In [8]: #Cast as set to get unique values
        set(lst)
```

```
Out[8]: {1, 2, 3, 4, 5, 6}
```

After converting the list to set with `set()` function, only unique items have remained. That's because a set is only concerned with unique elements.

- **Unordered**

```
In [9]: a = {89, 'hello', 2.0}
        b = {2.0, 89, 'hello'}
```

```
In [10]: a == b
```

```
Out[10]: True
```

7.3. Set Operators:

```
In [11]: c = set()

         c.add('world')
         c.add('20')
         c.add(89)
```

- **Union**: Sets union can be performed with the `|` operator. The union of two sets contains all elements that are in either set or both sets.

```
In [12]: a | c
```

```
Out[12]: {2.0, '20', 89, 'hello', 'world'}
```

- **Intersection**: Sets intersection can be performed with the `&` operator. The intersection of two sets contains only the elements that are present in both sets.

```
In [13]: a & c
```

```
Out[13]: {89}
```

- **Difference**: `a - c` return the set of all elements that are in a but not in c.

```
In [14]: a - c
```

```
Out[14]: {2.0, 'hello'}
```

- **Symmetric Difference**: Sets symmetric difference can be performed with the `^` operator. The symmetric difference of two sets contains the elements that are in either set but not in both sets.

```
In [15]: a ^ c
```

```
Out[15]: {2.0, '20', 'hello', 'world'}
```

- **Subset**: The `<=` operator is used to check if one set is a subset of another set in Python.

```
In [16]: d = set()
         d.add('hello')

         d <= a
```

```
Out[16]: True
```

- **Proper Subset**: The proper subset relationship between two sets can be determined using the `<` operator in Python. A proper subset is the same as a subset, except that the sets can’t be identical. While a set is considered a subset of itself, it is not a proper subset of itself.

```
In [17]: a < a
```

```
Out[17]: False
```

- **Superset**: The `>=` operator is used to check if one set is a superset of another. A superset contains all the elements of another set and possibly more.

```
In [18]: a >= d
```

```
Out[18]: True
```

- **Proper Superset**: The `>` operator is used to check if one set is a proper superset of another set in Python. A proper superset is the same as a superset, except that the sets can’t be identical.

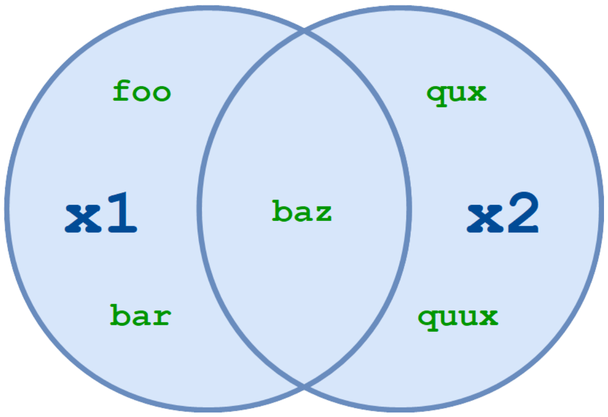
```
In [19]: a > d
```

```
Out[19]: True
```

7.4. Built-in Set Methods:

Method	Description
union(set)	merge sets and keep unique elements from all sets
intersection(set)	return the set of elements present in all sets
difference(set)	x1.difference(x2) return the set of all elements that are in x1 but not in x2
symmetric_difference(set)	return the set of all elements in either sets
isdisjoint(set)	determines whether or not two sets have any elements in common. returns True if they have no elements in common
issubset(set)	determine whether one set is a subset of the other
issuperset(set)	set a is considered as the superset of b, if all the elements of set b are the elements of set a
update(set)	adds any elements in new set that our set does not already have
intersection_update(set)	retain only elements found in both
difference_update(set)	it's like difference method except it updates the original set
symmetric_difference_update(set)	it's like symmetric difference method except it updates the original set
add(set)	add an item to the set
remove(m)	remove m from the set

Method	Description
discard(m)	remove m from the set. However, if m is not in set, discard does nothing instead of raising an exception
pop()	removes a random element from the set
clear()	removes all elements from the set



```
In [20]: a.union(c)
```

```
Out[20]: {2.0, '20', 89, 'hello', 'world'}
```

There is a subtle difference between `|` operator and `.union()`. When you use the `|` operator, both operands must be sets. The `.union()` method, on the other hand, will take any iterable as an argument, convert it to a set, and then perform the union.

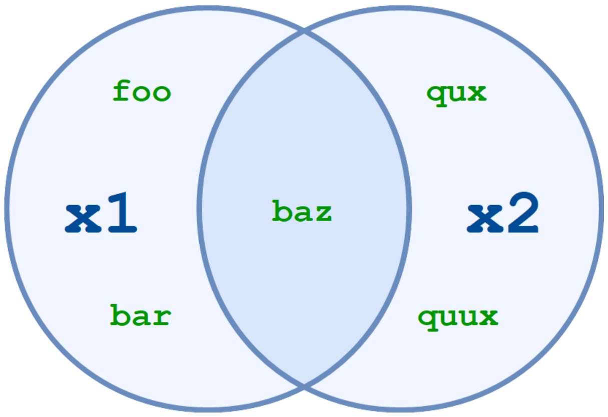
```
In [21]: a.union(('a', 'b', 28))
```

```
Out[21]: {2.0, 28, 89, 'a', 'b', 'hello'}
```

```
In [22]: a | ('a', 'b', 28)
```

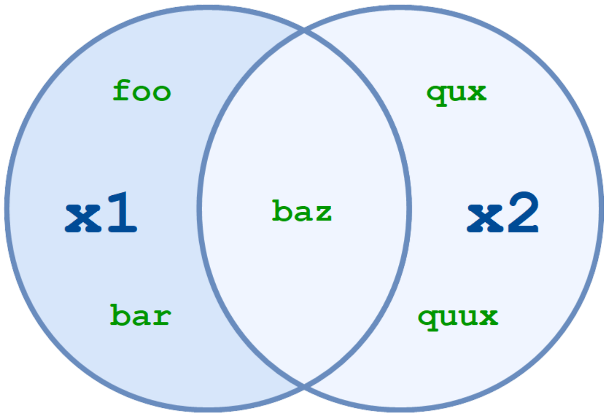
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10548\564823127.py in <module>
----> 1 a | ('a', 'b', 28)

TypeError: unsupported operand type(s) for |: 'set' and 'tuple'
```



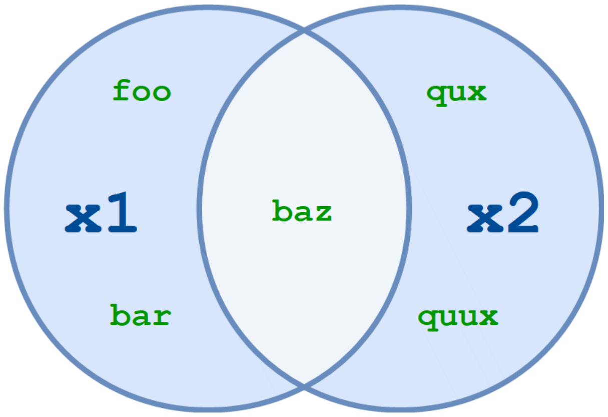
```
In [23]: a.intersection(c)
```

```
Out[23]: {89}
```



```
In [24]: a.difference(c)
```

```
Out[24]: {2.0, 'hello'}
```



```
In [25]: a.symmetric_difference(c)
```

```
Out[25]: {2.0, '20', 'hello', 'world'}
```

```
In [26]: a.isdisjoint(c)
```

```
Out[26]: False
```

```
In [27]: d.issubset(a)
```

```
Out[27]: True
```

```
In [28]: a.issuperset(d)
```

```
Out[28]: True
```

```
In [29]: a.update(['a', 'A'])

         a
```

```
Out[29]: {2.0, 89, 'A', 'a', 'hello'}
```

```
In [30]: #permanently changes the set
         a.intersection_update(d)
```

```
In [31]: a
```

```
Out[31]: {'hello'}
```

```
In [32]: a.remove('hello')
```

```
In [33]: a
```

```
Out[33]: set()
```

```
In [34]: a.discard('hello')
```

```
In [35]: a.remove('hello')
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10548\918437660.py in <module>
----> 1 a.remove('hello')

KeyError: 'hello'
```

7.5. Frozen Sets:

Python provides another built-in type called a frozenset, which is in all respects exactly like a set, except that a frozenset is immutable.

```
In [36]: x = frozenset(['a', 45, '78'])
```

```
In [37]: x
```

```
Out[37]: frozenset({45, '78', 'a'})
```

Any attempt to modify a frozenset will fail.

```
In [38]: x.add('b')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_10548\3997461639.py in <module>
----> 1 x.add('b')

AttributeError: 'frozenset' object has no attribute 'add'
```