Data Sctructures in Python

Lists Tuples Sets and Dictionaries

1. Lists

A list is a collection of items in a particular order.

You can make a list that includes the letters of the alphabet, the digits from 0–9, or the names of all the people in your family.

You can put anything you want into a list, and the items in your list don't have to be related in any particular way.

Because a list usually contains more than one element, it's a good idea to make the name of your list plural, such as letters, digits, or names.

They are **ordered**, **mutable (changeable)**, and **allow duplicate** elements.**

In [2]:
```
lst = [1,2,3,"hello", 3.45]
print(lst)
```
```
[1, 2, 3, 'hello', 3.45]
```

In [19]:
```
List_2D=[[0]*8]*10
print(List_2D)
```
```
[[0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0,
0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0,
0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0, 0, 0], [0, 0, 0, 0, 0, 0,
0, 0]]
```

Accessing Elements (Indexing & Splicing)

In [3]:
```
lst[-1]
```
Out[3]:
```
3.45
```

In [4]:
```
lst[2:4]
```
Out[4]:
```
[3, 'hello']
```

In [5]:
```
lst[-2:]
```
Out[5]:
```
['hello', 3.45]
```

In [6]:
```
lst[:-2]
```

```
Out[6]: [1, 2, 3]
```

```
In [7]: lst[::3]
```

```
Out[7]: [1, 'hello']
```

```
In [8]: lst[::-2]
```

```
Out[8]: [3.45, 3, 1]
```

```
In [1]: Languages = ["java", "c++", "python", "fortran"]
        print(Languages[0].title())
```

```
Java
```

```
In [3]: #f-strings
        print(f"my favourite programming language is {Languages[-1].title()}")
```

```
my favourite programming language is Fortran
```

Modify Elements in a List

```
In [4]: Languages = ["java", "c++", "python", "fortran"]
        print(Languages)

        Languages[0] = "R"
        print(Languages)
```

```
['java', 'c++', 'python', 'fortran']
['R', 'c++', 'python', 'fortran']
```

List Methods

> append(): Adds an element to the end of the list.
>
> extend(): Extends the list by appending elements from another list.
>
> insert(): Inserts an element at a specified position.
>
> remove(): Removes the first occurrence of a value.
>
> del statment: Removes the element by its index and it doesn't re-assign it.
>
> pop(): Removes and returns the element at a specified index.
>
> index(): Returns the index of the first occurrence of a value.
>
> count(): Returns the number of occurrences of a value.
>
> sort(): Sorts the list in ascending order.
>
> reverse(): Reverses the elements of the list.

```
In [9]: lstMethod = [1,2,3,3,4,4,4,5]
        lstMethod.append([1,2,3])

        print(lstMethod)
```

```
[1, 2, 3, 3, 4, 4, 4, 5, [1, 2, 3]]
```

```
In [9]: Languages = ["java", "c++", "python", "fortran"]
        Languages.append(["dart", "kotlin"])
        print(Languages)
```

```
['java', 'c++', 'python', 'fortran', ['dart', 'kotlin']]
```

```
In [10]: lstMethod = [1,2,3,3,4,4,4,5]
         lstMethod.extend([1,2,3])
         print(lstMethod)
```

```
[1, 2, 3, 3, 4, 4, 4, 5, 1, 2, 3]
```

```
In [10]: Languages = ["java", "c++", "python", "fortran"]
         Languages.extend(["dart", "kotlin"])
         print(Languages)
```

```
['java', 'c++', 'python', 'fortran', 'dart', 'kotlin']
```

```
In [11]: lstMethod.insert(5, 99)
         print(lstMethod)
```

```
[1, 2, 3, 3, 4, 99, 4, 4, 5, 1, 2, 3]
```

```
In [11]: Languages = ["java", "c++", "python", "fortran"]
         Languages.insert(0, "kotlin")
         print(Languages)
```

```
['kotlin', 'java', 'c++', 'python', 'fortran']
```

Removing elements using remove() method "by Value"

```
In [12]: lstMethod.remove(99)
         print(lstMethod)
```

```
[1, 2, 3, 3, 4, 4, 4, 5, 1, 2, 3]
```

```
In [19]: Languages = ["java", "c++", "python", "fortran"]
         Languages.remove("java")
         print(Languages)
```

```
['c++', 'python', 'fortran']
```

```
In [20]: Languages.remove("c")
         print(Languages)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[20], line 1
----> 1 Languages.remove("c")
      2 print(Languages)

ValueError: list.remove(x): x not in list
```

note:

The remove() method deletes only the first occurrence of the value you specify. If there's a possibility the value appears more than once in the list, you'll need to use a loop to make sure all occurrences of the value are removed.

Removing elements using del statement

If you know the position of the item you want to remove from a list, you can use the del statement.

In [12]:
```
Languages = ["java", "c++", "python", "fortran"]
print(Languages)

del Languages[0]
print(Languages)
```

```
['java', 'c++', 'python', 'fortran']
['c++', 'python', 'fortran']
```

In [ ]:
```
Languages = ["java", "c++", "python", "fortran"]
print(Languages)

del Languages[0]
print(Languages)
```

Removing elements using pop() method

The pop() method removes the last item in a list, but it lets you work with that item after removing it.

The term pop comes from thinking of a list as a stack of items and popping one item off the top of the stack. In this analogy, the top of a stack corresponds to the end of a list.

In [13]:
```
lstMethod.pop()
print(lstMethod)
```

```
[1, 2, 3, 3, 4, 4, 4, 5, 1, 2]
```

In [15]:
```
Languages = ["java", "c++", "python", "fortran"]
print(Languages)

popped_Language = Languages.pop()
print(Languages)
print(popped_Language)
```

```
['java', 'c++', 'python', 'fortran']
['java', 'c++', 'python']
fortran
```

Popping Items from any Position in a List

You can use pop() to remove an item from any position in a list by including the index of the item you want to remove in parentheses.

In [18]:
```
Languages = ["java", "c++", "python", "fortran"]
print(Languages)
```

```
popped_Language_1 = Languages.pop(0)
print(f"The first Prog lang I learned is {popped_Language_1.upper()}")
```

```
['java', 'c++', 'python', 'fortran']
The first Prog lang I learned is JAVA
```

note:

If you’re unsure whether to use the del statement or the pop() method, here’s a simple way to decide: when you want to delete an item from a list and not use that item in any way, use the del statement; if you want to use an item as you remove it, use the pop() method.

In [14]:
```
lstMethod.index(4)
```

Out[14]:
```
4
```

In [15]:
```
lstMethod.count(4)
```

Out[15]:
```
3
```

Printing a List in Reverse Order

In [17]:
```
lstMethod.reverse()
print(lstMethod)
```

```
[5, 4, 4, 4, 3, 3, 2, 2, 1, 1]
```

In [25]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print(cars)
cars.reverse()
print(cars)
```

```
['bmw', 'audi', 'toyota', 'subaru']
['subaru', 'toyota', 'audi', 'bmw']
```

Sorting a List Permanently with the sort() Method

In [21]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort()
print(cars)
```

```
['audi', 'bmw', 'subaru', 'toyota']
```

In [22]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
cars.sort(reverse=True)
print(cars)
```

```
['toyota', 'subaru', 'bmw', 'audi']
```

In [16]:
```
lstMethod.sort()
print(lstMethod)
```

```
[1, 1, 2, 2, 3, 3, 4, 4, 4, 5]
```

Sorting a List Temporarily with the sorted() Function

To maintain the original order of a list but present it in a sorted order, you can use the sorted() function.

In [23]:
```
cars = ['bmw', 'audi', 'toyota', 'subaru']
print("Here is the original list:")
print(cars)
print("\nHere is the sorted list:")
print(sorted(cars))
print("\nHere is the original list again:")
print(cars)
```

```
Here is the original list:
['bmw', 'audi', 'toyota', 'subaru']

Here is the sorted list:
['audi', 'bmw', 'subaru', 'toyota']

Here is the original list again:
['bmw', 'audi', 'toyota', 'subaru']
```

Use Looping in Lists

In [26]:
```
squares = []
for value in range(1, 11):
    square = value ** 2
    squares.append(square)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

In [27]:
```
squares = []
for value in range(1, 11):
    squares.append(value ** 2)
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Looping through slice:

In [29]:
```
players = ['Asser', 'Ramy', 'Ali', 'Samy', 'Emad']
print("Here are the first three players on my team:")
for player in players[:3]:
    print(player.title())
```

```
Here are the first three players on my team:
Asser
Ramy
Ali
```

Copying a List

To copy a list, you can make a slice that includes the entire original list by omitting the first index and the second index ([:]). This tells Python to make a slice that starts at the first item and ends with the last item, produc□ing a copy of the entire list

In [41]:
```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods[:]
print("My favorite foods are:")
print(my_foods)
print("My friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake']
```

In [34]:
```
my_foods.append('Koushari')
friend_foods.append('ice cream')

print("My favorite foods are:")
print(my_foods)
print("My friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'Koushari']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

In [39]:
```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods #wrong now friend food is just reference to my food

my_foods.append('koushari')
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("My friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'koushari', 'ice cream']

My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'koushari', 'ice cream']
```

In [42]:
```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = my_foods.copy()

my_foods.append('koushari')
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("My friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'koushari']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

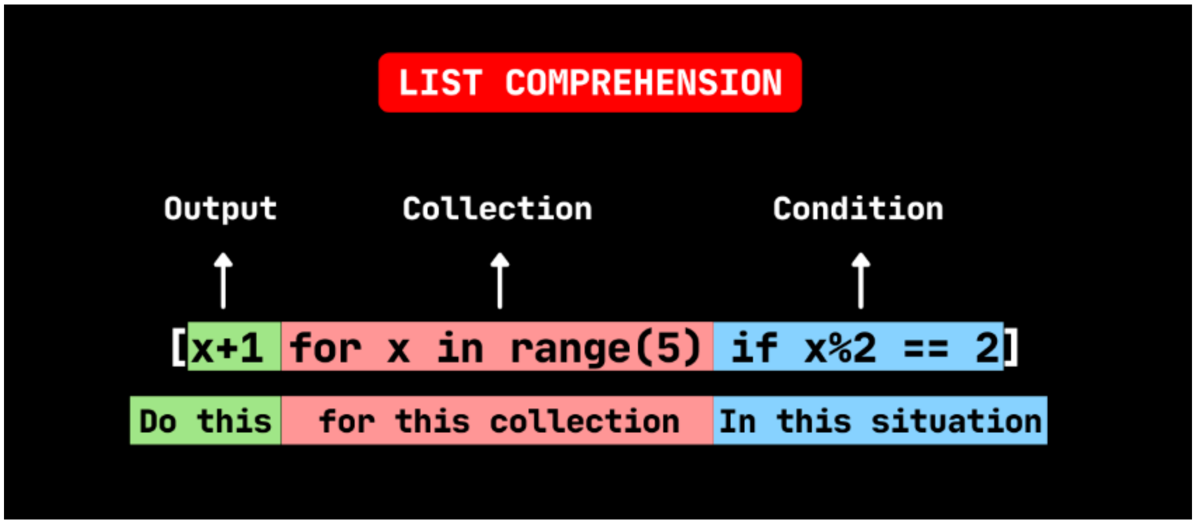
In [44]:
```
my_foods = ['pizza', 'falafel', 'carrot cake']
friend_foods = list(my_foods)

my_foods.append('koushariii')
```

```
friend_foods.append('ice cream')
print("My favorite foods are:")
print(my_foods)
print("My friend's favorite foods are:")
print(friend_foods)
```

```
My favorite foods are:
['pizza', 'falafel', 'carrot cake', 'koushariii']
My friend's favorite foods are:
['pizza', 'falafel', 'carrot cake', 'ice cream']
```

List Comprehensions



In [51]:
```
x = 10
lstComprehensions = [x+i for i in range(5) if i%2==0]
print(lstComprehensions)
```

```
[10, 12, 14]
```

In [28]:
```
squares = [value**2 for value in range(1, 11)]
print(squares)
```

```
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

2.Tuples

Python refers to values that cannot change as immutable, and an immutable list is called a tuple.

Tuples are **ordered**, ***immutable (unchangeable)*** and **allow duplicate** elements.

In [44]:
```
my_tuple = (1, 2, 3)
print(my_tuple)
```

```
(1, 2, 3)
```

In [40]:
```
my_tuple = (1, 2, 2)
print(my_tuple)
```

```
(1, 2, 2)
```

In [48]:
```
mixed_tuple=(5,4,10.5,"mostafa",True)
print(mixed_tuple)
```

```
(5, 4, 10.5, 'mostafa', True)
```

In [45]:
```
my_tuple = (10, 20, 30, "mohamed",True)
print(my_tuple[0])
print(my_tuple[-1])
print(my_tuple[:-2])
print(my_tuple[::-2])
```

```
10
True
(10, 20, 30)
(True, 30, 10)
```

In [102…
```
dimensions = (200, 50)
dimensions[0] = 250
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[102], line 2
      1 dimensions = (200, 50)
----> 2 dimensions[0] = 250

TypeError: 'tuple' object does not support item assignment
```

In [54]:
```
my_t = (3,)
print(type(my_t))
```

```
<class 'tuple'>
```

In [55]:
```
my_t = (3)
print(type(my_t))
```

```
<class 'int'>
```

In [57]:
```
my_t = 3,
print(type(my_t))
```

```
<class 'tuple'>
```

In [52]:
```
my_t = 3
print(type(my_t))
```

```
<class 'int'>
```

Tuples Methods

count() and index()

In [46]:
```
my_tuple = (1, 2, 3, 2)
print(my_tuple.count(2))
print(my_tuple.count(5))
print(my_tuple.index(3))
print(my_tuple.index(10)) # Output: Error
```

```
2
0
2
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[46], line 5
      3 print(my_tuple.count(5))
      4 print(my_tuple.index(3))
----> 5 print(my_tuple.index(10))

ValueError: tuple.index(x): x not in tuple
```

Writing over a Tuple

In [58]:
```python
dimensions = (200, 50)
print("Original dimensions:")
for dimension in dimensions:
    print(dimension)
```

```
Original dimensions:
200
50
```

In [59]:
```python
dimensions = (400, 100)
print("Modified dimensions:")
for dimension in dimensions:
    print(dimension)
```

```
Modified dimensions:
400
100
```

3.Dictionary

A dictionary in Python is a collection of key-value pairs. Each key is connected to a value, and you can use a key to access the value associated with that key. A key's value can be a number, a string, a list, or even another dictionary. In fact, you can use any object that you can create in Python as a value in a dictionary.

Dictionary is a collection which is **ordered** and **mutable (changeable)** and **No duplicate** members.

In [76]:
```python
my_dict = {"name": "John", "age": 30, "city": "New York"}
print(my_dict["name"])
emp_dict = {}
print(type(emp_dict))
emp_dict_2 = dict()
print(type(emp_dict_2))
```

```
John
<class 'dict'>
<class 'dict'>
```

Adding New Key-Value Pairs

In [60]:
```
line_0 = {'color': 'green', 'points': 5}
print(line_0)
line_0['x_position'] = 0
line_0['y_position'] = 25
print(line_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green', 'points': 5, 'x_position': 0, 'y_position': 25}
```

Modifying Values in a Dictionary

In [61]:
```
line_0 = {'color': 'green'}
print(f"The line is {line_0['color']}.")
line_0['color'] = 'yellow'
print(f"The line is now {line_0['color']}.")
```

```
The line is green.
The line is now yellow.
```

Removing Key-Value Pairs

In [63]:
```
line_0 = {'color': 'green', 'points': 5}
print(line_0)
del line_0['points']
print(line_0)
```

```
{'color': 'green', 'points': 5}
{'color': 'green'}
```

note:

Be aware that the deleted key-value pair is removed permanently

Using get() to Access Values

Using keys in square brackets to retrieve the value you’re interested in from a dictionary might cause one potential problem: if the key you ask for doesn’t exist, you’ll get an error.

In [64]:
```
line_0 = {'color': 'green', 'speed': 'slow'}
point_value = line_0.get('points', 'No point value assigned.')
print(point_value)
```

```
No point value assigned.
```

note:

If you leave out the second argument in the call to get() and the key doesn’t exist, Python will return the value None. The special value None means “no value exists.” This is not an error: it’s a special value meant to indicate the absence of a value.

Dictionary Methods

keys() , values() , items() , update() , pop() , popitem() , copy() , clear()

In [27]:
```
print(my_dict.keys())
```
```
dict_keys(['name', 'age', 'city', 'gender'])
```

In [28]:
```
print(my_dict.values())
```
```
dict_values(['John', 30, 'New York', 'Male'])
```

In [29]:
```
print(my_dict.items())
```
```
dict_items([('name', 'John'), ('age', 30), ('city', 'New York'), ('gender', 'Male')])
```

In [30]:
```
other_dict = {"gender" : "Male"}
my_dict.update(other_dict)
print(my_dict)
```
```
{'name': 'John', 'age': 30, 'city': 'New York', 'gender': 'Male'}
```

In [32]:
```
age = my_dict.pop("age")
print(age)
```
```
30
```

In [36]:
```
item = my_dict.popitem()
print(item)
```
```
('city', 'New York')
```

In [40]:
```
copied_dict=my_dict.copy()
copied_dict.update({"gender":"male"})
print(my_dict)
print(copied_dict)
```
```
{'name': 'John', 'age': 30, 'city': 'New York'}
{'name': 'John', 'age': 30, 'city': 'New York', 'gender': 'male'}
```

In [41]:
```
my_dict.clear()
```

In [42]:
```
print(my_dict)
print(copied_dict)
```
```
{}
{'name': 'John', 'age': 30, 'city': 'New York', 'gender': 'male'}
```

Nesting

A List in a Dictionary

In [91]:
```
favorite_languages = {
 'ahmed': ['python', 'ruby'],
 'mona': ['c'],
 'rana': ['ruby', 'go'],
 'gehad': ['python', 'haskell'],
 }
for name, languages in favorite_languages.items():
    print(f"\n{name.title()}'s favorite languages are:")
    for language in languages:
        print(f"\t{language.title()}")
```

```
Ahmed's favorite languages are:
        Python
        Ruby

Mona's favorite languages are:
        C

Rana's favorite languages are:
        Ruby
        Go

Gehad's favorite languages are:
        Python
        Haskell
```

A List of Dictionaries

In [95]:

```
line_0 = {'color': 'green', 'points': 5}
line_1 = {'color': 'yellow', 'points': 10}
line_2 = {'color': 'red', 'points': 15}
lines = [line_0, line_1, line_2]
for line in lines:
    print(line)
```

```
{'color': 'green', 'points': 5}
{'color': 'yellow', 'points': 10}
{'color': 'red', 'points': 15}
```

A Dictionary in a Dictionary

In [98]:

```
users = {
 'mosalah': {
 'first': 'mohamed',
 'last': 'salah',
 'location': 'liverpool',
 },
 'asadat': {
 'first': 'anwar',
 'last': 'sadat',
 'location': 'egypt',
 },
 }

for username, user_info in users.items():
    print(f"\nUsername: {username}")
    full_name = f"{user_info['first']} {user_info['last']}"
    location = user_info['location']
    print(f"\tFull name: {full_name.title()}")
    print(f"\tLocation: {location.title()}")
```

```
Username: mosalah
        Full name: Mohamed Salah
        Location: Liverpool

Username: asadat
        Full name: Anwar Sadat
        Location: Egypt
```

Looping Through a Dictionary

Looping Through All Key-Value Pairs

In [66]:
```
user_0 = {
 'username': 'mosalah',
 'first': 'mohamed',
 'last': 'salah',
 }
```

In [71]:
```
for key, value in user_0.items():
    print(f"\nKey: {key}")
    print(f"Value: {value}")
```

```
Key: username
Value: mosalah

Key: first
Value: mohamed

Key: last
Value: salah
```

Looping Through All the Keys in a Dictionary

In [73]:
```
favorite_languages = {
 'ahmed': 'python',
 'mona': 'c',
 'rana': 'ruby',
 'gehad': 'python',
 }

for name in favorite_languages.keys():
    print(name.title())
```

```
Ahmed
Mona
Rana
Gehad
```

In [79]:
```
friends = ['mona', 'gehad']
for name in favorite_languages.keys():
    print(f"Hi {name.title()}")

    if name in friends:
        language = favorite_languages[name].title()
        print(f"\t {name.title()}, I see you love {language}!")
```

```
Hi Ahmed
Hi Mona
	Mona, I see you love C!
Hi Rana
Hi Gehad
	Gehad, I see you love Python!
```

In [80]:
```
for name in sorted(favorite_languages.keys()):  #Looping Through a Dictionary’s Keys i
    print(f"{name.title()}, thank you for taking the poll.")
```

```
Ahmed, thank you for taking the poll.
Gehad, thank you for taking the poll.
Mona, thank you for taking the poll.
Rana, thank you for taking the poll.
```

Looping Through All Values in a Dictionary

In [83]:
```
favorite_languages = {
 'ahmed': 'python',
 'mona': 'c',
 'rana': 'ruby',
 'gehad': 'python',
 }

print("The following languages have been mentioned:")
for language in favorite_languages.values():
    print(language.title())
```

```
The following languages have been mentioned:
Python
C
Ruby
Python
```

In [85]:
```
print("The following languages are the unique mentions:")
for language in set(favorite_languages.values()):
    print(language.title())
```

```
The following languages are the unique mentions:
Python
Ruby
C
```

4.Sets

Set is a collection which is **unordered**, **unchangeable**, and **No duplicate** members.

In [101…
```
my_set = {5,4,3,2,0,1,0}
print(my_set)
empty_set = set()
print(empty_set)
print(type(empty_set))
empty_dict = {}
print(empty_dict)
print(type(empty_dict))
```

```
{0, 1, 2, 3, 4, 5}
set()
<class 'set'>
{}
<class 'dict'>
```

Sets Methods

add() , remove(), discard(), pop(), clear()

Mathematical set operations:

> union(), intersection(), difference(), symmetric difference()

In [61]:
```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)
```
```
{1, 2, 3, 4}
```

In [62]:
```
set1 = {1, 2, 3}
set2 = {4, 5, 6}
set1.update(set2)
print(set1)
```
```
{1, 2, 3, 4, 5, 6}
```

In [64]:
```
my_set = {1, 2, 3}
my_set.remove(2)
print(my_set)
```
```
{1, 3}
```

In [67]:
```
my_set = {1, 2, 3}
my_set.discard(1)
print(my_set)
```
```
{2, 3}
```

In [68]:
```
my_set = {1, 2, 3}
popped_element = my_set.pop()
print(my_set)
print(popped_element)
```
```
{2, 3}
1
```

In [70]:
```
my_set = {1, 2, 3}
my_set.clear()
print(my_set)
```
```
set()
```

In [71]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set)
```
```
{1, 2, 3, 4, 5}
```

In [72]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1.intersection(set2)
print(intersection_set)
```
```
{3}
```

In [73]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1.difference(set2)
print(difference_set)
```
```
{1, 2}
```

In [74]:
```
set1 = {1, 2, 3}
set2 = {3, 4, 5}
symmetric_difference_set = set1.symmetric_difference(set2)
print(symmetric_difference_set)
```
```
{1, 2, 4, 5}
```

Finally - Sources

https://www.w3schools.com/python/python_lists.asp

https://www.w3schools.com/Python/python_tuples.asp

https://www.w3schools.com/python/python_dictionaries.asp

https://w3schools.com/python/python_sets.asp

https://www.freecodecamp.org/news/list-comprehension-in-python/#:~:text=List%20comprehension%20is%20an%20easy,code%20enclosed%20in%20square%2

https://youtu.be/h3VCQjyaLws?si=5gujWHB63zTofaPI

In [ ]:
```
```