Set

A set is an unordered collection of unique elements.

1. Sets are mutable (can be modified), but they do not allow duplicate values.
2. Sets are defined using curly braces {} or the set() constructor.

In [1]:

```python
# Basic Set
fruits = {"apple", "banana", "cherry"}
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

```
{'apple', 'cherry', 'banana'}
```

In [ ]:

```python
# Set with Duplicates (Duplicates will be removed)
fruits = {"apple", "banana", "cherry", "apple", "banana"}
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

In [ ]:

```python
# Creating a Set from a List
numbers = set([1, 2, 3, 4, 5])
print(numbers)  # Output: {1, 2, 3, 4, 5}
```

1. add() Adds an element to the set.
2. clear() Removes all elements from the set.
3. copy() Returns a shallow copy of the set.
4. discard() Removes an element if it exists (does not raise an error).
5. remove() Removes an element (raises an error if element is not found).
6. union() Returns a new set with all elements from both sets.
7. intersection() Returns a new set with elements common to both sets.
8. difference() Returns a new set with elements in the first set but not in the second.
9. symmetric_difference() – Items in either set, but not both
10. issubset() Checks if the set is a subset of another set.
11. issuperset() Checks if the set is a superset of another set.

In [1]:

```python
# add() - Add an element
fruits = {"apple", "banana"}
fruits.add("cherry")
print(fruits)  # Output: {'apple', 'banana', 'cherry'}
```

```
{'apple', 'cherry', 'banana'}
```

In [3]:

```python
# clear() - Remove all elements
fruits = {"apple", "banana", "cherry"}
fruits.clear()
print(fruits)  # Output: set()
```

```
set()
```

In [4]:

```python
# copy() - Copy a set
fruits = {"apple", "banana", "cherry"}
```

```
fruits_copy = fruits.copy()
print(fruits_copy)  # Output: {'apple', 'banana', 'cherry'}
```

```
{'apple', 'cherry', 'banana'}
```

In [4]:

```
# discard() - Remove an element (if exists)
fruits = {"apple", "banana", "cherry"}
fruits.discard("banana")
print(fruits)  # Output: {'apple', 'cherry'}
```

```
{'apple', 'cherry'}
```

In [6]:

```
# remove() - Remove an element (raises error if element doesn't exist)
fruits = {"apple", "banana", "cherry"}
fruits.remove("banana")
print(fruits)  # Output: {'apple', 'cherry'}
# fruits.remove("orange")  # This will raise a KeyError
```

```
{'apple', 'cherry'}
```

In [5]:

```
# remove() - Remove an element (raises error if element doesn't exist)
fruits = {"apple", "banana", "cherry"}
fruits.remove("banana")
#print(fruits)  # Output: {'apple', 'cherry'}
#fruits.remove("orange")  # This will raise a KeyError
```

In [7]:

```
# union() - Combine two sets
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1.union(set2)
print(union_set)  # Output: {1, 2, 3, 4, 5}
```

```
{1, 2, 3, 4, 5}
```

In [8]:

```
# intersection() - Get common elements
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
intersection_set = set1.intersection(set2)
print(intersection_set)  # Output: {3, 4}
```

```
{3, 4}
```

In [9]:

```
# difference() - Get elements not in the second set
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
difference_set = set1.difference(set2)
print(difference_set)  # Output: {1, 2}
```

```
{1, 2}
```

In [19]:

```
# symmetric_difference() - Items in either set, but not both
set1 = {1, 2, 3, 4}
set2 = {3, 4, 5, 6}
difference_set = set1.symmetric_difference(set2)
print(difference_set)  # Output: {1, 2, 5, 6}
```

```
{1, 2, 5, 6}
```

In [10]:

```
# issubset() - Check if the set is a subset of another
set1 = {1, 2}
set2 = {1, 2, 3, 4}
print(set1.issubset(set2))  # Output: True
```

```
True
```

In [11]:

```
# issuperset() - Check if the set is a superset of another
set1 = {1, 2, 3, 4}
set2 = {1, 2}
print(set1.issuperset(set2))  # Output: True
```

```
True
```

Set Operations

In [12]:

```
# Set Union (|)
set1 = {1, 2, 3}
set2 = {3, 4, 5}
union_set = set1 | set2
print(union_set)  # Output: {1, 2, 3, 4, 5}
```

```
{1, 2, 3, 4, 5}
```

In [13]:

```
# Set Intersection (&)
set1 = {1, 2, 3}
set2 = {3, 4, 5}
intersection_set = set1 & set2
print(intersection_set)  # Output: {3}
```

```
{3}
```

In [14]:

```
# Set Difference (-)
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1 - set2
print(difference_set)  # Output: {1, 2}
```

```
{1, 2}
```

In [20]:

```
# Set Difference (-)
set1 = {1, 2, 3}
set2 = {3, 4, 5}
difference_set = set1 ^ set2
print(difference_set)  # Output: {1, 2}
```

```
{1, 2, 4, 5}
```

Touple

1. A tuple is a collection of ordered, immutable (unchangeable), and heterogeneous (different data types) elements.
2. Tuples are faster than lists because they are immutable.
3. Tuples are defined using parentheses ().

Tuples in Python are immutable, which means you can't change, add, or remove items once the tuple is created. Because of this, tuple methods are very limited compared to lists.

In [6]:

```
# Empty tuple
empty_tuple = ()

# Tuple with elements
numbers = (1, 2, 3, 4)

# Tuple with different data types
mixed_tuple = (1, "hello", 3.14, True)

# Tuple with one element (comma is required!)
single_element_tuple = (5,)
```

In [7]:

```
# count() - Count occurrences of a value
numbers = (1, 2, 3, 2, 2, 4)
print(numbers.count(2))  # Output: 3
```

```
3
```

In [8]:

```
# index() - Find the index of a value
fruits = ("apple", "banana", "cherry", "banana")
print(fruits.index("banana"))  # Output: 1 (first occurrence)
```

```
1
```

In [9]:

```
# Tuple Concatenation
tuple1 = (1, 2, 3)
tuple2 = (4, 5, 6)
result = tuple1 + tuple2
print(result)  # Output: (1, 2, 3, 4, 5, 6)
```

```
(1, 2, 3, 4, 5, 6)
```

In [11]:

```
# If you ever need to modify a tuple, you have to convert it to a list first:
t = (1, 2, 3)
temp = list(t)
temp.append(4)
t = tuple(temp)
print(t)  # Output: (1, 2, 3, 4)
```

```
(1, 2, 3, 4)
```

1. List: Use when the data will change (e.g., items in a cart).
2. Tuple: Use when data is fixed and should stay the same (e.g., coordinates, dates).

Tuple Unpacking (Very useful and common!)

In [12]:

```
t = (10, 20, 30)
a, b, c = t

print(a)  # 10
print(b)  # 20
print(c)  # 30
```

```
10
20
30
```

In [14]:

```python
# If you want to unpack a part of a tuple and collect the rest:
t = (1, 2, 3, 4, 5)

a, *b, c = t
print(a)  # 1
print(b)  # [2, 3, 4]
print(c)  # 5
```

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

Tuple Comparison

In [15]:

```python
# Tuples are compared element by element, from left to right.
t1 = (1, 2, 3)
t2 = (1, 2, 4)

print(t1 == t2)  # False (last elements are different)
print(t1 < t2)   # True (3 < 4)
```

```
False
True
```

In [16]:

```python
# Swapping Values with Tuple Unpacking
x, y = 10, 20
x, y = y, x
print(x, y)  # 20, 10
```

```
20 10
```

Class Assignment :01

Course Enrollment Analysis Using Sets Objective: Use set operations to find relationships between student groups. Problem: You are given two sets: course_A = {"Alice", "Bob", "Charlie", "David"} course_B = {"Charlie", "Eve", "David", "Frank"} Write a program that: 1. Finds students who are enrolled in both courses. 2. Finds students who are enrolled in only Course A. 3. Finds students who are enrolled in either Course A or Course B but not both. Expected Output: 1. Enrolled in both courses: {'Charlie', 'David'} 2. Only in Course A: {'Alice', 'Bob'} 3. In only one course: {'Alice', 'Bob', 'Eve', 'Frank'}

In [17]:

```python
# Given sets
course_A = {"Alice", "Bob", "Charlie", "David"}
course_B = {"Charlie", "Eve", "David", "Frank"}

# 1. Enrolled in both courses
both_courses = course_A & course_B
print("Enrolled in both courses:", both_courses)

# 2. Only in Course A
only_A = course_A - course_B
print("Only in Course A:", only_A)

# 3. In only one course
```

```
only_one = course_A ^ course_B
print("In only one course:", only_one)
```

```
Enrolled in both courses: {'Charlie', 'David'}
Only in Course A: {'Bob', 'Alice'}
In only one course: {'Bob', 'Eve', 'Alice', 'Frank'}
```

What is an f-string in Python? An f-string is a formatted string — introduced in Python 3.6 — that allows you to insert variables or expressions directly inside a string using {}.

In [23]:

```
name = "Alice"
age = 25

print(f"My name is {name} and I am {age} years old.")
```

```
My name is Alice and I am 25 years old.
```

In [24]:

```
# You Can Use Expressions Too!
a = 10
b = 5
print(f"The sum of {a} and {b} is {a + b}")
```

```
The sum of 10 and 5 is 15
```

In [25]:

```
# Format Numbers
price = 1234.5678
print(f"Price: ${price:.2f}")  # 2 decimal places
```

```
Price: $1234.57
```

In [26]:

```
# Adding Comma for Thousands Separator
large_number = 1234567890
print(f"Formatted with commas: {large_number:,}")
```

```
Formatted with commas: 1,234,567,890
```

In [27]:

```
# Percentage Formatting
percentage = 0.875
print(f"Formatted as percentage: {percentage:.2%}")
```

```
Formatted as percentage: 87.50%
```

In [28]:

```
# Exponential Notation
number = 1234567890
print(f"Exponential form: {number:.2e}")
```

```
Exponential form: 1.23e+09
```

In [30]:

```
num = 1234567890.98765
print(f"With commas: {num:,}")
print(f"With commas: {num:,.2f}")
print(f"Rounded to 2 decimals: {num:.2f}")
print(f"Percentage: {num:.2%}")
print(f"Scientific: {num:.2e}")
```

```
With commas: 1,234,567,890.98765
With commas: 1,234,567,890.99
Rounded to 2 decimals: 1234567890.99
Percentage: 123456789098.76%
Scientific: 1.23e+09
```

Class Assignment :02

Objective: Use tuple unpacking and comparison to manage and compare student data. Problem: You are given a list of students where each student is represented as a tuple:

In [32]:

```
students = [
    ("Alice", 3.75),
    ("Bob", 3.60),
    ("Charlie", 3.90),
]

# for i in students:
#     print(i)
```

Write a program that:

1. Unpacks and prints each student's name and CGPA.
2. Finds the student with the highest CGPA using tuple comparison.
3. Sorts the list based on CGPA in descending order.

In [21]:

```
students = [
    ("Alice", 3.75),
    ("Bob", 3.60),
    ("Charlie", 3.90),
]

# 1. Unpack and print
for name, cgpa in students:
    print(f"Name: {name}, CGPA: {cgpa}")

# 2. Find top student using max()
top_student = max(students, key=lambda x: x[1])
print(f"\nTop student: {top_student[0]} with CGPA {top_student[1]}")

# 3. Sort by CGPA descending
sorted_students = sorted(students, key=lambda x: x[1], reverse=True)
print("\nSorted List:")
print(sorted_students)
```

```
Name: Alice, CGPA: 3.75
Name: Bob, CGPA: 3.6
Name: Charlie, CGPA: 3.9

Top student: Charlie with CGPA 3.9

Sorted List:
[('Charlie', 3.9), ('Alice', 3.75), ('Bob', 3.6)]
```

In [ ]: