1. Decision Control Flow Statement in Python

A **decision control flow statement** in Python is used to execute a specific block of code based on conditions. It helps in making decisions within a program.

Key Points:

1. **if Statement** – Executes a block of code if a condition is true.
2. **if-else Statement** – Executes one block if the condition is true, otherwise another block.
3. **if-elif-else Statement** – Used for multiple conditions, executing different blocks based on which condition is true.
4. **Nested if Statement** – An `if` statement inside another `if` statement for complex decision-making.

Example:

```
python
CopyEdit
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is 5 or less")
```

String Slicing and Joining in Python

String Slicing

String slicing is a technique used to extract a portion of a string based on index positions. Since Python strings are **immutable** (cannot be changed after creation), slicing allows us to work with substrings without modifying the original string.

Types of String Slicing:

1. **Basic Slicing:** Extracts a specific part of the string.
2. **Omitting Start/End Index:** Defaults to beginning or end of the string.
3. **Negative Indexing:** Counts from the end of the string.
4. **Step Argument:** Skips characters at intervals.
5. **Reversing a String:** Uses `[::-1]` to reverse the order of characters.

Syntax:

```
string[start:end:step]
```

2. Traversing a List in Python

Definition

Traversing a list means accessing each element of the list one by one to perform operations such as reading, modifying, or processing data. It is an essential concept in Python as lists are one of the most commonly used data structures.

Methods of List Traversal

1. Using a `for` loop (Element-Based Traversal)

- The simplest way to traverse a list.
- Directly accesses each element without using an index.
- Preferred when no modifications to the list are required during traversal.

2. Using `for` loop with `range()` and Indexing

- Uses `range(len(list))` to access elements based on their index.
- Useful when both the index and the value are needed in operations.
- Allows modification of elements during traversal.

3. Using a `while` loop

- Uses a loop counter variable to track the index.
- The condition ensures traversal continues until the last index.
- Suitable when conditional traversal or modifications are required.

4. Using List Comprehension

- A concise, Pythonic way to traverse a list.
- Mostly used for transformations and filtering elements in a list.

3. Key Differences Between Single and Multiple Inheritance

Feature	Single Inheritance	Multiple Inheritance
Number of Parents	One parent class	Two or more parent classes
Simplicity	Simple and easy to understand	More complex due to multiple parent classes
Code Reusability	Reuses only one base class	Reuses multiple base classes
Risk of Conflicts	No conflicts, clear hierarchy	Possible method conflicts (MRO issues)
Example	`class Child(Parent):`	`class Child(Parent1, Parent2):`

4. Advantages of Using Web API

1. **Platform Independence** – Web APIs allow applications to communicate across different platforms (Windows, Linux, macOS, mobile, etc.) using standard protocols like HTTP.
2. **Reusability** – APIs enable code reuse, reducing development effort and allowing multiple applications to use the same functionality.
3. **Scalability** – Web APIs support high scalability, allowing applications to handle increased user traffic and requests efficiently.
4. **Security** – APIs use authentication methods like OAuth, API keys, and token-based security, ensuring controlled access to data.
5. **Faster Development** – APIs provide ready-made functionalities, reducing the time required to develop and integrate new features into applications.

5. Steps to Plot a Simple Line Graph in Python

1. **Import the required library**
   - `matplotlib.pyplot` is used for plotting graphs.

```
python
CopyEdit
import matplotlib.pyplot as plt
```

2. **Prepare the data**
   - Create two lists or arrays representing the `x` and `y` coordinates.

```
python
CopyEdit
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 40]
```

3. **Plot the graph**
   - Use `plt.plot(x, y)` to create a line graph.

```
python
CopyEdit
plt.plot(x, y)
```

4. **Add labels and title**
   - Use `plt.xlabel()`, `plt.ylabel()`, and `plt.title()` for better readability.

```
python
CopyEdit
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Line Graph")
```

5. **Display the graph**
   - Use `plt.show()` to display the graph.

```
python
```

```
CopyEdit
plt.show()
```

6. Python Program to Demonstrate Basic Data Types

Python has several built-in data types, including integers, floats, strings, lists, tuples, sets, and dictionaries. The following program demonstrates these basic data types.

```
python
CopyEdit
# Integer Data Type
num = 10
print("Integer:", num, "| Type:", type(num))

# Float Data Type
decimal = 10.5
print("Float:", decimal, "| Type:", type(decimal))

# String Data Type
text = "Hello, Python!"
print("String:", text, "| Type:", type(text))

# List Data Type (Ordered, Mutable Collection)
my_list = [1, 2, 3, 4, 5]
print("List:", my_list, "| Type:", type(my_list))

# Tuple Data Type (Ordered, Immutable Collection)
my_tuple = (10, 20, 30)
print("Tuple:", my_tuple, "| Type:", type(my_tuple))

# Set Data Type (Unordered, Unique Elements)
my_set = {1, 2, 3, 4, 5}
print("Set:", my_set, "| Type:", type(my_set))

# Dictionary Data Type (Key-Value Pairs)
my_dict = {"name": "Alice", "age": 25}
print("Dictionary:", my_dict, "| Type:", type(my_dict))
```

7. Function Definition and Function Call in Python

1. Function Definition

In Python, a **function** is defined using the `def` keyword followed by the function name, parentheses (which may include parameters), and a colon. The code block inside the function is indented.

Syntax:

```
def function_name(parameters):
    # Code block
    return result  # Optional, if the function returns a value
```

- `function_name` is the name of the function.
- `parameters` are optional values that can be passed to the function.
- The function can return a value using the `return` statement.

2. Function Call

To use or **call** a function, you simply use its name followed by parentheses. If the function requires arguments, they are passed inside the parentheses.

Syntax:

```
python
CopyEdit
function_name(arguments)
```

8. Theory: Purpose of the `return` Statement in Python

The `return` statement in Python serves several important purposes:

1. **Returning a Result**
   The `return` statement is used to send a result back from a function to the caller. It allows a function to provide output that can be used later in the program.
2. **Ending Function Execution**
   When a `return` statement is executed, the function immediately stops its execution, and no further code within the function is executed after the `return`.
3. **Returning Multiple Values**
   Python allows a function to return multiple values at once. These values are packed into a tuple, which can be unpacked and used by the caller.
4. **Returning `None`**
   If a function does not have a `return` statement, Python automatically returns `None` to indicate that no value was explicitly returned from the function.

9.Python Program to Compute the Area of a Circle Using User Input

```
python
CopyEdit
import math

# Accept radius input from the user
radius = float(input("Enter the radius of the circle: "))

# Compute the area of the circle
area = math.pi * radius ** 2

# Display the result
print(f"The area of the circle with radius {radius} is: {area}")
```

Explanation:

- The program uses `input()` to get the radius from the user and converts it into a `float` for precision.
- The area of the circle is calculated using the formula Area=π×r2\text{Area} = \pi \times r^2Area=π×r2, where `math.pi` gives the value of π\piπ.
- Finally, the result is printed using formatted strings.

10. Membership Operators (`in` and `not in`) with a Dictionary in Python

- The **`in`** operator checks if a specified **key** exists in a dictionary. It returns `True` if the key is present and `False` if the key is absent.
- The **`not in`** operator checks if a specified **key** does **not** exist in a dictionary. It returns `True` if the key is absent and `False` if the key is present.

These operators are used to check for the presence or absence of keys, not values, in a dictionary.

```
my_dict = {"name": "Alice", "age": 25, "city": "New York"}


# Checking if the key 'name' exists

if "name" in my_dict:

    print("The key 'name' is present in the dictionary.")


# Checking if the key 'address' exists

if "address" not in my_dict:

    print("The key 'address' is not present in the dictionary.")
```

11.Difference Between List and Tuple in Python (Table)

Feature	List	Tuple
Mutability	Mutable (can be modified)	Immutable (cannot be modified)
Syntax	Defined using square brackets `[]`	Defined using parentheses `()`
Performance	Slower due to mutability	Faster due to immutability
Usage	Used when data needs to be modified	Used for constant, fixed data
Methods	More methods like `.append()`, `.remove()`, etc.	Fewer methods (e.g., `.count()`, `.index()`)
Memory Consumption	Consumes more memory due to dynamic size	Consumes less memory due to fixed size

12. Access Modifiers in Python

1. **Public Members**
   - Public members are accessible from any part of the program, both inside and outside the class. They do not have any leading underscores.
2. **Protected Members**
   - Protected members are intended to be accessed within the class and by its subclasses. They are not meant to be accessed directly outside the class. These members have a single leading underscore `_`.
3. **Private Members**
   - Private members are intended to be accessed only within the class itself. They cannot be accessed from outside the class or its subclasses. These members have a double leading underscore `__`.
4. **Name Mangling**
   - Private members undergo *name mangling* in Python, where their names are internally changed to include the class name as a prefix. This prevents accidental access but does not strictly enforce privacy.
5. **Special Case: `__all__`**
   - The `__all__` list defines what is exposed when a module is imported using `from module import *`. Only the members listed in `__all__` are imported, restricting access to others.

13. Steps to Create and Read Text Files in Python

1. Creating a Text File

To create a text file, you can use the `open()` function with the mode `'w'` (write) or `'x'` (exclusive creation).

- **Step 1**: Use the `open()` function with the appropriate mode.
- **Step 2**: Write content to the file using the `write()` method.
- **Step 3**: Close the file using the `close()` method to ensure data is saved and resources are released.

2. Reading a Text File

To read from a text file, use the `open()` function with the mode `'r'` (read).

- **Step 1**: Use the `open()` function with mode `'r'`.
- **Step 2**: Read the content using methods like `read()`, `readline()`, or `readlines()`.
- **Step 3**: Close the file using the `close()` method after reading.

File Modes:

- `'w'`: Opens a file for writing. Creates the file if it doesn't exist, or truncates it if it exists.
- `'r'`: Opens a file for reading. The file must exist.

- `'x'`: Creates a new file and opens it for writing. If the file already exists, it raises an error.

14. Method Overriding in Python

Method overriding occurs when a subclass (child class) defines a method that already exists in its superclass (parent class). The method in the subclass has the same name, signature, and parameters as the one in the superclass, but it provides a different implementation.

Key Points:

1. **Purpose**: Method overriding is used to modify or extend the behavior of an inherited method in a subclass.
2. **Signature**: The method in the subclass should have the same name and parameters as the method in the parent class.
3. **Call**: The method in the subclass will override the one in the parent class, and the subclass version will be executed when the method is called on an instance of the subclass.

15. CSV Files in Python - Simple Explanation

CSV (Comma Separated Values) files store data in a tabular format, where each row represents a record and columns are separated by commas.

- **Reading a CSV file**: Use `csv.reader()` to read each row as a list.
- **Writing to a CSV file**: Use `csv.writer()` to write rows to a CSV file.
- **Header**: The first row often contains column names.
- **Modes**: `'r'` for reading, `'w'` for writing, and `'a'` for appending.

Key Functions:

- **`csv.reader()`**: Reads CSV files.
- **`csv.writer()`**: Writes to CSV files.
- **`csv.DictReader()`**: Reads CSV files as dictionaries.
- **`csv.DictWriter()`**: Writes dictionaries to CSV files.

The `csv` module simplifies working with CSV files by handling common formatting issues like quotes around fields.

16. Features of JSON (JavaScript Object Notation)

1. **Lightweight Data Format**
   JSON is a lightweight, text-based format that is easy to read and write for humans and easy to parse and generate for machines.
2. **Human-Readable**
   JSON files are formatted in a way that is easily readable by humans, making it a good choice for data interchange.

3. **Data Representation**
   JSON represents data in key-value pairs, similar to a dictionary in Python, making it suitable for representing structured data.
4. **Language-Independent**
   Although JSON is derived from JavaScript, it is language-independent. It can be used with almost any modern programming language, including Python, Java, and JavaScript.
5. **Supports Nested Data Structures**
   JSON allows nesting of objects and arrays, enabling the representation of complex data structures like lists of dictionaries, dictionaries of lists, and so on.
6. **Simple Syntax**
   JSON uses a simple syntax:
   - Data is represented as key-value pairs.
   - Keys must be strings enclosed in double quotes.
   - Values can be strings, numbers, arrays, objects, or boolean values.
7. **Efficient Data Transmission**
   JSON is commonly used for transmitting data between a server and web application, especially in web APIs. It is more compact and efficient than XML.
8. **Supports Arrays**
   JSON can include arrays (ordered lists), which are represented with square brackets `[]`.

17. Plot Parameters for Customizing Line Graph in Python

When creating line graphs using the `matplotlib` library in Python, several plot parameters allow for customization of the appearance of the graph. These parameters can modify aspects like line style, color, markers, labels, and more.

Here are some common plot parameters used to customize line graphs:

1. **Line Style (`linestyle`)**
   - Defines the style of the line. Common options include:
     - `'-'`: Solid line (default)
     - `'--'`: Dashed line
     - `':'`: Dotted line
     - `'-.'`: Dash-dot line
2. **Line Color (`color`)**
   - Defines the color of the line. You can use color names like `'red'`, `'blue'`, or hexadecimal values like `'#FF5733'`. You can also use RGB tuples like `(0.1, 0.2, 0.5)`.
3. **Line Width (`linewidth` or `lw`)**
   - Specifies the thickness of the line. Higher values result in thicker lines.
4. **Markers (`marker`)**
   - Specifies the shape and style of markers that appear at each data point on the line.
     - `'.'`: Point marker
     - `'o'`: Circle marker
     - `'^'`: Triangle marker
     - `'s'`: Square marker

        - `'D'`: Diamond marker
5. **Marker Size (`markersize` or `ms`)**
    - Defines the size of the markers. Larger values make the markers bigger.
6. **Marker Color (`markerfacecolor` or `mfc`)**
    - Specifies the color of the marker itself.
7. **Marker Edge Color (`markeredgecolor` or `mec`)**
    - Specifies the color of the marker's edge.
8. **Grid (`grid`)**
    - Controls the visibility of the grid. It can be turned on with `plt.grid(True)` and off with `plt.grid(False)`.
9. **Axis Labels (`xlabel`, `ylabel`)**
    - Defines the labels for the x-axis and y-axis.
10. **Title (`title`)**

- Adds a title to the plot.

11. **Legend (`legend`)**

- Adds a legend to the plot, which is helpful when plotting multiple lines. You can specify labels for each line and use `plt.legend()` to display the legend.

12. **Alpha (`alpha`)**

- Controls the transparency of the line. Values range from `0` (completely transparent) to `1` (completely opaque).

2023

Python Q&A 5&8 marks

Key Features of Python

1. **Easy to Learn and Read**
   Python's syntax is clean and easy to understand, making it an ideal choice for beginners. The code is highly readable, with an emphasis on simplicity and clarity.
2. **Interpreted Language**
   Python is an interpreted language, meaning the code is executed line by line by the Python interpreter. This allows for quick debugging and testing of code.
3. **Dynamically Typed**
   Python doesn't require explicit declaration of variable types. The interpreter dynamically assigns the type during runtime, which increases flexibility and reduces boilerplate code.
4. **Object-Oriented**
   Python supports object-oriented programming (OOP) principles like classes, inheritance, polymorphism, and encapsulation, enabling more organized and reusable code.
5. **Extensive Standard Library**
   Python comes with a comprehensive standard library that provides modules and functions for tasks like file I/O, regular expressions, networking, and more. This reduces the need to write code from scratch.
6. **Cross-Platform Compatibility**
   Python is platform-independent, meaning Python code can run on various operating systems such as Windows, Linux, and macOS without modification.
7. **Large Community Support**
   Python has a large and active community of developers. This ensures a wealth of tutorials, documentation, and third-party libraries are available for solving problems.
8. **Versatile and Multi-Paradigm**
   Python supports multiple programming paradigms, including procedural, functional, and object-oriented programming, making it versatile and adaptable to various needs.
9. **Extensibility**
   Python allows for the integration of code written in other languages, like C or C++, for performance-critical sections. It also supports the use of external libraries to extend its functionality.
10. **Garbage Collection**
    Python automatically handles memory management through garbage collection, which frees up memory by removing unused objects from memory.

Indentation in Python

Indentation refers to the practice of using spaces or tabs to define the structure of code blocks in Python. Unlike many programming languages that use braces `{}` or other delimiters to define code blocks, Python uses indentation to indicate the grouping of statements. Proper indentation is crucial for Python code to work correctly.

Key Points:

1. **Code Blocks**: In Python, blocks of code (like loops, conditionals, functions, classes) are defined by consistent indentation.
2. **No Braces**: Instead of using curly braces `{}`, Python relies on the level of indentation to determine where a block of code starts and ends.
3. **Consistent Indentation**: It is important to use consistent indentation (either spaces or tabs, but not both) throughout your code. The standard practice is to use **4 spaces per indentation level**.

Indexing and Slicing in Tuples

Indexing in Tuples

Indexing is used to access individual elements of a tuple. Python tuples are **ordered** collections, and indexing starts at **0**. You can also use **negative indexing**, where `-1` refers to the last element, `-2` refers to the second-last element, and so on.

- **Positive Indexing**: Access elements from the beginning of the tuple.
- **Negative Indexing**: Access elements from the end of the tuple.

Slicing in Tuples

Slicing allows you to access a subrange or subset of elements in a tuple. It is done by specifying a **start** index, an **end** index, and an optional **step**.

- **Start Index**: The position where the slice starts (inclusive).
- **End Index**: The position where the slice ends (exclusive).
- **Step**: The interval between elements you want to include in the slice (optional).

Example of Indexing and Slicing in Tuples

```
python
CopyEdit
# Example Tuple
my_tuple = (10, 20, 30, 40, 50)

# Indexing examples
print(my_tuple[0])  # Output: 10 (first element)
print(my_tuple[2])  # Output: 30 (third element)
print(my_tuple[-1]) # Output: 50 (last element)
print(my_tuple[-3]) # Output: 30 (third-last element)
```

Class and Object in Python

Class

A **class** is a blueprint or template for creating objects. It defines the structure (attributes) and behavior (methods) that objects of the class will have. A class does not store data directly; rather, it defines how objects will be constructed and what properties and methods they will contain.

Key features of a class:

1. **Attributes**: These are variables that are associated with a class or an object. They define the state or characteristics of the class.
2. **Methods**: Functions defined inside a class that represent the behavior of the class. These are used to manipulate or access the attributes of the class.
3. **Constructor**: The `__init__` method is a special method that is called when a new object of the class is created. It is used to initialize the attributes of the object.

Object

An **object** is an instance of a class. When a class is defined, no memory is allocated for its data. Memory is allocated when an object of the class is instantiated. Each object has its own set of attributes and can call the methods defined in the class.

Key features of an object:

1. **Instance**: Each object is an instance of the class and has its own copy of the attributes.
2. **Accessing Attributes and Methods**: Objects can access and modify their own attributes and call methods defined in the class.

File Reading and Writing in Python

Python provides built-in functions and methods for reading from and writing to files. The `open()` function is used to open a file, and the file is accessed through file objects. These files can be manipulated by using various methods.

Opening a File

To perform any operation on a file, you first need to open the file using the `open()` function. The `open()` function takes two main arguments:

1. **File name**: The name of the file you want to open (including path if necessary).
2. **Mode**: The mode in which to open the file:
    - `'r'`: Read mode (default). Opens the file for reading.
    - `'w'`: Write mode. Opens the file for writing (creates a new file if it doesn't exist).
    - `'a'`: Append mode. Opens the file for appending (creates a new file if it doesn't exist).
    - `'rb'`, `'wb'`: Read or write in binary mode.

- `'r+'`: Opens the file for both reading and writing.

After opening the file, Python provides several methods to read or write data to the file.

File Reading

There are different ways to read from a file:

1. **`read()`**: Reads the entire content of the file as a string.
2. **`readline()`**: Reads one line at a time.
3. **`readlines()`**: Reads all the lines in the file and returns a list where each item is a line in the file.

File Writing

You can write to a file using the `write()` or `writelines()` methods.

1. **`write()`**: Writes a string to the file.
2. **`writelines()`**: Writes a list of strings to the file

Built-in Functions in Python

1. **Mathematical Functions**:
   - `abs(x)`: Returns absolute value of `x`.
   - `round(x, n)`: Rounds `x` to `n` decimal places.
   - `min(iterable)`: Returns smallest item in an iterable.
   - `max(iterable)`: Returns largest item in an iterable.
   - `sum(iterable)`: Returns sum of all elements in iterable.
2. **Type Conversion Functions**:
   - `int(x)`: Converts `x` to an integer.
   - `float(x)`: Converts `x` to a floating-point number.
   - `str(x)`: Converts `x` to a string.
   - `list(iterable)`: Converts iterable to a list.
   - `tuple(iterable)`: Converts iterable to a tuple.
3. **String Functions**:
   - `len(s)`: Returns length of string `s`.
   - `str.upper()`: Converts string to uppercase.
   - `str.lower()`: Converts string to lowercase.
   - `str.strip()`: Removes leading/trailing whitespace from string.
   - `str.replace(old, new)`: Replaces `old` with `new` in string.
4. **Collection-related Functions**:
   - `len(iterable)`: Returns length of iterable.
   - `sorted(iterable)`: Returns sorted list of items.
   - `reversed(iterable)`: Returns reversed iterator.
   - `enumerate(iterable)`: Returns index-value pairs of iterable.
5. **Input/Output Functions**:
   - `input(prompt)`: Reads user input as a string.

Program to Print Positive and Negative Numbers

```
python
CopyEdit
# Function to separate positive and negative numbers
def separate_numbers(numbers):
    positive_numbers = []
    negative_numbers = []

    # Loop through the list of numbers
    for num in numbers:
        if num >= 0:
            positive_numbers.append(num)  # Append positive numbers
        else:
            negative_numbers.append(num)  # Append negative numbers

    # Print the positive and negative numbers
    print("Positive numbers:", positive_numbers)
    print("Negative numbers:", negative_numbers)

# Example list of numbers
numbers = [10, -5, 3, -2, 0, 7, -8]

# Call the function with the list of numbers
separate_numbers(numbers)
```

Concepts of Strings in Python

1. String Creation

- A string is a sequence of characters enclosed in quotes.
- You can use:
    - Single quotes: `'Hello'`
    - Double quotes: `"World"`
    - Triple quotes for multi-line strings:

      ```
      python
      CopyEdit
      '''This is a
      multi-line string'''
      ```

2. String Slicing

- String slicing allows you to extract a portion of a string using indices.
- Syntax: `string[start:end:step]`
    - **Start**: Where the slice begins (inclusive).
    - **End**: Where the slice ends (exclusive).
    - **Step**: Optional; it controls the spacing between indices.

3. String Comparison

- Strings can be compared using the following operators:
    - `==`: Checks if two strings are equal.
    - `!=`: Checks if two strings are not equal.

- `<`, `>`, `<=`, `>=`: Compares strings lexicographically (alphabetically).

4. Finding Substring

- You can check if a substring exists within a string using:
  - **`in`**: Returns `True` or `False`.
  - **`find()`**: Returns the index of the first occurrence or `-1` if not found.
  - **`index()`**: Similar to `find()`, but raises an error if the substring is not found.

Dictionary in Python

A **dictionary** in Python is a collection of key-value pairs. Each key is unique, and it maps to a specific value. Dictionaries are mutable, meaning their contents can be changed after creation. They are defined using curly braces `{}` with key-value pairs separated by colons `:`.

Syntax:

```
python
CopyEdit
dictionary = {key1: value1, key2: value2, key3: value3}
```

Using `get()` in Dictionaries

- The `get()` method is used to retrieve the value associated with a specific key.
- If the key does not exist, it returns `None` by default (or you can specify a default value).

Inheritance in Python

Inheritance is a mechanism in object-oriented programming (OOP) that allows one class (called the **child class** or **subclass**) to inherit the properties and behaviors (methods) of another class (called the **parent class** or **superclass**). This allows for code reusability and logical hierarchy between classes.

Types of Inheritance in Python

- **Single Inheritance**: A child class inherits from one parent class.
- **Multiple Inheritance**: A child class inherits from multiple parent classes.
- **Multilevel Inheritance**: A chain of inheritance, where a class inherits from a parent, which inherits from another parent.
- **Hierarchical Inheritance**: Multiple child classes inherit from a single parent class.
- **Hybrid Inheritance**: Combination of two or more types of inheritance.

Set Data Type in Python

A **set** is a built-in data type in Python that represents an unordered collection of unique elements. Sets are similar to lists and tuples, but they do not allow duplicate values, and they are unordered (meaning the elements have no specific order). Sets are also mutable, meaning you can add or remove elements.

Key Properties of Sets:

- **Create**: Use `{}` or `set()` to create a set.
- **Add**: Use `add()` to add an element.
- **Remove**: Use `remove()` or `discard()` to remove elements.
- **Union**: Combine two sets with `union()` or `|`.
- **Intersection**: Find common elements with `intersection()` or `&`.
- **Difference**: Find elements in one set but not the other with `difference()` or `-`.
- **Symmetric Difference**: Find elements that are in either set but not both with `symmetric_difference()` or `^`.

CSV in Python

A **CSV (Comma Separated Values)** file is a simple text file where data is separated by commas, often used to store tabular data like spreadsheets or databases. Python provides several ways to work with CSV files, primarily using the built-in `csv` module and the **pandas** library.

```
import pandas as pd


# Reading a CSV file into a DataFrame

df = pd.read_csv('example.csv')

print("Data from CSV file:")

print(df)


# Writing a DataFrame to a CSV file

df.to_csv('output.csv', index=False)  # index=False to avoid writing row numbers


# Modifying Data and Writing to CSV

df['Country'] = ['USA', 'USA', 'USA']  # Adding a new column

df.to_csv('modified_output.csv', index=False)
```