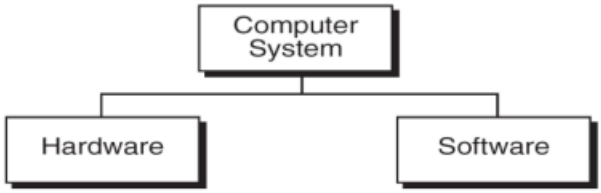
(Chapter- 01) Computer Systems

A computer system consists of hardware and software.



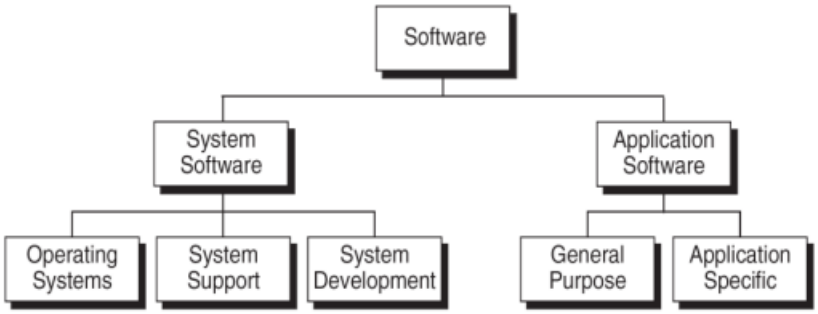
1. Hardware: Hardware refers to the physical components of a computer or any other electronic device that you can touch and see.

Examples:

- **Input Devices:** Keyboard, mouse, touchscreen, microphone (used to give commands to the computer).
- **Output Devices:** Monitor, printer, speakers (used to display or output the results of the computer's processing).
- **Processing Unit:** Central Processing Unit (CPU), often called the "brain" of the computer, processes data and instructions.
- **Storage Devices:** Hard drive, SSD (used to store data and programs permanently).
- **Memory:** RAM (Random Access Memory), temporarily stores data and instructions that the CPU needs while working.

2. Software: Software is a collection of instructions or programs that tell the hardware what to do. It is the non-physical part of a computer.

Or, It is consist of computer programs that instruct the execution of a computers



- **Types of Software:**
  - **System Software:** The most important software that manages hardware and software resources, examples include Windows, macOS, and Linux.
  - **Application Software:** Programs that help users do specific tasks like word processing (Microsoft Word), browsing the internet (Google Chrome), or playing games.

- **Examples of Software:**
    - **Mobile Applications:** WhatsApp, Instagram, Google Maps
    - **Web Applications:** Facebook (www.facebook.com), Amazon (www.amazon.com), YouTube (www.youtube.com)
    - **Gaming Software:** Free fire, GTA, Call of Duty
    - **Operating Systems:** Windows, macOS, Linux, Android, iOS

(Chapter- 02) Introduction of Programing

Computer Programs: A computer program is a set of instructions that execute the computers.

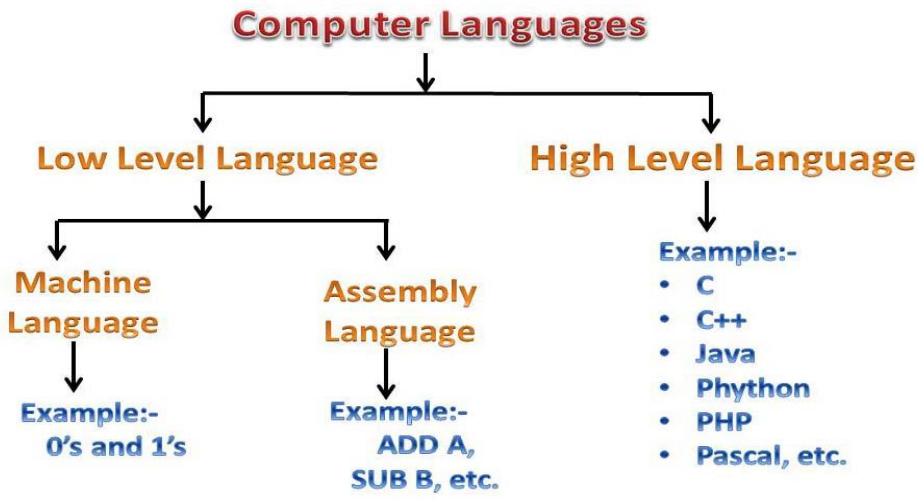
1. **Why is Programming Important?**

   Programming allows us to create software, apps, websites, and games. It's how we make computers do things, from simple tasks like adding numbers to complex ones like controlling a robot.

2. **How Does Programming Work?**
   1. **Writing Code:** Programmers write instructions, called code, in a programming language.
   2. **Executing Code:** The computer follows these instructions step by step to complete the task.
   3. **Debugging:** If something goes wrong, programmers find and fix the errors in the code, a process called debugging.

Computer Languages:

To communicate with a computer, users need a language that the computer can understand. Different languages have been developed to perform various tasks on a computer. These languages are mainly divided into two categories based on how they are interpreted.



Low-Level Languages

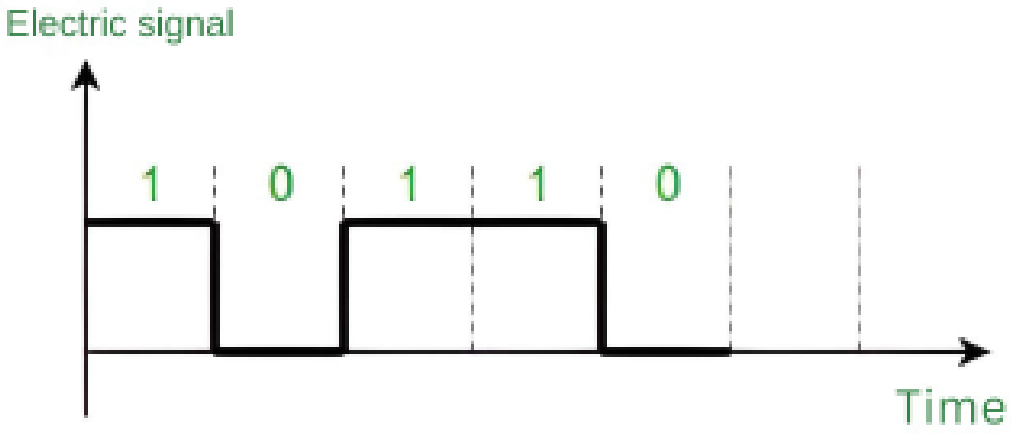
- Low-level languages are closer to the machine's hardware and offer less abstraction. They provide more control over hardware but are harder to read and write.

High-Level Languages

- High-level computer languages are designed to be easy to understand, using English-like words and symbols.
- Each instruction written in a high-level language is need to convert into machine language, which the computer can understand and execute.

Machine Language: Machine language is the simplest and most basic computer language. Computers can only understand this language, which uses 0s and 1s (binary code).

The number "0" means **no electric signal**, and "1" means there is **an electric signal**. Computers recognize these signals and understand machine language.



Computer Programming Language:

- A specific type of computer language designed to create software by providing instructions that a computer can execute.
- These languages are used to write algorithms and logic to perform specific tasks.

Side-by-side comparison between Machine Language and Computer Programing language

Aspect	Machine Language	Computer Programming Language
Definition	The lowest-level programming language in binary code	A higher-level language used to write software programs
Human Readability	Not human-readable (binary code)	Human-readable (using words and symbols)
Execution	Executed directly by the computer's hardware	Needs to be translated (compiled/interpreted) into machine language
Speed	Fastest execution, no translation needed	Slightly slower due to the translation process
Example	10110000 01100001	printf("Hello, World!"); in c- language
Portability	Not portable, specific to hardware/CPU	Portable, can run on different hardware with minimal changes

Compiler / Interpreter:

Both compilers and interpreters are tools used to convert programs written in high-level languages into machine code that a computer can understand and execute.

Aspect	Compiler	Interpreter
How It Works	Translates the entire program into machine code at once.	Translates and runs the program line by line.
Execution Time	Takes time to translate before running, but the program runs faster once translated.	Runs the program as it translates, which can be slower.
Error Handling	Shows all errors after translating the whole program.	Shows errors one at a time, as it runs through the program.
Example Languages	C, C++	Python, JavaScript
Use Case	Good for creating software that needs to run quickly.	Good for testing and debugging code easily.
Development Process	You need to recompile every time you make changes.	You can test parts of the code without recompiling.

File Extension

A file extension is a suffix at the end of a file name that indicates the type of files and folders.

It's usually is in small letters or numbers after a dot (.)

Examples:

- .txt – Text File (plain text)
- .docx – Microsoft Word Document
- .jpg or .png – Image Files
- .mp3 – Audio File
- .mp4 – Video File
- .pdf – Portable Document Format
- .html – HyperText Markup Language (web page)
- **.py – for Python Programing language**
- **.exe – for Software in windows**

(Chapter- 03) Introduction to Python

1. Python

Python is a **high-level**, **interpreted** programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, it emphasizes code readability using significant indentation. Python supports **multiple programming paradigms**, including procedural, object-oriented, and functional programming.

Key features of Python:

- **Easy Syntax:** Python's syntax is designed to be easy to read and write, making it accessible for beginners.
- **Interpreted Language:** Python code is executed line by line by the Python interpreter, without the need for compilation.
- **Extensive Libraries**: Python has a vast standard library and supports many third-party libraries for various fields, including data science, web development, automation, machine learning, and more.
- **Multi-Paradigm programming language:** Procedural, Object-Oriented, Functional and etc

2. Why we Learn Python Programing Language?

1. **Easy to Learn and Use**
2. **Versatility, Flexibility and Wide Range of Applications**
    - Web Development,
    - Data Science and Machine Learning,
    - Automation,
    - Game Development,
    - IoT and Robotics
3. **Huge Community Support & Open Source**
4. **High Demand in the Job Market (**Go to the Google Trends and compare different language for 5 or 10 years**)**
5. **Extensive Libraries and Frameworks**
6. **Strong Presence in AI, Machine Learning, and Data Science**

3. Installing Python and setting up the environment.

- **Download and Install Python:** https://www.python.org/downloads/
- **To check if Python was installed correctly, Type at CMD (Command Prompt) " python --version "**
- **Install any Python IDEs (Integrated Development Environment): IDLE, PyCharm, VSCode**

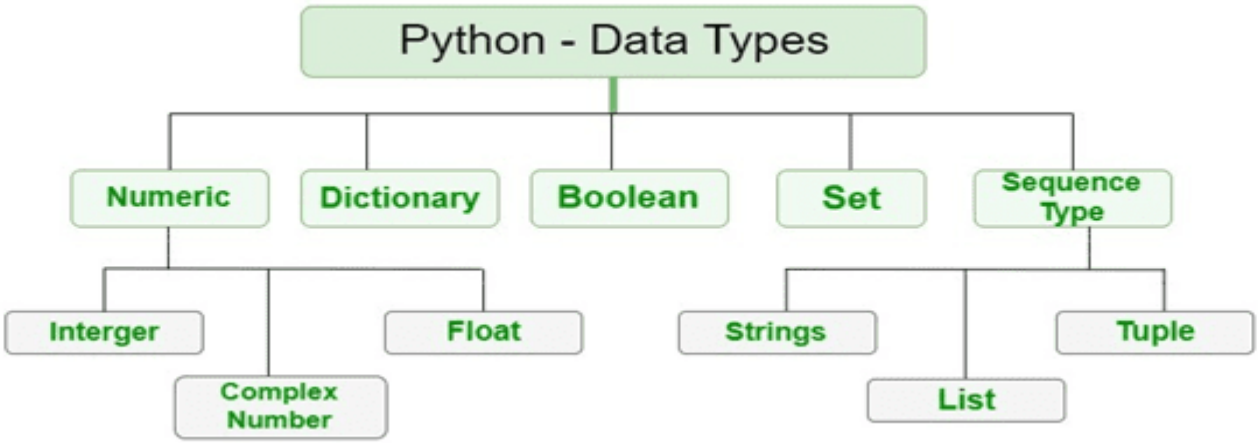
(Chapter- 04) Python Basics

Variable :- A variable is a name given to a memory location in Programs that holds a value. Variables allow you to store and manipulate data in a program.

```
age = 15
temperature = 36.5
grade = 'A'
```

What is Data types?

- Data types define the type of data that a variable can hold.
- They help the compiler to understand, how much memory to allocate for the variable and how to interpret the data.



Basic Data Types:

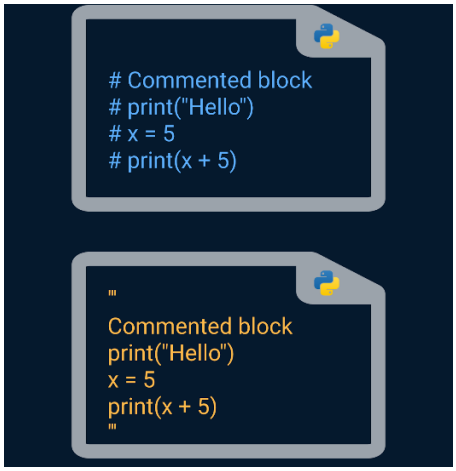
Name	Type	Description	Example
Integers	int	Whole numbers	3, 300, 200
Floating point	float	Numbers with a decimal point	2.3, 4.6, 100.0
Strings	str	Ordered sequence of characters	"hello", 'Sammy', "2000", "LL"
Lists	list	Ordered sequence of object	[10, "hello", 200.3]
Dictionaries	dict	Unordered Key: Value pairs	{"mykey": "value", "name" "Frankie"}
Tuples	tup	Ordered immutable sequence of objects	(10, "hello", 200.3)
Sets	set	Unordered collection of unique objects	{"a", "b"}

Comments:

In Python, a comment is a line of text that is ignored by the interpreter and is used to explain the code, make it more readable, or leave notes for yourself or other developers. Comments do not affect the execution of the program.

Single-line comment

''' Multi-line comment '''



Input and output functions:

- **Input Function: input():** It is used to take input from the user.

```
# Taking input from the user
name = input("Enter your name: ")
age = input("Enter your age: ")

print("Hello, " + name + "! You are " + age + " years old.")
```

- **Print Function: print() :** It is use to display outputs data to the console.

```
# Printing multiple values
print("Hello", "World!")  # Output: Hello World!

# Custom separator and ending
print("Hello", "World", sep=", ", end="!\n")  # Output: Hello, World!
```

Operators: It is special symbols that perform operations on variables and values.

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators
4. Assignment Operators
5. Bitwise Operators
6. Membership Operators
7. Identity Operators

1. Arithmetic Operators: These operators are used to perform basic mathematical operations.

Operator	Description	Example
+	Addition	`5 + 3` results in `8`
-	Subtraction	`5 - 3` results in `2`
*	Multiplication	`5 * 3` results in `15`
/	Division	`5 / 2` results in `2.5`
//	Floor Division	`5 // 2` results in `2`
%	Modulus (Remainder)	`5 % 2` results in `1`
**	Exponentiation	`5 ** 2` results in `25`

2. Comparison Operators: These operators compare two values and return a Boolean result (True or False).

Operator	Description	Example
==	Equal to	5 == 3 results in False
!=	Not equal to	5 != 3 results in True
>	Greater than	5 > 3 results in True
<	Less than	5 < 3 results in False
>=	Greater than or equal to	5 >= 5 results in True
<=	Less than or equal to	5 <= 5 results in True

3. Logical Operators : These operators are used to combine conditional statements.

Operator	Description	Example
and	Returns True if both statements are true	True and False results in False
or	Returns True if at least one statement is true	True or False results in True
not	Returns True if the statement is false	not True results in False

4. Assignment Operators: These operators are used to assign values to variables.

Operator	Description	Example
=	Assigns value	x = 5
+=	Adds and assigns	x += 2 (equivalent to x = x + 2)
-=	Subtracts and assigns	x -= 2 (equivalent to x = x - 2)
***=**	Multiplies and assigns	x *= 2 (equivalent to x = x * 2)
/=	Divides and assigns	x /= 2 (equivalent to x = x / 2)
//=	Floor divides and assigns	x //= 2 (equivalent to x = x // 2)
%=	Modulus and assigns	x %= 2 (equivalent to x = x % 2)
****=**	Exponentiates and assigns	x **= 2 (equivalent to x = x ** 2)

5. Bitwise Operators: These operators are used to perform operations on binary numbers.

Operator	Description	Example
&	Bitwise AND	5 & 3 results in 1
`	`	Bitwise OR
^	Bitwise XOR	5 ^ 3 results in 6
~	Bitwise NOT	~5 results in -6
<<	Left Shift	5 << 1 results in 10
>>	Right Shift	5 >> 1 results in 2

6. Membership Operators: These operators test for membership in a sequence (like lists, strings, or tuples).

Operator	Description	Example
in	Returns True if the value is found in the sequence	3 in [1, 2, 3] results in True
not in	Returns True if the value is not found in the sequence	4 not in [1, 2, 3] results in True

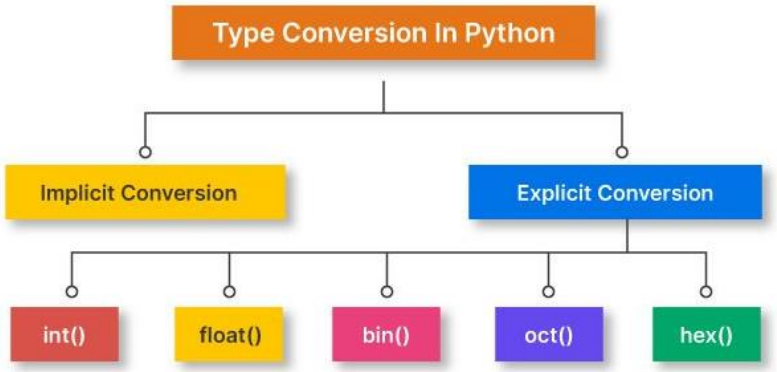
7. Identity Operators: These operators check if two variables point to the same object in memory.

Operator	Description	Example
is	Returns True if both variables point to the same object	x is y
is not	Returns True if both variables do not point to the same object	x is not y

Type conversion or Type Casting :

The conversion of one data type into the other data type is known as type casting in python or type conversion in python.

Python supports a wide variety of functions or methods like: int(), float(), str(), ord(), hex(), oct(), tuple(), set(), list(), dict(), etc. for the type casting in python.



Explicit typecasting:

The conversion of one data type into another data type, done via developer or programmer's intervention or manually as per the requirement, is known as explicit type conversion.

It can be achieved with the help of Python's built-in type conversion functions such as int(), float(), hex(), oct(), str(), etc .

Example of explicit typecasting:

```
string = "15"
string_number = int(string)          #throws an error if the string is not a valid
sum= 7 + string_number
print("The Sum of both the numbers is: ", sum)
```

Output:

The Sum of both the numbers is 22

Implicit type casting: According to the level, one data type is converted into other by the Python interpreter itself (automatically). This is called, implicit typecasting in python.

Python converts a smaller data type to a higher data type to prevent data loss.

Example of implicit type casting:

```
# Python automatically converts
# a to int
a = 7
print(type(a))

# Python automatically converts b to float
b = 3.0
print(type(b))

# Python automatically converts c to float as it is a float addition
c = a + b
print(c)
print(type(c))
```

Output:

```
<class 'int'>
<class 'float'>
10.0
<class 'float'>
```

(Chapter- 05) Control Structures

If-else statements

It is used for conditional execution of code. It allows to execute certain blocks of code based on whether a condition evaluates to True or False. This is essential for making decisions in your programs.

```
age = 18

if age >= 18:
    print("You are eligible to vote.")
else:
    print("You are not eligible to vote.")
```

Nested If-Else:

A **nested if-else** is an if-else statement placed inside another if or else, allowing multiple levels of condition checking.

```
id = "hemantraj"
pas = "hemant@123"

if id == input("Enter the user Id: "):
    if pas == input("Enter the password: "):
        print("Hurray! You have successfully logged in.")
    else:
        print("Oops! You entered the wrong password.")
else:
    print("Wrong user id: ")
```

Loops:

It allows a block of code to be executed repeatedly based on a condition. Loops are essential in automating repetitive tasks in programming.

Two primary types of loops in Python are **for loops** and **while loops.**

1. For Loop

It is used to iterate over a sequence (such as a list, tuple, dictionary, set, or string) or any other iterable object. It executes a block of code for every item in the sequence.

```python
numbers = [1, 2, 3, 4, 5]
for num in numbers:
    print(num)
```

Output:

```
1
2
3
4
5
```

Range Function:

The **range()** function is often used in for loops to generate a sequence of numbers.

```python
for i in range(start, stop, step):
    # code block to execute
```

start: (optional) Starting value (default is 0).

stop: The value at which the loop stops (exclusive).

step: (optional) The increment between numbers (default is 1).

Nested For Loops:

It is a loop inside another loop, where the inner loop runs fully for each iteration of the outer loop and use for more complex iterations.

```python
for i in range(3):  # Outer loop
    for j in range(3):  # Inner loop
        print("*", end=" ")
    print()  # Moves to the next line after inner loop finishes
```

Loop Control Statements:

Break: Exits the loop when a certain condition is met.	`for i in range(10):`
	`  if i == 5:`
	`  break`
	`print(i)`
	Output: `0 1 2 3 4`

Continue: Skips the current iteration and moves to the next.	`for i in range(5):`
	`  if i == 3:`
	`  continue`
	`print(i)`
	Output: `0 1 2 4`

Else with For Loop: The else block is executed when the loop finishes normally (i.e., without encountering a break statement).

```
for i in range(3):
    print(i)
else:
    print("Loop finished")
```

Output:
0
1
2
Loop finished

2. While Loop

A while loop continues to execute a block of code as long as a given condition is True.

Syntax: `while condition:`

condition: The loop will keep running as long as this condition evaluates to True.

```
x = 1
while x <= 5:
    print(x)
    x += 1
```

Output:
1
2
3
4
5

Infinite While Loop:

If the condition in a while loop never becomes False, the loop will continue indefinitely, causing an infinite loop.

```
while True:
    print("This is an infinite loop")
```

Differences between For and While Loops

For Loop	While Loop
Used for iterating over a sequence or range.	Used when the number of iterations is unknown, and the loop depends on a condition.
Stops automatically when the sequence ends.	Stops when the condition becomes False.
Ideal for iterating through a known number of elements.	Ideal for conditions where you don't know how many iterations are required.

(Chapter- 06) Data Structures in Python

Data structures are essential for organizing and storing data efficiently, enabling quick access and modifications. Python provides several built-in data structures, which can be classified into:

1. List 2. Tuple 3. Set 4. Dictionary

Python Lists: A list is a collection of ordered, mutable elements in Python. Lists can hold any data type (integers, floats, strings, objects, etc.)

```python
my_list = [1, 'apple', 3.14, [1, 2, 3]]
```

List Index: Each item/element in a list has its own unique index. This index can be used to access any particular item from the list. The first item has index [0], second item has index [1], third item has index [2] and so on.

```
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
#          [0]     [1]     [2]      [3]      [4]
```

- **Positive Indexing:** Start from 0 to positive whole number.

```
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
#          [0]     [1]     [2]      [3]      [4]
print(colors[2])
print(colors[4])
print(colors[0])
```

Output:

```
Blue
Green
Red
```

- **Negative Indexing:** Similar to positive indexing, negative indexing is also used to access items, but from the end of the list. The last item has index [-1], second last item has index [-2], third last item has index [-3] and so on.

```
colors = ["Red", "Green", "Blue", "Yellow", "Green"]
#          [-5]    [-4]    [-3]     [-2]      [-1]
print(colors[-1])
print(colors[-3])
print(colors[-5])
```

Output:

```
Green
Blue
Red
```

Slicing and Indexing: Slicing allows access to sub lists using the format **list[start: stop: step]**.

```python
lst = [1, 2, 3, 4, 5, 6]
print(lst[1:5])  # Output: [2, 3, 4, 5]
print(lst[::-1])  # Reverse the list: [6, 5, 4, 3, 2, 1]
```

List Operations and Methods:

- **Adding elements:**
  - **append():** Adds a single element.
  - **extend():** Adds multiple elements from another list or iterable.
  - **insert():** Inserts an element at a specific index

```python
lst = [1, 2, 3]
lst.append(4)  # [1, 2, 3, 4]
lst.extend([5, 6])  # [1, 2, 3, 4, 5, 6]
lst.insert(1, 'a')  # [1, 'a', 2, 3, 4, 5, 6]
```

- **Removing elements:**
  - **remove():** Removes the first occurrence of the element.
  - **pop():** Removes and returns the element at the given index.
  - **clear():** Removes all elements from the list.

```python
lst.remove(3)  # Removes the first '3' in the list.
lst.pop(0)  # Removes the first element and returns it.
lst.clear()  # Empties the list.
```

Introduction to Tuples

A **tuple** is a collection data type in Python, similar to a list. However, unlike lists, tuples are **immutable**, meaning once a tuple is created, its elements cannot be changed, modified, or updated.

Tuples are often used when you need to group multiple related values together and want to ensure that the values remain constant throughout the program.

Characteristics of Tuples:

- **Ordered**: The elements in a tuple maintain their order.
- **Immutable**: Once defined, elements in a tuple cannot be changed.
- **Allows Duplicates**: Tuples can have duplicate elements.
- **Heterogeneous**: A tuple can contain different data types (int, float, string, etc.).

1. Creating Tuples

Tuples are defined by placing a comma-separated sequence of values within parentheses ().

```
my_tuple = (1, 2, 3, 'a', 'b', 'c')
```

Note: To create a tuple with only one element, a comma is required after the element; otherwise, it will not be recognized as a tuple.

2. Accessing Tuple Elements

You can access tuple elements using **indexing**, similar to lists. The index starts at 0 for the first element, 1 for the second, and so on.

Example:

```
my_tuple = ('apple', 'banana', 'cherry')
print(my_tuple[1])  # Output: banana
```

We can perform Slicing and indexing in Tuple also.

3. Tuple Methods

Tuples support only two built-in methods: count() and index().

- **count()**: Returns the number of times a specified value appears in the tuple.

```
my_tuple = (1, 2, 2, 3, 4)
print(my_tuple.count(2))  # Output: 2
```

- **index()**: Returns the index of the first occurrence of the specified value.

```
my_tuple = (1, 2, 3, 4)
print(my_tuple.index(3))  # Output: 2
```

4. Tuple Operations

Just like lists, tuples support a variety of operations, including concatenation, repetition, and membership testing.

- **Concatenation**: Use + to combine two tuples.

```
t1 = (1, 2)
t2 = (3, 4)
print(t1 + t2)  # Output: (1, 2, 3, 4)
```

- **Repetition**: Use * to repeat the elements in a tuple.

```
t = (1, 2)
print(t * 3)  # Output: (1, 2, 1, 2, 1, 2)
```

- **Membership Testing**: Use in to check if an element exists in a tuple.

```
my_tuple = (1, 2, 3, 4)
print(3 in my_tuple)  # Output: True
```

5. Tuples vs Lists: Key Differences

Feature	Tuple	List
Mutability	Immutable (Cannot change)	Mutable (Can change)
Syntax	Uses parentheses ()	Uses square brackets []
Methods	Limited (Only count() and index())	Many (append, remove, etc.)
Use Case	Fixed collections	Dynamic collections
Performance	Faster due to immutability	Slower due to mutability

Introduction to Sets:

A **set** is a built-in data structure in Python that represents a collection of unique elements. Sets are unordered, meaning that the items do not have a defined order, and they cannot contain duplicate elements.

They are particularly useful for performing mathematical set operations like union, intersection, and difference.

Characteristics of Sets:

- **Unordered**: The elements do not maintain any specific order.
- **Mutable**: Sets can be modified after creation (you can add or remove elements).
- **No Duplicates**: Sets automatically remove duplicate values.
- **Heterogeneous**: A set can contain elements of different data types (integers, strings, etc.).

---

1. Creating Sets

Sets can be created using curly braces {} or the set() function.

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

Using the set() Function:

```
my_set = set([1, 2, 3, 4])  # Creating a set from a list
```

2. Accessing Elements in a Set

Sets are unordered collections, so **you cannot access elements by index**. However, you can iterate through the elements in a set.

```
my_set = {1, 2, 3}
for item in my_set:
    print(item)
```

3. Adding and Removing Elements

You can add or remove elements from a set using the following methods:

- **Adding Elements**:
    - add(): Adds a single element to a set.

```
my_set = {1, 2, 3}
my_set.add(4)
print(my_set)  # Output: {1, 2, 3, 4}
```

- **Removing Elements**:
    - remove(): Removes a specified element from a set. Raises a KeyError if the element is not found.

```
my_set.remove(2)
print(my_set)  # Output: {1, 3, 4}
```

    - discard(): Removes a specified element but does not raise an error if the element is not found.

```
my_set.discard(5)  # No error even if 5 is not in the set
```

- **Clearing a Set**:
    - clear(): Removes all elements from the set.

```
my_set.clear()
print(my_set)  # Output: set()
```

4. Set Operations

Sets support several mathematical operations:

- **Union**: Combines two sets to form a new set with all unique elements.
    - Using **|** operator:

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

    - Using the **union()** method:

```
union_set = set1.union(set2)
```

- **Intersection**: Creates a new set with elements that are common to both sets.
    - Using **&** operator:

```
intersection_set = set1 & set2
print(intersection_set)  # Output: {3}
```

    - Using the **intersection()** method:

```
intersection_set = set1.intersection(set2)
```

- **Difference**: Creates a new set with elements in the first set but not in the second.
    - Using - operator: ans = {1, 2, 3} - {3, 4, 5} /// "ans" will be {1, 2}
    - Using the **difference()** method: **set1.difference(set2)**

- **Symmetric Difference**: Creates a new set with elements in either set but not in both.
    - Using **^** operator:
    - Using the symmetric_difference() method:

---

5. Set Methods

Sets come with a variety of built-in methods for manipulation and analysis:

- **copy()**: Returns a shallow copy of the set.
- **pop()**: Removes and returns an arbitrary element from the set. Raises a Key Error if the set is empty.
- **update()**: Adds multiple elements from another set or iterable.
- **issubset()**: Checks if one set is a subset of another.
- **issuperset()**: Checks if one set is a superset of another.
- **isdisjoint()**: Checks if two sets have no elements in common.

---

6. Set Comprehension

Similar to list comprehensions. **L = set(i for in range(10))**

7. Advantages of Sets

- **Uniqueness**: Automatically removes duplicates, ensuring all elements are unique.
- **Efficiency**: Sets are optimized for membership testing, making operations like checking if an item exists much faster than lists.
- **Convenience**: Useful for mathematical operations such as unions, intersections, and differences.

---

8. When to Use Sets vs Lists

- Use a **set** when you need to maintain a collection of unique items or perform mathematical set operations.
- Use a **list** when you need to maintain the order of items, allow duplicates, or when you need to frequently change the collection.

---

9. Sets vs Other Data Structures

Feature	Set	List	Tuple
Order	Unordered	Ordered	Ordered
Mutability	Mutable	Mutable	Immutable
Duplicates	No	Yes	Yes
Syntax	Curly braces {}	Square brackets []	Parentheses ()
Performance	Fast membership testing	Slower membership testing	Faster than lists