Chapter 1: Introduction to Python

1.1 What is Python?

Python is a general-purpose, dynamic, high-level and interpreted programming language.

- It is designed to be simple and easy to learn, making it an ideal choice for beginners.
- One of the key strengths of Python is its versatility.
- Python supports the object-oriented programming approach, allowing developers to create applications with organized and reusable code.

1.2 Features of Python

Readability:

Python's syntax is designed to be clear and readable, making it easy for both begineers and

- **Simplicity:**

Python emphasizes simplicity and avoids complex syntax, making it easier to learn and use compared to other programming languages.

- **Dynamic Typing:**

Python is dynamically typed, meaning you don't need to explicitly declare variable types.

- **Large Standard Library:**

Python provides a vast standard library with ready-to-use modules and functions for various tasks, saving developers time and function efforts in implementing common functiondlities.

- **Object-Oriented Programming (OOP):**

Python supports the object-oriented programming paradigm, allowing for the creation and manipulation of objects, classess and inheritance.

- **Cross-Platform Compatibility:**

Python is available on multiple platforms, including Windows, macOS, and Linux, making it

Extensive Third-Party Libraries :-

Python has a vast ecosystem of third-party libraries and frameworks that expand its capabilities in different domains, such as web development, data analysis and machine learning.

Interpreted Nature:-

Python is an interpreted language, meaning it does not require compilation. This results in a faster development cycle as codecan be executed directly without the need for a seprate compilation step.

Integration Capabilities:-

Python can easily integrate with other languages like C, C++ and Java, allowing developers to leverage existing codebases and libraries.

1.3 Applications of Python

Python is widely used in various domains and offers numerous applications due to its flexibility and ease of use. Here are some key areas where Python finds applications :

- **Web Development :**

Python is extensively used in web development frameworks such as Django and Flask. These frameworks provide efficient tools and libraries to build dynamic websites and web applications.

- **Data Analysis and Visualization :**

Python's rich ecosystem of libraries, including NumPy, Pandas, and Matplotlib, make it a popular choice for data analysis and visualization. It enables professionals to process, manipulate and visualize data effectively.

- **Machine Learning and Artificial Intelligence :**

Python has become the go-to language for machine learning and AI projects. Libraries like Tensorflow, keras and scikit-learn provide powerful tools for implementing complex algorithms and training models.

- **Automation and Scripting:**

Python's easy-to-read syntax and rapid development cycle make it an ideal choice for automation and scripting tasks. It is commonly used

1.4 Python Installation

To download and install Python, follow these steps:

For Windows:

- Visit the official Python website at www.python.org/downloads/
- Download the Python installer that matches your system requirements.
- On the Python Releases for Window page, select the link for the latest Python 3.x.x release.
- Scroll down and choose either the "Windows x86-64 executable installer" for 64-bit or the "Windows x86 executable installer" for 32 bit.
- Run the dowloaded installer and follow the instructions to install Python on your Windows system.

For Linux (specifically Ubuntu):

- Open the Ubunty software Center folder on your Linux system.
- From the All software drop-down list box, select

- Locate the entry for Python 3.x.x and double-click on it.
- Click on the install button initiate the installation process.
- Once the installation is complete, close the Ubuntu Software Centre Folder.

1.5 First Python Program

Writing your first Python program is an exciting step toward learning the language. Here's a simple example to get you started:

```
# Printing Hello World Using Python
print ("Hello World")
```

Let's break down the code:

- The print() function is used to display the specified message or value on the console.
- In this case, we pass the string "Hello World" as an argument to the print() function.

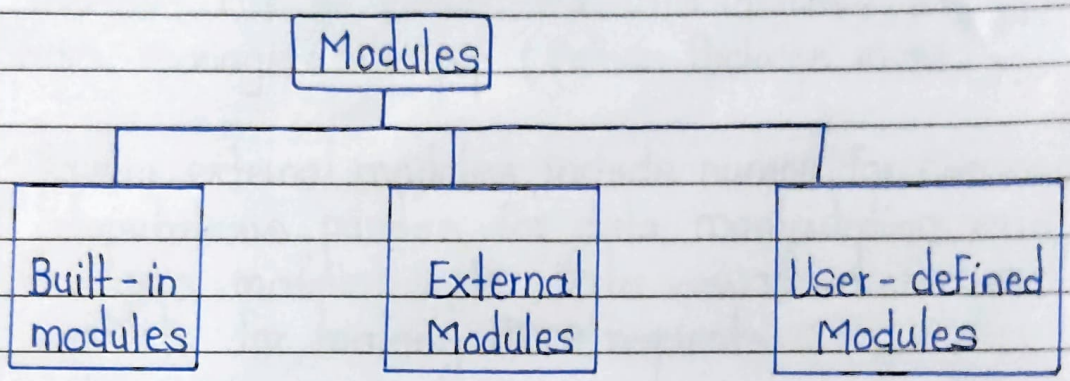
- The string is enclosed in double quotes.
- The # symbol indicates a comment in Python. Comments are ignored by the interpreter and are used to provide explainations or notes to the code.

Chapter 2: Modules Comment & Pip

2.1 Modules in Python:

- Modules provide a way to organize your code logically. Instead of having all your code in a single file, you can split it into multiple modules based on their purpose.
- For example, you might have one module for handling input/output operations, another for mathematical calculations, and another for data manipulation.
- When you want to use the functionality from a module, you can import it into your current program or another module.
- This allows you to access and use the functions, classes and variables defined within that module. By importing a module, you can avoid writing the same code repeatedly and instead reuse the code defined in the module.

2.2 Three Main Types of Modules



Built-in modules:

- These are modules that come pre-installed with python. They are part of the standard library and provide a wide range of functionalities.

- Examples include modules like math for mathematical operations, random for generating random numbers, DateTime for working with dates and times, and os for interacting with the operating system

- Built-in modules are readily available for use without the need for additional installations.

External Modules:

- These are modules that are created by third-party developers and are not part of the standard library.

- They extend Python's capabilities by providing additional functionalities for specific purpose. External modules can be downloaded and installed using packages managers like pip (Python Package Index)

- Popular external modules include numpy for numerical computations, pandas for data manipulation and analysis, matplotlib for data visualization, and requests for making HTTP requests.

User-Defined Modules:

- These are modules created by the Python programmers themselves. They allow users to organize their code into seprate files and reuse functionality across multiple programs.

- User-defined modules can contain functions, classes variables, and other code that can be imported and used in other Python scripts or modules.

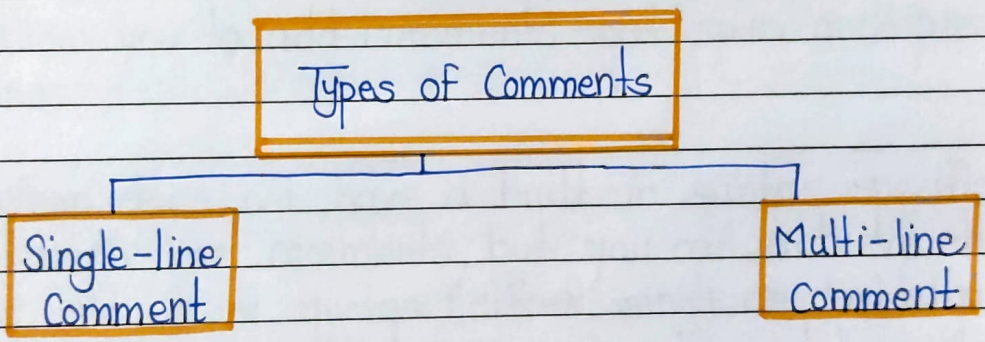
2.3 Comments in Python

Comments in Python are used to provide explanatory notes within the code that are not executed or interpreted by the computer.

They are helpful for improving code readability and for leaving reminders or explanations for other developers who might work with the code in the future.

In Python, Comments are denoted by the hash symbol (#) followed by the comment text.

It's important to note that comments are meant for human readers and are not executed by the Python interpreter. Therefore they have no impact on the program's functionality or performance.



1. Single line Comments

Single-line Comments are used to add explanatory notes or comments on a single line of code.

They start with a hash symbol (#) and continue untill the end of the line.

Anything written after the hash symbol is considered a comment and is ignored by the Python interpreter.

Here's an example:

```
# This is a single-line comment
x=5  # Assigning a value to the variable x
```

2. Multi-line Comments

Multi-line comments, also known as block comments, allow you to add comments that span multiple lines.

Python does not have a built-in syntax specifically for multi-line comments, but you can achieve this by using triple quotes (either single or double quotes) to create string that is a not assigned to any variable. Since it is that not used elsewhere in the code, it acts as a comment.

Here's an example:

```
""" 
This is a multi-line comment.
It can span across multiple lines.
"""
```

2.3 What is a pip?

- In Simple terms, pip is a package for Python. It Stands for "Pip Installs Pakages" or "Pip Installs Python".

- When working with Python, you may need to use external libraries or modules that provide additional functionalities beyond what the standard library afters. These libraries are often developed by the Python community and are available for anyone to use.

- Pip makes it easy to install, manage, and unistall these external libraries. It helps you find and download the libraries you need from the Python Package Index (pypi) which is a repository of Python packages maintained by the community.

- With pip, you can install a package by running a simple command in your terminal or command prompt.

Chapter 3 : Variables, Data Types & Keywords

3.1 Variables in Python

In Python, variables are used to store values that can be used later in a program. You can think of variables as containers that hold data. What makes Python unique is that you don't need to explicitly declare the type of a variable. You simply assign a value to a variable using the "=" operator.

- For example, you can create a variable called "name" and assign it a value like this:

```
name = "Yadnyesh"
```

Here, "name" is the variable name, and "Yadnyesh" is the value assigned to it. Python will automatically determine the type of the variable

based on the value assigned to it. In this case, the type of the variable "name" is a string.

Variables in Python can hold different types of data such as numbers, strings, lists, or even more complex objects. You can change the value of a variable at any time by assigning a new value to it. For instance:

```
age = 25
age = 26    # updating the value of 'age' variable
```

Python also allows you to perform operations on variables. for example, you can add, subtract, multiply, or divide variables containing numbers. You can even combine variables containing numbers of different types using operators? For instance

```
x = 5
y = 3
z = x + y  # The value of 'z' will be 8

greeting = "Hello"
name = "John"
message = greeting + " " + name  # The value of "message" will be "Hello John"
```

Variables provide a way to store and manipulate data in Python, making it edsier to work with information throughout your program. By giving meaningful names to variables, you can make your code more readable and understanble.

3.1.1 Identifier in Python

In Python, an identifier is a name used to identif a variable, function, class, module or any other user-defined object. An identifier can be made up of letters (both uppercase and lowercase), digits, and underscores (_). However, it must start with a letter or and underscores.

Here are some important rules to keep in mind when working with identifiers in Python.

- **Valid characters :** An identifiers can contain letters (a-z, A-Z), digits (0-9), and underscores (_). It cannot contain spaces or special characters like @, #, or $.

- **Case Sensitivity :** Python is case-sensitive, meaning uppersase and lowercase letters are considered different. So, "myvar" and "myVar" are treated as two different identifiers.

- **Reserved words :** Python has reserved words, also known as keywords that have predefined meanings

in the language. These words cannot be used as ide
fiers. Examples of reserved words include "if", "while
and "def".

• **Length:** Identifiers can be of any length. However,
is recommendended to use meaningful and descript
names that are non-excessively long.

• **Readability:** It is good practice to choose desc
ptive names for identifiers that convey their purp
or meaning. This helps make the code more redd
le and understandable.

Here are some examples of valid identifiers in Pytho

- my_variable
- count
- total_sum
- PI
- Myclass

And here are some examples of Invalid identifiers:

- 123abc (starts with a digit)
- my-variable (contains a bypen)
- if (a reserved word)
- my var (contains a space)

3.2 Data Types in Python :

Data types in Python refer to the different kinds of values that can be assigned to variables. They determine the nature of the data and the operations that can be performed on them. Python provides several built-in data types, including,

1. Numeric :

Python supports different numerical data types, including integers (whole numbers), floating-point numbers (decimal numbers), and complex numbers(numbers with real and imaginary parts).

a. integers (int) : Integers represent whole numbers without any fractional part. For example, age = 25.

b. Floating-Point Numbers (float) : Floating-point numbers represent numbers with decimal points or fractions. For example, pi = 3.14.

c. Complex Numbers (complex): Complex numbers have a real and imaginary part. They are denoted by a combination of a real and imaginary number, suffixes with j or J. for example $z = 2+3j$.

2. Dictionary :

Dictionaries are key-value pairs enclosed in

curly braces. Each values is associated with a unique key, allowing for efficient lookup and retrieval. For example, person = {'name': 'John', 'age': 25, 'city': 'New York'}.

3. Boolean:

Boolean (bool): Booleans represent thuth values, either True or False. They are used for logical operations and conditions. For example, is_valid = True

4. Set:

Sets (set): Sets are enclosed unordered collections o unique elements enclosed in curly braces. They are useful for mathematical operations such as union, intersection and difference. For example, fruits = {'apple', 'banana', 'orange'}.

5. Sequence Type:

Sequences represent a collection of elements and include data types like strings, lists and tuples. Strings are used to store textual data, while lists and tuples are used to stored ordered collections of items.

- Strings (str): Strings are sequences of

enclosed within single or double quotes. For examp
name = "John".

• Lists(list): Lists are ordered sequences of elements enclosed in square brackets. Each elem can be of any data type. For example, number = [,3,4]

• Tuple(tuple): Tuples are similar to lists bu are immutable, meaning their elements cannot be changed once defined. They are enclosed in paran ses. For example. coordinates = (3,4)

3.3 keywords in Python

• keywords in Python are special words that have spe fic meanings and purposes within the Python lang ge. They are reserved and cannot be used as var able names or identifiers.

• keywords play a crucial role in defining the struc and behaviour of python programs.

• keywords are like building blocks that allow us t create conditional statements, loops, functions, c handle errors, and perform other important opera

• They help in controlling the flow of the program

For example, the if keyword is used to check condition and perform specific actions based on those conditions. The for and while keywords are used to create loops that repeat a block of code multiple times.

The def keyword is used to define functions, which are reusable blocks of code that perform specific tasks

List of all keyword in Python:

False	await	else	import	pass
None	break	except	in	raise
True	class	Finally	is	return
and	continue	For	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yeild

Chapter :4 Operators in Python

4.1 Operators in Python

Operators in Python are symbols or special characters that are used to perform specific operations on variables and values.

Python provides various types of operators to manipulate and work with different data types. Here are some important categories of operators in Python:

- Arithmetic operators
- Comparison operators
- Assignment operators
- Logical operators
- Bitwise operators
- Membership operators
- Identity operators
- Arithmetic operators.

The basic arithmetic operators include:

- **Addition (+):**
  Adds two operands together. For example, if we have a = 10 and b = 10, then a+b equals 20.

- **Subtraction (-):**
  Substracts the second operand from the first operand. If the first operand is smaller than the second operand, the result will be negative. For example, if we have a = 20 and b = 5, then a-b equals 15.

- **Division (/):**
  Divides the first operand by the second operand and returns the quotient. For example, if we have a = 20 and b = 10, then a/b equals 2.0.

- **(Multiplication) *:**
  Multiplies one operand by the other. for example, if we have a = 20 and b = 4 then a*b = 80.

- **Modulus (%):**
  Returns the remainder after dividing the first operand by the second operand. For example, if we have a = 20 and b = 10, then a%b equals 0.

- **Exponentiation (**) or Power:**
  Raises the first operand to the power of the second operand. for example, if we have a = 2 and

- Floor Division (//):

Provides the floor value of the quotient obtained by dividing the two operands. It returns the largest integer that is less than or equal to the result. For example, if we have a = 20 and b = 3, then a // b equals 6.

Comparison Operators:

Comparison operators in Python are used to compare two values and return a Boolean value (True or False) based on the comparison. Common comparison operators include:

- Equal to (==): checks if two operands are ~~not~~ equal.
- Not equal to (!=): checks if two operands are not equal.
- Greater than (>): checks if the left operand is greater than the right operand.
- Less than (<): checks if the left operand is less than the right operand.
- Greater than or equal to (>=): checks if the left operand is greater than or equal to the right operand.

- Less than or equal to (<=): checks if the left operand is less than or equal to the right operand.

Assignment Operators:

Assignment operators are used to assign values to variables. They include:

- Equal to (=): Assigns the value on the right to the variable on the left.
- Compound assignment operators (+=, -=, *=, /=): Perform the specified arithmetic operation and assign the result to the variable.

Logical Operators:

Logical operators in Python are used to perform logical operations on Boolean values. The main logical operators are:

- Logical AND (and): Returns True if both operands are True, otherwise False.
- Logical OR (or): Returns True if at least one of the operands is True, otherwise False.
- Logical NOT (not): Returns the opposite Boolean value of the operand.

Bitwise Operators:

Bitwise operators perform operations on individual bits of binary numbers. Some common bitwise operators in Python are:

- Bitwise AND (&): Performs a bitwise AND operation on the binary representations of the operands.
- Bitwise OR (|): Performs a bitwise OR operation on the binary representations of the operands.
- Bitwise XOR (^): Performs a bitwise exclusive OR operation on the binary representations of the operands.
- Bitwise complement (~): Inverts the bits of the operand.
- Left shift (<<): Shifts the bits of the left operand to the left by the number of positions specified by the right operand.
- Right shift (>>): Shifts the bits of the left operand to the right by the number of positions specified by the right operand.

Membership Operators:

Membership operators are used to test whether a value is a member of a sequence (e.g string, list, tuple)

They include:

- In : Returns True if the value is found in the sequence
- Not in : Returns True if the value is not found in the sequence.

Identity Operators :

Identity operators are used to compare the identity of two objects. They include:

- Is : Returns True if both operands refer to the same object.
- Is not : Returns True if both operands do not refer to the same object.

Chapter 5 Python Basics to Advanced Control Flow

Control Flow directs program execution through structures like loops, conditionals, and functions, determining the order and path of operations.

IF Statements:

An if statement in Python checks whether a condition is true or false. If the condition is true, the code inside the if block runs. If False, the code is skipped. It's used to make decisions in the program, executing specific actions based on conditions.

example:

```
age = 18
if age >= 18:
    print("You are an adult")
else:
    print("You are a minor")
```

Else and elif statements:

In Python, else and elif statement are used alongside if to handle multiple conditions and alternative actions.

- **elif (else if):**
  checks another condition if the previous if was False. You can have multiple elif statements

- **else:**
  Runs when none of the elif or if conditions are same true. Its the 'default' action.

example:

```
temperature = 75

if temperature > 85:
    print ("Its too hot outside")
elif temperature > 76:
    print ("Its a nice day")
elif temperature > 50:
    print ("Its a bit chilly")
else:
    print("Its cold outside")
```

Nested if Statement:

A nested 'if statement in Python is an 'if statement inside and if. It lets you check multiple related conditions in sequence.

For example:

If you first check the weather and its sunny, you can then check how many guests are coming. Depending on the number of guests, you decide between different activities, like a barbecue or picnic.

If the weather isn't sunny, you skip the nested checks and go straight to an alternative action, like staying indoors.

```
temperature = 78
humidity = 65

if temperature > 70:
    printf "The weather is warm")
    if humidity > 60:
        print ("Its also quite humid")
    else:
        print("The humidity is low.")
else:
    print("The weather is cool")
```

```
if humidity > 60:
    print("Its humid despite the cool")
else:
    print("The weather is cool and dry")
```

The while loop:

A 'while loop' in Python repeatedly executes a block of code as long as a specified condition is true.

It first checks the condition; if true, the code inside runs. After each iteration, the condition is rechecked. The loop continues until the condition becomes false.

For example:

A while loop can keep counting up as long as the count is below a certain number. It's useful for scenarios where you don't know in advance how many iterations are needed.

```
count = 0
while count < 5:
    print("Count is :", count)
    count += 1

print("Loop ended")
```

The For Loop :

A For loop in Python is used to iterate over a sequence, such as a list, tuple, or range, executing a block of code for each item in the sequence

Unlike a while loop which runs untill a condition is false, a for loop runs a set number of times based on the length of the sequence.

For example:

It can go through a list of numbers, processing each one in turn. It's ideal for repetitive tasks like iterating over data collections.

example:

```
Fruits = ["apple","banana","cherry"]

For fruit in fruits:
    print (fruit)

print("Loop ended")
```

Loop Control Statements:

Loop control statements in Python allow

you to alter the flow of loop execution. They include:

break:

This is statement immediately exits the loop, regardless of whether the loop's condition is still true. It's useful for stopping a loop when a specific condition is met, like when searching for an item in a list and finding it before the loop has iterated through the entire list.

Example:

```
for i in range(10):
    if i==5:
        break
    print("Current number:", i)
print("Loop ended")
```

Continue:

This statement skips the rest of the current loop iteration and proceeds to the next iteration. It's helpful for bypassing certain parts of the loop based on a condition, like skipping even numbers in a loop that processes a range of numbers.

```
for i in range(10):
    if i % 2 == 0:
        continue
    print("Odd number:", i)
print("Loop ended")
```

Using Range() in for loop:

The range() function in Python is commonly used in for loops to iterate over a sequence of numbers.

```
for i in range(5):
    print("Number:", i)
```

Here's a basic rundown of how it works:

Start: The starting value of the sequence (inclusive). If ommitted, it defaults to 0.

Stop: The ending value of the sequence (exclusive). The loop will run until it reaches this value.

Step: The amount by which the sequence is incremented. If ommitted, it defaults to 1.

You can use range() in a for loop for various tasks like iterating through lists, generating sequencing of numbers, or performing repetitive actions a specific number of times.

example:

```
For i in range(10, 0, -1):
    print("Number:", i)
```

Chapter 6 Lists and Tuples

Lists and Tuples:

Lists and Tuples in Python both store collections of items. Lists are mutable, allowing changes like adding or removing elements, and use square brackets ([]).

Tuple are immutable, meaning they can't be modified after creation, and use parantheses (()). Use lists for dynamic data and tuples for fixed collections

Lists and Methods:

Lists in python are ordered, mutable collections used to store multiple items in a single variable. They are defined using square brackets ([]) and can contain elements of different data types, such as integers, strings, or even other lists.

You can add, remove or modify elements within a list, making them highly versatile for managing

Basic Operations:

- Create a list : my_list = [1,2,3,4]
- Access elements : my_list [0] (returns 1)
- modify elements : my_list [1] = "new_value"
- Append elements : my_list.append (5)
- Remove elements : my_list.remove(2) (removes the first occurence of 2)

1. Append()

Adds a single element to the end of the list
syntax: list.append(element)

2. extend()

Extends the list by appending elements from an iterable (like another list)
syntax : list.extend(iterable)

3. insert()

Inserts an element at a specified position in the list
syntax: list.insert(index, element)

4. remove()

Removes the first occurences of a specified element

syntax : list.remove(element)

5. PoP()

Removes and returns the element at a specified position (index). If no index is specified, it removes and returns the last element.
syntax : list.pop(index)

Indexing :

Indexing in Python refers to accessing individual elements within a sequence, such as a list, tuple, string or other iterable objects.

Each element in a sequence is assigned a numerical index starting from 0 for the first element and increasing by 1 for each subsequent element.

key points:

1. Positive thinking

- The first element has an index of 0.
- The second element has an index of 1, and so on.

2. Negative thinking

the sequence.
- The last element has an index of -1, the second last is -2, and so on.

3. Indexing in Strings:

- Indexing works similarly with strings, where each character has an index.

4. Out-of-Range Index:

- Accessing an index that is beyond the length of the sequence raises an IndexError.

Slicing

Slicing in Python is a technique used to access a subset of elements from sequences like lists, tuples or strings.

It allows you to retrieve a portion of the sequence by specifying a start, stop and optional step index

- **Start:**

The index where the slice begins (inclusive). If omitted, it defaults to the beginning of the sequence (0).

• Stop():

The index where the slice ends (exclusive). If omitted, it defaults to the end of the sequence.

• step:

The step size or interval between elements in the slice. If omitted, it defaults to 1.

List Comprehsion:

List comprehsion is a concise and powerful feature in Python that allows you to create lists in a single line of code.

It combines the process of creating and populating a list with an expression and optionally, one or more loops and conditions.

Syntax:

[expression for item in iterable if condition]

expression:

The value or operation applied to each item.

item:

element in the iteration.

- **iterable:**

The collection (like a list, tuple or range) that you are iterating over.

- **Condition:**

A filter that decides whether the expression should be applied to the current item.

Benefits of List Comprehsion:

- **Concise:**

It reduces the lines of code needed to create and populate lists.

- **Readable:**

Once you get familiar with the syntax, it can be easier to read and understand.

- **Efficient:**

It often runs faster than traditional for-loop approaches due to its optimized implementation.

Tuples and their Immutabilty

Tuples in Python are ordered collections of items, similar to lists, but with one key difference tuples are immutable. This means that once a tuple is created, its elements cannot be modified, added or removed. This immutability makes tuples useful for representing fixed data that should not change throughout the program.

Key Features of Tuples:

Ordered:

Like lists tuples maintain the order of elements and you can access elements by their index.

Immutable:

Once a tuple is created, you cannot change its content - This includes:

- Modifying elements: You cannot change the value of any item in the tuple.
- Adding elements: You cannot append or insert new items.
- Removing elements: You cannot remove items

Define with Parentheses: Tuples are created by placing a sequence of values seprated by commas inside parentheses ().

Tuple Packing and Unpacking:

Tuple packing and unpacking are two related concepts in python that make working with tuples more convenient and intuitive.