**INDEX**

1. [Python Intro](#Chapter)
2. [Python Variables and types](#Chapter2)
3. [Python Built in Functions and Modules](#Chapter3)
4. [Python Strings](#Chapter4)
5. [Python Comparison operator and logical operator](#Chapter5)
6. [IF ELSE](#Chapter6)
7. [Nested If Else](#Chapter7)
8. [Python Lists](#Chapter8)
9. [Python Tuples](#Chapter9)
10. [Python Sets](#Chapter10)
11. [Python Dictionaries](#Chapter11)
12. [Slice and Dictionary](#Chapter12)
13. [Python While Loop](#Chapter13)
14. [Python For Loop](#Chapter14)
15. [Python Break & Continue](#Chapter15)
16. [Python Function](#Chapter16)
17. [Python Arguments](#Chapter17)
18. [Python Objects](#Chapter18)
19. [Python classes & objects](#Chapter19)
20. [Python init & self in class](#Chapter20)
21. [Python Multiple Constructor](#Chapter21)
22. [Python Encapsulation](#Chapter22)
23. [Python Private Methods](#Chapter23)
24. [Python Inheritance](#Chapter24)
25. [Python Multiple Inheritance](#Chapter25)
26. [Python Super](#Chapter26)
27. [Python Composition](#Chapter27)
28. [Difference in Aggregation and Composition](#Chapter28)
29. [Python Abstract Classes](#Chapter29)
30. [Exception Handling](#Chapter30)
31. [Try Except Else Finally](#Chapter31)
32. [If Name](#Chapter32)
33. [Python Iterators](#Chapter33)
34. [Python Generator](#Chapter34)
35. [Command Line Arguments](#Chapter35)
36. [Lambda Filter Reduce Map](#Chapter36)
37. [Python Closure Nested Function](#Chapter37)
38. [Python Decorators](#Chapter38)
39. [Operators Overloading](#Chapter39)

**GitHub link For all the codes:**

[**https://github.com/bishalpaul777/python-tutorial**](https://github.com/bishalpaul777/python-tutorial)

**Chapter 1 -- Python Intro**

**Q**: **Define Python**.

**Ans**:

* Python is a **Multi-Paradigm**, **Interpreted** programming language.
* Supports **Dynamic Data Types.**
* Independent of **Platforms.**
* **Open Source.**
* **High Level** language.

**Q: Why learn python**.

**Ans**:

* **Reduce development time.**
* **Reduce code length.**
* **Easy to understand, to do team projects.**

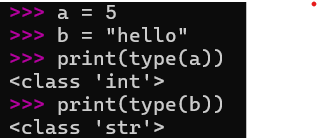
**Q: Where to use python**.

**Ans**:

1. **Web Development (Django and Flask)**
2. **Data Science.**
3. **Scripting.**
4. **GUI.**

**Chapter 2 -- Python Variabl****es & Types**

**Q:** **What is variable**?

**Ans**: A variable is a named place or storage location in a program where value can be stored, to perform operations.

a = 5

b = “Hello”

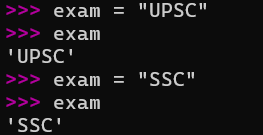
**Q**: **Variable naming rules**.

**Ans**:

* Must start with a letter or underscore
* Can only contain letters, digits, and underscores
* Case-sensitive (lowercase and uppercase are different)
* Cannot use reserved keywords
* Cannot contain spaces.

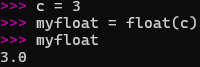
**Q**: **Is Re assignment of variable in python possible?**

**Ans**: Yes it’s possible. We can rename a variable.



**Q**: **What is type casting?**

**Ans**: Floating type casting in Python means converting another data type (like int or string) into a float using float().

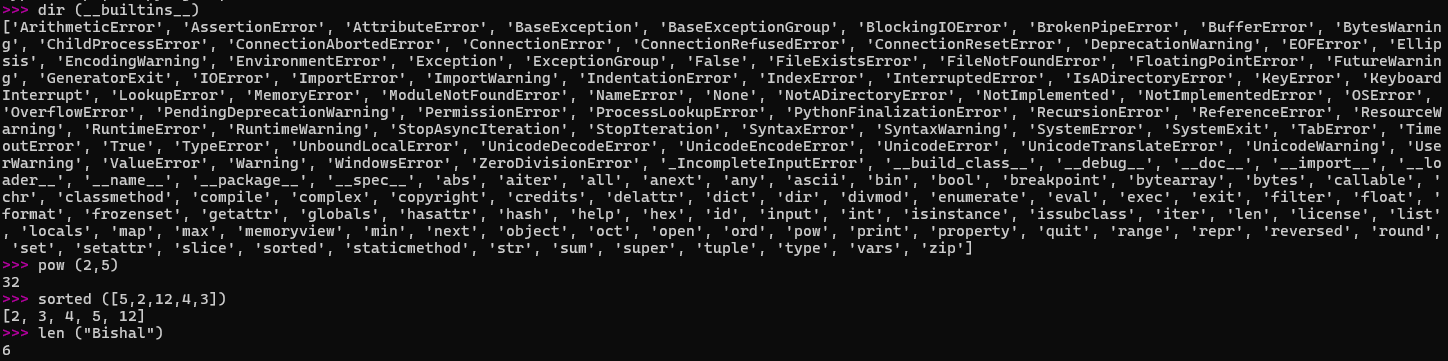


**Chapter 3 -- Python Buit in Functions & Modules**

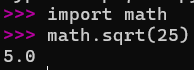
**Q**: **Built in Functions.**

**Ans**:

**Built-in functions** in Python are the functions that come pre-installed and are always available for use without importing anything. They help perform common tasks like printing output, converting data types, finding length, or doing calculations.



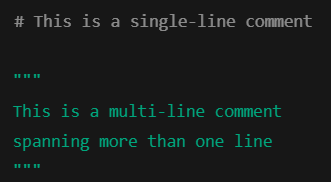
A **module** in Python is a file that contains reusable code such as functions, classes, or variables. Python provides many **built-in modules** (like math, random, os) that can be imported, and you can also create your own custom modules.



**Chapter 4 -- Python Strings & Comments**

**Q**: **Python Comments.**

**Ans**: **Comments** are notes you write in your code to explain what it does. Python ignores comments when running the program.

There are different types of comments:

1. **Single-line Comment** 
   * Starts with # and runs till the end of the line.
   * Used for short explanations.
2. **Multi-line Comment**
   * Python doesn’t have a special syntax for block comments, but we can use triple quotes ''' or """ as a workaroun

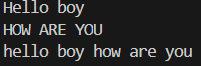
**Q: Python Strings.**

**Ans:**

A string is a sequence of characters enclosed in single quotes ('), double quotes (") or triple quotes (''' ''' / """ """). Strings are used to store and work with text in Python.

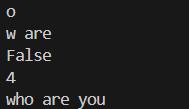
a = "hello boy"

b = "how are you"

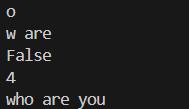
print(a.capitalize())

print(b.upper())

print(a + " " + b) # Concat two strings

print(a[7]) # indexing

print(b[2:7]) # Slicing

print(a.isupper())

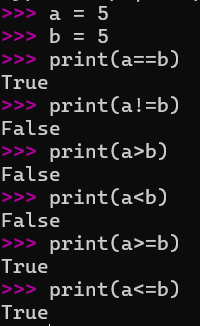
print(b.find("are")) # Finding Substring

print(b.replace("how","who")) # replace substring

**Chapter 5 -- Python Comparision Operators and Logical Operators**

* **Comparison Operators:** These operators are used to compare two variables. They return value as **TRUE** or **FALSE**.

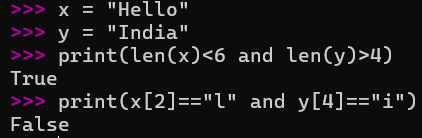
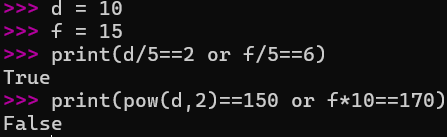
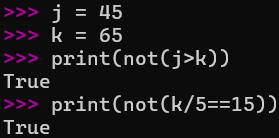
Some comparison operators are:



1. == Equal to
2. != Not equal to
3. > Greater than
4. < Less than
5. >= Greater than or equal to
6. <= Less than or equal to

* **Logical Operators:** Logical operators in Python are operators that are used to combine conditional statements and return a Boolean value (True or False). They help in making decisions based on multiple conditions.

The three logical operators are:

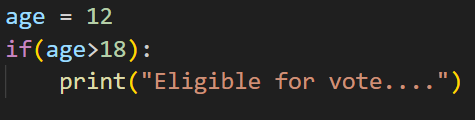
1. **and** → True if both conditions are true
2. **or** → True if at least one condition is true
3. **not** → Reverses the result of a condition

**Chapter 6 -- Python IF ELSE statements**

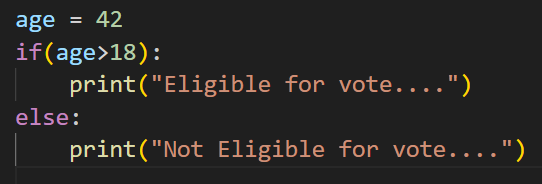
**Q: Define IF Else statement.**

**Ans**:

* IF statement: **IF** statement in Python is a conditional statement used to execute a block of code only when the given condition is true.



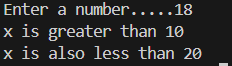
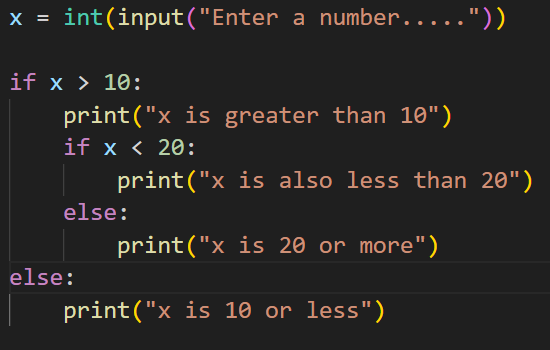
* IF-ELSE statement: **IF-ELSE** statements in Python are conditional statements that let you execute a block of code if a condition is true, and another block of code if the condition is false.

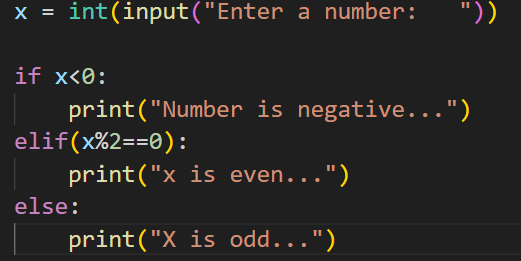


**Chapter 7 -- Python NESTED IF**

**Q**: **Define Nested If.**

**Ans**: **Nested IF** statement in Python means placing one if statement inside another. It is used when we need to check multiple conditions one after another in a hierarchy.





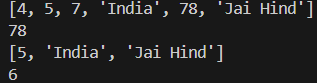


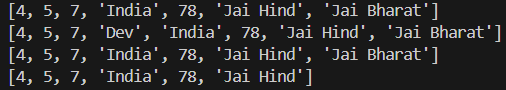
**Chapter 8 -- Python LISTS**

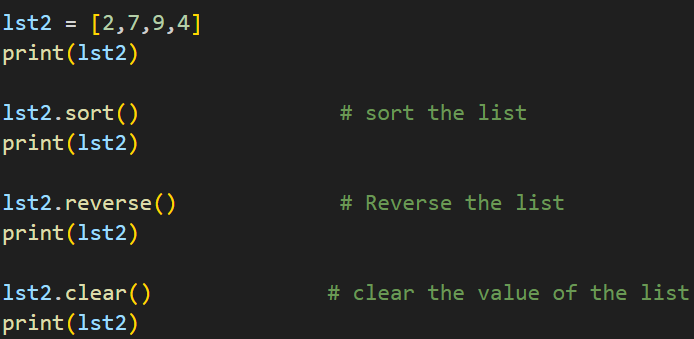
**Q: Define List.**

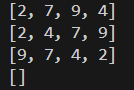
**Ans:**

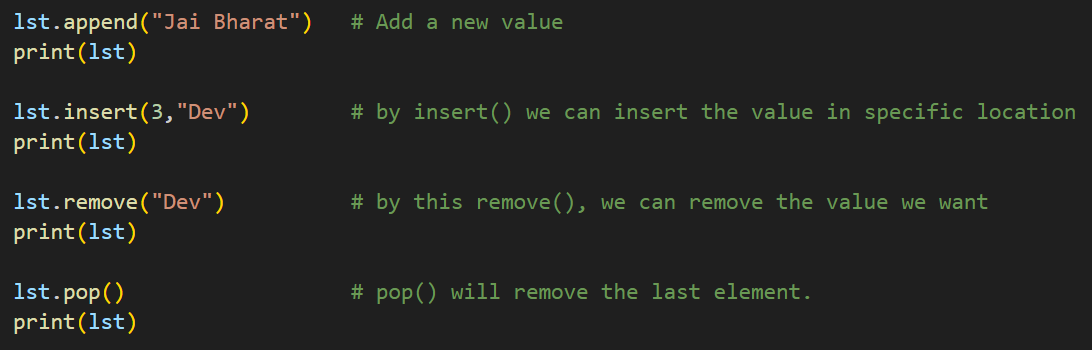
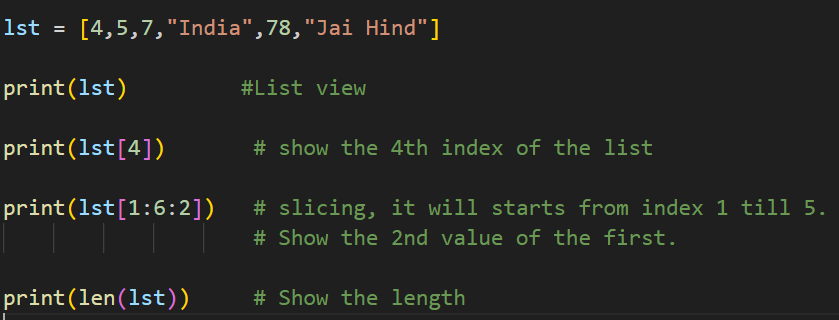
List in Python is a collection data type that can store multiple items in a single variable. Lists are ordered, mutable (can be changed), and can hold mixed data types like numbers, strings, or even other lists.







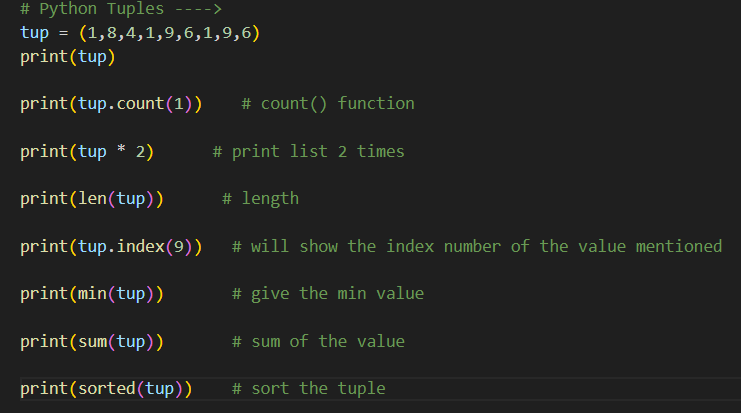
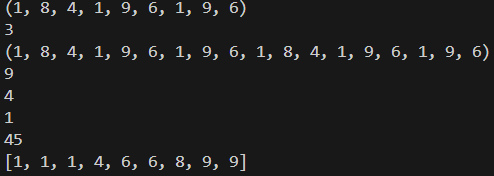


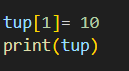
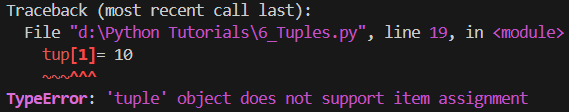
**Chapter 9 -- Python TUPLE**

**Q: Define python Tuples.**

**Ans**: Tuple in Python is a collection data type that is ordered and immutable (cannot be changed after creation). Tuples can store multiple items of mixed data types, just like lists, but once created, their elements cannot be modified, added, or removed.



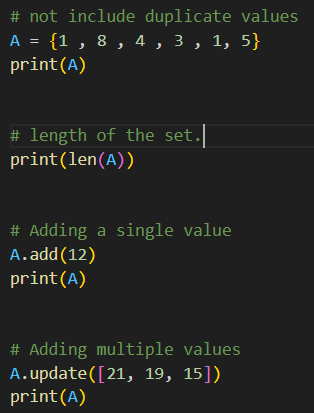
As we tuples are immutable, so if we want to change or append any value, it will give error.

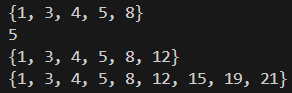


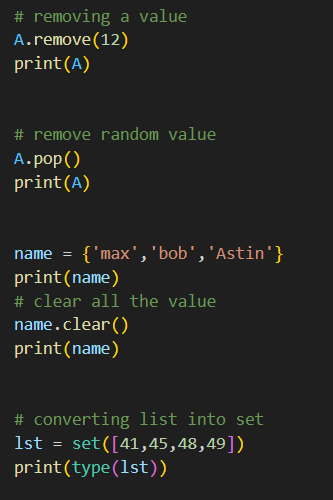
**Chapter 10 -- Python SETS**

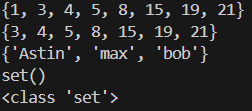
**Q: Define Sets.**

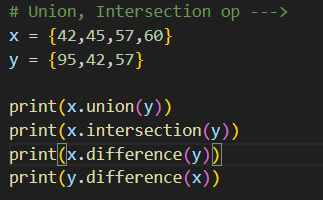
**Ans**: **SET** in Python is a collection data type that is unordered, unindexed, and contains only unique elements. Sets are mainly used when you want to store multiple items but automatically remove duplicates.

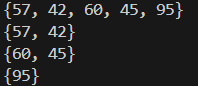
👉 Sets are written inside curly braces {}, with items separated by commas.











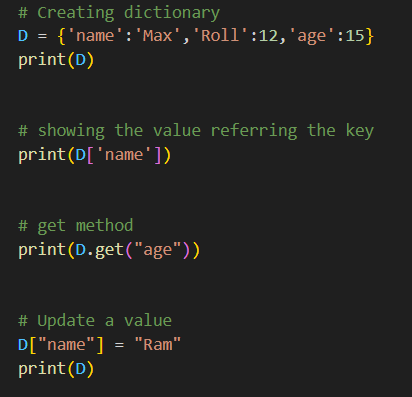
**Chapter 11 --** **Python DICTIONARIES**

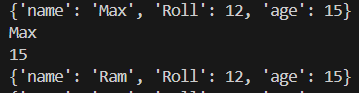
**Q: Define Dictionary.**

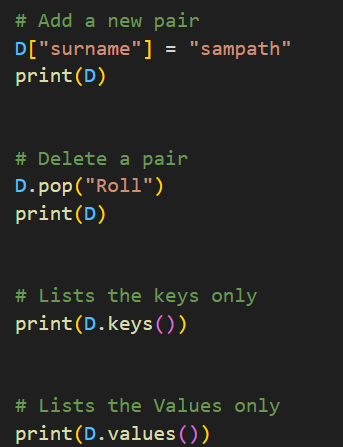
**Ans**: Dictionary in Python is a collection data type that stores data in the form of key–value pairs. It is unordered, mutable, and indexed by keys, where each key must be unique and immutable, while values can be of any data type.

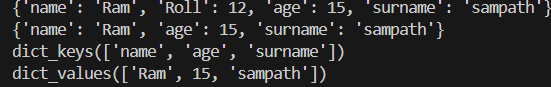
👉 Dictionaries are written inside curly braces {} with the format:

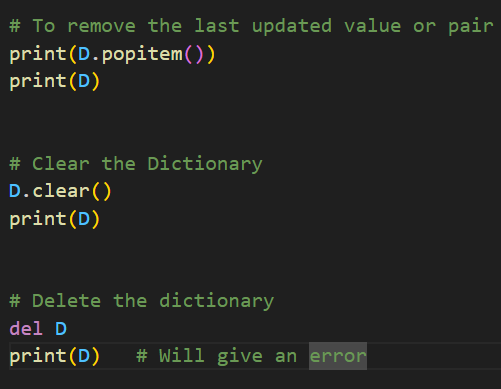
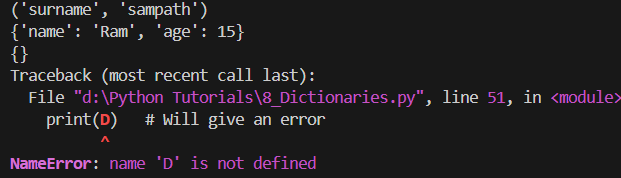
{key: value, key: value}







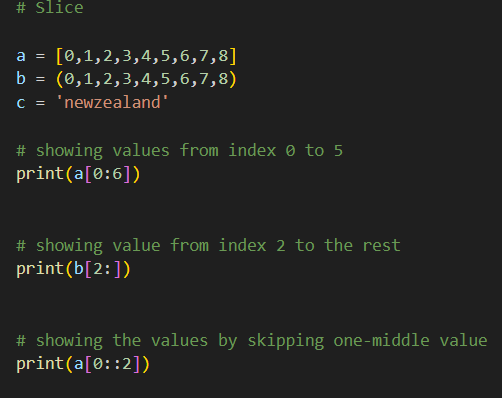


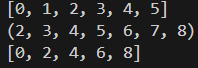


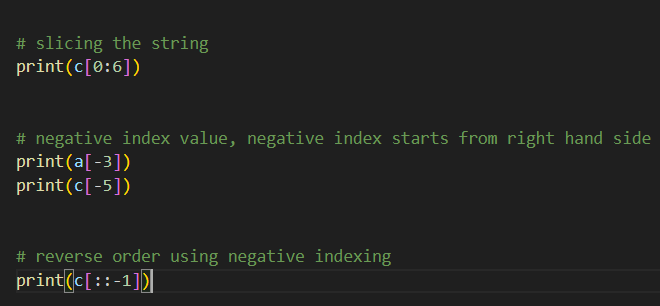
**Chapter 12 -- Python Slice & Dictionary**

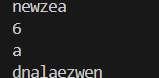
**Q: Define Slicing.**

**Ans**: **Slicing in Python** is a technique used to extract a specific portion (substring, sub list, etc.) from a sequence like a string, list, or tuple by specifying a **start index, end index, and step**.









**Chapter 13 -- Python While Loop**

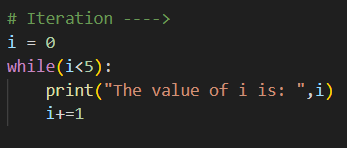
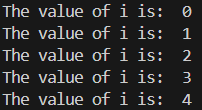
**Q: Define While Loop.**

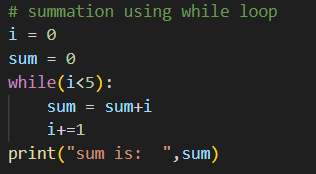
**Ans:** While loop in Python is a control flow statement that repeatedly executes a block of code as long as the given condition is True. When the condition becomes False, the loop stops.

Syntax:

while condition:

# code block







**Chapter 14 -- Python For Loop**

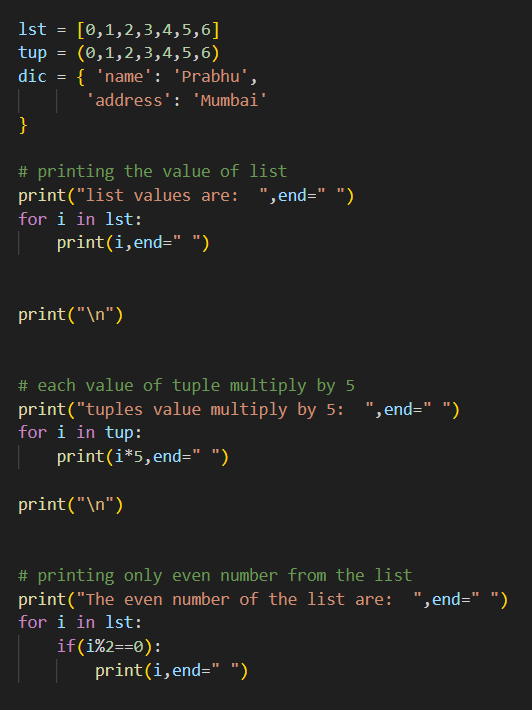
**Q: Define For Loop.**

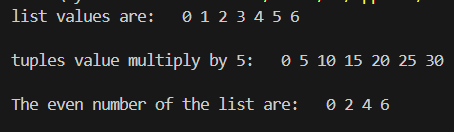
**Ans: For loop in Python** is a control flow statement used to **iterate over a sequence** (like a list, tuple, string, or range) and execute a block of code for each item in that sequence.

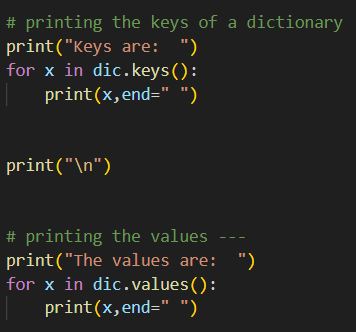
Syntax:

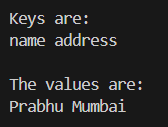
for variable in sequence:

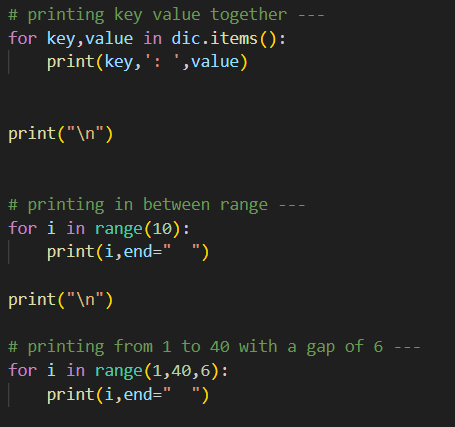
# code block

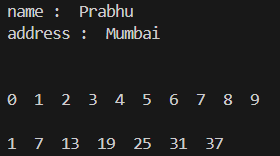








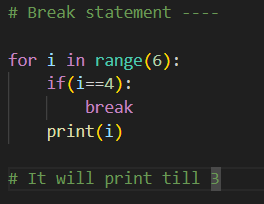




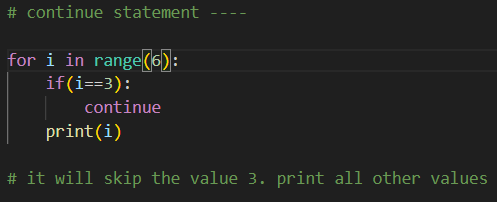
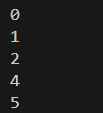
**Chapter 15 -- Python Break & Continue**

**Q: Define Break.**

**Ans:** The **break statement** is used inside loops to immediately stop the loop when a certain condition is met. Once break is executed, control moves to the first line after the loop.



**Q: Define Continue.**

**Ans:** The **continue statement** is used inside loops to skip the current iteration when a condition is met. Instead of terminating the loop, it moves directly to the next iteration of the loop.

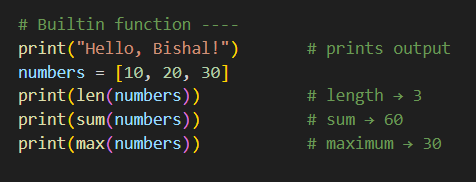
**Chapter 16 -- Python Function**

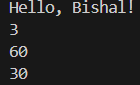
**Q: Define Python Functions and its types.**

**Ans:** A **FUNCTION** in Python is a block of reusable code that performs a specific task. Functions help reduce code repetition, improve readability, and make programs modular. You define a function using the def keyword and call it by its name.

There are 2 types of functions in python:

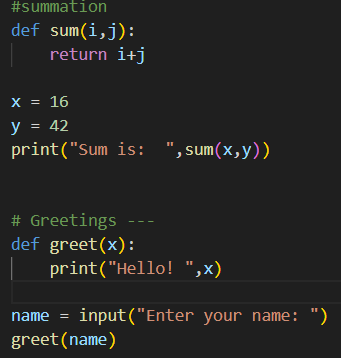
1. **Built-in Functions**: Functions that come pre-defined in Python (e.g., print(), len(), sum()).





1. **User-defined Functions**: Functions created by the programmer using

“**def function\_name(arg1, arg2,…..):”**

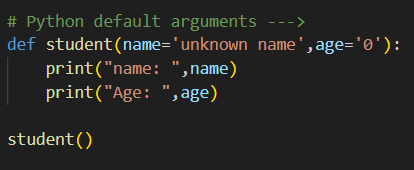




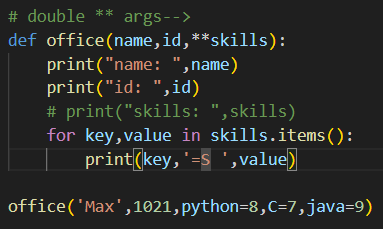
**Chapter 17** **-- Python Arguments**

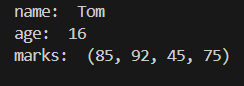
**Q: Define Python Arguments.**

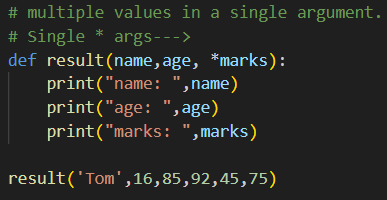
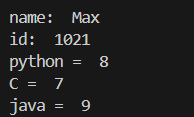
**Ans: Arguments** in Python are the values or data you pass to a function when calling it. They provide input for the function so it can perform its task.











**Chapter 18 -- Python Objects**

**Q: Define Python Objects.**

**Ans:** Object in Python is an instance of a class that bundles together data (attributes/variables) and behavior (methods/functions). Everything in Python (like numbers, strings, lists, functions) is treated as an object, because Python follows the object-oriented programming (OOP) approach.

class cab{ 🡨class

cabservice, location, numberplate 🡨data

book(), arrival(), start() 🡨methods

}

Class passenger{

name, address 🡨 data

openApp(), bookCab(), walk() 🡨 methods

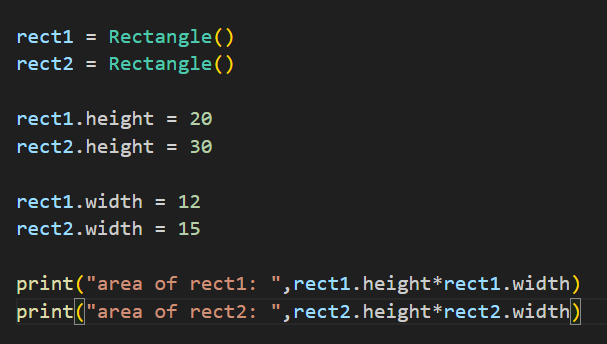
}

**Chapter 19 -- Python Classes & Objects**

**Q: Python Class.**

**Ans: Class** in Python is a blueprint or template used to create objects. It defines the structure and behavior of objects by grouping together variables (called attributes) and functions (called methods) under one unit.





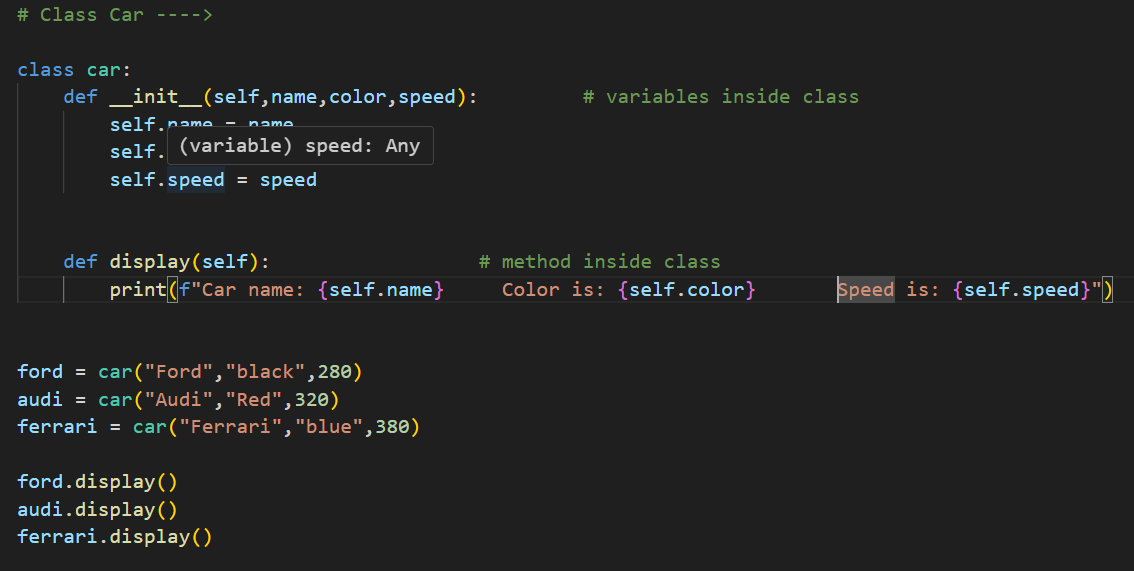


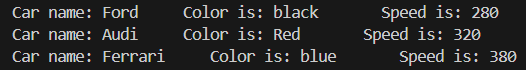
**Chapter 20 -- Python Classes & Objects**

**Q: Define “\_\_init\_\_” and “self” in class.**

**Ans:** The **\_\_init\_\_** method is a special function inside a class, also known as a constructor. It is called automatically whenever a new object of the class is created, and it is mainly used to initialize the object’s attributes with given values.

The **self** keyword represents the current instance of the class. It is used inside class methods to access the object’s attributes and methods, ensuring that each object maintains its own separate data.



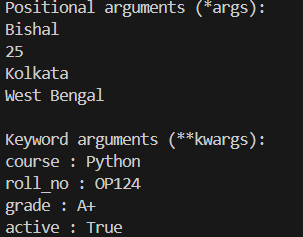


**Chapter 21 -- Python Multiple constructor**

**Q: Define Multiple constructors in python.**

**Ans: Multiple constructors** in Python refer to having more than one way to initialize objects of a class. Since Python does not directly support multiple constructors, this is usually achieved by using default arguments, \*args/\*\*kwargs.





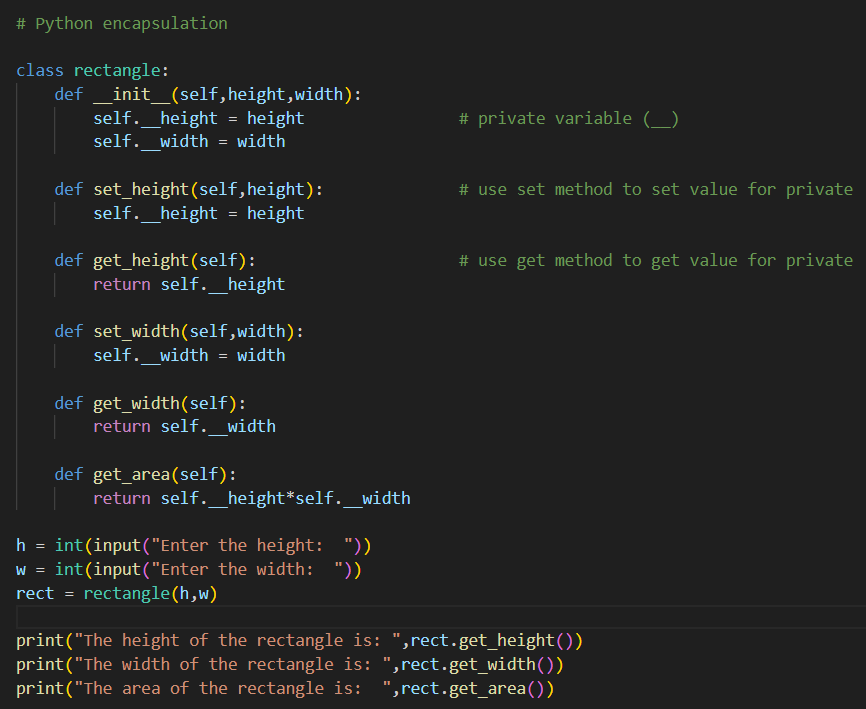
**Chapter 22 -- Python Encapsulation**

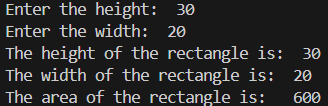
**Q: Define Python Encapsulation.**

**Ans: Encapsulation** in Python is an object-oriented programming (OOP) concept where the internal details of a class (its data and methods) are hidden from direct access and can only be accessed or modified through controlled interfaces (like methods).

In Python, encapsulation is implemented using access specifiers:

* Public → accessible everywhere.
* Protected (\_var) → a convention meaning “internal use only” (still accessible, but discouraged).
* Private (\_\_var) → name mangling makes it harder to access directly from outside the class.

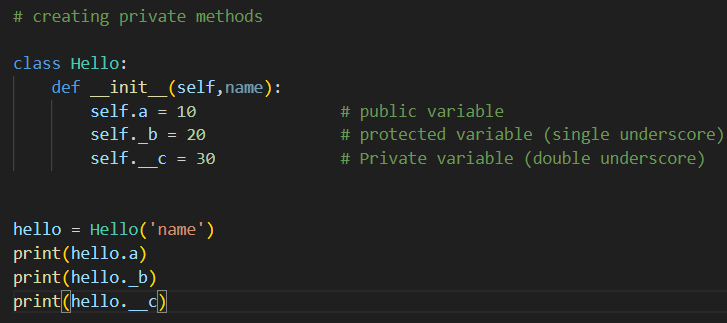


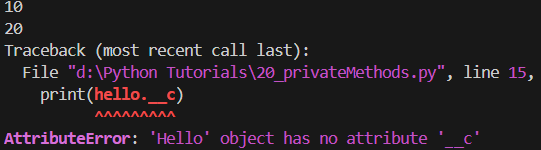


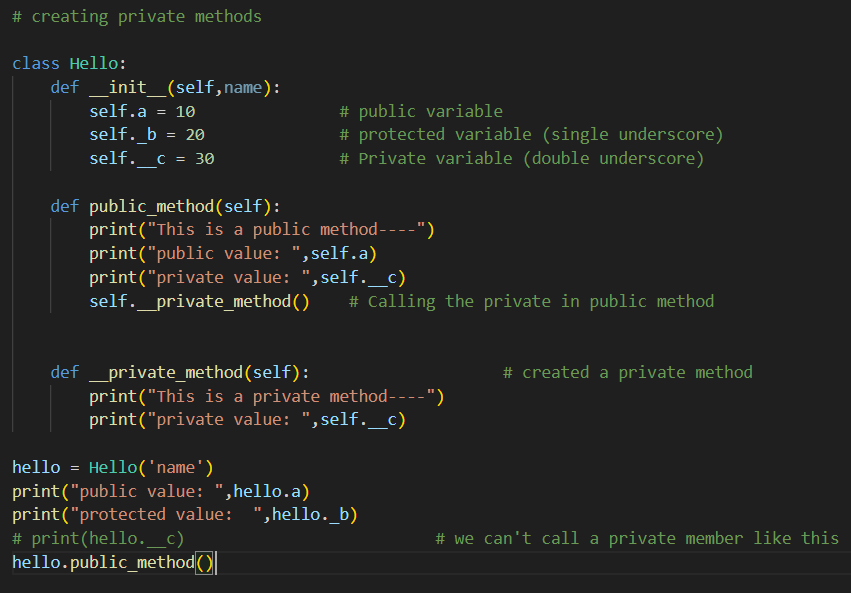
**Chapter 23 -- Python Private Methods**

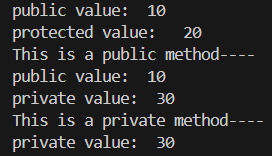
**Q: Private Methods.**

**Ans:** We can’t call private variable of a class simply just “object.call()” method. It will give us an error.



 To access the private members, we need private methods.



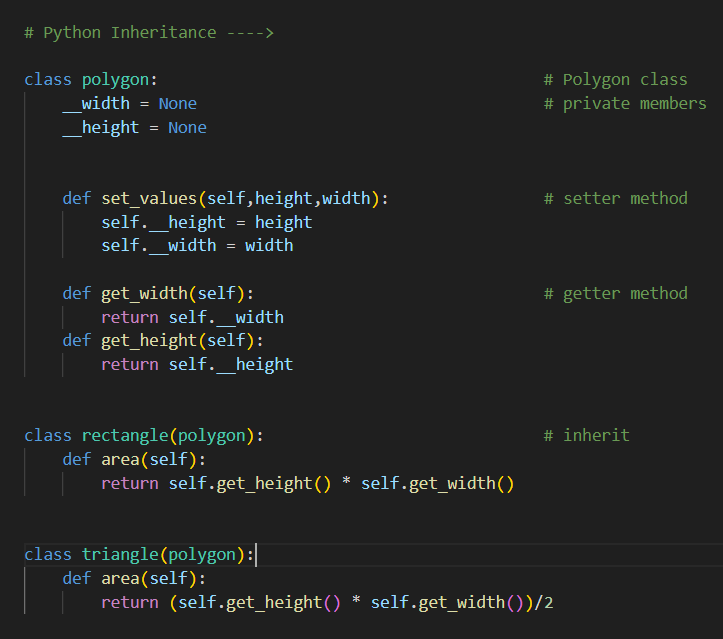


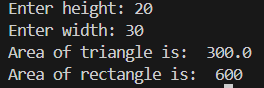
**Chapter 24 -- Python Inheritance**

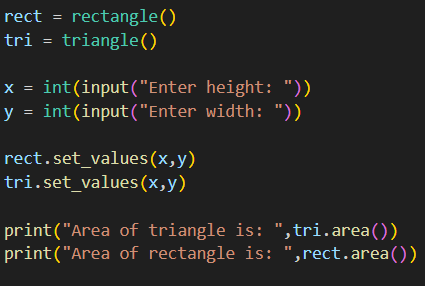
**Q: Define Python Inheritance.**

**Ans: Inheritance** in Python is an Object-Oriented Programming (OOP) concept where one class (called the child or subclass) can inherit properties and methods from another class (called the parent or superclass).

This allows code reusability and lets us extend or modify the behavior of the parent class in the child class.





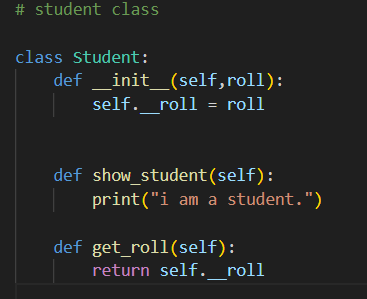
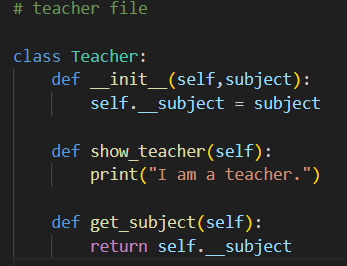


**Chapter 25 -- Python Multiple Inheritance**

**Q: Define Multiple Inheritance.**

**Ans: Multiple Inheritance** in Python means a class can inherit from more than one parent class. This allows the child class to have attributes and methods from multiple classes.

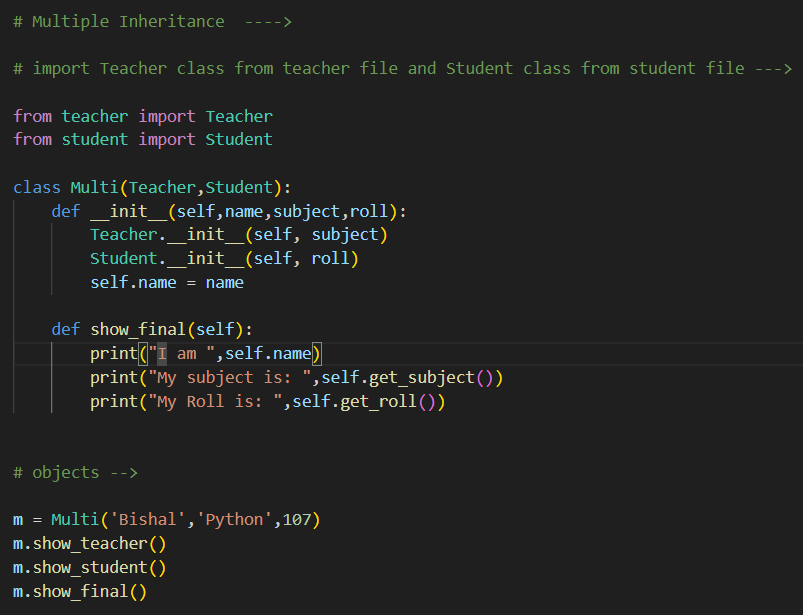
We have two different files: **Teacher.py** & **Student.py.** We have to inherit the properties of the class in these two files to the main files.

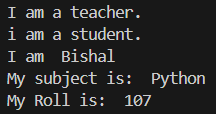


Teacher.py

Student.py

Now import these two files to the main file, where we will do inheritance.

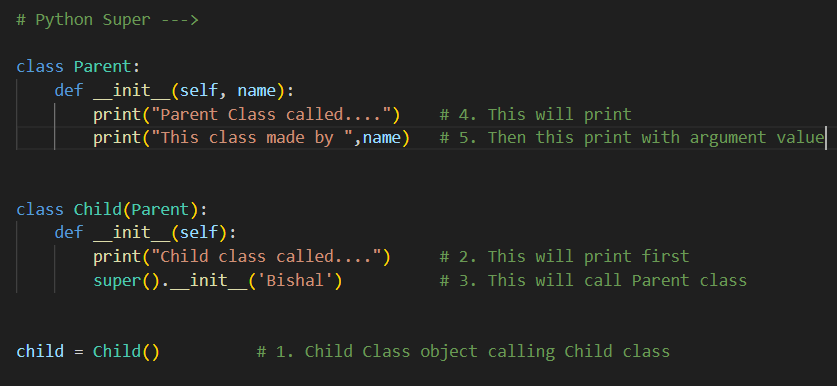


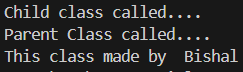


**Chapter 26 -- Python Super**

**Q: Define Super.**

**Ans: SUPER** in Python is a built-in function used inside a class to call methods or constructors from its parent class. It’s mainly used in inheritance to avoid repeating code and to make sure the parent’s initialization or methods are properly executed in the child class.





**Chapter 27 -- Python Composition**

**Q: Define Composition.**

**Ans: Composition** is an Object-Oriented Programming concept where one class is built using objects of other classes. Instead of inheriting from a parent class, the class contains other class instances as its members.

**“Inheritance = "is-a"** relationship, while **Composition = "has-a"** relationship.”



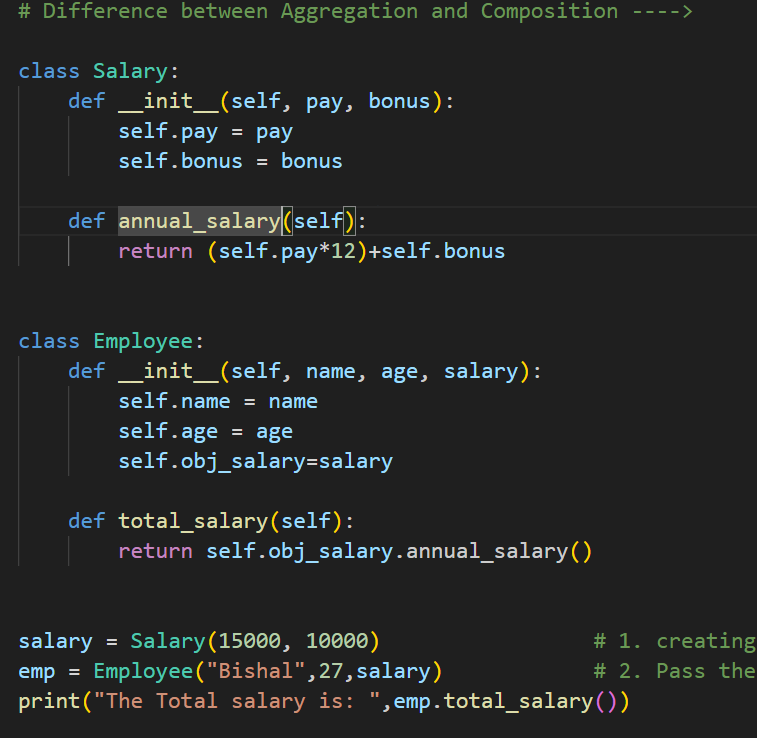


**Chapter 28 -- difference in aggregation and composition**

**Q: Define Aggregation.**

**Ans: Aggregation** is a special form of composition where one class contains another class’s object, but the contained object can exist independently of the container.

* **Composition** = strong relationship → if the container is destroyed, the parts are also destroyed.
* **Aggregation** = weak relationship → the parts can still exist even if the container is destroyed.





**Chapter 29 -- Python Abstract Classes**

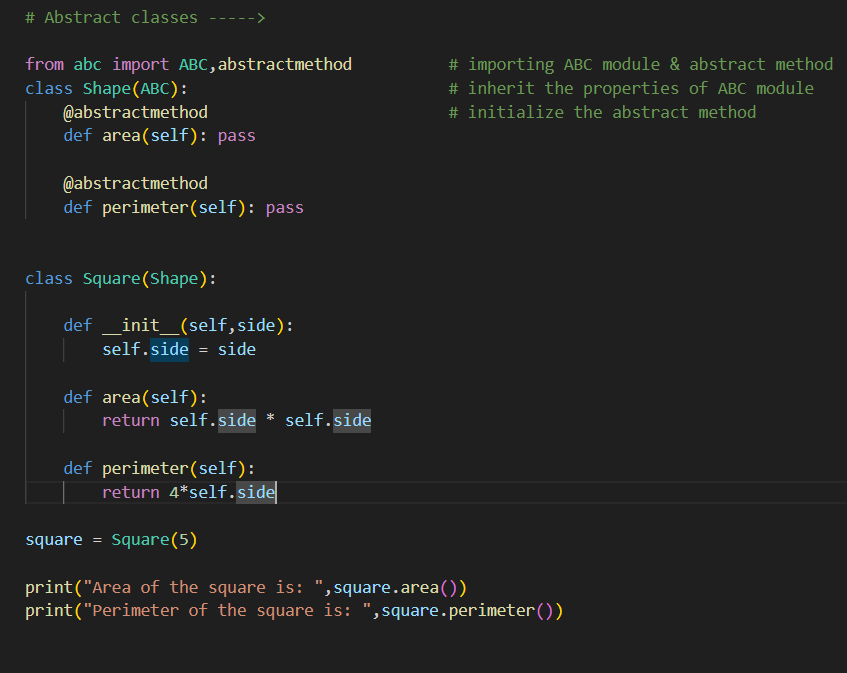
**Q: Define Python Abstract Classes.**

**Ans:** An **abstract class** in Python is a class that cannot be instantiated directly and is meant to serve as a blueprint for other classes.

It can contain:

* **Abstract methods** → methods declared but not implemented (subclasses must implement them).
* Concrete methods → normal methods with implementation.

Abstract classes are created using the abc module with the ABC class and **@abstractmethod** decorator.

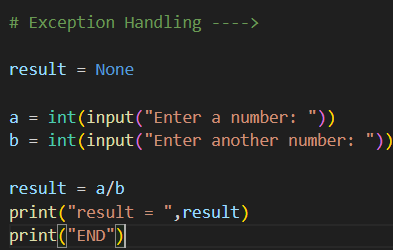


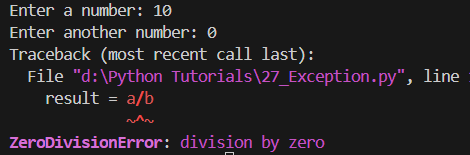
**Chapter 30 -- Python Exception Handling**

**Q: Define Exception Handling.**

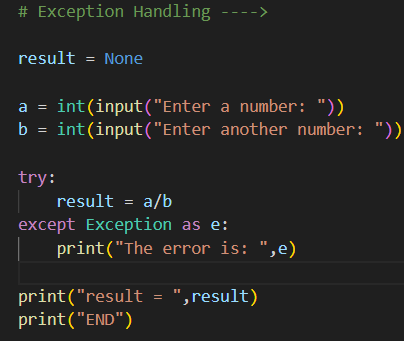
**Ans: Exception handling** in Python is a way to deal with errors that occur at runtime without stopping the whole program. Instead of crashing, the program can catch the error, handle it gracefully, and continue running.

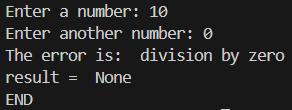
Python uses **try**, **except**, **else**, and **finally** blocks to manage exceptions.



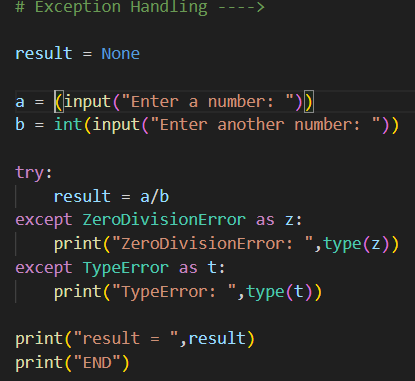


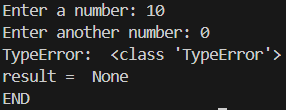
It gives an error, and stop the flow of the program as well. We need throw an **exception** there to handle the error.



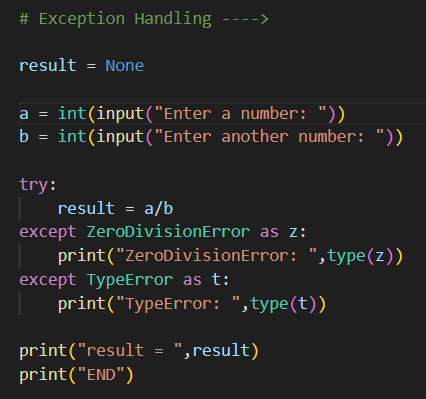


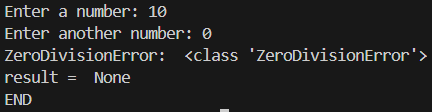
Or we can segregate the errors 🡪

* Typed one string, instead of integer. So it will give **TypeError:**



* Divided by 0, so it will give **Division by Zero** error**:**





**Chapter 31 -- Try Except Else Finally**

**Q: Define Try.**

**Ans:** The **try** block is where you put the code that might cause an exception.

**Q: Define Except.**

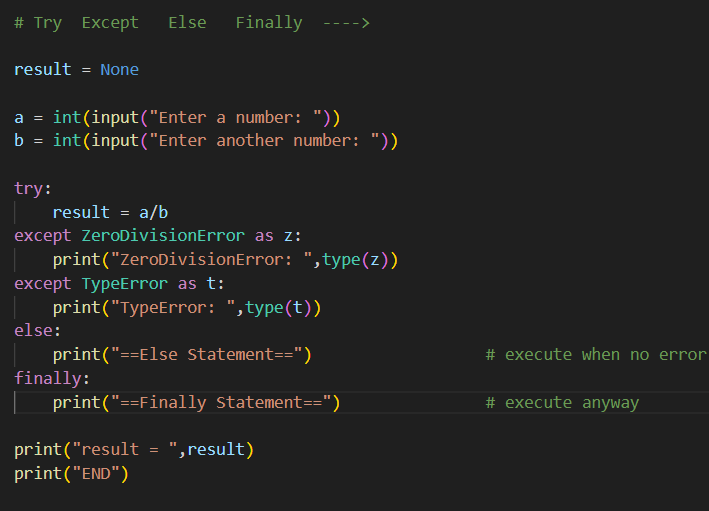
**Ans:** The **except** block is where you handle the error if it occurs inside the try.

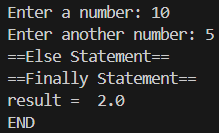
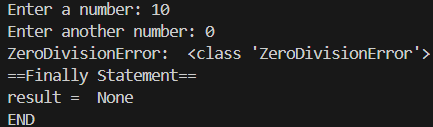
**Q: Define Else.**

**Ans:** The **else** block runs only if no exception occurs in the try block.

**Q: Define Finally.**

**Ans:** The **finally** block runs no matter what happens (whether an exception occurred or not). It’s often used for cleanup tasks like closing files or releasing resources.



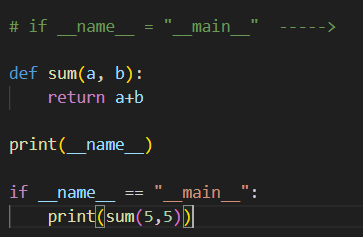
No error: Error:

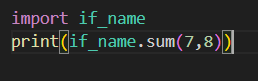
**Chapter 32 -- If Name**

**Q: Define “if \_\_name\_\_ = “\_\_main\_\_” ”.**

**Ans:** This line is used to check whether a Python file is being **run directly** or being **imported as a module** in another file.

* If the file is **run directly**, the special variable \_\_name\_\_ is set to "\_\_main\_\_", and the code inside this block will execute.
* If the file is imported somewhere else, \_\_name\_\_ is set to the module’s name, and the block will not run.

main file: import to another file:







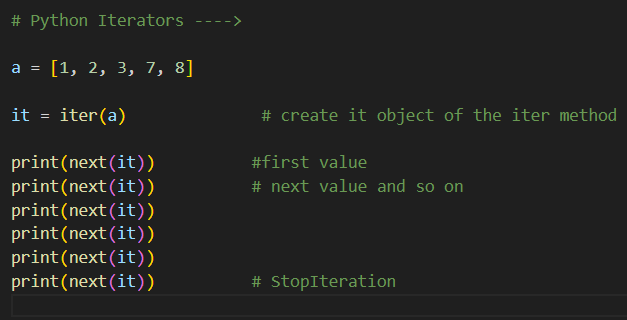
**Chapter 33 -- Python Iterators**

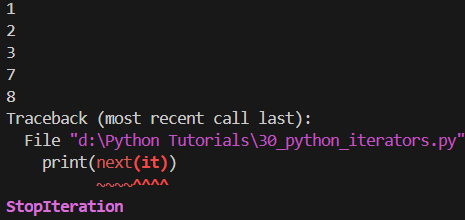
**Q: Define Python Iterators.**

**Ans:** An **iterator** in Python is an object that allows you to traverse (loop through) elements one by one.

Iterators implement two special methods:

* \_\_iter\_\_() → returns the iterator object itself.
* \_\_next\_\_() → returns the next value; raises StopIteration when there are no more items.



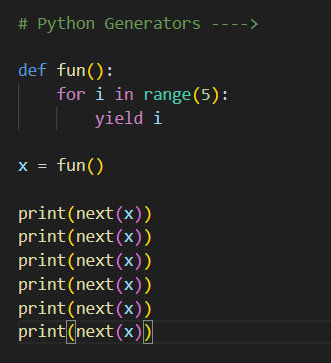


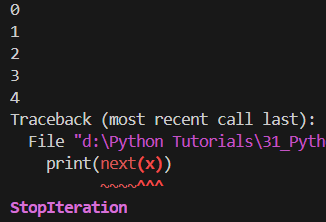
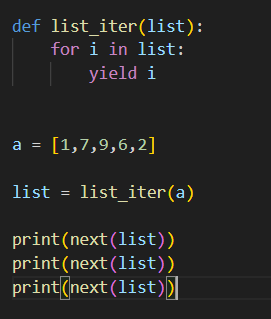
**Chapter 34 -- Python Generator**

**Q: Define Python Generator.**

**Ans: GENERATOR** in Python is a special type of function that produces values one at a time using the **YIELD** keyword, instead of returning them all at once with return.

* They are memory efficient because values are generated on the fly (lazy evaluation).
* A generator is both an iterator and an iterable (so we can use next() or loop through it).
* Once exhausted, a generator cannot be reused unless recreated.

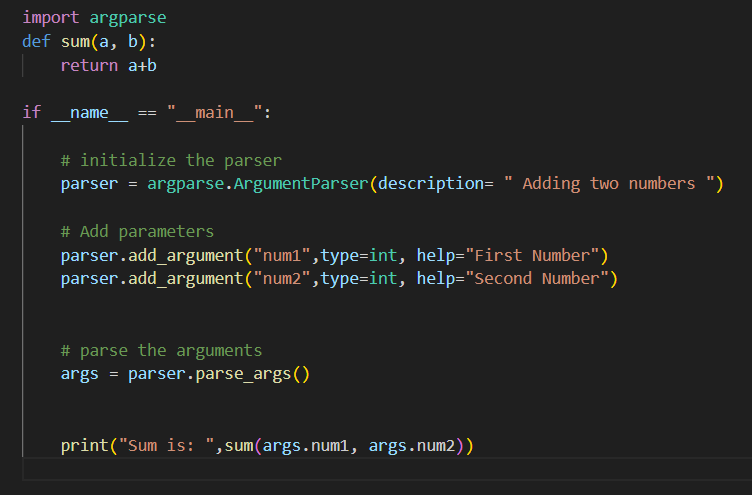
****

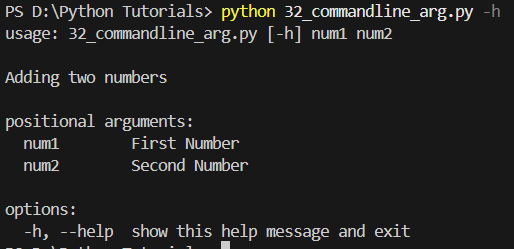
****

**Chapter 35 -- Python Commandline Arguments**

**Q: Define Command Line Argument.**

**Ans: Command line arguments** are the values you pass to a Python program when running it from the terminal/command prompt.



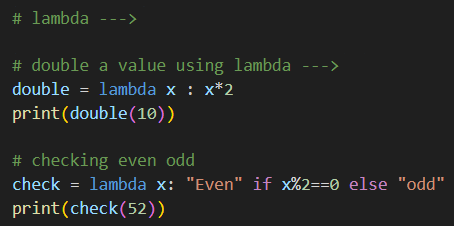
****

**Chapter 36 -- Lambda filter reduce map**

**Q: Define Lambda.**

**Ans:** A **lambda** is a small, anonymous (nameless) function in Python. It’s created using the lambda keyword instead of def.

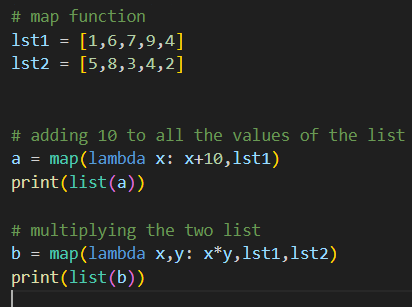
* **Syntax:** “lambda arguments: expression”
* It can take any number of arguments but only one expression.
* Commonly used for short, throwaway functions (like in map(), filter(), sorted(), etc.).





**Map():** In Python, **map()** is used when you want to apply a function to every item in a sequence, like a list or tuple, and get a new sequence back.

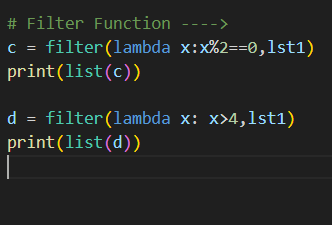
For example, if we want to double every number in a list, **map(**) does it without writing a loop.





**Filter(): filter()** is used when we want to pick only the elements from a sequence that meet a certain condition.

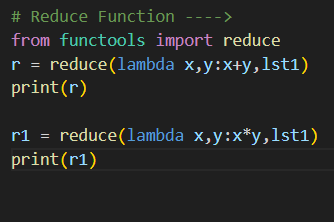
For example, if we want only the even numbers from a list, you can use filter() with a condition.





**Reduce(): reduce()**, which comes from the functools module, is used when we want to apply a function cumulatively to all the items in a sequence and reduce them to a single value.

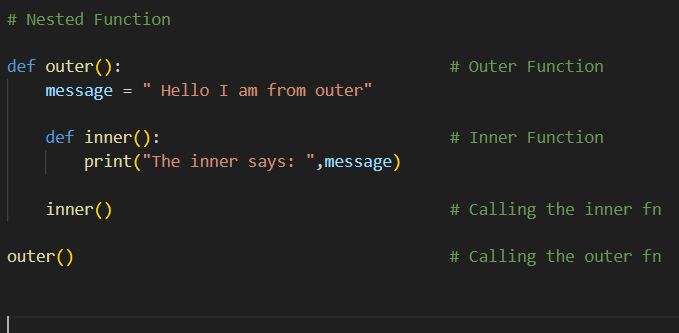
For example, adding up all the numbers in a list can be done with reduce().



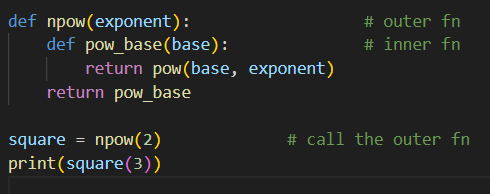
**Chapter 37 -- Python Nested Functions**

**Q: Define Nested Function.**

**Ans:** A nested function is simply a function defined inside another function. The inner function can access variables of the outer function, which makes it useful for organizing code and creating function closures.







**Chapter 38 -- Python Decorators**

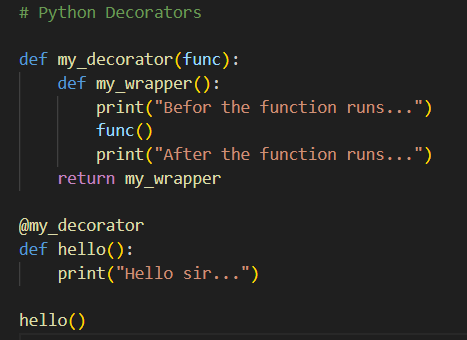
**Q: Define Decorators.**

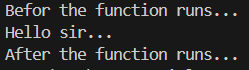
**Ans:** A **decorator** is a special function that lets you add extra features or modify the behavior of another function without changing its actual code.

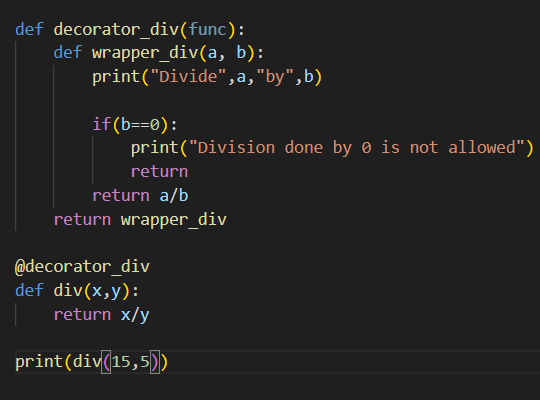
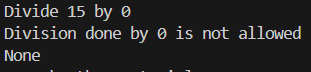
They are written using the **@decorator\_name** syntax and are often used for things like logging, authentication, timing, or validation.

**A decorator**

**A wrapper around a function that runs some extra code before or after the original function.**







**Chapter 39 -- Python Operator Overloading**

**Q: Define Operator Overloading.**

**Ans: Operator overloading** means **using the same operator (like +, -, \*) for user-defined objects** in a way that makes sense for them.

