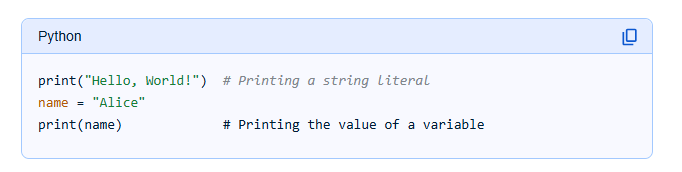
**Assignment (Module – 3)**

**Q1) Introduction to the print() function in Python.**

**ANS)** The print() function in Python is a built-in function used to display output to the console or standard output device. It is fundamental for communicating with users, debugging code, and presenting results during program execution.

Basic Usage:

* To print a simple message or the value of a variable, enclose the desired output within the parentheses of the print() function.

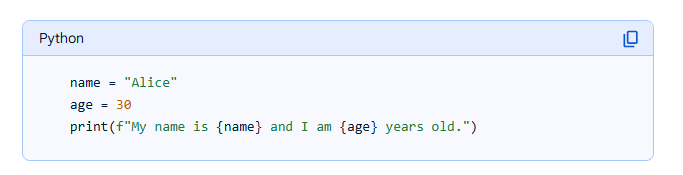


**Q2) Formatting outputs using f-strings and format().**

**ANS**) Python provides two primary methods for formatting output strings: f-strings (formatted string literals) and the format() method.

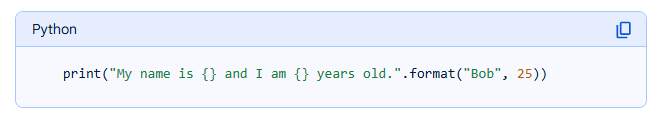
1. F-strings (Formatted String Literals):

F-strings, introduced in Python 3.6, are a concise and readable way to embed expressions inside string literals.



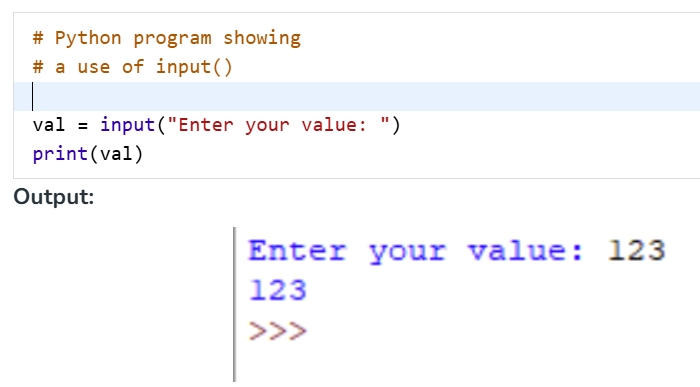
1. The format() Method:

The format() method, available on string objects, provides a flexible way to format strings using positional or named arguments.



**Q3) Using the input() function to read user input from the keyboard.**

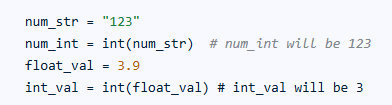
**ANS)** The input() function pauses program execution to allow you to type in a line of input from the keyboard.

****

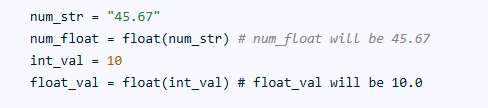
**Q4) Converting user input into different data types (e.g., int, float, etc.).**

**ANS)** Converting user input into different data types is a common necessity in programming, as user input is often initially received as a string. This conversion, known as type casting or explicit type conversion, allows the data to be used in calculations or operations specific to its intended type.

1. **int():** Converts a value to an integer. This function truncates decimal values when converting from floats and can convert numeric strings.



1. **float():** Converts a value to a floating-point number. This is used for numbers with decimal places.



1. **str():** Converts a value to a string. This is useful when displaying non-string data or concatenating it with other strings.



**Q5) Opening files in different modes ('r', 'w', 'a', 'r+', 'w+').**

**ANS)**

* ‘r’ : Read-only. Raises I/O error if file doesn't exist.



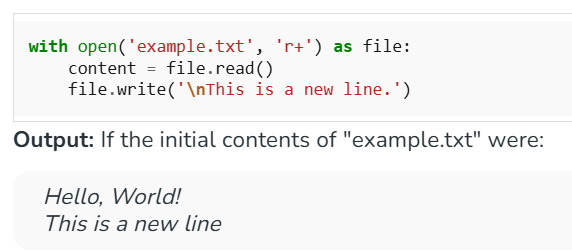
* ‘w’ : Opens the file for writing only. If the file exists, its content is deleted. If not, a new file is created.



* ‘a’ : Opens the file to add content at the end without deleting existing data. If the file doesn’t exist, it creates a new one.



* ‘r+’ : Opens the file for both reading and writing. Starts at the beginning of the file. Raises FileNotFoundError if the file doesn’t exist.



* ‘w+’ : This mode allows you to open a file for both reading and writing. If the file already exists, it will truncate the file to zero length. If the file does not exist, it will create a new file.



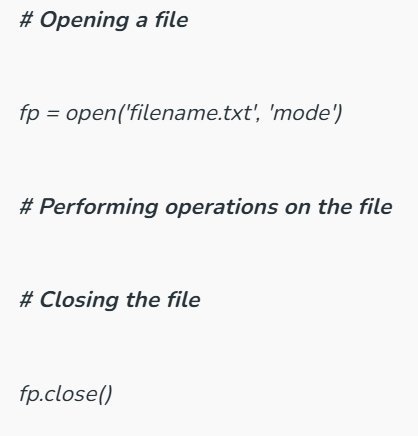
**Q6) Using the open() function to create and access files.**

**ANS)** Opening a file refers to getting the file ready either for reading or for writing. This can be done using the open() function. This function returns a file object and takes two arguments, one that accepts the file name and another that accepts the mode(Access Mode).



**Q7) Closing files using close().**

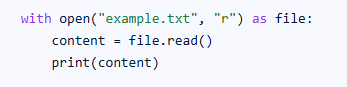
**ANS)** Closing a file object in Python is crucial to prevent resource leaks and ensure that changes are saved. Always use the close() method in conjunction with a try-finally block to guarantee proper closure, whether you are reading from or writing to a file.



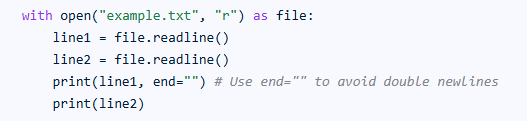
**Q8) Reading from a file using read(), readline(), readlines().**

**ANS)** When working with files in Python, the read(), readline(), and readlines() methods offer different ways to access the content:

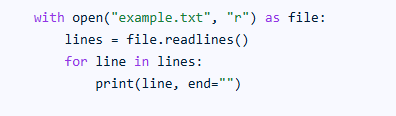
* read(size=-1): This method reads the entire content of the file and returns it as a single string. If an optional size argument is provided, it reads up to that many bytes from the file. If size is omitted or set to -1, the entire file is read.



* readline(size=-1): This method reads a single line from the file at a time and returns it as a string. The returned string includes the newline character (\n) if present at the end of the line.



* **readlines():** This method reads all the lines from the file and returns them as a list of strings, where each element in the list represents a single line from the file, including the newline characters.

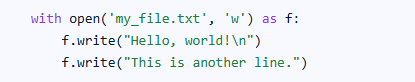


**Q9) Writing to a file using write() and writelines().**

**ANS**) In Python, the write() and writelines() methods are used to write content to a file. Both methods operate on file objects opened in a write-compatible mode (e.g., 'w', 'a', 'w+').

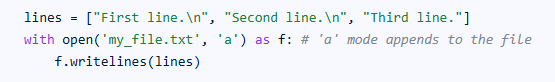
* **write() method:**

The write() method is used to write a single string to a file**.**

****

* **writelines() method:**

The writelines() method is used to write multiple strings from an iterable (like a list or tuple of strings) to a file.



**Q10) Introduction to exceptions and how to handle them using try, except, and finally.**

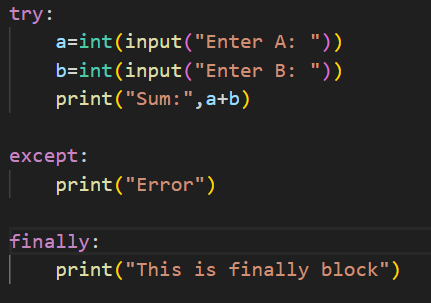
**ANS**)

**Try:** This block will test the excepted error to occur

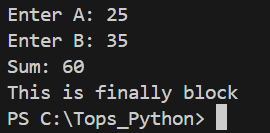
**Except:** Here you can handle the error

**Else:** If there is no exception then this block will be executed

**Finally:** Finally block always gets executed either exception is generated or not.



OUTPUT:



**Q11) Understanding multiple exceptions and custom exceptions.**

**ANS)** Multiple exceptions are handled by using multiple catch blocks, each designed to handle a specific exception type. Custom exceptions, which are user-defined exceptions, allow you to create specific exception classes tailored to your application's needs, improving error handling and code clarity.

* Handling Multiple Exceptions:

In programming, especially in languages like Java or Python, you can encounter different types of errors during the execution of your code. To handle these effectively, you use try...catch blocks. The try block encloses the code that might raise an exception, and the catch blocks handle specific exception types.

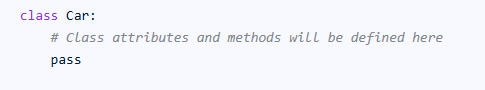
* Custom Exceptions:

Custom exceptions, also known as user-defined exceptions, are created by extending the built-in Exception class (or RuntimeException for unchecked exceptions). This allows you to define your own exception types with specific meanings relevant to your application.

**Q12) Understanding the concepts of classes, objects, attributes, and methods in Python.**

**ANS)** In Python's object-oriented programming, the fundamental concepts are:

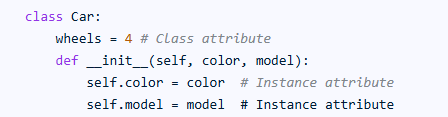
1. **Class:** A class serves as a blueprint or a template for creating objects. It defines the structure and behavior that objects of that class will possess. Think of it as a design document that specifies what attributes (data) and methods (functions) an object will have. Classes are defined using the class keyword.



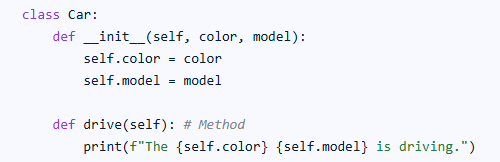
1. **Object:** An object is an instance of a class. It is a concrete realization of the blueprint defined by the class. When you create an object, you are creating a specific entity that adheres to the class's definition. Objects are created by calling the class name as if it were a function.



1. **Attribute:** Attributes are variables defined within a class that store data associated with an object.

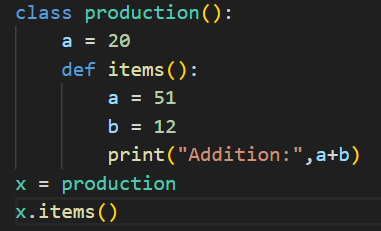


1. **Method:** Methods are functions defined within a class that define the behaviors or actions an object can perform. They operate on the object's attributes and can also accept arguments. Methods are defined within the class using the def keyword, similar to regular functions, but they always take self as their first argument, which refers to the instance of the object itself.



**Q13) Difference between local and global variables.**

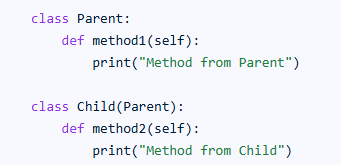
**ANS)** Local and global variables differ in their scope, which determines where they can be accessed within a program. Global variables are declared outside any function and can be accessed from anywhere in the code. Local variables, on the other hand, are declared inside a specific function or block of code and are only accessible within that function or block.



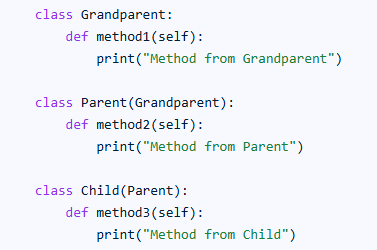
**Q14) Single, Multilevel, Multiple, Hierarchical, and Hybrid inheritance in Python.**

**ANS)** Inheritance in Python allows a class (child/derived class) to inherit attributes and methods from another class (parent/base class), promoting code reusability and establishing a hierarchical relationship between classes. The main types of inheritance in Python are:

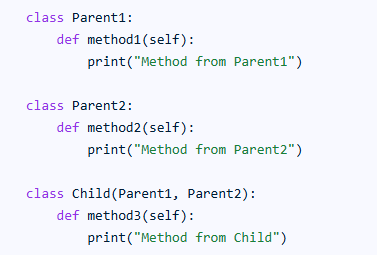
* **Single Inheritance:** A child class inherits from only one parent class. This is the simplest form of inheritance.



* **Multilevel Inheritance:** A class inherits from a class, which in turn inherits from another class, forming a chain.

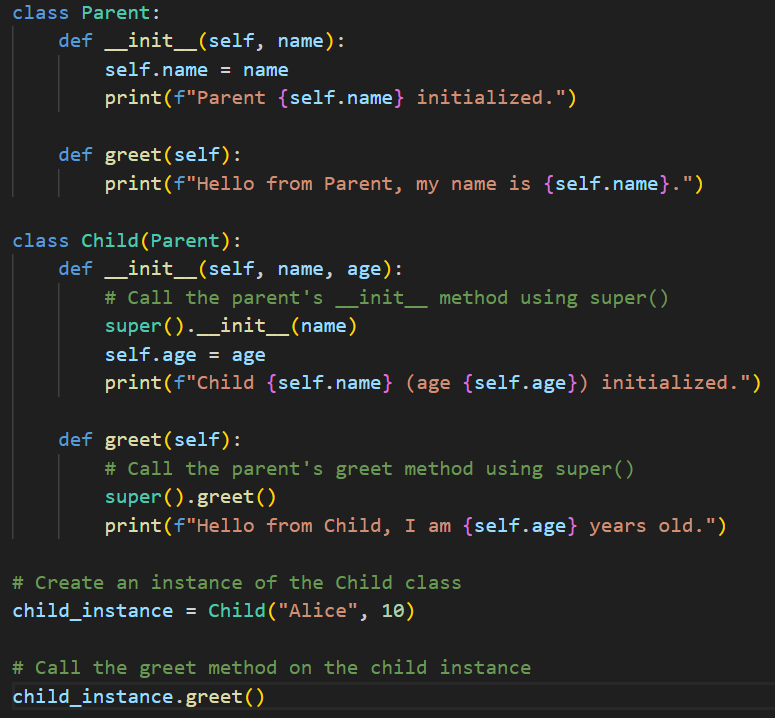


* **Multiple Inheritance:** A child class inherits from more than one parent class, combining functionalities from multiple sources.



**Q15) Using the super() function to access properties of the parent class.**

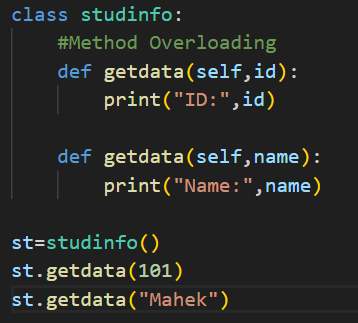
**ANS)** The super() function in Python provides a way to access methods and properties of a parent or superclass from a child or subclass. This is particularly useful in object-oriented programming with inheritance, enabling subclasses to leverage or extend functionality defined in their parent classes.

****

**Q16) Method overloading: defining multiple methods with the same name but different**

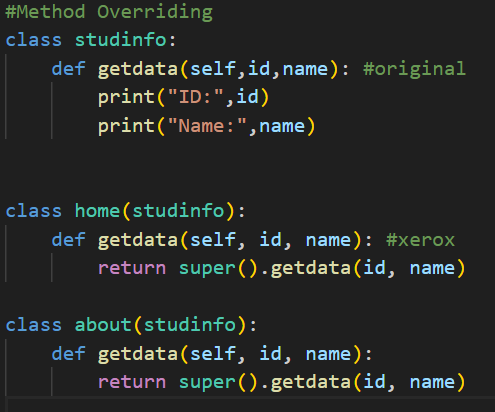
**parameters.**

**ANS)** Method overloading is a feature in object-oriented programming that allows a class to have multiple methods with the same name, but with different parameter lists. This is a form of compile-time polymorphism, also known as static polymorphism or early binding.



**Q17) Method overriding: redefining a parent class method in the child class.**

**ANS)** Method overriding is a fundamental concept in object-oriented programming (OOP) where a subclass (child class) provides its own specific implementation for a method that is already defined in its superclass (parent class). This allows for polymorphism, meaning that objects of different classes can be treated as objects of a common type, while still exhibiting their unique behaviors**.**

****

**Q18) Introduction to SQLite3 and PyMySQL for database connectivity.**

**ANS**) SQLite is a lightweight, serverless, self-contained, and highly reliable SQL database engine. It is widely used due to its simplicity, ease of setup and zero-configuration nature.

Because SQLite is an embedded database, you actually don't need to 'download' it in the same way that you would download MySQL or PostgreSQL for example.

PyMySQL:

* MySQL is an open-source Relational Database Management System (RDMS), which uses Structured Query Language (SQL) to manage and use data in the database. It has various features such as:
* Data storage and retrieval using the tables and rows.
* Indexing and searching capabilities.
* User authentication and access control.
* It supports various data types, including strings, numbers, etc.

**Q19) Creating and executing SQL queries from Python using these connectors.**

**ANS**) Install the Connector. For MySQL: Use pip install mysql-connector-python .

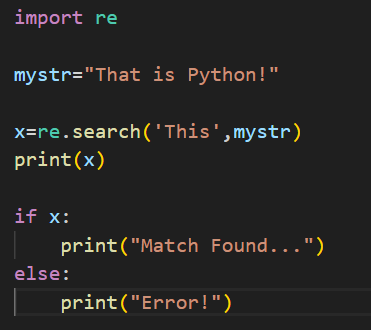
Set Up the Database. Ensure the database server is running and accessible.

Import the Connector Library. Use the appropriate Python module to work with the database, such as mysql.

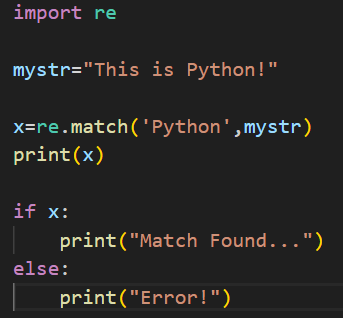
**Q20) Using re.search() and re.match() functions in Python’s re module for pattern matching.**

**ANS)** Python's re module provides re.search() and re.match() functions for pattern matching using regular expressions. While both functions aim to find a match for a given pattern within a string, they differ in their search scope.

* **Re.search():**

****

* **re.match():**

****

**Q21) Difference between search and match.**

**ANS)** When working with regular expressions (regex) in Python, re.search() and re.match() are two commonly used methods for pattern matching. Both are part of the re module but function differently. The key difference is that re.match() checks for a match only at the beginning of the string, while re.search() searches the entire string for the pattern.