1. What is the difference between list and tuple in Python?

List	Tuple
Lists are mutable, We can modify their elements, add new elements, or remove existing elements after the list is created. We can use methods like `append()`, `insert()`, and `remove()` to modify a list.	Tuples are immutable. Once a tuple is created, you cannot change, add, or remove elements. This immutability provides some level of data integrity.
Lists are created using square brackets `[ ]`. For example: `my_list = [1, 2, 3]`.	Tuples are created using parentheses `()`. For example: `my_tuple = (1, 2, 3)`.
Due to their mutability, lists may consume more memory and have a slightly slower performance compared to tuples.	Tuples are generally faster and use less memory since they are immutable.
Use lists when you have a collection of items that might need to be modified or updated. Lists are suitable for situations where you want to perform operations like adding, removing, or modifying elements.	Use tuples when you have a collection of items that should remain constant throughout the program. Tuples are suitable for situations where immutability and the integrity of data are important.

2. Explain the concept of PEP 8.

PEP 8 is the official style guide for Python code. "PEP" stands for Python Enhancement Proposal, and PEP 8 is one of the many PEPs that provide information to the Python community or describe a new feature for Python. PEP 8 focuses specifically on the style and conventions for writing readable and maintainable Python code.

Key principles and recommendations outlined in PEP 8 include:

1. Indentation:
   - Use 4 spaces per indentation level.
   - Avoid tabs for indentation.

2. Maximum Line Length:
   - Limit all lines to a maximum of 79 characters for code and 72 for docstrings and comments.

3. Imports:
   - Imports should usually be on separate lines.
   - Imports should be grouped in the following order:
     1. Standard library imports.

2. Related third-party imports.
3. Local application/library specific imports.
- Within each group, imports should be sorted in alphabetical order.

4. Whitespace in Expressions and Statements:
- Avoid extraneous whitespace in the following situations:
  - Immediately inside parentheses, brackets, or braces.
  - Immediately before a comma, semicolon, or colon.
  - Immediately before the open parenthesis that starts an argument list.

5. Comments:
- Comments should be complete sentences and should be used sparingly.
- Inline comments should be separated by at least two spaces from the statement.

6. Naming Conventions:
- Variables, functions, and methods should be lowercase with words separated by underscores (`snake_case`).
- Constants should be uppercase with words separated by underscores (`UPPER_CASE`).
- Classes should be named using `CapWords` convention.

7. Function and Method Arguments:
- Function arguments should be separated by spaces after the commas.

8. Programming Recommendations:
- Use blank lines to separate functions, classes, and blocks of code inside functions.
- Avoid extraneous whitespace at the end of lines.
- Avoid using wildcard imports (`from module import *`).

3. What is the purpose of the __init__ method in Python classes?

The **__init__** method in Python is a special method, often called the "initializer" or "constructor," and it is automatically called when an object is created from a class. The primary purpose of the **__init__** method is to initialize the attributes or properties of an object.

4. How does inheritance work in Python? Provide an example.
   Inheritance is a fundamental concept in object-oriented programming that allows a class (subclass/derived class) to inherit attributes and behaviors from another class (superclass/base class). In Python, inheritance is implemented using the class definition, and it allows you to create a new class that is a modified version of an existing class.

   Python code in Google colab

5. Explain the difference between staticmethod and classmethod.

staticmethod:

A static method is a method that belongs to a class rather than an instance of the class.

It does not have access to the instance or class itself (i.e., it doesn't have the `self` or `cls` parameter).

It is defined using the `@staticmethod` decorator.

Static methods are often used for utility functions that don't depend on the state of the instance or class.

classmethod:

A class method is a method that takes the class itself as its first parameter (`cls`).

It can access and modify class-level attributes, but it doesn't have access to the instance-specific attributes.

It is defined using the `@classmethod` decorator.

Class methods are often used as alternative constructors or for operations that involve the class itself.

Python code in colab

6. What is Polymorphism in Python? Give an example.

Polymorphism is a fundamental concept in object-oriented programming that refers to the ability of objects of different classes to be treated as objects of a common base class. It allows a single interface (method or operator) to represent different types of objects. There are two main types of polymorphism: compile-time (or static) polymorphism and runtime (or dynamic) polymorphism.

Python code in colab

7. How do you handle exceptions in Python?

In Python, exceptions are used to handle runtime errors or unexpected situations that may occur during the execution of a program. Exception handling is a mechanism that allows you to gracefully manage these errors, preventing the program from crashing and providing a way to recover or handle the exceptional situation.

The **try**, **except**, **else**, and **finally** blocks are used for exception handling in Python.

Python code in colab

8. Explain the Global Interpreter Lock (GIL) in Python.

   The Global Interpreter Lock (GIL) is a mechanism in CPython, the reference implementation of Python, that ensures only one thread executes Python bytecode at a time within a single process. The purpose of the GIL is to simplify memory management and avoid certain race conditions in CPython's memory management implementation. While the GIL has benefits, it also introduces limitations, especially in scenarios involving multi-core processors and CPU-bound tasks

9. What is a decorator in Python? Provide an example.

   In Python, a decorator is a design pattern that allows you to extend or modify the behavior of callable objects (functions or methods) without altering their source code. Decorators are applied using the **@decorator** syntax and are commonly used for tasks such as logging, timing, access control, or modifying the behavior of functions.

   A decorator is a higher-order function that takes a function as input, typically wraps it with additional functionality, and returns a new function. The **@decorator** syntax is a shorthand way of applying a decorator to a function.

   Python code in colab

10. How do you implement encapsulation in Python?

    Encapsulation is one of the fundamental principles of object-oriented programming (OOP), and it refers to the bundling of data (attributes) and methods that operate on the data within a single unit known as a class. In Python, encapsulation is achieved through the use of private and protected access modifiers and property methods.

    Python code in colab

11. Explain the concept of duck typing.

    Duck typing is a programming concept in which the type or class of an object is determined by its behavior (methods and properties) rather than its explicit inheritance or type declarations. The term "duck typing" comes from the saying, "If it looks like a duck, swims like a duck, and quacks like a duck, then it probably is a duck." In the context of programming, it means that the type or class of an object is determined by its behavior rather than its explicit type.

    Python code in colab

12. What is the difference between append() and extend() methods for lists?

append() Method:

- The **append()** method is used to add a single element to the end of a list.
- It takes one argument, which is the element to be added.
- The element is added as a single item, whether it's a single value, a variable, or another list.

extend() Method:

- The **extend()** method is used to add elements from an iterable (e.g., a list, tuple, or string) to the end of a list.
- It takes one argument, which is the iterable whose elements will be added.
- The elements of the iterable are added individually to the list

13. How does the with statement work in Python?

    The **with** statement in Python is used for simplifying resource management, particularly in cases where resources need to be acquired and released properly, such as file handling or working with database connections. The primary purpose of the **with** statement is to ensure that certain operations are performed before and after a block of code, guaranteeing that resources are properly managed and released, even if an exception occurs.

14. Discuss the use of self in Python classes.

    In Python, **self** is a conventional name used to represent the instance of a class within the class itself. It is the first parameter in the definition of a method within a class and refers to the instance of that class. The use of **self** is a way for class methods to access and manipulate the instance variables and other methods of the class.
    Python code in colab

15. Explain the purpose of the __slots__ attribute.

    The **__slots__** attribute in Python is used to explicitly declare the attributes (instance variables) that a class can have. It allows you to restrict the set of attributes that can be assigned to instances of a class, providing some benefits in terms of memory efficiency and preventing accidental creation of new attributes.

16. What is the difference between an instance variable and a class variable?

- **Scope:**

  Instance Variable:

- Instance variables are associated with instances (objects) of a class.
- Each instance of the class has its own copy of instance variables.
- Changes to an instance variable of one object do not affect the same variable in other objects.

Class Variable:

- Class variables are associated with the class itself rather than instances.
- There is only one copy of a class variable shared among all instances of the class.
- Changes to a class variable are reflected in all instances of the class.
- **Declaration:**

Instance Variable:

- Instance variables are declared inside methods, typically within the **__init__** method.
- They are prefixed with **self** to indicate that they belong to the instance.

Class Variable:

- Class variables are declared outside of any method, typically at the top level of the class.
- They are shared by all instances and are not prefixed with **self**.
- **Initialization:**

Instance Variable:

- Instance variables are initialized within the **__init__** method and can have different values for each instance.

Class Variable:

- Class variables are initialized outside of methods and are typically set directly within the class body.
- **Access:**

Instance Variable:

- Instance variables are accessed using the **self** keyword within instance methods.
- They can also be accessed using the instance itself, but it's a convention to use **self** for clarity.

Class Variable:

- Class variables are accessed using the class name, not through instances.
- They can also be accessed within instance methods using the class name.

17. How do you implement Encapsulation, Abstraction, Polymorphism?

Encapsulation : Encapsulation is the concept of bundling the data (attributes) and the methods (functions) that operate on the data into a single unit known as a class. This helps in controlling access to the data and prevents unintended external interference.

Abstraction : Abstraction involves simplifying complex systems by modeling classes based on the essential properties and behaviors they share, while hiding unnecessary details.

Polymorphsim : Polymorphism allows objects of different types to be treated as objects of a common type. It provides a way to use a single interface for different types of objects.

Python code in colab