1. What is Object-Oriented Programming (OOP)?

**Object-Oriented Programming (OOP)** is a method of programming centered around objects and their interactions. It models complex systems by describing a collection of interacting objects through their data and behavior. Python is an object-oriented language that uses objects and classes to design and build applications.

**Object-Oriented Programming (OOP)** is like designing a real-world model in code. Imagine you’re creating a game. You have players, enemies, and weapons. Each of them has attributes (name, strength) and actions (attack, move, defend). OOP lets you code them as objects that work together.

2. What is the difference between procedural programming and OOP?

Procedural based programming is derived from structural programming based on the concepts of

functions/procedure/routines. It is easy to access and change the data in procedural oriented programming. On the other hand, Object Oriented Programming (OOP) allows decomposition of a problem into several units called objects and then build the data and functions around these objects. Its emphasis more on the data than procedure or functions. Also, in OOP, data is hidden and cannot be accessed by external procedure.

Programming Paradigms:

1. **Procedural Programming** - A programming paradigm based on the concept of procedure calls, where programs are structured as a sequence of instructions grouped into procedures (also known as functions, subroutines, or methods).

Key Characteristics:

1. **Top-down approach** – Problem-solving starts from the highest-level tasks and breaks them down into smaller subtasks.

2. **Sequence, Selection, Iteration** – Control structures form the backbone:

Sequence: **Execution step by step.**

Selection: **If–Else, Switch statements**.

Iteration**: Loops (for, while, do-while).**

3**. Global and local variables** – Variables can be declared globally or within a function’s scope.

4. **Modularity** – Code is divided into procedures for reusability and better organization.

Advantages:

• Easy to read and understand for small programs.

• Straightforward to implement.

• Good for tasks with a clear sequence of steps.

Limitations:

• Difficult to maintain for large projects (code can become “spaghetti”).

• Poor modeling for real-world entities.

• Code reuse limited compared to OOP.

2**. Functional Programming (FP)** - A paradigm where computation is treated as the evaluation of mathematical functions and avoids changing state and mutable data. Programs are built by composing pure functions.

Key Characteristics:

1. **Pure functions** – Functions that always produce the same output for the same input and have no side effects.

2**. Immutability** – Data cannot be modified once created; new data is created instead.

3**. First-class and higher-order functions** – Functions are treated as values and can be passed as arguments or returned by other functions.

4. **Declarative style** – Focuses on what to do rather than how to do it.

5. **Recursion over loops –** Often uses recursion for iteration instead of mutable loops.

**Advantages:**

• Easier to debug and test due to pure functions. • Supports concurrency and parallelism.

• Predictable behavior due to immutability.

**Limitations:**

• May have a steeper learning curve. • Sometimes less efficient in terms of memory.

•Not always intuitive for problems that map better to

3. **OBJECT-ORIENTED PYTHON (OOP)**

What is Object-Oriented Programming in Python?

**Object-Oriented Programming (OOP) in Python** is a programming style that uses objects and classes to structure code. It models real-world entities through concepts like inheritance, polymorphism, and encapsulation. The main idea is to bind data and the functions that operate on that data into a single unit, so no other part of the code can directly access it. This approach helps in organizing and protecting data within your programs.

Why Use OOP in Python?

• **Clean Code**: OOP helps you keep your code as tidy as a librarian's bookshelf.

• **Scalability:** It lets you build programs as big as a skyscraper without losing your mind.

• **Reusability**: You can reuse your code like your favorite pair of socks (but way less smelly).

• **Maintenance**: It makes fixing bugs as easy as pie (and who doesn't love pie?).

**Four Pillars of OOP**

1. **Encapsulation**

• Bundles related data (attributes) and methods (functions) into a single unit (class).

• Restricts direct access to some of an object's components, often using private/protected attributes.

2. **Abstraction**

• Hides complex internal details and shows only essential features to users.

• Achieved in Python with abstract classes and methods.

3. **Inheritance**

• Enables one class (child/subclass) to inherit attributes and methods from another class (parent/superclass).

• Promotes code reusability and hierarchical relationships.

4**. Polymorphism**

• Allows objects of different classes to be treated as objects of a common superclass.

• Supports method overriding and the ability for different classes to respond to the same method call in different ways.

**OOP Concepts in Python**

1. **Class** **– The Blueprint**

A **class** is like a template or drawing of a toy. It tells us what the toy should look like and what it can do.

Example: A class called Student can tell us that all students have a name, and age.

A close-up of a computer code

AI-generated content may be incorrect.

1. **Object – the real thing**
2. **Attributes** – The Features

These are like labels or traits on your toy. A student has a **name or age**. In programming, we call them attributes.

1. A computer code with text

   AI-generated content may be incorrect.A white background with black text

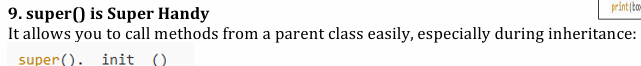
   AI-generated content may be incorrect.A screenshot of a computer code

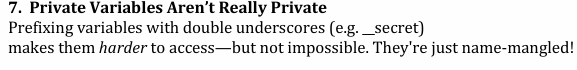
   AI-generated content may be incorrect.A computer code with text

   AI-generated content may be incorrect.**Methods** – The Actions These are like buttons on the toy. When you press them, the toy does

A close up of a website

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

A screen shot of a computer code

AI-generated content may be incorrect.

**SAS WEEK 2 – CLASSES AND OBJECT**

**A screenshot of a computer

AI-generated content may be incorrect.**A **class** is a blueprint for creating objects. It defines a logical entity that bundles together attributes (state) and methods (behavior).

A screen shot of a computer program

AI-generated content may be incorrect.

**Class attributes** are properties shared by all objects of a class and are accessed using the class name, while instance attributes are unique to each object and are typically defined within the \_\_init\_\_ method.

A white paper with black text

AI-generated content may be incorrect.A white paper with black text

AI-generated content may be incorrect.

A close-up of a document

AI-generated content may be incorrect.

A close-up of a program

AI-generated content may be incorrect.

An **object** (or instance) is a particular implementation of a class; it holds its own data and can use the class’s methods. • Objects are created by calling the class as if it were a function.

A close-up of a document

AI-generated content may be incorrect.

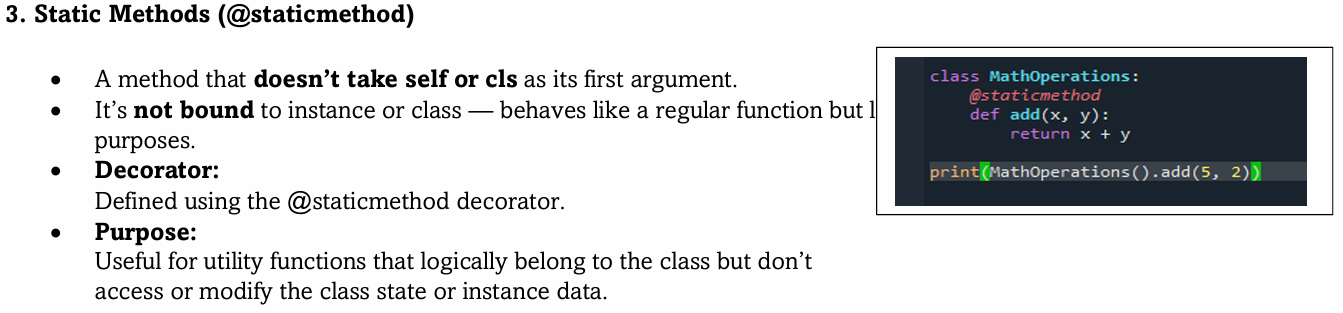
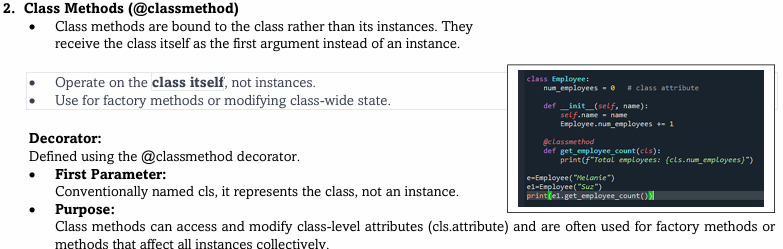
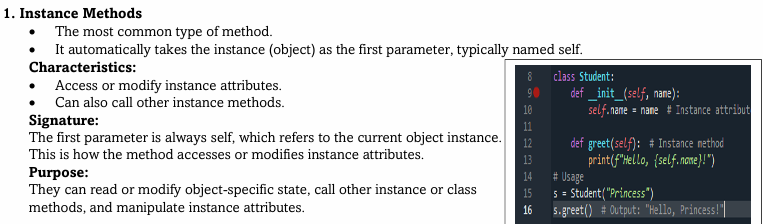
A close-up of a text

AI-generated content may be incorrect.A computer screen shot of a program code

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

In Python, methods are functions defined inside a class that describe the behaviors of the objects created from that class. There are three primary types of methods that differ based on whether they operate on an instance, the class itself, or neither: 1. Instance Methods 2. Class Methods (@classmethod) 3. Static Methods (@staticmet

hod)