1. What is Object-Oriented Programming, and how does it differ from procedural programming?

1. Object-Oriented Programming (OOP): This paradigm uses "objects" – data structures consisting of data fields and methods together.

2. Procedural Programming: This style of programming is centered around procedures or functions. It involves writing a list of instructions to tell the computer what to do step by step.

3. Difference: OOP is more about modeling complex items and relationships using objects, while procedural programming is about writing a sequence of steps to complete a task.

2. Explain the principles of OOP and how they are implemented in Python. Describe the concepts of encapsulation, inheritance, and polymorphism in Python.

Principles of OOP in Python:

1. Encapsulation: Bundling data with methods that operate on the data. In Python, this is done by creating classes.

2. Inheritance: A way to form new classes using classes that have already been defined. Python supports inheritance, allowing multiple base classes.

3. Polymorphism: The ability to present the same interface for differing underlying data types. In Python, it's achieved by using inheritance and method overriding.

3. What is the purpose of the self keyword in Python class methods?

self represents the instance of the class and is used to access the attributes and methods of the class in Python.

4. How does method overriding work in Python, and why is it useful?

This occurs when a method in a subclass has the same name, parameters, and return type as a method in its superclass. It's used to provide specific implementation of a method already defined in its superclass.

5. What is the difference between class and instance variables in Python?

Class Variables: Shared across all instances of the class. They have the same value for every instance.
Instance Variables: Unique to each instance. They represent the data of an individual object.

6. Discuss the concept of abstract classes and how they are implemented in Python.

An abstract class is a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the abstract class. In Python, they are created using the abc module.

7. Explain the importance of the super() function in Python inheritance.

super() is used to give access to methods of a parent class. It returns a temporary object of the superclass that allows you to call its methods.

8. How does Python support multiple inheritance, and what challenges can arise from it?

Python supports multiple inheritance, where a class can be derived from more than one base class. Challenges include complexity and the "diamond problem" where the hierarchy structure forms a diamond shape.

9. What is a decorator in Python, and how can it be used in the context of OOP?

A decorator in Python is a function that adds functionality to an existing function or method without changing its structure. In OOP, it's used to add functionality to methods and classes.

10. What do you understand about Descriptive Statistics? Explain by Example.

It involves summarizing and organizing the data so it can be easily understood. Common examples include mean, median, mode, range, and standard deviation. For instance, summarizing the performance of students in a class using their test scores.

11. What do you understand by Inferential Statistics? Explain by Example

It involves making predictions or inferences about a population based on a sample of data taken from it. For example, using a sample of voter opinions to predict the outcome of an election.