1. What is OOP, and why is it important in Python?

Answer:

OOP, or Object-Oriented Programming, is a programming paradigm used in many languages, including Python. In OOP, you can model real-world concepts using classes and objects. You can break down complex problems into smaller, more manageable parts. This makes the code reusable, maintainable, and scalable.

In Python, OOP is vital because it allows you to write more structured and efficient code. OOP in Python supports code reusability through inheritance. Using OOP in Python can make your development process more efficient and your code more readable.

2. What is a class in Python?

Answer:

A class in Python is like a blueprint for creating objects. A class defines properties and methods that are common to all objects created from it.

In Python, you use the *class* keyword to define a class.

Classes help organize code by grouping related attributes and functions, promoting reusability and modularity. For instance, if you’re creating a program to manage a library, you might have classes for books, authors, and borrowers, each with its specific attributes and methods.

3. What is an object in Python?

Answer:

An object in Python is an instance of a class. You can think of it as a specific realization of the blueprint provided by the class.

An object contains data in the form of attributes and code in the form of methods. When you create an object, you are essentially creating a variable that has all the properties and behaviors defined in the class.

For example, if you have a class for cars, an object could represent a specific car, such as a Honda Civic, with attributes like color and speed, and methods like start and stop.

4. How do you create an instance of a class in Python?

Answer:

In Python, you can create a class instance by calling the class as a function and providing necessary arguments. This process is known as instantiation. Once you create an instance, you can access the attributes and methods defined in the class using the instance.

For example, if you have a class named *Dog*, you could create an instance of that class by writing *my\_dog = Dog()*. Now, *my\_dog* is an object of the *Dog* class, and you can access its attributes and methods through the *my\_dog* variable. This allows you to represent specific real-world entities in your code.

5. What is the constructor in a Python class?

Answer:

In Python, a constructor is a special method used to initialize objects. You can think of it as blueprint instructions for creating objects.

When you create an instance of a class, the constructor method, *\_\_init\_\_*, is automatically called. It allows you to set up properties or execute any setup code needed for the object.

If you don’t define a constructor in your class, Python provides a default one. However, defining your own constructor gives you more control over the object initialization process.

6. What is a decorator in Python? How is it related to OOP?

Answer:

A decorator in Python is a function that you can use to modify or extend the behavior of other functions or methods without changing their code. In the context of OOP, decorators can be used to add functionality to methods within classes, such as logging or access control.

You can think of decorators as wrappers around a function or method. When you call a decorated function, the decorator’s code runs first, allowing you to perform actions before or after the original function. It helps keep the code clean and adhere to the OOP reusability principle.

7. What is the purpose of the *self* keyword in methods?

Answer:

The *self* keyword in Python is used within a class to refer to the instance of the object itself. When you define a method inside a class, the first parameter is usually named *self*. It helps you access the attributes and methods of the class within the current object’s context.

For example, you can use *self.name* to refer to the object’s name attribute.

When calling a method, you don’t have to pass the *self* argument; Python automatically sends the current object reference to the method.

8. How is inheritance implemented in Python?

Answer:

Inheritance allows a class to inherit attributes and methods from another class. You can create a new class – known as the child class – based on an existing class – known as the parent class – and add new features or modify existing ones.

To implement inheritance in Python, you define the new class and put the name of the parent class in parentheses.

For example, if you want a class *Dog* to inherit from a class *Animal*, you would write *class Dog(Animal):*. This way, the *Dog* class would have access to all the attributes and methods defined in *Animal*, allowing for code reusability and better organization.

9. What is the use of the *super()* function?

Answer:

Python’s *super()* function is used within a class to call a method from a parent class, often within the context of method overriding.

If you have a method in a child class with the same name as a method in the parent class, you can use *super()* to call the parent method within the child method. This is particularly useful when you want to extend or modify the behavior of the parent method in the child class.

By using *super()*, you ensure that your code follows the inheritance hierarchy. It also makes the code more maintainable, as changes in the parent class can be easily propagated to child classes.

10. What is the purpose of the *@property* decorator?

Answer:

In Python, the *@property* decorator allows you to treat a method as a property of the class. By using this, you can create a “getter” method, which enables you to access a class method as though it’s an attribute without needing to write parentheses when you call it. This means you can control how the attribute is accessed without directly exposing it.

The *@property*decorator provides a way to implement encapsulation – a key concept in Object-Oriented Programming (OOP). It helps in managing the attributes of a class and making the code more readable.

11. How can you achieve multiple inheritance in Python?

Answer:

Multiple inheritance in Python means that a class can inherit characteristics and features from more than one parent class. You can achieve this by defining a class with more than one parent class inside parentheses in the class definition. This lets the derived class access attributes and methods from all its parent classes.

While this can make code more flexible and powerful, you should use it cautiously. Multiple inheritance can lead to complex class structures.

12. What is a static method and how is it different from a class method?

Answer:

A static method in Python belongs to a class rather than an instance of the class. You can define it using the *@staticmethod* decorator. It doesn’t require reference to the class or its instance and can be called on the class itself. Unlike regular instance methods, it doesn’t take a *self* parameter.

A class method, on the other hand, is defined with the *@classmethod* decorator and takes a reference to the class itself as its first parameter – usually named “cls.” It can access and modify class-level attributes, while a static method can’t.

A class method is more versatile as subclasses can override it, whereas a static method remains unchanged.

13. How do you implement abstract classes and methods in Python?

Answer:

In Python, you can implement abstract classes and methods using the *ABC* module. You would first import the module and then create a class that inherits from *ABC*, which stands for abstract base class.

Within this class, you can define abstract methods using the *@abstractmethod* decorator. These abstract methods serve as a blueprint for other classes, enforcing that the derived classes must provide a concrete implementation of these methods.

If a derived class does not implement the abstract methods, an instantiation error will occur.

14. How can you prevent method overriding in Python?

Answer:

In Python, method overriding is a common feature that allows a subclass to provide a different implementation of a method defined in its superclass. However, if you want to prevent method overriding, you can do so by defining the method as private.

By prefixing the method’s name with a double underscore (*\_\_*), you make it private to the class, and it cannot be overridden in a subclass.

Though not strictly enforced, this convention signals that the method should not be accessed or overridden outside the class, and it helps maintain the integrity of the original method within the hierarchy.

15. How does polymorphism work in Python?

Answer:

Polymorphism in Python allows different objects to be treated as instances of the same class, even if they belong to different classes. You can achieve this through inheritance, where a subclass can have methods with the same name as the methods in the superclass.

When you call a method on an object, Python will automatically use the correct method based on the object’s class, even if the object is referred to by a variable of the superclass type.

This enables more flexible and reusable code, as you can write functions that work with different classes, provided they adhere to the same interface or method signature.

Links --🡪

1. <https://www.geeksforgeeks.org/quizzes/python-oops-quiz/>
2. <https://www.vskills.in/practice/python-oop-test>