What is class in Python, and how it is used to create objects?

In Python, a **class** is a blueprint for creating objects. It encapsulates data (attributes) and behavior (methods) that objects created from the class will have.

Key Points about Classes:

1. **Definition**: Classes are defined using the class keyword followed by the class name.
2. **Attributes**: Variables defined within a class that hold data.
3. **Methods**: Functions defined within a class that describe the behaviors of an object.

Syntax for Defining a Class:

```python
class ClassName:
    # Constructor: Initializes the object
    def __init__(self, attribute1, attribute2):
        self.attribute1 = attribute1  # Instance variable
        self.attribute2 = attribute2  # Instance variable

    # Method: Defines behavior
    def method_name(self):
        return f"Attribute1 is {self.attribute1} and Attribute2 is {self.attribute2}"

```

Creating Objects from a Class:

An **object** is an instance of a class. Objects are created by calling the class as if it were a function.

Example:

```python
# Define a class
class Car:
    def __init__(self, brand, model):
        self.brand = brand  # Instance attribute
        self.model = model  # Instance attribute

    def get_description(self):
        return f"{self.brand} {self.model}"

# Create an object (instance of the class)
my_car = Car("Toyota", "Corolla")

# Access attributes and call methods
print(my_car.brand)          # Output: Toyota
print(my_car.get_description())  # Output: Toyota Corolla
```

Steps to Use a Class to Create Objects:

1. **Define the class** with a constructor (__init__) and methods.
2. **Create objects** by calling the class with required arguments (if any).
3. **Access or modify attributes** using dot notation (object_name.attribute).
4. **Call methods** using dot notation (object_name.method()).

This is a foundational concept in **Object-Oriented Programming (OOP)**, allowing you to model real-world entities and their behaviors in Python.

What are methods and attributes in Python Classes?

Attributes:

- **Definition**: Variables that store data or properties of a class or object.
- **Types**:
    1. **Instance Attributes**: Specific to each object and defined using self.
    2. **Class Attributes**: Shared across all instances of the class.

Example:

```
class Car:
  wheels = 4  # Class attribute
  def __init__(self, color):
    self.color = color  # Instance attribute
```

Methods:

- **Definition**: Functions defined inside a class that perform actions or behaviors related to the class or its objects.
- **Types**:
    1. **Instance Methods**: Operate on object-specific data; use self.
    2. **Class Methods**: Operate on class-level data; use @classmethod and cls.
    3. **Static Methods**: Independent of class or instance; use @staticmethod.

Example:

```
class Car:
  def start(self):  # Instance method
    print("Car started")
```

What is abstraction in OOPS and how does it simplify complex systems?

Abstraction in OOPS (Object-Oriented Programming):

- **Definition**: Abstraction is the process of **hiding the internal details** of an object and exposing only the relevant functionalities.
- It allows you to interact with an object without needing to understand its internal complexities.

How Abstraction Simplifies Complex Systems:

1. **Focus on Essentials**: Users see only the necessary details, making the system easier to understand.
2. **Reduce Complexity**: Internal logic and implementation details are hidden, so interactions remain simple.
3. **Improve Maintainability**: Changes to internal implementations don't affect external code.
4. **Enhance Reusability**: Abstraction allows code reuse through interfaces and abstract classes.

Example in Python:

```python
from abc import ABC, abstractmethod

# Abstract class
class Animal(ABC):
    @abstractmethod
    def speak(self):
        pass

# Concrete class inheriting from the abstract class
class Dog(Animal):
    def speak(self):
        return "Barking"

# Creating an object
dog = Dog()
print(dog.speak())  # Output: Barking
```

- Here, **Animal is abstract**, hiding implementation details, while **Dog provides concrete behavior**.

Abstraction helps developers work at a **higher level**, reducing cognitive load and improving code clarity and maintenance.