OOP Approach

The programming where we create a template and create copies from that template is called object-oriented programming. This style allows us to code for scenarios closely linked with real life. The template we create is called a Class and the copies we create out of it is called an object.

Object is the instance of the class used to access the properties of the class.

1. **Class**: Think of a class as a blueprint or a template for creating objects. It defines the characteristics (attributes) and behaviours (methods) that an object of that class can have. For example, if we're talking about a "Dog" class, the class would define attributes such as "breed", "age", and "color", and methods such as "bark" and "fetch".
2. **Object**: An object is an instance of a class, which means it's a specific realization of the class. It's like a real-life object that you can touch and interact with. For example, if we create an object of the "Dog" class, it would represent a specific dog with its own breed, age, and color.
3. **Attributes**: Attributes are the characteristics or properties of an object that are defined in the class. They store information about the object. For example, in the "Dog" class, the attributes could be "breed", "age", and "color".
4. **Methods**: Methods are the behaviours or actions that an object of a class can perform. They are defined in the class and can be called on objects of that class. For example, in the "Dog" class, the methods could be "bark", "fetch", and "sleep".
5. **Inheritance**: Inheritance allows a class to inherit attributes and methods from another class. It's like passing on traits from parents to children. For example, you could have a "Pet" class that has attributes such as "name" and "species", and the "Dog" class can inherit these attributes from the "Pet" class, as a dog is a type of pet.
6. **Encapsulation**: Encapsulation is the concept of hiding the internal details of a class from the outside world and only exposing what's necessary. It helps to keep the code organized and prevents unauthorized access to data. For example, you can define attributes as private in a class, which means they can only be accessed within the class itself.
7. **Polymorphism**: Polymorphism allows objects of different classes to be treated as if they were of the same class. It allows for flexibility in code design and promotes code reusability. For example, you could have different classes like "Dog", "Cat", and "Bird" that all have a "speak" method, and you can call the "speak" method on any of these objects, even though they are of different classes.
8. Overriding:

Overriding occurs when a child class provides a new implementation for a method that is already defined in its parent class. The child class “overrides” the method in the parent class with its own implementation.

1. Overloading:

Unlike some other programming languages, Python does not support method overloading in the traditional sense, where multiple methods with the same name but the different parameters can coexist in the same class.

However, we can achieve similar functionality by using default arguments or variable-length arguments.

* + - 1. Default arguments

class Calculator:

def add(self, num1, num2=0):

return num1 + num2

calc = Calculator()

result1 = calc.add(5, 3) # Output: 8

result2 = calc.add(5) # Output: 5

1. Variable-length arguments (Yet to Read)

class Printer:

def print\_values(self, \*args):

for value in args:

print(value)

printer = Printer()

printer.print\_values(1, 2, 3) # Output: 1 2 3

printer.print\_values("a", "b") # Output: a b