Fundamentals of OOP in Python

This page introduces Object-Oriented Programming (OOP) concepts and their implementation in Python. You’ll explore the benefits of OOP and how it structures code for better maintainability and re-usability.

Concept Overview

Topics:

* Introduction to OOP Concepts
* Getting Started with Python Classes and Objects

Learning Objectives

* Understand the core principles of OOP: classes, objects, encapsulation, and abstraction.
* Be able to define classes and create objects in Python.
* Grasp the use of class attributes, instance methods, and the self keyword.

Introduction

Object-Oriented Programming (OOP) is a programming paradigm based on the concept of objects. It allows you to structure your code in a way that models real-world entities, making it easier to manage and maintain large-scale projects. Think of OOP as a way of writing code that mimics how things work in the real world. In real life, we deal with objects all the time—like cars, dogs, or phones. Each object has characteristics (like color, size, or brand) and things it can do (like drive, bark, or make calls). OOP lets us represent these objects and their actions in our code.

Detailed Explanation

Key Concepts in OOP:

**Objects**

Objects are like the things you see and interact with in everyday life (cars, dogs, etc.). Each object has properties (like color, size) and behaviors (like drive, bark).

**Classes**: A class is like a blueprint or template for creating objects. It defines what properties an object will have (attributes) and what actions it can perform (methods).

**Encapsulation**:

Encapsulation means bundling data (attributes) and the methods (functions) that operate on that data together within a class. It helps keep related things together and hides the inner workings of an object from the outside, showing only what’s necessary.

**Inheritance**:

Inheritance is like passing down traits from parent to child. It allows a new class (child) to inherit properties and behaviors from an existing class (parent), promoting code re-usability.

**Polymorphism**: Polymorphism means many forms. It allows objects to take on different forms or behaviors based on the context or the class they belong to.

class Car:

def \_\_init\_\_(self, make, model, year):

self.make = make

self.model = model

self.year = year

self.odometer\_reading = 0 # Default attribute value

def get\_descriptive\_name(self):

full\_name = f"{self.year} {self.make} {self.model}"

return full\_name

def read\_odometer(self):

print(f"This car has {self.odometer\_reading} miles on it.")

def update\_odometer(self, mileage):

if mileage >= self.odometer\_reading:

self.odometer\_reading = mileage

else:

print("You can't roll back an odometer!")

def increment\_odometer(self, miles):

self.odometer\_reading += miles

**In this Car class**:

* We have an \_\_init\_\_ method (constructor) that initializes attributes like make, model,year, and odometer\_reading.
* get\_descriptive\_name method returns a formatted string representing the car’s name.
* read\_odometer method prints the car’s mileage.
* update\_odometer method updates the odometer reading with a new value (if it’s higher). increment\_odometer method increments the odometer reading by a specified amount.

You can create object (instances) of the Car class and access its methods and attributes as follows:

my\_car = Car("Toyota", "Camry", 2020)

print(my\_car.get\_descriptive\_name()) # Output: 2020 Toyota Camry

my\_car.read\_odometer() # Output: This car has 0 miles on it.

my\_car.update\_odometer(100) # Update odometer reading

my\_car.read\_odometer() # Output: This car has 100 miles on it.

my\_car.increment\_odometer(50) # Increment odometer reading

my\_car.read\_odometer() # Output: This car has 150 miles on it.

This Car class demonstrates basic attributes and methods commonly associated with a car object, such as its make, model, year, and odometer reading

Practice Exercises

**Exercise 1**: Creating a Student Class

Instructions:

Define a Student class with attributes like name and age. Include a method to display student information.

**Exercise 2**: Creating a Product Catalog

Instruction:

Define a Product class with attributes like name, price, and quantity. Implement a method to calculate the total value of products in stock.