OOP in Python

🧠 **What is OOP (Object-Oriented Programming)?**OOP is a programming paradigm based on the concept of "objects", which can contain:  
- Data (in the form of attributes/properties)  
- Code (in the form of methods/functions)  
  
**Key Pillars of OOP:**1. Class – Blueprint for creating objects.  
2. Object – Instance of a class.  
3. Encapsulation – Hiding internal data, showing only necessary parts.  
4. Abstraction – Hiding complex logic and showing only relevant info.  
5. Inheritance – Reusing code from other classes.  
6. Polymorphism – Same function name behaving differently based on context.  
  
**🧰 Use Cases of OOP in Python**  
- Real-world modeling – Students, Employees, Vehicles, etc.  
- Game Development – Characters, scores, enemies, etc.  
- GUI Applications – Buttons, windows as objects.  
- Web Development (Django/Flask) – Models and Views are classes.  
- Code Reusability – Create base classes, extend when needed.  
- Modularity – Large applications become easy to manage.  
  
**🧪 Syntax and Code Examples**  
**1. Creating a Class & Object**  
class Student:  
 def \_\_init\_\_(self, name, roll):  
 self.name = name  
 self.roll = roll  
 def display(self):  
 print(f"Name: {self.name}, Roll: {self.roll}")  
  
s1 = Student("Pratyush", 101)  
s1.display()  
  
**2. Encapsulation**  
class BankAccount:  
 def \_\_init\_\_(self, balance):  
 self.\_\_balance = balance  
 def deposit(self, amount):  
 self.\_\_balance += amount  
 def get\_balance(self):  
 return self.\_\_balance  
  
**3. Inheritance**  
class Animal:  
 def speak(self):  
 print("Animal speaks")  
class Dog(Animal):  
 def bark(self):  
 print("Dog barks")  
  
**4. Polymorphism**  
class Bird:  
 def fly(self):  
 print("Bird can fly")  
class Ostrich(Bird):  
 def fly(self):  
 print("Ostrich can't fly")  
  
**5. Abstraction using Abstract Base Class:-**

from abc import ABC, abstractmethod  
class Shape(ABC):  
 @abstractmethod  
 def area(self):  
 pass  
  
class Circle(Shape):  
 def \_\_init\_\_(self, radius):  
 self.radius = radius  
 def area(self):  
 return 3.14 \* self.radius \*\* 2

**✅ Top Benefits of OOP in Python (and in general)**  
  
1. Modularity – Organized into classes, easy to debug and maintain.  
2. Reusability – Inheritance allows using existing code.  
3. Scalability – Easy to expand without rewriting.  
4. Encapsulation – Protect sensitive data using private attributes.  
5. Abstraction – Hides complexity, provides a simple interface.  
6. Polymorphism – One function/method works in multiple ways.  
7. Maintainability – Isolate issues and update parts independently.  
8. Improved Productivity – Reusable components save time.  
9. Better Collaboration in Teams – Teams can work on independent modules.  
10. Real-World Mapping – Classes model real-world objects.  
  
  
**PRACTICE QUESTIONS**

Q-1> Write a definition for a class named Circle with attributes center and radius, where center is a Point object and radius is a number. Instantiate a Circle object that represents a circle with its center at (150,100) and radius 75. Write a function named point\_in\_circle that takes a Circle and a Point and returns True if the Point lies in or on the boundary of the circle. Write a function named rect\_in\_circle that takes a Circle and a Rectangle and returns True if the Rectangle lies entirely in or on the boundary of the circle. Write a function named rect\_circle\_overlap that takes a Circle and a Rectangle and returns True if any of the corners of the Rectangle fall inside the Circle. Or as a more challenging version, return True if any part of the Rectangle falls inside the Circle.

SOLUTION:- [Answer](https://thinkpython.com/code/Circle.py)

Q-2> Write a function called draw\_rect that takes a Turtle object and a Rectangle and uses the Turtle to draw the Rectangle. See Chapter 4 for examples using Turtle objects. Write a function called draw\_circle that takes a Turtle and a Circle and draws the Circle

SOLUTION: [Answer](https://thinkpython.com/code/draw.py.)

**🔹 \_\_init\_\_ Method**

**✅ Definition:**

The \_\_init\_\_ method is the **constructor** in Python. It automatically runs when you create a new object from a class and is used to **initialize the object’s attributes**.

**🧠 Syntax:**

class ClassName:

def \_\_init\_\_(self, parameters):

# Initialization code

self.attribute = value

**🔧 Example:**

class Student:

def \_\_init\_\_(self, name, roll):

self.name = name

self.roll = roll

s1 = Student("Pratyush", 101)

print(s1.name) # Output: Pratyush

**📌 Notes:**

* You can think of it as a **setup function** for the object.
* It doesn't return anything (None by default).

**🔹 \_\_str\_\_ Method**

**✅ Definition:**

The \_\_str\_\_ method defines the **string representation of the object** — what should be displayed when the object is printed using print() or str().

**🧠 Syntax:**

class <ClassName>:

def \_\_str\_\_(self):

return "String to display"

**🔧 Example:**

class Student:

def \_\_init\_\_(self, name, roll):

self.name = name

self.roll = roll

def \_\_str\_\_(self):

return f"Student(Name: {self.name}, Roll: {self.roll})"

s1 = Student("Pratyush", 101)

print(s1) # Output: Student(Name: Pratyush, Roll: 101)

**📌 Notes:**

* It helps in debugging and logging by showing meaningful info about the object.
* If \_\_str\_\_ is not defined, print(obj) shows a default object representation like <\_\_main\_\_.Student object at 0x...>.

**QUESTIONS**

Q-1> Download the code from this chapter from <Link: [Code](https://thinkpython.com/code/%20Time2.py)>. Change the attributes of Time to be a single integer representing seconds since mid night. Then modify the methods (and the function int\_to\_time) to work with the new implemen tation. You should not have to modify the test code in main. When you are done, the output should be the same as before.

Solution:- [Answer](https://thinkpython.com/code/Time2_soln.py%20.)

Q-2> This exercise is a cautionary tale about one of the most common, and difficult to f ind, errors in Python. Write a definition for a class named Kangaroo with the following methods:

1. An\_\_init\_\_ method that initializes an attribute named pouch\_contents to an empty list.

2. A method named put\_in\_pouch that takes an object of any type and adds it to pouch\_contents.

3. A \_\_str\_\_ method that returns a string representation of the Kangaroo object and the con tents of the pouch. Test your code by creating two Kangaroo objects, assigning them to variables named kanga and roo, and then adding roo to the contents of kanga’s pouch.

Solution:- [Answer](https://github.com/AllenDowney/ThinkPython2/blob/master/code/BadKangaroo.py)