**1. What is the primary goal of Object-Oriented Programming (OOP)?**

Ans: The primary objective of Object-Oriented Programming (OOP) is to organize and structure code in a way that models real-world entities and their interactions.

**2. What is an object in Python?**

Ans: In Python, an object is an instance of a class, which is a blueprint for creating data structures.

**3. What is a class in Python?**

Ans: In Python, a class is a blueprint for creating objects. It defines a structure (attributes) and behavior (methods) that the objects created from it will have.

**4. What are attributes and methods in a class?**

Ans: In Python classes, attributes and methods are the building blocks that define an object's data and behavior.

**5. What is the difference between class variables and instance variables in Python?**

Ans:

Class Variables: Variables that are shared across all instances of a class.

Instance Variables: Variables that are unique to each object.

**6. What is the purpose of the self parameter in Python class methods?**

Ans:

a) Access Instance Variables

b) Call Other Methods

c) Distinguish Between Instance and Local Variables

**7. For a library management system, you have to design the "Book" class with OOP**

**principles in mind. The “Book” class will have following attributes:**

**a. title: Represents the title of the book.**

**b. author: Represents the author(s) of the book.**

**c. isbn: Represents the ISBN (International Standard Book Number) of the book.**

**d. publication\_year: Represents the year of publication of the book.**

**e. available\_copies: Represents the number of copies available for checkout.**

**The class will also include the following methods:**

**a. check\_out(self): Decrements the available copies by one if there are copies**

**available for checkout.**

**b. return\_book(self): Increments the available copies by one when a book is**

**returned.**

**c. display\_book\_info(self): Displays the information about the book, including its**

**attributes and the number of available copies.**

Ans:

class Book:

def \_\_init\_\_(self, title, author, isbn, publication\_year, available\_copies):

# Instance variables

self.title = title

self.author = author

self.isbn = isbn

self.publication\_year = publication\_year

self.available\_copies = available\_copies

# Method to check out a book

def check\_out(self):

if self.available\_copies > 0:

self.available\_copies -= 1

print(f"Book '{self.title}' checked out successfully. Available copies: {self.available\_copies}")

else:

print(f"Sorry, '{self.title}' is not available for checkout.")

# Method to return a book

def return\_book(self):

self.available\_copies += 1

print(f"Book '{self.title}' returned successfully. Available copies: {self.available\_copies}")

# Method to display book information

def display\_book\_info(self):

print(f"Title: {self.title}")

print(f"Author: {self.author}")

print(f"ISBN: {self.isbn}")

print(f"Publication Year: {self.publication\_year}")

print(f"Available Copies: {self.available\_copies}")

print("-" \* 40)

**8. For a ticket booking system, you have to design the "Ticket" class with OOP**

**principles in mind. The “Ticket” class should have the following attributes:**

**a. ticket\_id: Represents the unique identifier for the ticket.**

**b. event\_name: Represents the name of the event.**

**c. event\_date: Represents the date of the event.**

**d. venue: Represents the venue of the event.**

**e. seat\_number: Represents the seat number associated with the ticket.**

**f. price: Represents the price of the ticket.**

**g. is\_reserved: Represents the reservation status of the ticket.**

**The class also includes the following methods:**

**a. reserve\_ticket(self): Marks the ticket as reserved if it is not already reserved.**

**b. cancel\_reservation(self): Cancels the reservation of the ticket if it is already**

**reserved.**

**c. display\_ticket\_info(self): Displays the information about the ticket, including its**

**attributes and reservation status.**

Ans:

class Ticket:

def \_\_init\_\_(self, ticket\_id, event\_name, event\_date, venue, seat\_number, price):

# Instance variables

self.ticket\_id = ticket\_id

self.event\_name = event\_name

self.event\_date = event\_date

self.venue = venue

self.seat\_number = seat\_number

self.price = price

self.is\_reserved = False # By default, the ticket is not reserved

# Method to reserve the ticket

def reserve\_ticket(self):

if not self.is\_reserved:

self.is\_reserved = True

print(f"Ticket {self.ticket\_id} for '{self.event\_name}' has been successfully reserved.")

else:

print(f"Ticket {self.ticket\_id} is already reserved.")

# Method to cancel the reservation

def cancel\_reservation(self):

if self.is\_reserved:

self.is\_reserved = False

print(f"Reservation for ticket {self.ticket\_id} has been successfully canceled.")

else:

print(f"Ticket {self.ticket\_id} is not reserved yet.")

# Method to display ticket information

def display\_ticket\_info(self):

print(f"Ticket ID: {self.ticket\_id}")

print(f"Event Name: {self.event\_name}")

print(f"Event Date: {self.event\_date}")

print(f"Venue: {self.venue}")

print(f"Seat Number: {self.seat\_number}")

print(f"Price: ${self.price}")

print(f"Reserved: {'Yes' if self.is\_reserved else 'No'}")

print("-" \* 40)

**9. You are creating a shopping cart for an e-commerce website. Using OOP to model**

**the "ShoppingCart" functionality the class should contain following attributes and**

**methods:**

**a. items: Represents the list of items in the shopping cart.**

**The class also includes the following methods:**

**a. add\_item(self, item): Adds an item to the shopping cart by appending it to the**

**list of items.**

**b. remove\_item(self, item): Removes an item from the shopping cart if it exists in**

**the list.**

**c. view\_cart(self): Displays the items currently present in the shopping cart.**

**d. clear\_cart(self): Clears all items from the shopping cart by reassigning an**

**empty list to the items attribute.**

Ans:

class ShoppingCart:

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

# Attribute to store items in the cart

self.items = []

# Method to add an item to the cart

def add\_item(self, item):

self.items.append(item)

print(f"'{item}' has been added to your shopping cart.")

# Method to remove an item from the cart

def remove\_item(self, item):

if item in self.items:

self.items.remove(item)

print(f"'{item}' has been removed from your shopping cart.")

else:

print(f"Item '{item}' not found in your shopping cart.")

# Method to view items in the cart

def view\_cart(self):

if self.items:

print("Items in your shopping cart:")

for idx, item in enumerate(self.items, start=1):

print(f"{idx}. {item}")

else:

print("Your shopping cart is empty.")

# Method to clear all items from the cart

def clear\_cart(self):

self.items = []

print("Your shopping cart has been cleared.")

**10. Imagine a school management system. You have to design the "Student" class using**

**OOP concepts.The “Student” class has the following attributes:**

**a. name: Represents the name of the student.**

**b. age: Represents the age of the student.**

**c. grade: Represents the grade or class of the student.**

**d. student\_id: Represents the unique identifier for the student.**

**e. attendance: Represents the attendance record of the student.**

**The class should also include the following methods:**

**a. update\_attendance(self, date, status): Updates the attendance record of the**

**student for a given date with the provided status (e.g., present or absent).**

**b. get\_attendance(self): Returns the attendance record of the student.**

**c. get\_average\_attendance(self): Calculates and returns the average**

**attendance percentage of the student based on their attendance record.**

Ans:

class Student:

def \_\_init\_\_(self, name, age, grade, student\_id):

# Instance variables

self.name = name

self.age = age

self.grade = grade

self.student\_id = student\_id

self.attendance = {} # Dictionary to store attendance {date: status}

# Method to update attendance

def update\_attendance(self, date, status):

if status.lower() in ['present', 'absent']:

self.attendance[date] = status.lower()

print(f"Attendance for {self.name} on {date} has been marked as '{status}'.")

else:

print("Invalid status. Please use 'present' or 'absent'.")

# Method to get the attendance record

def get\_attendance(self):

if self.attendance:

print(f"Attendance record for {self.name}:")

for date, status in self.attendance.items():

print(f"{date}: {status.capitalize()}")

else:

print(f"No attendance records found for {self.name}.")

# Method to calculate average attendance percentage

def get\_average\_attendance(self):

total\_days = len(self.attendance)

present\_days = sum(1 for status in self.attendance.values() if status == 'present')

if total\_days == 0:

print(f"No attendance records available for {self.name}.")

return 0

average\_attendance = (present\_days / total\_days) \* 100

return average\_attendance