In [ ]:
1. What is the primary goal of Object-Oriented Programming (OOP)?

In [ ]:
Ans:- Primary goal of Object-Oriented Programming (OOP)

OOP's primary goal is to organize and structure code using objects and classes, emphasizing modularity, reusability, and abstraction.

In [ ]:
2. What is an object in Python?

In [ ]:
ANS:- Object in Python

In Python, an object is an instance of a class, representing a real-world entity with attributes (data) and methods (functions).

In [ ]:
3. Class in Python?

A class in Python is a blueprint or template defining the structure and behavior of objects, encapsulating data and methods.

In [ ]:
4. Attributes and methods in a class?

Attributes are data members of a class (e.g., variables), while methods are functions that operate on those attributes.

In [ ]:
5. Class variables vs. instance variables?

Class variables are shared among all instances of a class, while instance variables are unique to each instance.

In [ ]:
6. Purpose of self parameter?

The self parameter refers to the instance of the class, allowing methods to access and modify instance attributes.

In [ ]:
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.

In [38]:
class Book:
    def __init__(self, title, author, isbn, publication_year, available_copies):
        # Initialize the attributes
        self.title = title
        self.author = author
        self.isbn = isbn
        self.publication_year = publication_year
        self.available_copies = available_copies

    def check_out(self):
        # Decrease available copies if there are any left
        if self.available_copies > 0:
            self.available_copies -= 1
            print(f"Book '{self.title}' checked out successfully.")
        else:
            print(f"Sorry, no available copies of '{self.title}' for checkout.")

    def return_book(self):
        # Increase the available copies when a book is returned
        self.available_copies += 1
        print(f"Book '{self.title}' returned successfully.")

    def display_book_info(self):
        # Display information about the book
        print(f"Book 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}")

Example usage:

# Create a book instance
book1 = Book("The Great Gatsby", "F. Scott Fitzgerald", "9780743273565", 1925, 3)

# Display book information
book1.display_book_info()

# Check out the book
book1.check_out()

# Return the book
book1.return_book()

# Display updated book information
book1.display_book_info()

Book Title: The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 9780743273565
Publication Year: 1925
Available Copies: 3
Book 'The Great Gatsby' checked out successfully.
Book 'The Great Gatsby' returned successfully.
Book Title: The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 9780743273565
Publication Year: 1925
Available Copies: 3

In [ ]:
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.

In [42]:
from datetime import datetime

class Ticket:
    def __init__(self, ticket_id, event_name, event_date, venue, seat_number, price):
        # Initialize the attributes
        self.ticket_id = ticket_id
        self.event_name = event_name
        # Store the event date as a datetime object for easier date manipulation
        self.event_date = datetime.strptime(event_date, '%Y-%m-%d') # Format: 'YYYY-MM-DD'
        self.venue = venue
        self.seat_number = seat_number
        self.price = price
        self.is_reserved = False # Initially, the ticket is not reserved

    def reserve_ticket(self):
        # Mark the ticket as reserved if not already reserved
        if not self.is_reserved:
            self.is_reserved = True
            print(f"Ticket {self.ticket_id} for '{self.event_name}' has been reserved successfully.")
        else:
            print(f"Ticket {self.ticket_id} for '{self.event_name}' is already reserved.")

    def cancel_reservation(self):
        # Cancel the reservation if the ticket is reserved
        if self.is_reserved:
            self.is_reserved = False
            print(f"Reservation for ticket {self.ticket_id} has been canceled.")
        else:
            print(f"Ticket {self.ticket_id} for '{self.event_name}' is not reserved.")

    def display_ticket_info(self):
        # Display ticket details including reservation status
        reserved_status = "Reserved" if self.is_reserved else "Not Reserved"
        print(f"Ticket ID: {self.ticket_id}")
        print(f"Event Name: {self.event_name}")
        print(f"Event Date: {self.event_date.strftime('%Y-%m-%d')}")
        print(f"Venue: {self.venue}")
        print(f"Seat Number: {self.seat_number}")
        print(f"Price: ${self.price:.2f}")
        print(f"Reservation Status: {reserved_status}")

Example usage:

# Create a ticket instance
ticket1 = Ticket("T001", "Rock Concert", "2024-12-01", "Arena Stadium", "A12", 100.00)

# Display ticket information
ticket1.display_ticket_info()

# Reserve the ticket
ticket1.reserve_ticket()

# Try to reserve the ticket again (should show already reserved)
ticket1.reserve_ticket()

# Cancel the reservation
ticket1.cancel_reservation()

# Display updated ticket information
ticket1.display_ticket_info()

# Try to cancel reservation again (should show not reserved)
ticket1.cancel_reservation()

Ticket ID: T001
Event Name: Rock Concert
Event Date: 2024-12-01
Venue: Arena Stadium
Seat Number: A12
Price: $100.00
Reservation Status: Not Reserved
Ticket T001 for 'Rock Concert' has been reserved successfully.
Ticket T001 for 'Rock Concert' is already reserved.
Reservation for ticket T001 has been canceled.
Ticket ID: T001
Event Name: Rock Concert
Event Date: 2024-12-01
Venue: Arena Stadium
Seat Number: A12
Price: $100.00
Reservation Status: Not Reserved
Ticket T001 for 'Rock Concert' is not reserved.

In [ ]:
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.

In [36]:
class ShoppingCart:
    def __init__(self):
        # Initialize an empty list for items in the cart
        self.items = []

    def add_item(self, item):
        # Add an item to the shopping cart
        self.items.append(item)
        print(f"'{item}' has been added to your shopping cart.")

    def remove_item(self, item):
        # Remove an item from the shopping cart if it exists
        if item in self.items:
            self.items.remove(item)
            print(f"'{item}' has been removed from your shopping cart.")
        else:
            print(f"'{item}' is not in your shopping cart.")

    def view_cart(self):
        # Display the items currently in the shopping cart
        if not self.items:
            print("Your shopping cart is empty.")
        else:
            print("Items in your shopping cart:")
            for item in self.items:
                print(f"- {item}")

    def clear_cart(self):
        # Clear all items from the shopping cart
        self.items = []
        print("Your shopping cart has been cleared.")

# Example usage:
cart = ShoppingCart()

# Add items to the shopping cart
cart.add_item("Laptop")
cart.add_item("Headphones")
cart.add_item("Smartphone")

# View current items in the cart
cart.view_cart()

# Remove an item from the cart
cart.remove_item("Headphones")

# View the updated cart
cart.view_cart()

# Clear the cart
cart.clear_cart()

# View the empty cart
cart.view_cart()

'Laptop' has been added to your shopping cart.
'Headphones' has been added to your shopping cart.
'Smartphone' has been added to your shopping cart.
Items in your shopping cart:
- Laptop
- Headphones
- Smartphone
'Headphones' has been removed from your shopping cart.
Items in your shopping cart:
- Laptop
- Smartphone
Your shopping cart has been cleared.
Your shopping cart is empty.

In [ ]:

In [ ]:
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 also includes 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.

In [62]:
class Student:
    def __init__(self, name, age, grade, student_id):
        # Initialize the student's attributes
        self.name = name
        self.age = age
        self.grade = grade
        self.student_id = student_id
        self.attendance = {} # A dictionary to store attendance (date: status)

    def update_attendance(self, date, status):
        """
        Updates the attendance record for a given date and status.
        status should be either "present" or "absent".
        """
        if status not in ["present", "absent"]:
            print("Invalid status. Please use 'present' or 'absent'.")
        else:
            self.attendance[date] = status
            print(f"Attendance for {self.name} on {date} has been marked as {status}.")

    def get_attendance(self):
        """Returns the full attendance record of the student."""
        return self.attendance

    def get_average_attendance(self):
        """Calculates and returns the average attendance percentage."""
        total_days = len(self.attendance)
        if total_days == 0:
            return 0 # If no attendance record is available

        present_days = sum(1 for status in self.attendance.values() if status == "present")
        average_attendance = (present_days / total_days) * 100
        return average_attendance

    def display_student_info(self):
        """Displays the information of the student."""
        print(f"Student ID: {self.student_id}")
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Grade: {self.grade}")
        print(f"Attendance Record: {self.get_attendance()}")
        print(f"Average Attendance: {self.get_average_attendance():.2f}%")

Example usage:

# Create a Student instance
student1 = Student("John Doe", 16, "Grade 10", "S123456")

# Update attendance for various dates
student1.update_attendance("2024-11-01", "present")
student1.update_attendance("2024-11-02", "absent")
student1.update_attendance("2024-11-03", "present")

# Display the student's information
student1.display_student_info()

# Check attendance and average attendance
print(f"Attendance record: {student1.get_attendance()}")
print(f"Average Attendance: {student1.get_average_attendance():.2f}%")

Attendance for John Doe on 2024-11-01 has been marked as present.
Attendance for John Doe on 2024-11-02 has been marked as absent.
Attendance for John Doe on 2024-11-03 has been marked as present.
Student ID: S12345
Name: John Doe
Age: 16
Grade: Grade 10
Attendance Record: {'2024-11-01': 'present', '2024-11-02': 'absent', '2024-11-03': 'present'}
Average Attendance: 66.67%

```
Attendance record: {"2024-11-01": "present", "2024-11-02": "absent", "2024-11-03": "present"}
Average Attendance: 66.67%
```