01 01_July_OOPs

August 27, 2023

0.1 01_July_OOPs

1. What is the primary goal of Object-Oriented Programming (OOP)? The primary goal of Object-Oriented Programming (OOP) is to model real-world entities and their interactions in a software system. OOP aims to organize code into objects, which are instances of classes. Each class represents a blueprint that defines the structure and behavior of its objects.

The key principles of OOP are encapsulation, inheritance, and polymorphism

2. What is an object in Python? In Python, an object is a fundamental concept representing a specific instance of a data structure or a user-defined class. Everything in Python, including simple data types like integers and strings, as well as more complex structures like lists and dictionaries, is an object.

3. What is a class in Python? In Python, a class is a blueprint or a template that defines the structure and behavior of objects. It serves as a user-defined data type, allowing you to create objects based on its specifications. A class encapsulates data (attributes) and functions (methods) that operate on that data.

4. What are attributes and methods in a class? Attributes are variables that store data associated with a class. They represent the state of the objects created from the class. These variables are defined within the class and are used to hold specific information about each object of that class. Each object created from the class has its own set of attributes, which can have different values for different instances. Methods are functions defined within a class that define the behavior of the objects created from the class. They operate on the class's attributes and can perform various actions or provide specific functionalities. A method can access and modify the instance attributes through the self parameter.

5. What is the difference between class variables and instance variables in Python? Class Variables:

Class variables are variables that are shared among all instances (objects) of a class. They are defined at the class level, outside any method, and are accessible using the class name. These variables have the same value for all instances of the class, and modifying their value affects all objects created from that class. Class variables are usually used to store data that is common to all instances and remains constant throughout the lifetime of the class. You can access and modify class variables both from the class itself and from any instance of the class.

Instance Variables:

Instance variables are variables that belong to individual instances (objects) of a class. They are defined within the class's methods using the self keyword. Each instance of the class has its own set of instance variables, and they can have different values for each object. Instance variables represent the unique state of each object and are used to store data that varies from one instance to another. Since they are specific to each instance, instance variables are used to hold data that can be different for each object.

6. What is the purpose of the self parameter in Python class methods? The purpose of the self parameter is to allow class methods to access and manipulate the instance attributes (instance variables) of the object. It acts as a reference to the object itself and enables you to work with its specific data. Without self, the method wouldn't know which instance's attributes to operate on, as there could be multiple instances of the class.

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.

[1]:
```python
class Book:
    def __init__(self, title, author, isbn, publication_year, available_copies):
        self.title = title
        self.author = author
        self.isbn = isbn
        self.publication_year = publication_year
        self.available_copies = available_copies

    def check_out(self):
        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):
        self.available_copies += 1
        print(f"Book '{self.title}' returned successfully.")

    def display_book_info(self):
        print("Book Information:")
        print(f"Title: {self.title}")
        print(f"Author(s): {self.author}")
        print(f"ISBN: {self.isbn}")
        print(f"Publication Year: {self.publication_year}")
        print(f"Available Copies: {self.available_copies}")


# Usage example:
book1 = Book("The Catcher in the Rye", "J.D. Salinger", "978-0-316-76948-7",␣
 ↪1951, 3)
book1.display_book_info()

book1.check_out()
book1.display_book_info()

book1.check_out()
book1.check_out()

book1.return_book()
book1.display_book_info()
```

```
Book Information:
Title: The Catcher in the Rye
Author(s): J.D. Salinger
ISBN: 978-0-316-76948-7
Publication Year: 1951
Available Copies: 3
Book 'The Catcher in the Rye' checked out successfully.
Book Information:
Title: The Catcher in the Rye
Author(s): J.D. Salinger
ISBN: 978-0-316-76948-7
Publication Year: 1951
Available Copies: 2
Book 'The Catcher in the Rye' checked out successfully.
Book 'The Catcher in the Rye' checked out successfully.
Book 'The Catcher in the Rye' returned successfully.
Book Information:
Title: The Catcher in the Rye
Author(s): J.D. Salinger
```

```
ISBN: 978-0-316-76948-7
Publication Year: 1951
Available Copies: 1
```

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.

[2]:
```python
class Ticket:
    def __init__(self, ticket_id, event_name, event_date, venue, seat_number,␣
 ↪price):
        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

    def reserve_ticket(self):
        if not self.is_reserved:
            self.is_reserved = True
            print(f"Ticket {self.ticket_id} for '{self.event_name}' is now␣
 ↪reserved.")
        else:
```

```
            print(f"Ticket {self.ticket_id} is already reserved.")

    def cancel_reservation(self):
        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} is not reserved.")

    def display_ticket_info(self):
        print("Ticket Information:")
        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"Reservation Status: {'Reserved' if self.is_reserved else 'Not␣
 ↪Reserved'}")


# Usage example:
ticket1 = Ticket("T123", "Concert Night", "2023-08-15", "City Arena", "A-25",␣
 ↪50.00)
ticket1.display_ticket_info()

ticket1.reserve_ticket()
ticket1.display_ticket_info()

ticket1.reserve_ticket()

ticket1.cancel_reservation()
ticket1.display_ticket_info()
```

```
Ticket Information:
Ticket ID: T123
Event Name: Concert Night
Event Date: 2023-08-15
Venue: City Arena
Seat Number: A-25
Price: 50.0
Reservation Status: Not Reserved
Ticket T123 for 'Concert Night' is now reserved.
Ticket Information:
Ticket ID: T123
Event Name: Concert Night
Event Date: 2023-08-15
```

```
Venue: City Arena
Seat Number: A-25
Price: 50.0
Reservation Status: Reserved
Ticket T123 is already reserved.
Reservation for ticket T123 has been canceled.
Ticket Information:
Ticket ID: T123
Event Name: Concert Night
Event Date: 2023-08-15
Venue: City Arena
Seat Number: A-25
Price: 50.0
Reservation Status: Not Reserved
```

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.

[3]:
```python
class ShoppingCart:
    def __init__(self):
        self.items = []

    def add_item(self, item_name, price, quantity):
        item = {"item_name": item_name, "price": price, "quantity": quantity}
        self.items.append(item)
        print(f"{quantity} {item_name}(s) added to the shopping cart.")

    def remove_item(self, item_name):
        for item in self.items:
            if item["item_name"] == item_name:
                self.items.remove(item)
                print(f"{item_name} removed from the shopping cart.")
                return
```

```
        print(f"{item_name} not found in the shopping cart.")

    def get_total_price(self):
        total_price = 0
        for item in self.items:
            total_price += item["price"] * item["quantity"]
        return total_price

    def display_cart(self):
        print("Shopping Cart Contents:")
        for item in self.items:
            print(f"{item['item_name']} - Quantity: {item['quantity']} - Price␣
↪per item: ${item['price']}")

    def clear_cart(self):
        self.items.clear()
        print("Shopping cart cleared.")

# Usage example:
cart = ShoppingCart()

cart.add_item("Shirt", 25.00, 2)
cart.add_item("Shoes", 50.00, 1)
cart.add_item("Hat", 15.00, 3)

cart.display_cart()
total_price = cart.get_total_price()
print(f"Total Price: ${total_price}")

cart.remove_item("Shoes")
cart.remove_item("Pants")  # Not in the cart

cart.display_cart()
total_price = cart.get_total_price()
print(f"Total Price: ${total_price}")

cart.clear_cart()
cart.display_cart()
total_price = cart.get_total_price()
print(f"Total Price: ${total_price}")
```

```
2 Shirt(s) added to the shopping cart.
1 Shoes(s) added to the shopping cart.
3 Hat(s) added to the shopping cart.
Shopping Cart Contents:
Shirt - Quantity: 2 - Price per item: $25.0
Shoes - Quantity: 1 - Price per item: $50.0
Hat - Quantity: 3 - Price per item: $15.0
```

```
Total Price: $145.0
Shoes removed from the shopping cart.
Pants not found in the shopping cart.
Shopping Cart Contents:
Shirt - Quantity: 2 - Price per item: $25.0
Hat - Quantity: 3 - Price per item: $15.0
Total Price: $95.0
Shopping cart cleared.
Shopping Cart Contents:
Total Price: $0
```

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.

[4]:
```python
class Student:
    def __init__(self, name, age, grade, student_id):
        self.name = name
        self.age = age
        self.grade = grade
        self.student_id = student_id
        self.attendance = {}

    def update_attendance(self, date, status):
        if status.lower() == "present" or status.lower() == "absent":
            self.attendance[date] = status.lower()
            print(f"Attendance updated for {self.name} on {date}. Status:␣
↪{status.lower()}")
        else:
```

```
            print("Invalid status. Status should be 'present' or 'absent'.")

    def get_attendance(self):
        return self.attendance

    def get_average_attendance(self):
        total_days = len(self.attendance)
        if total_days == 0:
            return 0.0
        present_days = list(self.attendance.values()).count("present")
        return (present_days / total_days) * 100.0

# Usage example:
student1 = Student("John Smith", 15, "10th Grade", "S12345")

student1.update_attendance("2023-07-30", "present")
student1.update_attendance("2023-07-31", "absent")

attendance_record = student1.get_attendance()
print("Attendance Record:")
for date, status in attendance_record.items():
    print(f"{date}: {status}")

average_attendance = student1.get_average_attendance()
print(f"Average Attendance: {average_attendance:.2f}%")
```

```
Attendance updated for John Smith on 2023-07-30. Status: present
Attendance updated for John Smith on 2023-07-31. Status: absent
Attendance Record:
2023-07-30: present
2023-07-31: absent
Average Attendance: 50.00%
```

[ ]: