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

The primary goal of OOP is to improve code organization, maintainability, reusability, and flexibility in software development leading to more efficient and powerful applications.

2. What is an object in Python?

In Python, an object is a fundamental concept that represents specific instance of a class that cotains data and methods.

3. What is a class in Python?

In Python, a class is a blueprint or template for creating objects. It is a fundamental concept of object-oriented programming that defines attributes and methods.

4. What are attributes and methods in a class?

Attributes: Attributes are variables that store data associated with a class or its instances (objects).

Methods: Methods are functions defined within a class that define the behavior and actions that objects which are created from the class can perform.

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

Class variables are shared among all instances of a class and hold data that is common to all instances, while Instance variables are specific to each instance and hold unique data for each object.

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

```
The "self" parameter is a conventionally used parameter that refers to the instance of the class.
The purpose of self parameter is to allow access to instance variables and other methods of the class within those methods.
```

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

In [2]:
```python
book=Book("My Book","Hritik Kadam",123456,2000,100)
book.display_book_info()
```

```
Title: My Book
Author: Hritik Kadam
ISBN: 123456
Publication Year: 2000
Available Copies: 100
```

In [3]:
```python
book.check_out()
book.display_book_info()
```

```
Title: My Book
Author: Hritik Kadam
ISBN: 123456
Publication Year: 2000
Available Copies: 99
```

In [4]:
```
book.return_book()
book.display_book_info()
```

```
Title: My Book
Author: Hritik Kadam
ISBN: 123456
Publication Year: 2000
Available Copies: 100
```

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 [13]:
```
class Ticket:
    def __init__(self,ticket_id,event_name,event_date,venue,seat_number,pr
        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=is_reserved
    def reserve_ticket(self):
        if self.is_reserved =="N":
            self.is_reserved="Y"
        else:
            print("The ticket is already reserved")

    def cancel_reservation(self):
        if self.is_reserved=="Y":
            self.is_reserved="N"
    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"Reservation Status: {self.is_reserved}")
```

In [20]:
```
ticket=Ticket('01',"FIFA",'07/09/2023','Dubai',100,'$500','N')
ticket.reserve_ticket()
ticket.display_ticket_info()
```

```
Ticket ID: 01
Event Name: FIFA
Event Date: 07/09/2023
Venue: Dubai
Seat Number: 100
Price: $500
Reservation Status: Y
```

In [21]:
```
ticket=Ticket('01',"FIFA",'07/09/2023','Dubai',100,'$500','Y')
ticket.cancel_reservation()
ticket.display_ticket_info()
```

```
Ticket ID: 01
Event Name: FIFA
Event Date: 07/09/2023
Venue: Dubai
Seat Number: 100
Price: $500
Reservation Status: N
```

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 [47]:
```
class Shoppingcart:
    def __init__ (self):
        self.items=[]

    def add_item(self,item):
        self.items.append(item)
    def remove_item(self,item):
        self.items.remove(item)
    def view_cart(self):
        print(self.items)
    def clear_cart(self):
        self.items.clear()
```

In [48]:

```python
shop=Shoppingcart()
shop.add_item('Cake')
shop.add_item('Milk')
shop.view_cart()

shop.remove_item('Cake')
shop.view_cart()

shop.clear_cart()
shop.view_cart()
```

```
['Cake', 'Milk']
['Milk']
[]
```

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.

In [3]:

```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):
        self.attendance[date]=status
    def get_attendance(self):
        return self.attendance
    def get_average_attendance(self):
        tot_days=len(self.attendance)
        present=[]
        for i in self.attendance.values():
           if i=='Present':
            present.append(i)
        tot_present=len(present)
        avg_att=(tot_present/tot_days)*100
        return avg_att
```

In [9]:
```python
student=Student('Hritik',23,'A',22909)
student.update_attendance('07/01/2023','Present')
student.update_attendance('07/02/2023','Present')
student.update_attendance('07/03/2023','Absent')
student.update_attendance('07/04/2023','Present')
student.get_attendance()
```

Out[9]:
```
{'07/01/2023': 'Present',
 '07/02/2023': 'Present',
 '07/03/2023': 'Absent',
 '07/04/2023': 'Present'}
```

In [10]:
```python
student.get_average_attendance()
```

Out[10]:
```
75.0
```