lab6

August 24, 2025

1 Lab 6

Define a class Student with attributes name, roll_number, and marks. Implement a method display_info() that prints the details of the student. Create an instance of Student and call the display_info() method to display the student's details.

[1]:
```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_info(self):
        print("Student Details:")
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_number}")
        print(f"Marks: {self.marks}")

student1 = Student("Abhishek Kharel", "81BEL005", 92)
student1.display_info()
```

```
Student Details:
Name: Abhishek Kharel
Roll Number: 81BEL005
Marks: 92
```

Create a base class Animal with a method speak() that prints "Animal makes a sound". Derive a class Dog from Animal and override the speak() method to print "Dog barks". Instantiate the Dog class and call its speak() method.

[2]:
```
class Animal:
    def speak(self):
        print("Animal makes a sound")
class Dog(Animal):
    def speak(self):
        print("Dog barks")
dog1 = Dog()
dog1.speak()
```

```
Dog barks
```

Define a class BankAccount with private attributes account_number and balance. Implement methods to deposit and withdraw money, ensuring that the balance cannot go below zero. Provide a method to get the account details. Test the class by performing deposit and withdrawal operations.

[3]:
```python
class BankAccount:
    def __init__(self, account_number, balance=0):
        self.__account_number = account_number
        self.__balance = balance
    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount
            print(f"Deposited: {amount}. New balance = {self.__balance}")
        else:
            print("Deposit amount must be positive!")

    def withdraw(self, amount):
        if 0 < amount <= self.__balance:
            self.__balance -= amount
            print(f"Withdrew: {amount}. New balance = {self.__balance}")
        else:
            print("Insufficient balance or invalid amount!")

    def get_account_details(self):
        return f"Account Number: {self.__account_number}, Balance: {self.
 ↪__balance}"

account1 = BankAccount("123456789", 1000)

print(account1.get_account_details())
account1.deposit(500)
account1.withdraw(200)
account1.withdraw(2000)
print(account1.get_account_details())
```

```
Account Number: 123456789, Balance: 1000
Deposited: 500. New balance = 1500
Withdrew: 200. New balance = 1300
Insufficient balance or invalid amount!
Account Number: 123456789, Balance: 1300
```

Create a base class Shape with a method area(). Derive two classes Rectangle and Circle from Shape. Implement the area() method in both derived classes. Instantiate Rectangle and Circle, and demonstrate polymorphism by calling their area() methods.

[4]:
```python
import math
class Shape:
    def area(self):
        print("Area method not implemented in base class")
```

```python

class Rectangle(Shape):
    def __init__(self, length, width):
        self.length = length
        self.width = width

    def area(self):
        return self.length * self.width

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius * self.radius


shapes = [Rectangle(5, 3), Circle(4)]
for s in shapes:
    print(f"{s.__class__.__name__} area = {s.area()}")
```

```
Rectangle area = 15
Circle area = 50.26548245743669
```

Define a class Person with attributes name and age. Derive a class Employee from Person with additional attributes employee_id and salary. Implement a method display_employee() in Employee that prints all the details. Create an instance of Employee and display the information.

[5]:
```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class Employee(Person):
    def __init__(self, name, age, employee_id, salary):
        super().__init__(name, age)
        self.employee_id = employee_id
        self.salary = salary

    def display_employee(self):
        print(f"Name: {self.name}")
        print(f"Age: {self.age}")
        print(f"Employee ID: {self.employee_id}")
        print(f"Salary: {self.salary}")


emp1 = Employee("Abhishek", 22, "E123", 50000)
emp1.display_employee()
```

```
Name: Abhishek
Age: 22
Employee ID: E123
Salary: 50000
```

Define a class Vector with attributes x and y. Overload the + operator to add two Vector objects. Implement the **add**() method and test it by adding two Vector instances.

[6]:
```
class Vector:
    def __init__(self, x, y):
        self.x = x
        self.y = y
    def __add__(self, other):
        return Vector(self.x + other.x, self.y + other.y)
    def __str__(self):
        return f"Vector({self.x}, {self.y})"

v1 = Vector(3, 4)
v2 = Vector(1, 2)

v3 = v1 + v2

print("v1 =", v1)
print("v2 =", v2)
print("v1 + v2 =", v3)
```

```
v1 = Vector(3, 4)
v2 = Vector(1, 2)
v1 + v2 = Vector(4, 6)
```

Create a class Book with attributes title and author. Overload the **str**() method to return a string representation of the Book object in the format "Title by Author". Test this method by printing a Book instance.

[7]:
```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def __str__(self):
        return f'"{self.title}" by {self.author}'


book1 = Book("Python Basics", "John Doe")
book2 = Book("OOP in Depth", "Jane Smith")

print(book1)
print(book2)
```

```
"Python Basics" by John Doe
```

"OOP in Depth" by Jane Smith

Define a class Time with attributes hours, minutes, and seconds. Overload the == operator to compare two Time objects for equality. Implement the **eq**() method and test it by comparing two Time instances.

[8]:
```
class Time:
    def __init__(self, hours, minutes, seconds):
        self.hours = hours
        self.minutes = minutes
        self.seconds = seconds

    def __eq__(self, other):
        return (self.hours == other.hours and
                self.minutes == other.minutes and
                self.seconds == other.seconds)

    def __str__(self):
        return f"{self.hours:02d}:{self.minutes:02d}:{self.seconds:02d}"


t1 = Time(10, 30, 15)
t2 = Time(10, 30, 15)
t3 = Time(12, 45, 0)

print(f"t1 = {t1}")
print(f"t2 = {t2}")
print(f"t3 = {t3}")

print("t1 == t2 ?", t1 == t2)
print("t1 == t3 ?", t1 == t3)
```

```
t1 = 10:30:15
t2 = 10:30:15
t3 = 12:45:00
t1 == t2 ? True
t1 == t3 ? False
```

Define a class Person with attributes name and age. Define another class Address with attributes street, city, and zipcode. Create a Contact class that contains an instance of Person and Address. Implement methods to display the contact details. Create a Contact object and display its information.

[9]:
```
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age
class Address:
    def __init__(self, street, city, zipcode):
```

```python
        self.street = street
        self.city = city
        self.zipcode = zipcode
class Contact:
    def __init__(self, person, address):
        self.person = person
        self.address = address

    def display_contact(self):
        print("Contact Details:")
        print(f"Name: {self.person.name}")
        print(f"Age: {self.person.age}")
        print(f"Street: {self.address.street}")
        print(f"City: {self.address.city}")
        print(f"Zipcode: {self.address.zipcode}")

p = Person("Abhishek Kharel", 20)
a = Address("Old Baneshwor", "Kathmandu", "44600")

contact1 = Contact(p, a)
contact1.display_contact()
```

```
Contact Details:
Name: Abhishek Kharel
Age: 20
Street: Old Baneshwor
City: Kathmandu
Zipcode: 44600
```