



Python

Experiment No. 9

Aim: To implement classes, objects, constructor, inner class, inheritance and polymorphism.

Problem Statements:

1. Create a BankAccount class that allows you to deposit, withdraw, and check the balance of an account.
2. Create an Employee class with the following characteristics:
A constructor (`__init__`) that initializes the attributes (`employee_id`, `first_name`, `last_name`, `email`, and `salary`) when an object is created. A method called `display_employee_details` that prints the employee's details in a well-formatted manner. Create several instances of the Employee class, each representing different employees in a company. Calculate and print the average salary of all employees. Add a class variable called `employee_count` to keep track of the number of employee objects created. Update this variable within the constructor. Create a class method called `get_employee_count` that returns the total number of employees.
3. Create a Python class called Product that represents products in an inventory. Use a static variable called `total_products` to keep track of the total number of product instances created. Implement a static method to retrieve the total count.
4. Create a Library class that manages a catalog of books. Use an inner class called Book to represent individual books with attributes like title, author, and ISBN. The Library class should have methods for adding books, searching for books by title and displaying the book catalog.
5. Implement a polymorphic behavior by creating a base class Shape with a method `area`. Derive subclasses like Circle, Rectangle, and Triangle, each with its implementation of the `area` method. Demonstrate polymorphism by calculating and printing the areas of different shapes

Theory:

Python is an “object-oriented programming language.” This means that almost all the code is implemented using a special construct called classes.

Programmers use classes to keep related things together. This is done using the keyword “class,” which is a grouping of object-oriented constructs.

What is a class?

A **class** is a code template for creating objects. Objects have member variables and have behaviour associated with them. In python a class is created by the keyword class.

```
class ClassName:  
    # class definition  
  
class Bike:  
    name = ""  
    gear = 0
```

An **object** is created using the constructor of the class. This object will then be called the instance of the class. In Python we create instances in the following manner :

```
Instance = class(arguments)
```

Access Class Attributes Using Objects

We use the . notation to access the attributes of a class.

For example,

```
# modify the name attribute  
bike1.name = "Mountain Bike"  
  
# access the gear attribute  
bike1.gear
```

Constructors in Python:

Class functions that begins with double underscore (__) are called special functions as they have special meaning.

Of one particular interest is the __init__() function. This special function gets called whenever a new object of that class is instantiated.

This type of function is also called constructors in Object Oriented Programming (OOP). We normally use it to initialize all the variables.

```
class Bike:  
    # constructor function  
    def __init__(self, name = ""):
```

```
self.name = name  
bike1 = Bike()
```

The Self Parameter does not call it to be Self, You can use any other name instead of it.

__str__() method

Python has a particular method called `__str__()` that is used to define how a class object should be represented as a string. It is often used to give an object a human-readable textual representation, which is helpful for logging, debugging, or showing users object information. When a class object is used to create a string using the built-in functions `print()` and `str()`, the `__str__()` function is automatically used. You can alter how objects of a class are represented in strings by defining the `__str__()` method.

Static method

In Python, a static variable is a variable that is shared among all instances of a class, rather than being unique to each instance. It is also sometimes referred to as a class variable, because it belongs to the class itself rather than any instance of the class.

Static variables are defined inside the class definition, but outside of any method definitions. They are typically initialized with a value, just like an instance variable, but they can be accessed and modified through the class itself, rather than through an instance.

Features of Static Variables

- Static variables are allocated memory once when the object for the class is created for the first time.
- Static variables are created outside of methods but inside a class
- Static variables can be accessed through a class but not directly with an instance.
- Static variables behavior doesn't change for every object.

The Python approach is simple; it doesn't require a static keyword.

Note: All variables which are assigned a value in the class declaration are class variables. And variables that are assigned values inside methods are instance variables.

Inner Class :

A nested or inner class is contained within another class. This could be for reasons of encapsulation, where the inner class is not useful by itself.

- Chaitanya Shah

Syntax :

```
class A:  
    class B:  
        pass  
Pass
```

What is Inheritance?

Inheritance is a powerful feature in object oriented programming. It refers to defining a new class with little or no modification to an existing class. The new class is called derived (or child) class and the one from which it inherits is called the base (or parent) class.

Python Inheritance Syntax:

```
class BaseClass:  
    #Body of base class  
  
class DerivedClass(BaseClass):  
    #Body of derived class
```

Python Polymorphism:

Polymorphism means the ability to take various forms. In Python, Polymorphism allows us to define methods in the child class with the same name as defined in their parent class.

As we know, a child class inherits all the methods from the parent class. However, you will encounter situations where the method inherited from the parent class doesn't quite fit into the child class. In such cases, you will have to re-implement method in the child class. This process is known as Method Overriding.

If you have overridden a method in child class, then the version of the method will be called based upon the type of the object used to call it. If a child class object is used to call an overridden method then the child class version of the method is called. On the other hand, if parent class object is used to call an overridden method, then the parent class version of the method is called.

Q.1 Create a BankAccount class that allows you to deposit, withdraw, and check the balance of an account.

Code:

```
class BankAccount():
    bal = 0
    def deposit(self):
        amount = int(input("Enter amount to deposit: "))
        print("\nBefore Deposit balance is: ",
BankAccount.bal)
        BankAccount.bal = BankAccount.bal+ amount
        print("After Deposit balance is: ", BankAccount.bal)

    def withdraw(self):
        wdr_amt = int(input("Enter amount to withdraw: "))
        print("\nAmount to withdraw: ", wdr_amt)
        print("Before Withdrawal balance is: ",
BankAccount.bal)
        if wdr_amt < BankAccount.bal:
            BankAccount.bal = BankAccount.bal - wdr_amt
            print("After Withdrawal balance is: ",
BankAccount.bal)
        else:
            print("Less amount")

    def check_bal(self):
        print("\nCurrent balance is:", BankAccount.bal)

print("Choose Option : \n1.Deposit\n2.Withdraw\n3.Check
Balance\n0. EXIT")
choice = int(input("Enter your choice: "))

while(choice!=0):
    u1 = BankAccount()
    if (choice == 1):
        u1.deposit()
    if (choice == 2):
        u1.withdraw()
    if (choice == 3):
        u1.check_bal()
    choice = int(input("Enter your choice: "))
```

Output:

```
➡ Choose Option :  
1.Deposit  
2.Withdraw  
3.Check Balance  
0. EXIT  
Enter your choice: 1  
Enter amount to deposit: 5000  
  
Before Deposit balance is: 0  
After Deposit balance is: 5000  
Enter your choice: 2  
Enter amount to withdraw: 200  
  
Amount to withdraw: 200  
Before Withdrawal balance is: 5000  
After Withdrawal balance is: 4800  
Enter your choice: 3  
  
Current balance is: 4800  
Enter your choice: 0
```

Q.2 Create an Employee class with the following characteristics:

- A constructor (`__init__`) that initializes the attributes (`employee_id`, `first_name`, `last_name`, `email`, and `salary`) when an object is created.
- A method called `display_employee_details` that prints the employee's details in a well-formatted manner.
- Create several instances of the Employee class, each representing different employees in a company.
- Calculate and print the average salary of all employees.
- Add a class variable called `employee_count` to keep track of the number of employee objects created.
- Update this variable within the constructor.
- Create a class method called `get_employee_count` that returns the total number of employees.

Code:

```
class Employee:
    employee_count = 0

    def __init__(self, employee_id, first_name, last_name, email, salary):
        self.employee_id = employee_id
        self.first_name = first_name
        self.last_name = last_name
        self.email = email
        self.salary = salary
        Employee.employee_count += 1

    def display_employee_details(self):
        print(f"Employee ID: {self.employee_id}")
        print(f"First Name: {self.first_name}")
        print(f>Last Name: {self.last_name}")
        print(f>Email: {self.email}")
        print(f"Salary: {self.salary}")
        print("-" * 20)

    def get_employee_count():
        return Employee.employee_count

employee1 = Employee(1, "Chaitanya", "Shah",
                    "chaitanya.shah@example.com", 100000)
employee2 = Employee(2, "Krisha", "Shah",
                    "krisha.shah@example.com", 60000)
employee3 = Employee(3, "Komal", "Kasat",
                    "komal.kasat@example.com", 50000)
```

```

employee4 = Employee(4, "Dev", "Mittal",
"dev.mittal@example.com", 80000)

employee1.display_employee_details()
employee2.display_employee_details()
employee3.display_employee_details()
employee4.display_employee_details()

total_salary = employee1.salary + employee2.salary +
employee3.salary
average_salary = total_salary / Employee.employee_count
print(f"Average Salary: {average_salary}")

total_employees = Employee.get_employee_count()
print(f"Total Employees: {total_employees}")

```

Output:

```

➡ Employee ID: 1
First Name: Chaitanya
Last Name: Shah
Email: chaitanya.shah@example.com
Salary: 100000
-----
Employee ID: 2
First Name: Krisha
Last Name: Shah
Email: krisha.shah@example.com
Salary: 60000
-----
Employee ID: 3
First Name: Komal
Last Name: Kasat
Email: komal.kasat@example.com
Salary: 50000
-----
Employee ID: 4
First Name: Dev
Last Name: Mittal
Email: dev.mittal@example.com
Salary: 80000
-----
Average Salary: 52500.0
Total Employees: 4

```


Q.3 Create a Python class called Product that represents products in an inventory. Use a static variable called total_products to keep track of the total number of product instances created. Implement a static method to retrieve the total count.

Code:

```
class Product:
    total_products = 0

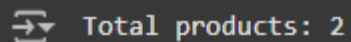
    def __init__(self, name, price, quantity):
        self.name = name
        self.price = price
        self.quantity = quantity
        Product.total_products += 1

    def get_total_products():
        return Product.total_products

product1 = Product("Shirt", 25, 100)
product2 = Product("Pants", 50, 50)

total_count = Product.get_total_products()
print(f"Total products: {total_count}")
```

Output:



- Q.4 Create a Library class that manages a catalog of books.
Use an inner class called Book to represent individual books with attributes like title, author, and ISBN.
The Library class should have methods for adding books, searching for books by title and displaying the book catalog.

Code:

```
class Library:
    def __init__(self):
        self.catalog = []

    class Book:
        def __init__(self, title, author, isbn):
            self.title = title
            self.author = author
            self.isbn = isbn

        def __str__(self):
            return f"Title: {self.title}, Author: {self.author}, ISBN: {self.isbn}"

    def add_book(self, title, author, isbn):
        book = self.Book(title, author, isbn)
        self.catalog.append(book)

    def search_book(self, title):
        for book in self.catalog:
            if book.title == title:
                return book
        return None # Book not found

    def display_catalog(self):
        for book in self.catalog:
            print(book)

my_library = Library()

my_library.add_book("Da Vinci Code", "Dan Brown", "9745-5484")
my_library.add_book("Pride & Prejudice", "Jane Austen", "9712-5641")

book = my_library.search_book("Da Vinci Code")
if book:
```

```
    print("Book found:", book)
else:
    print("Book not found.")

print("\nLibrary Catalog:")
my_library.display_catalog()
```

Output:

```
➞ Book found: Title: Da Vinci Code, Author: Dan Brown, ISBN: 9745-5484

Library Catalog:
Title: Da Vinci Code, Author: Dan Brown, ISBN: 9745-5484
Title: Pride & Prejudice, Author: Jane Austen, ISBN: 9712-5641
```

Q.5 Implement a polymorphic behavior by creating a base class Shape with a method area. Derive subclasses like Circle, Rectangle, and Triangle, each with its implementation of the area method. Demonstrate polymorphism by calculating and printing the areas of different shapes

Code:

```
import math

class Shape:
    def area(self):
        pass

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

    def area(self):
        return math.pi * self.radius**2

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

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

class Triangle(Shape):
    def __init__(self, base, height):
        self.base = base
        self.height = height

    def area(self):
        return 0.5 * self.base * self.height

circle = Circle(5)
rectangle = Rectangle(4, 6)
triangle = Triangle(3, 8)

shapes = [circle, rectangle, triangle]
for shape in shapes:
    print(f"Area of {type(shape).__name__}: {shape.area()}")
```

Output:

```
⇒ Area of Circle: 78.53981633974483  
Area of Rectangle: 24  
Area of Triangle: 12.0
```

Conclusion: Thus studied classes, objects, constructors and inner class.

- Chaitanya Shah