GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)

Accredited with A+ Grade by NAAC

12-B Status from UGC

DEPARTMENT: – COMPUTER ENGINEERING AND APPLICATIONS

PROGRAM: –B. TECH CSE(AIML & IOT)

SEMESTER: – 2ND

SUBJECT: – COMPUTER PROGRAMMING – II (ADVANCE PYTHON)

ASSIGNMENT: - 2ND

CLASS ROLL NO.: - 37

UNIVERSITY ROLL NO.: – 2315510150

SECTION: – 'CB'-2

Submitted to: -
Md. Amir Khan

Submitted By: -
Prem Narayan Sharma

---

1). Problem: Calculator Description: Create a simple calculator class that can perform basic arithmetic operations (addition, subtraction, multiplication, division).

```
class Calculator:
    def add(self, x, y):
        return x + y

    def subtract(self, x, y):
        return x - y

    def multiply(self, x, y):
        return x * y

    def divide(self, x, y):
        return x / y

    def Error(self, x, y):
        if y == 0:
            return "Error: Division by zero"

# Example usage:
calculator = Calculator()

result_addition = calculator.add(5, 3)
result_subtraction = calculator.subtract(10, 4)
result_multiplication = calculator.multiply(2, 6)
result_division = calculator.divide(8, 2)

print("Addition:", result_addition)
print("Subtraction:", result_subtraction)
print("Multiplication:", result_multiplication)
print("Division:", result_division)
```

Output: -

Addition: 8
Subtraction: 6
Multiplication: 12
Division: 4.0

---

2. Problem: Student Management System Description: Implement a Student class with attributes like name, roll number, and methods for displaying student details.

```
class Student:
    def __init__(self,name,roll_no,course,branch):
        self.name=name
        self.roll_no=roll_no
        self.course=course
        self.branch=branch
    def getDetails(self):
        print(f'''Name : {self.name}\nRoll No. : {self.roll_no}\nCourse :
{self.course}\nBranch : {self.branch}''')
Student1=Student("Prem Narayan Sharma","2315510150","B-Tech","CSE(AIML &
IOT)")

Student1.getDetails()
```

Output: -

```
Name : Prem Narayan Sharma
Roll No. : 2315510150
Course : B-Tech
Branch : CSE(AIML & IOT)
```

---

3. Problem: Employee Management System Description: Design an Employee class with attributes like name, employee ID, and methods for calculating salary based on hours worked.

```python
class Employee:
    def __init__(self, name, employee_id):
        self.name = name
        self.employee_id = employee_id

    def calculate_salary(self, hours_worked, hourly_rate):
        salary = hours_worked * hourly_rate
        return salary

    def display_details(self):
        print("Employee Details:")
        print("Name:", self.name)
        print("Employee ID:", self.employee_id)

employee1 = Employee("Prem Sharma", "2315510150")
employee2 = Employee("Tony Stark", "2315510151")

hourly_rate = 500

hours_worked_employee1 = 40
hours_worked_employee2 = 30

salary_employee1 = employee1.calculate_salary(hours_worked_employee1,
hourly_rate)
salary_employee2 = employee2.calculate_salary(hours_worked_employee2,
hourly_rate)

employee1.display_details()
print("Salary:", salary_employee1,'rupee')
print("--------------------")
employee2.display_details()
print("Salary:", salary_employee2,'rupee')
```

Output: -

```
Employee Details:
Name: Prem Sharma
Employee ID: 2315510150
Salary: 20000 rupee
--------------------
Employee Details:
Name: Tony Stark
Employee ID: 2315510151
Salary: 15000 rupee
```

---

4. Problem: ToDo List Description: Create a ToDo class with methods to add tasks, mark tasks as completed, and display the list of tasks.

```
class ToDo:
    def __init__(self):
        self.tasks = []

    def add_task(self, task):
        self.tasks.append({"task": task, "completed": False})
        print(f"Task '{task}' added successfully.")

    def mark_as_completed(self, task_index):
        if 0 <= task_index < len(self.tasks):
            self.tasks[task_index]["completed"] = True
            print(f"Task '{self.tasks[task_index]['task']}' marked as
completed.")
        else:
            print("Invalid task index.")

    def display_tasks(self):
        """Display the list of tasks."""
        if not self.tasks:
            print("No tasks in the to-do list.")
        else:
            print("To-Do List:")
            for index, task_info in enumerate(self.tasks):
                status = " [x] " if task_info["completed"] else " [ ] "
                print(f"{index + 1}.{status} {task_info['task']}")

# Example usage:
todo_list = ToDo()

todo_list.add_task("Buy groceries")
todo_list.add_task("Complete homework")
todo_list.add_task("Exercise")

todo_list.display_tasks()

print("--------------------")
todo_list.mark_as_completed(2)

print("--------------------")
todo_list.display_tasks()
```

Output: -

```
Task 'Buy groceries' added successfully.
Task 'Complete homework' added successfully.
Task 'Exercise' added successfully.
To-Do List:
1. [ ]  Buy groceries
2. [ ]  Complete homework
3. [ ]  Exercise
--------------------
```

Task 'Exercise' marked as completed.

To-Do List:
1. [ ] Buy groceries
2. [ ] Complete homework
3. [x] Exercise

---

5. Problem: Car Rental System Description: Develop a Car class for a rental system, including attributes like model, color, and methods for calculating rental charges.

```
class Car:
    def __init__(self, model, color, base_rate):

        self.model = model
        self.color = color
        self.base_rate = base_rate

    def calculate_rental_charge(self, days, discount=0):
        if days < 0:
            raise ValueError("Number of days cannot be negative.")

        daily_rate = self.base_rate * (1 - discount)
        total_cost = days * daily_rate
        return total_cost

    def __str__(self):

        return f"Car Model: {self.model}, Color: {self.color}, Base Rate:
{self.base_rate}"

# Example Usage
car1 = Car("Toyota Corolla", "Blue", 100)
print(car1)
print("Rental Charge for 5 days:", car1.calculate_rental_charge(5))
print("Rental Charge for 5 days with 10% discount:",
car1.calculate_rental_charge(5,0.1))
```

Output : -

```
Car Model: Toyota Corolla, Color: Blue, Base Rate: 100
Rental Charge for 5 days: 500
Rental Charge for 5 days with 10% discount: 450.0
```

---

6. Problem: Bank Transaction System Description: Build a BankAccount class with methods for deposit, withdraw, and check balance.

```
class BankAccount:
    def __init__(self, account_holder, initial_balance=0):
        self.account_holder = account_holder
        self.balance = initial_balance

    def deposit(self, amount):
        if amount > 0:
            self.balance += amount
            print(f"Deposited {amount} rupees . New balance: {self.balance}
rupees")
```

```
        else:
            print("Invalid deposit amount. Please enter a positive value.")

    def withdraw(self, amount):
        if amount > 0 and amount <= self.balance:
            self.balance -= amount
            print(f"Withdrew {amount} rupees . New balance: {self.balance}
rupees")
        else:
            print("Invalid withdrawal amount or insufficient funds.")

    def check_balance(self):
        print(f"Current balance for {self.account_holder}: {self.balance}
rupees")

# Example usage:
account1 = BankAccount("Prem Sharma", initial_balance=1000)

account1.check_balance()
account1.deposit(500)
account1.withdraw(200)
account1.check_balance()
```

Output: -

```
Current balance for Prem Sharma: 1000 rupees
Deposited 500 rupees . New balance: 1500 rupees
Withdrew 200 rupees . New balance: 1300 rupees
Current balance for Prem Sharma: 1300 rupees
```

---

7. Problem: Shape Area Calculator Description: Implement a Shape class with methods to calculate the area for different shapes like square, rectangle, and circle.

```
import math

class Shape:
    def calculate_area(self):
        pass

class Square(Shape):
    def __init__(self, side_length):
        self.side_length = side_length

    def calculate_area(self):
        return self.side_length ** 2

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

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

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

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

# Example usage:
square = Square(5)
rectangle = Rectangle(4, 6)
circle = Circle(3)

print("Area of Square:", square.calculate_area())
print("Area of Rectangle:", rectangle.calculate_area())
print("Area of Circle:", circle.calculate_area())
```

Output: -

```
Area of Square: 25
Area of Rectangle: 24
Area of Circle: 28.274333882308138
```

---

8. Problem: Time Converter Description: Create a Time class that converts seconds to minutes, minutes to hours, and vice versa.

```
class Time:
    def __init__(self, seconds=0, minutes=0, hours=0):
        self.total_seconds = seconds + minutes * 60 + hours * 3600

    def to_seconds(self):
        return self.total_seconds

    def to_minutes(self):
        return self.total_seconds / 60

    def to_hours(self):
        return self.total_seconds / 3600

    @classmethod
    def from_seconds(cls, seconds):
        return cls(seconds=seconds)

    @classmethod
    def from_minutes(cls, minutes):
        return cls(minutes=minutes)

    @classmethod
    def from_hours(cls, hours):
        return cls(hours=hours)

# Example usage:
# Convert 2 hours to minutes and seconds
time_obj = Time.from_hours(2)
print(f"{time_obj.to_hours()} hours is equal to {time_obj.to_minutes()}
minutes or {time_obj.to_seconds()} seconds.")

# Convert 180 seconds to minutes and hours
time_obj = Time.from_seconds(180)
print(f"{time_obj.to_seconds()} seconds is equal to {time_obj.to_minutes()}
minutes or {time_obj.to_hours()} hours.")
```

Output; -

```
2.0 hours is equal to 120.0 minutes or 7200 seconds.
180 seconds is equal to 3.0 minutes or 0.05 hours.
```

---

9. Problem: Dice Simulator Description: Develop a Dice class that simulates rolling a six-sided die.

```python
import random

class Dice:
    def __init__(self):
        self.sides = 6

    def roll(self):
        return random.randint(1, self.sides)

# Example usage:
dice = Dice()

# rolling the die 6 times
for _ in range(6):
    result = dice.roll()
    print(f"Rolling the die... and result is: {result}")
```

Output: -

```
Rolling the die... and result is: 5
Rolling the die... and result is: 4
Rolling the die... and result is: 2
Rolling the die... and result is: 6
Rolling the die... and result is: 1
Rolling the die... and result is: 5
```

---

10. Problem: Library Book Management Description: Design a LibraryBook class with attributes like title, author, and availability status. Implement methods for borrowing and returning books.

```python
class LibraryBook:
    def __init__(self, title, author):

        self.title = title
        self.author = author
        self.is_available = True

    def borrow_book(self):

        if self.is_available:
            self.is_available = False
            print(f"{self.title} by {self.author} has been successfully
borrowed.")
        else:
            print(f"Sorry, {self.title} by {self.author} is currently
unavailable.")

    def return_book(self):

        self.is_available = True
```

```
        print(f"{self.title} by {self.author} has been returned and is now
available.")

    def __str__(self):

        availability = "Available" if self.is_available else "Unavailable"
        return f"Title: {self.title}, Author: {self.author}, Status:
{availability}"

# Example Usage
book = LibraryBook("Concepts of Physics", "H.C. verma")
print(book)

book.borrow_book()
print(book)

book.return_book()
print(book)
```

Output: -

```
Title: Concepts of Physics, Author: H.C. verma, Status: Available
Concepts of Physics by H.C. verma has been successfully borrowed.
Title: Concepts of Physics, Author: H.C. verma, Status: Unavailable
Concepts of Physics by H.C. verma has been returned and is now available.
Title: Concepts of Physics, Author: H.C. verma, Status: Available
```