🐍 Problem Solving Using Python

💻 Python Programming Lab Manual

Course Title: Problem Solving Using Python

Prepared by:

Mr. Mohammed Siraj B

School of Engineering

St Aloysius (Deemed to be University), Mangalore

Academic Year: 2025 – 2026

💡 Manual Highlights:

✅ Simple, step-by-step algorithms
✅ Beginner-friendly, well-commented Python programs
✅ Neatly formatted outputs for quick understanding
✅ Ideal for lab submissions or classroom learning

🌟 **"Practice coding daily — logic grows with every line you write!"** 🌟

⚙️ SUGGESTED LIST OF EXPERIMENTS

🧩 Experiment 1 – Display All Prime Numbers Within a Range

🎯 Aim:

To write a Python program to display all prime numbers within a given range.

🧠 Algorithm / Steps:

1. Accept the start and end of the range from the user.
2. Loop through each number in the given range.
3. For each number, check if it has exactly two factors (1 and itself).
4. Display all numbers that satisfy the prime condition.

💻 Program:

```
# Program to display all prime numbers within a range

start = int(input("Enter start of range: "))
end = int(input("Enter end of range: "))

print(f"Prime numbers between {start} and {end} are:")

for num in range(start, end + 1):
    if num > 1:
        for i in range(2, num):
            if num % i == 0:
                break
        else:
            print(num, end=" ")
```

🖥️ Expected Output:

```
Enter start of range: 10
Enter end of range: 30
Prime numbers between 10 and 30 are:
11 13 17 19 23 29
```

✅ Result:

The program successfully displays all prime numbers within the given range.

🧩 Experiment 2 – Library Fine Calculation

🎯 Aim:

To write a Python program to calculate the fine for returning library books late.

🧠 Algorithm / Steps:

1. Accept issue and return dates from the user.
2. Assume the due period is 15 days.
3. Calculate the number of late days.
4. Apply fine rules based on delay duration:

Delay (days)	Fine (per day)	Action
1–5	₹0.50	Fine applicable
6–10	₹1.00	Fine applicable
11–30	₹5.00	Fine applicable
>30	–	Membership cancelled

💻 Program:

```
from datetime import date

issue_day = int(input("Enter issue day: "))
issue_month = int(input("Enter issue month: "))
return_day = int(input("Enter return day: "))
return_month = int(input("Enter return month: "))

issue_date = date(2025, issue_month, issue_day)
return_date = date(2025, return_month, return_day)

days_diff = (return_date - issue_date).days
due_days = 15
late_days = days_diff - due_days
```

```
if late_days <= 0:
    print("Book returned on time. No fine!")
elif late_days <= 5:
    print("Fine = ₹", late_days * 0.5)
elif late_days <= 10:
    print("Fine = ₹", late_days * 1)
elif late_days <= 30:
    print("Fine = ₹", late_days * 5)
else:
    print("Membership Cancelled!")
```

🖥️ Expected Output:

```
Enter issue day: 1
Enter issue month: 3
Enter return day: 25
Enter return month: 3
Fine = ₹ 50
```

✅ Result:

The program calculates and displays the correct fine or membership cancellation notice.

🧩 Experiment 3 – Count Occurrence of Elements in a List

🎯 Aim:

To create a list of random numbers, count occurrences of each element, and display them using a dictionary.

🧠 Algorithm / Steps:

1. Create a list with sample numbers.
2. Initialize an empty dictionary to count occurrences.
3. Traverse each element of the list.
4. Update count values for each unique number.
5. Display the final dictionary.

💻 Program:

```
# Create list and count element occurrences

numbers = [1, 2, 3, 2, 4, 1, 2, 5, 3, 1]
count_dict = {}

for num in numbers:
    if num in count_dict:
        count_dict[num] += 1
    else:
        count_dict[num] = 1

print("Number occurrences:")
for key, value in count_dict.items():
    print(f"{key}: {value}")
```

🖥️ Expected Output:

```
Number occurrences:
1: 3
2: 3
3: 2
4: 1
5: 1
```

✅ Result:

The program counts and displays occurrences of all elements using a dictionary.

🧩 Experiment 4 – Overtime Pay Calculation

🎯 Aim:

To calculate overtime pay for 10 employees at ₹12.00 per hour for every hour worked beyond 40 hours.

🧠 Algorithm / Steps:

1. Set total employees = 10.
2. For each employee, input total working hours.
3. If hours > 40, calculate overtime hours.
4. Multiply overtime hours by ₹12 to get total pay.
5. Display results for each employee.

💻 Program:

```
# Calculate overtime pay for 10 employees

for i in range(1, 11):
    hours = int(input(f"Enter hours worked by employee {i}: "))
    if hours > 40:
        overtime = (hours - 40) * 12
        print(f"Employee {i} Overtime Pay: ₹{overtime}")
    else:
        print(f"Employee {i} has no overtime pay.")
```

🖥️ Expected Output:

```
Enter hours worked by employee 1: 45
Employee 1 Overtime Pay: ₹60
Enter hours worked by employee 2: 38
Employee 2 has no overtime pay.
...
```

✅ Result:

The program correctly computes overtime pay for employees.

🧩 Experiment 5 – Student Information Dictionary

🎯 Aim:

To create and manage a student dictionary containing register numbers and names.

🧠 Algorithm / Steps:

1. Create a dictionary of register numbers and names.
2. Input a register number to search.
3. If it exists, display the corresponding student name.
4. Otherwise, display "Record not found".
5. Finally, display the complete sorted list.

💻 Program:

```
# Student dictionary program

Student_info = {
    101: "John",
    102: "Sara",
    103: "Ali",
    104: "Priya"
}

reg_no = int(input("Enter register number: "))

if reg_no in Student_info:
    print("Record Found:", Student_info[reg_no])
else:
    print("Record not found!")

print("\nSorted Student List:")
for reg, name in sorted(Student_info.items()):
    print(f"{reg}: {name}")
```

🖥️ Expected Output:

```
Enter register number: 103
Record Found: Ali

Sorted Student List:
101: John
102: Sara
103: Ali
104: Priya
```

✅ Result:

The program successfully manages student information and displays the sorted list.

🧩 Experiment 6 – Division with Exception Handling

🎯 Aim:

To divide two numbers and handle division-by-zero errors using exception handling.

🧠 Algorithm / Steps:

1. Accept numerator and denominator as inputs.
2. Use `try-except` blocks to handle exceptions.
3. If denominator = 0, show custom error.
4. Otherwise, display the division result.

💻 Program:

```
# Division with exception handling

try:
    a = int(input("Enter numerator: "))
    b = int(input("Enter denominator: "))
    result = a / b
    print("Result:", result)
except ZeroDivisionError:
    print("❌ Error: Division by zero is not allowed.")
except ValueError:
    print("⚠️ Invalid input. Please enter numeric values only.")
```

🖥️ Expected Output:

```
Enter numerator: 10
Enter denominator: 0
❌ Error: Division by zero is not allowed.
```

✅ Result:

The program performs division safely, handling exceptions gracefully.