

Name: C.Srihan Reddy

H.No:2303A51962

Batch:24

## LAB-ASSIGNMENT -6

### Q1: Task Description #1 (Loops – Automorphic Numbers in a Range)

- Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
- Instructions:
  - o Get AI-generated code to list Automorphic numbers using a for loop.
  - o Analyze the correctness and efficiency of the generated logic.
  - o Ask AI to regenerate using a while loop and compare both implementations.

Prompt: #generate a function that displays all automorphic numbers between 1 and 1000 usig a for loop

### Code:

```
import time as tt

def is_automorphic(num):
    square = num ** 2
    return str(square).endswith(str(num))

automorphic_numbers = []
start_time = tt.time()
for i in range(1, 1001):
    if is_automorphic(i):
        automorphic_numbers.append(i)
print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)

end_time = tt.time()
print("Time taken to execute the program:", end_time - start_time, "seconds")

#generate a function that displays all automorphic numbers between 1 and 1000 using while use time module
```

```

import time as tt

def is_automorphic(num):
    square = num ** 2
    return str(square).endswith(str(num))

automorphic_numbers = []
start_timee = tt.time()

i = 1

while i <= 1000:
    if is_automorphic(i):
        automorphic_numbers.append(i)
    i += 1

print("Automorphic numbers between 1 and 1000 are:", automorphic_numbers)

end_timee = tt.time()

print("Time taken to execute the program:", end_timee - start_timee, "seconds")

```

### **output:**

Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]

Time taken to execute the program: 0.0003712177276611328 seconds

PS C:\Us> & C:/Users/nikhi/AppData/Local/Microsoft/WindowsApps/python3.13.exe  
c:/Users/nikhi/OneDrive/Attachments/Desktop/AI\_assist\_coding/prac5.py

Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]

Time taken to execute the program: 0.0004649162292480469 seconds

Justification:

If range is known for loop is preferred

If range unknown while loop is best.

### **Q2:**

#### **Task Description #2 (Conditional Statements – Online Shopping Feedback**

##### **Classification)**

- **Task: Ask AI to write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1–5).**

- **Instructions:**

- o Generate initial code using nested if-elif-else.
- o Analyze correctness and readability.
- o Ask AI to rewrite using dictionary-based or match-case structure.

**Code:**

**Prompt:**#generate a python code that classify feedback as positive,neutral,negative based on rating 1-5

```
def classify_feedback(rating):
    if rating >= 4:
        return "positive"
    elif rating == 3:
        return "neutral"
    elif rating <= 2:
        return "negative"
    else:
        return "invalid rating"

# Example usage
ratings = [5, 4, 3, 2, 1, 0, 6]
for rate in ratings:
    print(f"Rating: {rate}, Feedback: {classify_feedback(rate)}")

#rewrite the code using dictionaries
def classify_feedback_dict(rating):
    feedback_dict = {
        range(4, 6): "positive",
        range(3, 4): "neutral",
        range(0, 3): "negative"
    }
    for key in feedback_dict:
        if rating in key:
            return feedback_dict[key]
    return "invalid rating"
```

```
# Example usage

for rate in ratings:

    print(f"Rating: {rate}, Feedback: {classify_feedback_dict(rate)}")

output: Rating: 5, Feedback: positive

Rating: 4, Feedback: positive

Rating: 3, Feedback: neutral

Rating: 2, Feedback: negative

Rating: 1, Feedback: negative

Rating: 0, Feedback: negative

Rating: 6, Feedback: positive

Rating: 5, Feedback: positive

Rating: 4, Feedback: positive

Rating: 3, Feedback: neutral

Rating: 2, Feedback: negative

Rating: 1, Feedback: negative

Rating: 0, Feedback: negative

Rating: 6, Feedback: invalid rating

PS C:\Users\nikhi\OneDrive\Attachments\Desktop\AI_assist_coding> &
C:/Users/nikhi/AppData/Local/Microsoft/WindowsApps/python3.13.exe
c:/Users/nikhi/OneDrive/Attachments/Desktop/AI_assist_coding/prac5.py

Rating: 5, Feedback: positive

Rating: 4, Feedback: positive

Rating: 3, Feedback: neutral

Rating: 2, Feedback: negative

Rating: 1, Feedback: negative

Rating: 0, Feedback: negative

Rating: 6, Feedback: positive
```

--- Using Dictionary Approach ---

```
Rating: 5, Feedback: positive

Rating: 4, Feedback: positive
```

**Rating: 3, Feedback: neutral**

**Rating: 2, Feedback: negative**

**Rating: 1, Feedback: negative**

**Rating: 0, Feedback: negative**

**Rating: 6, Feedback: invalid rating**

### **Q3: Task 3: Statistical\_operations**

**Define a function named statistical\_operations(tuple\_num) that performs the**

**following statistical operations on a tuple of numbers:**

- **Minimum, Maximum**
- **Mean, Median, Mode**
- **Variance, Standard Deviation**

**While writing the function, observe the code suggestions provided by GitHub**

**Copilot. Make decisions to accept, reject, or modify the suggestions based on**

**their relevance and correctness**

#### **Prompt:**

```
#generate a function named statistical_operations(tuple_num) that performs the  
min,max,mean,median,mode,variance,std
```

```
import statistics
```

```
def statistical_operations(tuple_num):
```

```
    result = {}
```

```
    result['min'] = min(tuple_num)
```

```
    result['max'] = max(tuple_num)
```

```
    result['mean'] = statistics.mean(tuple_num)
```

```
    result['median'] = statistics.median(tuple_num)
```

```
    try:
```

```
        result['mode'] = statistics.mode(tuple_num)
```

```
    except statistics.StatisticsError:
```

```
        result['mode'] = "No unique mode"
```

```
    result['variance'] = statistics.variance(tuple_num)
```

```
    result['std_dev'] = statistics.stdev(tuple_num)
```

```
    return result
```

```
# Example usage  
numbers = (1, 2, 2, 3, 4, 5, 5, 5)  
stats = statistical_operations(numbers)  
for key, value in stats.items():  
    print(f'{key}: {value}')
```

#### **Output:**

```
min: 1  
max: 5  
mean: 3.375  
median: 3.5  
mode: 5  
variance: 2.5535714285714284  
std_dev: 1.5979898086569353
```

#### **Q4: Task 4: Teacher Profile**

**Create a class Teacher with attributes teacher\_id, name, subject, and experience. Add a method to display teacher details.**

- **Expected Output: Class with initializer, method, and object creation.**

#### **Code:**

**#Create a class Teacher with attributes teacher\_id, name,subject, and experience. Add a method to display teacher details.**

```
class Teacher:  
  
    def __init__(self, teacher_id, name, subject, experience):  
        self.teacher_id = teacher_id  
        self.name = name  
        self.subject = subject  
        self.experience = experience
```

```
def display_details(self):  
    print(f"Teacher ID: {self.teacher_id}")  
    print(f"Name: {self.name}")  
    print(f"Subject: {self.subject}")
```

```
print(f"Experience: {self.experience} years")  
# Example usage  
teacher1 = Teacher(101, "Alice Smith", "Mathematics", 10)
```

## Output:

**Teacher ID: 101**  
**Name: Alice Smith**  
**Subject: Mathematics**  
**Experience: 10 years**

## Q6: Task Description #6 (Loops – Armstrong Numbers in a Range)

**Task:** Write a function using AI that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).

### Instructions:

- Use a for loop and digit power logic.
- Validate correctness by checking known Armstrong numbers (153, 370, etc.).
- Ask AI to regenerate an optimized version (using list comprehensions).

### Expected Output #7:

- Python program listing Armstrong numbers in the range.
- Optimized version with explanation.

**Prompt:#write a function that finds all armstrong numbers in given range**

### Code:

```
def is_armstrong(number):  
    num_str = str(number)  
    num_digits = len(num_str)  
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)  
    return sum_of_powers == number  
  
def find_armstrong_in_range(start, end):  
    armstrong_numbers = []  
    for num in range(start, end + 1):
```

```

if is_armstrong(num):
    armstrong_numbers.append(num)

return armstrong_numbers

start_range = int(input("Enter the start of the range: "))

end_range = int(input("Enter the end of the range: "))

armstrong_numbers = find_armstrong_in_range(start_range, end_range)

print(f"Armstrong numbers between {start_range} and {end_range}: {armstrong_numbers}")

```

**output:**

**Enter the start of the range: 1**

**Enter the end of the range: 1000**

**Armstrong numbers between 1 and 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]**

**Q8: ask Description #8 (Loops – Strong Numbers in a Range)**

**Task: Generate a function using AI that displays all Strong Numbers (sum of factorial of digits equals the number, e.g.,  $145 = 1!+4!+5!$ ) within a given range.**

**Instructions:**

- Use loops to extract digits and calculate factorials.
- Validate with examples (1, 2, 145).
- Ask AI to regenerate an optimized version (precompute digit factorials).

**Expected Output #9:**

- Python program that lists Strong Numbers.
- Optimized version with explanation.

**Code:**

```

#write a function that finds all armstrong numbers in given range ptimzized

def is_armstrong(number):

    num_str = str(number)

    num_digits = len(num_str)

    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    return sum_of_powers == number

def find_armstrong_in_range(start, end):

    armstrong_numbers = []

    for num in range(start, end + 1):

```

```
if is_armstrong(num):
    armstrong_numbers.append(num)
return armstrong_numbers

start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
armstrong_numbers = find_armstrong_in_range(start_range, end_range)
print(f"Armstrong numbers between {start_range} and {end_range}: {armstrong_numbers}")
```

**Output:**

**Enter the start of the range: 1**

**Enter the end of the range: 1000**

**Armstrong numbers between 1 and 1000: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407].**

## **Q10: Few-Shot Prompting for Nested Dictionary Extraction**

### **Objective**

**Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.**

### **Requirements**

- **The function should extract and return:**
  - **Full Name**
  - **Branch**
  - **SGPA**

### **Expected Output**

**A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples**

**Code: ""**

```
'personal_info': {
    'name': {
        'full_name': 'Nikhil yamsani',
        'first_name': 'Yamsani',
        'last_name': 'Nikhil'
```

```
        },
        'age': 21
    },
    'academic_info': {
        'branch': 'Computer Science',
        'sgpa': 9.0
    }
},
'personal_info': {
    'name': {
        'full_name': 'Ganneboina Samshray',
        'first_name': 'Samshray',
        'last_name': 'Ganneboina'
    },
    'age': 20
},
'academic_info': {
    'branch': 'Computer Science',
    'sgpa': 8.7
}
"""

def extract_student_info(student_dict):
    full_name = student_dict['personal_info']['name']['full_name']
    branch = student_dict['academic_info']['branch']
    sgpa = student_dict['academic_info']['sgpa']
    return full_name, branch, sgpa

# Example usage
student_data = {
    'personal_info': {
        'name': {
            'full_name': 'John Doe',
            'first_name': 'John',
            'last_name': 'Doe'
        }
    }
}
```

```
    },
    'age': 20
}
'academic_info': {
    'branch': 'Computer Science',
    'sgpa': 8.5
}
}
```

### **Output:**

```
name, branch, sgpa = extract_student_info(student_data)
print(f"Full Name: {name}")
print(f"Branch: {branch}")
print(f"SGPA: {sgpa}")
```