

AI ASSISTED CODING

ASSIGNMENT - 6.3

Name: Varshitha. G

HtNo: 2303A51103

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.

Expected Output #1:

- Correct implementation that lists Automorphic numbers using both loop types, with explanation.

```
1 #Generate a function that displays all automorphic numbers between 1 to 1000 using a for loop
2 def is_automorphic(num):
3     square=str(num**2)
4     return square.endswith(str(num))
5 l=[]
6 def display_automorphic_numbers():
7     for i in range(1, 1001):
8         if is_automorphic(i):
9             l.append(i)
10 display_automorphic_numbers()
11 print("Automorphic numbers between 1 and 1000 are:", l)
12
13 # using while loop
14 def is_automorphic(num):
15     square=str(num**2)
16     return square.endswith(str(num))
17 l=[]
18 def display_automorphic_numbers():
19     i = 1
20     while i <= 1000:
21         if is_automorphic(i):
22             l.append(i)
23         i += 1
24 display_automorphic_numbers()
25 print("Automorphic numbers between 1 and 1000 are:", l)
26
27
```

PROBLEMS	OUTPUT	TERMINAL	PORTS
<p>625</p> <p>PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Pyth</p> <p>Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]</p> <p>Automorphic numbers between 1 and 1000 are: [1, 5, 6, 25, 76, 376, 625]</p> <p>PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> █</p>			

Both versions work correctly and produce the same output by finding all automorphic numbers from 1 to 1000.

Both run equally fast because they loop 1000 times, use the same logic, and store results in a list.

Time Complexity: Overall: $O(n)$. Since $n = 1000$ is fixed, it behaves like $O(1)$ for this problem.

The for loop is better because it is cleaner, easier to read, safer (no infinite loop risk), and clearly shows a fixed number of iterations.

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:

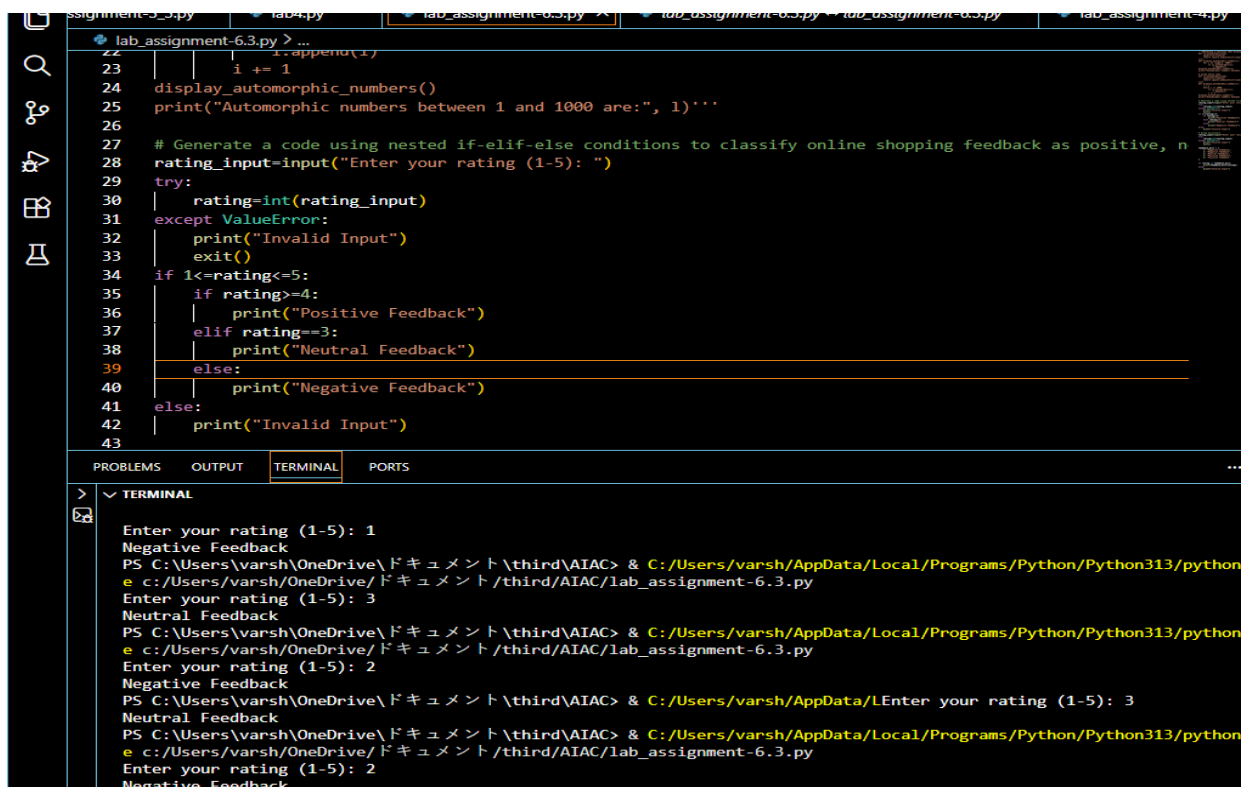
- o Generate initial code using nested if-elif-else.

- o Analyze correctness and readability.

- o Ask AI to rewrite using dictionary-based or match-case structure.

Expected Output #2:

- Feedback classification function with explanation and an alternative approach.



```
lab_assignment-6.3.py > ...
22         i.append(i)
23         i += 1
24     display_automorphic_numbers()
25     print("Automorphic numbers between 1 and 1000 are:", l)'''
26
27 # Generate a code using nested if-elif-else conditions to classify online shopping feedback as positive, n
28 rating_input=input("Enter your rating (1-5): ")
29 try:
30     rating=int(rating_input)
31 except ValueError:
32     print("Invalid Input")
33     exit()
34 if 1<=rating<=5:
35     if rating==4:
36         print("Positive Feedback")
37     elif rating==3:
38         print("Neutral Feedback")
39     else:
40         print("Negative Feedback")
41 else:
42     print("Invalid Input")
43
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
Enter your rating (1-5): 1
Negative Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python
e c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter your rating (1-5): 3
Neutral Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python
e c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter your rating (1-5): 2
Negative Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/LEnter your rating (1-5): 3
Neutral Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python
e c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter your rating (1-5): 2
Negative Feedback
```

```
44 # Using dictionary
45 rating_input=input("Enter your rating (1-5): ")
46 try:
47     rating=int(rating_input)
48 except ValueError:
49     print("Invalid Input")
50     exit()
51
52 feedback_dict = {
53     1: "Negative Feedback",
54     2: "Negative Feedback",
55     3: "Neutral Feedback",
56     4: "Positive Feedback",
57     5: "Positive Feedback"
58 }
59
60 if rating in feedback_dict:
61     print(feedback_dict[rating])
62 else:
63     print("Invalid Input")
```

PROBLEMS OUTPUT **TERMINAL** PORTS

> ▼ TERMINAL

```
Negative Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/L/third/AIAC
Enter your rating (1-5): 2
Negative Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/LNegative Fe
ocal/Programs/Python/Python313/python.exe c:/Users/varsh/OneDrive/ドキュメント/third/AIAC
Enter your rating (1-5): 5
Positive Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Progra
e c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter your rating (1-5): 3
Neutral Feedback
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Progra
e c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter your rating (1-5): -9
Invalid Input
Enter your rating (1-5): 0
Invalid Input
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> |
```

Dictionary-based approach is more suitable for this program.

The dictionary approach is cleaner and easier to read than long if-elif-else chains.

It is faster because dictionary works in $O(1)$ time.

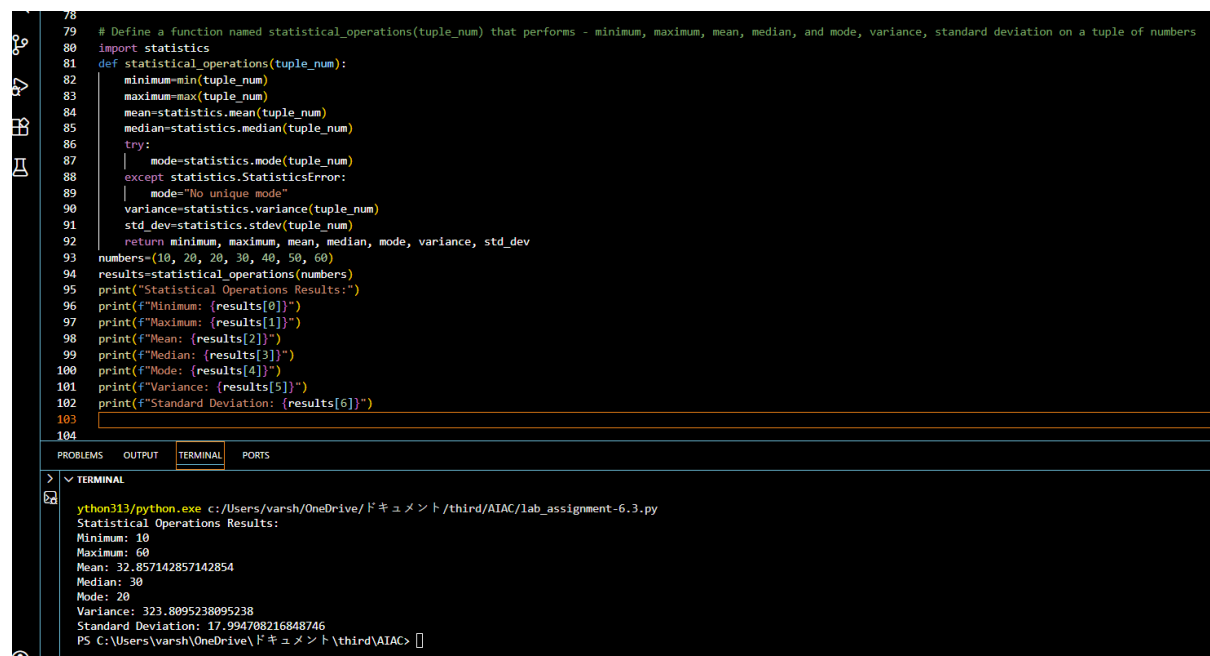
It is easy to maintain—adding or changing values only needs updating the dictionary.

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



```
78
79 # Define a function named statistical_operations(tuple_num) that performs - minimum, maximum, mean, median, and mode, variance, standard deviation on a tuple of numbers
80 import statistics
81 def statistical_operations(tuple_num):
82     minimum=min(tuple_num)
83     maximum=max(tuple_num)
84     mean=statistics.mean(tuple_num)
85     median=statistics.median(tuple_num)
86     try:
87         mode=statistics.mode(tuple_num)
88     except statistics.StatisticsError:
89         mode="No unique mode"
90     variance=statistics.variance(tuple_num)
91     std_dev=statistics.stdev(tuple_num)
92     return minimum, maximum, mean, median, mode, variance, std_dev
93 numbers=(10, 20, 20, 30, 40, 50, 60)
94 results=statistical_operations(numbers)
95 print("Statistical Operations Results:")
96 print(f"Minimum: {results[0]}")
97 print(f"Maximum: {results[1]}")
98 print(f"Mean: {results[2]}")
99 print(f"Median: {results[3]}")
100 print(f"Mode: {results[4]}")
101 print(f"Variance: {results[5]}")
102 print(f"Standard Deviation: {results[6]}")
103
104
```

PROBLEMS OUTPUT TERMINAL PORTS

TERMINAL

```
ython313/python.exe c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Statistical Operations Results:
Minimum: 10
Maximum: 60
Mean: 32.857142857142854
Median: 30
Mode: 20
Variance: 323.8095238095238
Standard Deviation: 17.994708216848746
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC>
```

Task 4: Teacher Profile

- Prompt: 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.

```
104
105 # Create a class teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.
106 class Teacher:
107     def __init__(self, teacher_id, name, subject, experience):
108         self.teacher_id = teacher_id
109         self.name = name
110         self.subject = subject
111         self.experience = experience
112
113     def display_details(self):
114         print(f"Teacher ID: {self.teacher_id}")
115         print(f"Name: {self.name}")
116         print(f"Subject: {self.subject}")
117         print(f"Experience: {self.experience} years")
118 teacher1 = Teacher(1, "Alice Smith", "Mathematics", 10)
119 teacher1.display_details()
120 teacher2 = Teacher(2, "Bob Johnson", "Science", 8)
121 teacher2.display_details()
122 teacher3 = Teacher(3, "Cathy Brown", "English", 5)
123 teacher3.display_details()
```

PROBLEMS OUTPUT **TERMINAL** PORTS

▼ TERMINAL

```
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Users/varsh/OneDrive/ird/AIAC/lab_assignment-6.3.py
Teacher ID: 1
Name: Alice Smith
Subject: Mathematics
Experience: 10 years
Teacher ID: 2
Name: Bob Johnson
Subject: Science
Experience: 8 years
Teacher ID: 3
Name: Cathy Brown
Subject: English
Experience: 5 years
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> □
```

Task #5 – Zero-Shot Prompting with Conditional Validation

Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.

Requirements: The function must ensure the mobile number: o Starts with 6, 7, 8, or 9 , Contains exactly 10 digits

Expected Output: A valid Python function that performs all required validations without using any input-output examples in the prompt.

```
125
126 # Generate a function that validates an Indian mobile number (10 digits, starts with 6, 7, 8, or 9)
127 import re
128 def validate_mobile_number(mobile_number):
129     pattern = r'^[6-9]\d{9}$'
130     if re.match(pattern, mobile_number):
131         return "Valid Indian Mobile Number"
132     else:
133         return "Invalid Indian Mobile Number"
134 mobile_number_input = input("Enter an Indian mobile number: ")
135 result = validate_mobile_number(mobile_number_input)
136 print(result)
```

PROBLEMS OUTPUT **TERMINAL** PORTS

▼ TERMINAL

```
Enter an Indian mobile number: 6123045879
Valid Indian Mobile Number
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/U
Drive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter an Indian mobile number: 0000000000
Invalid Indian Mobile Number
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/U
Drive/ドキュメント/third/AIAC/lab_assignment-6.3.py
Enter an Indian mobile number: 5
Invalid Indian Mobile Number
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> □
```

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.

```
139
140 def is_armstrong(num):
141     order = len(str(num))
142     sum_of_powers = sum(int(digit) ** order for digit in str(num))
143     return sum_of_powers == num
144 start_range = int(input("Enter the start of the range: "))
145 end_range = int(input("Enter the end of the range: "))
146 armstrong_numbers = []
147 for number in range(start_range, end_range + 1):
148     if is_armstrong(number):
149         armstrong_numbers.append(number)
150 print(f"Armstrong numbers between {start_range} and {end_range} are: {armstrong_numbers}")
```

PROBLEMS	OUTPUT	TERMINAL	PORTS
<p>▼ TERMINAL</p> <pre>PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python39-64/python.exe c:/Users/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py c:\Users\varsh\OneDrive\ドキュメント\third\AIAC\lab_assignment-6.3.py:127: SyntaxWarning: invalid escape sequence '\d' '''import re Enter the start of the range: 1 Enter the end of the range: 1000 Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407] PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> █</pre>			

```
169 def find_armstrong_numbers_generator(start, end):
170     """
171     Memory-efficient version using generator.
172     Yields Armstrong numbers one at a time instead of storing all in memory.
173     """
174     for num in range(start, end + 1):
175         if is_armstrong(num):
176             yield num
177
178 # Input with validation
179 while True:
180     try:
181         start_range = int(input("Enter the start of the range: "))
182         end_range = int(input("Enter the end of the range: "))
183         if start_range > end_range:
184             print("Error: Start range should be less than or equal to end range.")
185             continue
186         break
187     except ValueError:
188         print("Error: Please enter valid integers.")
189
190 # Using optimized list comprehension approach
191 armstrong_numbers = find_armstrong_numbers(start_range, end_range)
192 print(f"Armstrong numbers between {start_range} and {end_range} are: {armstrong_numbers}")
193
194 # Alternative: Memory-efficient generator approach (uncomment to use)
195 # armstrong_numbers_gen = find_armstrong_numbers_generator(start_range, end_range)
196 # print(f"Armstrong numbers between {start_range} and {end_range} are: {list(armstrong_numbers_gen)}")
```

PROBLEMS OUTPUT **TERMINAL** PORTS

> **TERMINAL**

```
'''import re
Enter the start of the range: 1
Enter the end of the range: 1000
Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC>
```

Input validation: Avoids errors when the range is invalid or reversed.

Separate function: `find_armstrong_numbers()` improves reusability and keeps code organized.

Error handling: try-except ensures the program doesn't go wrong on wrong input.

Comments make the logic easy to understand and maintain.

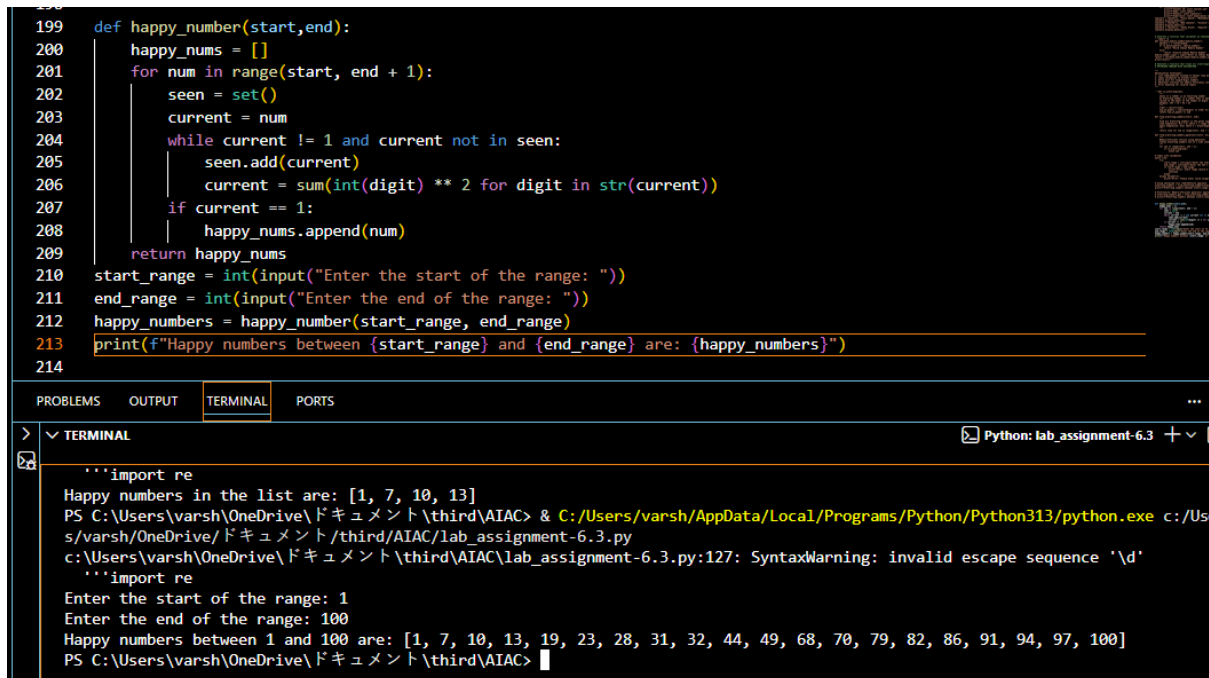
Task Description #7 (Loops – Happy Numbers in a Range)

Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).

Instructions:

- Implement the logic using a loop: repeatedly replace a number with the sum of the squares of its digits until the result is either 1 (Happy Number) or enters a cycle (Not Happy).

- Validate correctness by checking known Happy Numbers (e.g., 1, 7, 10, 13, 19, 23, 28...).
- Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).



```

199 def happy_number(start,end):
200     happy_nums = []
201     for num in range(start, end + 1):
202         seen = set()
203         current = num
204         while current != 1 and current not in seen:
205             seen.add(current)
206             current = sum(int(digit) ** 2 for digit in str(current))
207         if current == 1:
208             happy_nums.append(num)
209     return happy_nums
210 start_range = int(input("Enter the start of the range: "))
211 end_range = int(input("Enter the end of the range: "))
212 happy_numbers = happy_number(start_range, end_range)
213 print(f"Happy numbers between {start_range} and {end_range} are: {happy_numbers}")
214

```

PROBLEMS OUTPUT **TERMINAL** PORTS

Python: lab_assignment-6.3

```

> ✓ TERMINAL
'''import re
Happy numbers in the list are: [1, 7, 10, 13]
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python313/python.exe c:/Us
s/varsh/OneDrive/ドキュメント/third/AIAC/lab_assignment-6.3.py
c:\Users\varsh\OneDrive\ドキュメント\third\AIAC\lab_assignment-6.3.py:127: SyntaxWarning: invalid escape sequence '\d'
'''import re
Enter the start of the range: 1
Enter the end of the range: 100
Happy numbers between 1 and 100 are: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC>

```

Expected Output #8

- Python program that prints all Happy Numbers within a range.
- Optimized version using cycle detection with explanation.


```
def is_happy(num, seen=None):
    """
    Check if a number is happy using set-based cycle detection.

    Args:
        num: The number to check
        seen: Set tracking visited numbers in sequence

    Returns:
        True if happy (reaches 1), False if unhappy (cycle detected)
    Time Complexity: O(log n) - sequence converges quickly
    Space Complexity: O(1) - set size is bounded (typically < 20 elements)
    """
    if seen is None:
        seen = set()
    # Base case: reached 1 (happy number)
    if num == 1:
        return True
    # Cycle detected: number appears again in sequence (unhappy)
    if num in seen:
        return False
    # Add current number to set to track visited numbers
    seen.add(num)
    # Compute sum of squares of digits
    next_num = sum(int(digit) ** 2 for digit in str(num))
    # Recursively check next number with same set
    return is_happy(next_num, seen)

def find_happy_numbers_optimized(numbers_list):
    """
    Find all happy numbers in a list using list comprehension and cycle detection.

    Args:
        numbers_list: List of numbers to check

    Returns:
        List of happy numbers
    Time Complexity: O(n * log m) where n = list size, m = avg number value
    Space Complexity: O(k) where k = happy numbers found
    """
```

```
    return [num for num in numbers_list if is_happy(num)]

# Input validation: Get range from user
while True:
    try:
        start_range = int(input("Enter the start of the range: "))
        end_range = int(input("Enter the end of the range: "))

        # Validate range
        if start_range > end_range:
            print("Error: Start range should be less than or equal to end range.")
            continue
        if start_range < 1:
            print("Error: Range must contain positive integers.")
            continue
        break
    except ValueError:
        print("Error: Please enter valid integers.")

# Generate list from range
numbers_list = list(range(start_range, end_range + 1))
# Find happy numbers using optimized function
happy_numbers = find_happy_numbers_optimized(numbers_list)
print(f"\nHappy numbers between {start_range} and {end_range} are: {happy_numbers}")

# Example trace for unhappy number (2):
print("\nExample - Cycle detection for 2:")
print("2 → 4 → 16 → 37 → 58 → 89 → 145 → 42 → 20 → 4 (CYCLE DETECTED! Set prevents infinite loop)")
print("\nExample - Happy number sequence for 7:")
print("7 → 49 → 97 → 130 → 10 → 1 (HAPPY! Reaches 1)")
```

Cycle detection using a set: Already-visited numbers are stored, if a number repeats, it means a loop and the number is not happy.

Using a separate `is_happy()` function makes the code cleaner and easier to read.

Shorter and faster way to build lists compared to manual `append()`.

Ensures only valid, positive numbers and correct ranges are processed.

Each number finishes quickly because digit-square sums reduce fast, avoiding long loops.

Task 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.

```
275
276 def strong_numbers(start, end):
277     strong_nums = []
278     for num in range(start, end + 1):
279         sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
280         if sum_of_factorials == num:
281             strong_nums.append(num)
282     return strong_nums
283 def factorial(n):
284     if n == 0 or n == 1:
285         return 1
286     result = 1
287     for i in range(2, n + 1):
288         result *= i
289     return result
290 start_range = int(input("Enter the start of the range: "))
291 end_range = int(input("Enter the end of the range: "))
292 strong_numbers_list = strong_numbers(start_range, end_range)
293 print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers_list}")
```

PROBLEMS OUTPUT **TERMINAL** PORTS

> **TERMINAL**

```
'''import re
Enter the start of the range: 1
Enter the end of the range: 200
Strong numbers between 1 and 200 are: [1, 2, 145]
Strong numbers between 1 and 200 are: [1, 2, 145]
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC>
```

```
288 FACTORIAL_MAP = {}
289     0: 1,
290     1: 1,
291     2: 2,
292     3: 6,
293     4: 24,
294     5: 120,
295     6: 720,
296     7: 5040,
297     8: 40320,
298     9: 362880
299 }
300 def is_strong(num):
301     sum_of_factorials = sum(FACTORIAL_MAP[int(digit)] for digit in str(num))
302     return sum_of_factorials == num
303
304 def find_strong_numbers(start, end):
305     return [num for num in range(start, end + 1) if is_strong(num)]
306
307 # Input validation
308 while True:
309     try:
310         start_range = int(input("Enter the start of the range: "))
311         end_range = int(input("Enter the end of the range: "))
312         # Validate range
313         if start_range > end_range:
314             print("Error: Start range should be less than or equal to end range.")
315             continue
316         if start_range < 1:
317             print("Error: Range must contain positive integers.")
318             continue
319         break
320     except ValueError:
321         print("Error: Please enter valid integers.")
322
323 # Find and display strong numbers
324 strong_numbers_list = find_strong_numbers(start_range, end_range)
325 print(f"\nStrong numbers between {start_range} and {end_range} are: {strong_numbers_list}")
326
```

PROBLEMS OUTPUT **TERMINAL** PORTS

▼ **TERMINAL**

```
Strong numbers between 1 and 1000 are: [1, 2, 145]

Examples of strong numbers:
1 = 1! = 1
2 = 2! = 2
145 = 1! + 4! + 5! = 1 + 24 + 120 = 145
40585 = 4! + 0! + 5! + 8! + 5! = 24 + 1 + 120 + 40320 + 120 = 40585
PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> 
```

Precomputed factorials: Factorials of digits 0–9 are calculated once and stored, avoiding repeated work.

Factorials are fetched directly from a dictionary in $O(1)$ time instead of recalculating.

Cleaner and faster way to build the result list than using loops.

Checks the range and handles errors to prevent incorrect inputs.

Task #9 – 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

```
325 '''
326
327 '''
328 {
329     "name": {"first": "Aarav", "last": "Sharma"},
330     "branch": "CSE",
331     "sgpa": 9.1
332 }
333 display ("Aarav Sharma", "CSE", 9.1)
334
335 {
336     "student": {
337         "name": {"first": "Neha", "last": "Patel"},
338         "details": {
339             "branch": "ECE",
340             "sgpa": 8.6
341         }
342     }
343 }
344 display ("Neha Patel", "ECE", 8.6)'''
345 def student_info(student_dict):
346     try:
347         if "student" in student_dict:
348             first_name = student_dict["student"]["name"]["first"]
349             last_name = student_dict["student"]["name"]["last"]
350             branch = student_dict["student"]["details"]["branch"]
351             sgpa = student_dict["student"]["details"]["sgpa"]
352         else:
353             first_name = student_dict["name"]["first"]
354             last_name = student_dict["name"]["last"]
355             branch = student_dict["branch"]
356             sgpa = student_dict["sgpa"]
357         full_name = f"{first_name} {last_name}"
358         print(f"Name: {full_name}, Branch: {branch}, SGPA: {sgpa}")
359     except KeyError as e:
360         print(f"Missing key in dictionary: {e}")
361     student1 = {
362         "name": {"first": "Aarav", "last": "Sharma"},
363         "branch": "CSE",
364         "sgpa": 9.1
365     }
366     student2 = {
367         "student": {
368             "name": {"first": "Neha", "last": "Patel"},
369             "details": {
370                 "branch": "ECE",
371                 "sgpa": 8.6
372             }
373         }
374     }
375     student_info(student1)
```

PROBLEMS	OUTPUT	TERMINAL	PORTS
<div>> TERMINAL</div> <div>Name: Aarav Sharma, Branch: CSE, SGPA: 9.1 Name: Neha Patel, Branch: ECE, SGPA: 8.6 Name: Neha Patel, Branch: ECE, SGPA: 8.6</div>			

Task Description #10 (Loops – Perfect Numbers in a Range)

Task: Generate a function using AI that displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000).

Instructions:

- A Perfect Number is a positive integer equal to the sum of its proper divisors (excluding itself).
 - o Example: $6 = 1 + 2 + 3$, $28 = 1 + 2 + 4 + 7 + 14$.
- Use a for loop to find divisors of each number in the range.
- Validate correctness with known Perfect Numbers (6, 28, 496...).
- Ask AI to regenerate an optimized version (using divisor check only up to \sqrt{n}).

```
378 def perfect_numbers(start, end):
379     perfect_nums = []
380     for num in range(start, end + 1):
381         if num < 2:
382             continue
383         divisors_sum = sum(i for i in range(1, num) if num % i == 0)
384         if divisors_sum == num:
385             perfect_nums.append(num)
386     return perfect_nums
387 start_range = int(input("Enter the start of the range: "))
388 end_range = int(input("Enter the end of the range: "))
389 perfect_nums_list = perfect_numbers(start_range, end_range)
390 print(f"Perfect numbers between {start_range} and {end_range} are: {perfect_nums_list}")
```

PROBLEMS OUTPUT **TERMINAL** PORTS

> **TERMINAL**

perfect

SyntaxError: '{' was never closed

PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC> & C:/Users/varsh/AppData/Local/Programs/Python/Python311/Python.exe C:\Users\varsh\OneDrive\ドキュメント\third\AIAC\lab_assignment-6.3.py

c:\Users\varsh\OneDrive\ドキュメント\third\AIAC\lab_assignment-6.3.py:127: SyntaxWarning: invalid escape sequence '\r'

'''import re

Enter the start of the range: 1

Enter the end of the range: 100

Perfect numbers between 1 and 100 are: [6, 28]

```
390 def is_perfect(num):
391     if num < 2:
392         return False
393     divisors_sum = 1
394     i = 2
395     while i * i <= num:
396         if num % i == 0:
397             divisors_sum += i
398             if i != num // i:
399                 divisors_sum += num // i
400             i += 1
401     return divisors_sum == num
402 def find_perfect_numbers(start, end):
403     return [num for num in range(start, end + 1) if is_perfect(num)]
404 while True:
405     try:
406         start_range = int(input("Enter the start of the range: "))
407         end_range = int(input("Enter the end of the range: "))
408
409         # Validate range
410         if start_range > end_range:
411             print("Error: Start range should be less than or equal to end range.")
412             continue
413         if start_range < 1:
414             print("Error: Range must contain positive integers.")
415             continue
416         break
417     except ValueError:
418         print("Error: Please enter valid integers.")
419 # Find and display perfect numbers
420 perfect_nums_list = find_perfect_numbers(start_range, end_range)
421 print(f"\nPerfect numbers between {start_range} and {end_range} are: {perfect_nums_list}")
```

PROBLEMS	OUTPUT	TERMINAL	PORTS
<p>> TERMINAL</p> <p>Enter the start of the range: 1 Enter the end of the range: 1000</p> <p>Perfect numbers between 1 and 1000 are: [6, 28, 496]</p> <p>PS C:\Users\varsh\OneDrive\ドキュメント\third\AIAC></p>			

Square root optimization: Divisors are checked only up to \sqrt{n} , making the code much faster.

When a number divides n , both the divisor and its pair (n / i) are counted together.

Ensures the square root is added only once for perfect squares.

Cleaner and faster way to collect results than manual loops.

Handles invalid inputs safely and avoids runtime errors.