

AI ASSISTED CODING

ASSIGNMENT-6.3

Sony Dodla

2303A51666

Batch-23

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.

```
2 #generate all automorphic numbers within a given range using for loop
3 import time as t
4 def is_automorphic(num):
5     square = num * num
6     num_str = str(num)
7     square_str = str(square)
8     return square_str.endswith(num_str)
9
10 def generate_automorphic_numbers(start, end):
11     automorphic_numbers = []
12     for i in range(start, end + 1):
13         if is_automorphic(i):
14             automorphic_numbers.append(i)
15     return automorphic_numbers
16 # Example usage
17 start_time = t.time()
18 start_range = 1
19 end_range = 100
20 result = generate_automorphic_numbers(start_range, end_range)
21 print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
22 end_time = t.time()
23 print(f"Execution time: {end_time - start_time} seconds")
```

```
● PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment(6.3).py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.00017976760864257812 seconds
```

```

25 #generate all auomorphic numbeers within a given range using while loop
26 import time as t
27 def is_automorphic(num):
28     square = num * num
29     num_str = str(num)
30     square_str = str(square)
31     return square_str.endswith(num_str)
32 def generate_automorphic_numbers(start, end):
33     automorphic_numbers = []
34     i = start
35     while i <= end:
36         if is_automorphic(i):
37             automorphic_numbers.append(i)
38         i += 1
39     return automorphic_numbers
40 # Example usage
41 start_time = t.time()
42 start_range = 1
43 end_range = 100
44 result = generate_automorphic_numbers(start_range, end_range)
45 print(f"Automorphic numbers between {start_range} and {end_range}: {result}")
46 end_time = t.time()
47 print(f"Execution time: {end_time - start_time} seconds")

```

```

• PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment(6.3).py"
Automorphic numbers between 1 and 100: [1, 5, 6, 25, 76]
Execution time: 0.0002658367156982422 seconds

```

1. They use very little memory since only a small list of automorphic numbers is stored.
2. A for loop is faster because it is optimized inside Python.
3. A for loop makes the code cleaner and easier to understand because no manual counter is needed.
4. A for loop is safer and less error-prone since it avoids skipping numbers or running into infinite loops.

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.

```
49 #generate a nested if-elif-else to classify shopping feedback as positive negative or neutral based
50 # on rating (1-5)
51 def classify_feedback(rating):
52     if rating >= 4:
53         return "Positive"
54     elif rating == 3:
55         return "Neutral"
56     else:
57         return "Negative"
58 # Example usage
59 ratings = [5, 4, 3, 2, 1]
60 for rating in ratings:
61     classification = classify_feedback(rating)
62     print(f"Rating: {rating}, Feedback: {classification}")
63 #rewrite the above code using dictionary mapping
64 def classify_feedback_dict(rating):
65     feedback_map = {
66         5: "Positive",
67         4: "Positive",
68         3: "Neutral",
69         2: "Negative",
70         1: "Negative"
71     }
72     return feedback_map.get(rating, "Invalid rating")
73 # Example usage
74 ratings = [5, 4, 3, 2, 1]
75 for rating in ratings:
76     classification = classify_feedback_dict(rating)
77     print(f"Rating: {rating}, Feedback: {classification}")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment(6.3).py"
Rating: 5, Feedback: Positive
Rating: 4, Feedback: Positive
Rating: 3, Feedback: Neutral
Rating: 2, Feedback: Negative
Rating: 1, Feedback: Negative
```

1. A dictionary makes things clear because all rating-to-feedback pairs are visible in one place.
2. It is easy to maintain since you can add or change ratings without changing the main logic.
3. It handles invalid ratings more safely by letting you check or give a default message.
4. It follows Python's standard and clean way of mapping values, so the code looks natural.
5. It stays clean and readable even if the number of rating categories increases.

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

```
79 #generate a function named statistical_operations that takes a list of numbers as input and calculate
80 # minimum,maximum,mean,median ,mode ,standard deviation,variance of the list
81 import statistics as stats
82 def statistical_operations(numbers):
83     minimum = min(numbers)
84     maximum = max(numbers)
85     mean = stats.mean(numbers)
86     median = stats.median(numbers)
87     try:
88         mode = stats.mode(numbers)
89     except stats.StatisticsError:
90         mode = "No unique mode"
91     variance = stats.variance(numbers)
92     std_deviation = stats.stdev(numbers)
93
94     return {
95         "Minimum": minimum,
96         "Maximum": maximum,
97         "Mean": mean,
98         "Median": median,
99         "Mode": mode,
100        "Variance": variance,
101        "Standard Deviation": std_deviation
102    }
103
104 # Example usage
105 data = [10, 20, 20, 30, 40, 50]
106 results = statistical_operations(data)
107 for key, value in results.items():
108     print(f"{key}: {value}")
```

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment(6.3).py"
Minimum: 10
Maximum: 50
Mean: 28.333333333333332
Median: 25.0
Mode: 20
Variance: 216.66666666666666
Standard Deviation: 14.719601443879744
```

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.

```
66 #Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to
67 # display teacher details.
68 class Teacher:
69     def __init__(self, teacher_id, name, subject, experience):
70         self.teacher_id = teacher_id
71         self.name = name
72         self.(variable) experience: Any
73         self.experience = experience
74
75     def display_details(self):
76         print(f"Teacher ID: {self.teacher_id}")
77         print(f"Name: {self.name}")
78         print(f"Subject: {self.subject}")
79         print(f"Experience: {self.experience} years")
80
81 # Example usage
82 teacher = Teacher(1, "Alice Smith", "Mathematics", 10)
83 teacher.display_details()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "C:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

Teacher ID: 1
Name: Alice Smith
Subject: Mathematics
Experience: 10 years

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

```
84 ##generate a python code that validates an Indian mobile number.starts with 6,7,9 or 9
85 import re
86 def validate_indian_mobile_number(mobile_number):
87     pattern = r'^[6-9]\d{9}$'
88     if re.match(pattern, mobile_number):
89         return True
90     else:
91         return False
92 # Example usage
93 mobile_number = input("Enter an Indian mobile number: ")
94 if validate_indian_mobile_number(mobile_number):
95     print("Valid Indian mobile number.")
96 else:
97     print("Invalid Indian mobile number.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "C:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

Enter an Indian mobile number: 7093374718
Valid Indian mobile number.

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "C:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

Enter an Indian mobile number: 567890123
Invalid Indian mobile number.

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.

```

171 #Write a python code that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).using f
172 def is_armstrong_number(num):
173     (function) def armstrong_numbers_in_range(
174         start: Any,
175         end: Any
176     ) -> list                                for digit in num_str)
177 def armstrong_numbers_in_range(start, end):
178     armstrong_numbers = []
179     for num in range(start, end + 1):
180         if is_armstrong_number(num):
181             armstrong_numbers.append(num)
182     return armstrong_numbers
183 start_range = 1
184 end_range = 1000
185 result = armstrong_numbers_in_range(start_range, end_range)
186 print(f"Armstrong numbers between {start_range} and {end_range}: {result}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + - [] [X] ... | [C] X

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\assignment-6.3(task3).py:87: SyntaxWarning: invalid escape sequence '\d'

pattern = r'^[6-9]\d{9}\$'

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

1. It does the same work in fewer lines, so there is less overhead and faster execution.
2. It calculates the digit powers directly instead of using extra variables, reducing unnecessary steps.
3. List comprehension is faster than manually appending values in a loop.
4. It avoids storing extra temporary values, so memory usage stays low.
5. The code is cleaner and easier to read while giving the same correct result.

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

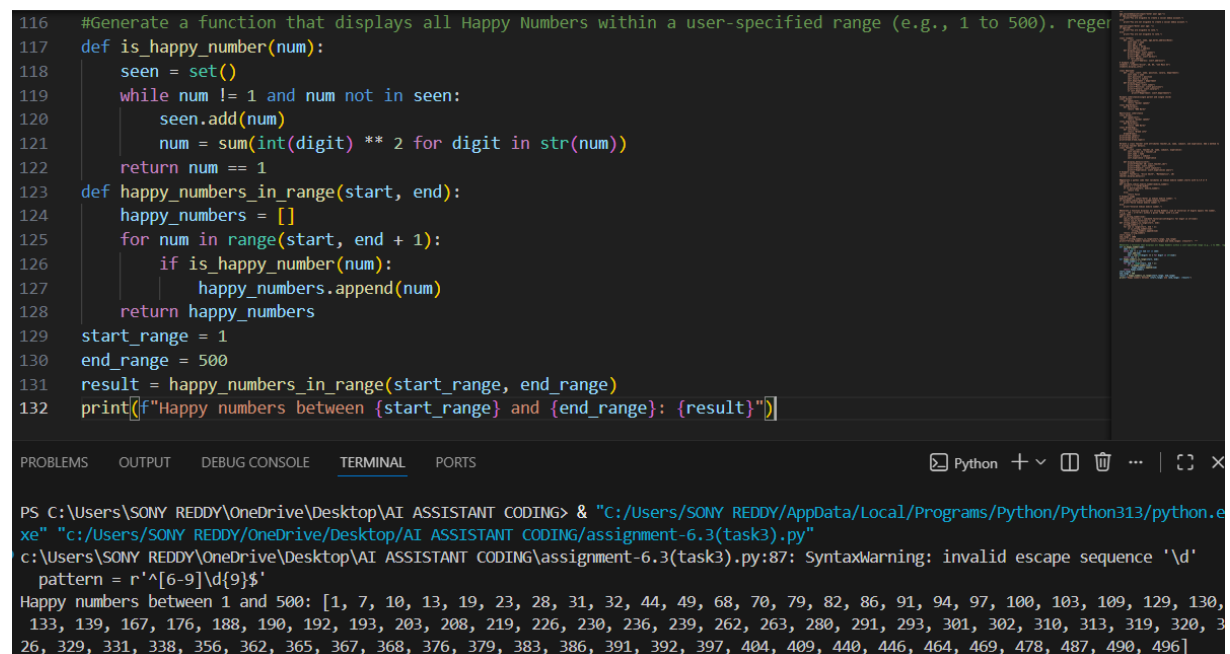
Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).

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).

Expected Output #8:

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



```
116 #Generate a function that displays all Happy Numbers within a user-specified range (e.g., 1 to 500). reger
117 def is_happy_number(num):
118     seen = set()
119     while num != 1 and num not in seen:
120         seen.add(num)
121         num = sum(int(digit) ** 2 for digit in str(num))
122     return num == 1
123 def happy_numbers_in_range(start, end):
124     happy_numbers = []
125     for num in range(start, end + 1):
126         if is_happy_number(num):
127             happy_numbers.append(num)
128     return happy_numbers
129 start_range = 1
130 end_range = 500
131 result = happy_numbers_in_range(start_range, end_range)
132 print(f"Happy numbers between {start_range} and {end_range}: {result}")
```

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\assignment-6.3(task3).py:87: SyntaxWarning: invalid escape sequence '\d'

pattern = r'^[6-9]\d{9}\$'

Happy numbers between 1 and 500: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100, 103, 109, 129, 130, 133, 139, 167, 176, 188, 190, 192, 193, 203, 208, 219, 226, 230, 236, 239, 262, 263, 280, 291, 293, 301, 302, 310, 313, 319, 320, 326, 329, 331, 338, 356, 362, 365, 367, 368, 376, 379, 383, 386, 391, 392, 397, 404, 409, 440, 446, 464, 469, 478, 487, 490, 496]

1. It saves results of numbers already checked, so the same calculations are not done again and again.
2. This avoids repeated loops for the same digit sequences, which makes it much faster for large ranges.
3. Using a set still prevents infinite loops, but caching adds an extra speed boost.

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.

```

99 #Generate a function displays all Strong Numbers (sum of factorial of digits equals the number,
00 # e.g., 145 = 1!+4!+5!) within a given range. with 1,2,145
01 import math
02 def is_strong_number(num):
03     sum_of_factorials = sum(math.factorial(int(digit)) for digit in str(num))
04     return sum_of_factorials == num
05 def strong_numbers_in_range(start, end):
06     strong_numbers = []
07     for num in range(start, end + 1):
08         if is_strong_number(num):
09             strong_numbers.append(num)
10     return strong_numbers
11 start_range = 1
12 end_range = 1000
13 result = strong_numbers_in_range(start_range, end_range)
14 print(f"Strong numbers between {start_range} and {end_range}: {result}")

```

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"
 c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\assignment-6.3(task3).py:87: SyntaxWarning: invalid escape sequence '\d'
 pattern = r'^[6-9]\d{9}\$'
 Strong numbers between 1 and 500: [1, 2, 145]

Time Complexity Comparison:

For larger ranges, this makes a significant performance difference since you're trading a tiny bit of extra memory (10 dictionary entries) for substantial computation savings across the entire loop.

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:

- o Full Name

- o Branch o

SGPA

Expected Output

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

```
134 #generate a python code that parses a nested dictionary representing student information.
135 #the dictionary contains Full Name Branch SGPA
136 def parse_student_info(student_dict):
137     for student_id, info in student_dict.items():
138         full_name = info.get("Full Name", "N/A")
139         branch = info.get("Branch", "N/A")
140         sgpa = info.get("SGPA", "N/A")
141         print(f"Student ID: {student_id}")
142         print(f"Full Name: {full_name}")
143         print(f"Branch: {branch}")
144         print(f"SGPA: {sgpa}")
145         print("-" * 20)
146 # Example usage
147 students = {
148     101: {"Full Name": "John Doe", "Branch": "Computer Science", "SGPA": 8.5},
149     102: {"Full Name": "Jane Smith", "Branch": "Electrical Engineering", "SGPA": 9.0},
150     103: {"Full Name": "Alice Johnson", "Branch": "Mechanical Engineering", "SGPA": 8.8},
151 }
152 parse_student_info(students)
```

PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"

Student ID: 101
Full Name: John Doe
Branch: Computer Science
SGPA: 8.5
SGPA: 8.5

Student ID: 102
Full Name: Jane Smith
Branch: Electrical Engineering
SGPA: 9.0

Student ID: 103
Full Name: Alice Johnson
Branch: Mechanical Engineering
SGPA: 8.8

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 root n

```

154 #Generate a python code displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000). usi
155 def is_perfect_number(num):
156     if num < 2:
157         return False
158     divisors_sum = sum(i for i in range(1, num) if num % i == 0)
159     return divisors_sum == num
160 def perfect_numbers_in_range(start, end):
161     perfect_numbers = []
162     for num in range(start, end + 1):
163         if is_perfect_number(num):
164             perfect_numbers.append(num)
165     return perfect_numbers
166 start_range = 1
167 end_range = 1000
168 result = perfect_numbers_in_range(start_range, end_range)
169 print(f"Perfect numbers between {start_range} and {end_range}: {result}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING> & "C:/Users/SONY REDDY/AppData/Local/Programs/Python/Python313/python.exe" "c:/Users/SONY REDDY/OneDrive/Desktop/AI ASSISTANT CODING/assignment-6.3(task3).py"
c:\Users\SONY REDDY\OneDrive\Desktop\AI ASSISTANT CODING\assignment-6.3(task3).py:87: SyntaxWarning: invalid escape sequence '\d'
  pattern = r'^[6-9]\d{9}$'
Perfect numbers between 1 and 1000: [6, 28, 496]
```