AI ASSISTANCE CODING

LAB ASSIGNMENT – 6.3

NAME: CH VISHWANATH RAO

BATCH:02

HT NO: 2303A51095

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 Analyse 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.

PROMPT:

```
# write a python program that displays all Automorphic numbers between 1 and 1000 using a for loop without list
```

FOR LOOP CODE:

```
import time
start_time = time.time()
def  is_automorphic(num):
    square = num * num
    return str(square).endswith(str(num))
for i in range(1, 1001):
    if is_automorphic(i):
        print(f"the num {i} is automorphic number")
end_time = time.time()
print(f"Time taken: {end_time - start_time} seconds")
```

PROMPT:

```
#generate THE code also FOR  while loop
```

WHILE LOOP CODE:

```
import time
start_time = time.time()
def is_automorphic(num):
    square = num * num
    return str(square).endswith(str(num))
i = 1
while i <= 1000:
    if is_automorphic(i):
        print(f"the num {i} is automorphic number")
    i += 1
end_time = time.time()
print(f"Time taken: {end_time - start_time} seconds")
```

Outputs:

For loop:-

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task1.p
the num 1 is automorphic number
the num 5 is automorphic number
the num 6 is automorphic number
the num 25 is automorphic number
the num 76 is automorphic number
the num 376 is automorphic number
the num 625 is automorphic number
Time taken: 0.0007503032684326172 seconds
```

While loop:-

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task1.p
the num 1 is automorphic number
the num 5 is automorphic number
the num 6 is automorphic number
the num 25 is automorphic number
the num 76 is automorphic number
the num 376 is automorphic number
the num 625 is automorphic number
Time taken: 0.0007503032684326172 seconds
```

Comparision:

Both for loop and while loop implementations are correct.

for loop: Cleaner, shorter, and preferred for fixed ranges.

while loop: More control, but requires manual increment (i += 1).

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.

PROMPT:

```
#generate a python program to write nested if-elif-else conditions to classify online
# shopping feedback as Positive, Neutral, or Negative based on a
# numerical rating (1-5).
```

Code (if elif else):

```
number = int(input("Enter your rating (1-5): "))
if number == 1:
    print("Negative feedback")
elif number == 2:
    print("Neutral feedback")
elif number == 3:
    print("Neutral feedback")
elif number == 4:
    print("Positive feedback")
elif number == 5:
    print("Positive feedback")
else:
    print("Invalid rating. Please enter a number between 1 and 5.")
```

PROMPT:

```
#generate the  python basic code to classify online
# shopping feedback as Positive, Neutral, or Negative based on a
# numerical rating (1-5) using dictionary-based to do the task.
```

CODE (DICTIONARY BASED):

```
try:
    rating = int(input("Enter your rating (1-5): "))
    feedback_dict = {
        1: "Negative feedback",
        2: "Neutral feedback",
        3: "Neutral feedback",
        4: "Positive feedback",
        5: "Positive feedback"
    }
    feedback = feedback_dict.get(rating, "Invalid rating. Please enter a number between 1 and 5.")
    print(feedback)
except ValueError:
    print("Invalid input. Please enter a numerical value between 1 and 5.")
```

OUTPUTS:

IF ELIF ELSE:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> &
Enter your rating (1-5): 1
Negative feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> &
Enter your rating (1-5): 2
Neutral feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> &
Enter your rating (1-5): 4
Positive feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> &
Enter your rating (1-5): 5
Positive feedback
```

DICTIONARY

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
Enter your rating (1-5): 5
Positive feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
Enter your rating (1-5): 3
Neutral feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
Enter your rating (1-5): 4
Positive feedback
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
Enter your rating (1-5): 2
Neutral feedback
```

CORRECTNESS AND READABILITY:

- Classifies a numeric rating into three categories (Negative, Neutral, Positive) by direct mapping.
- **Why dict works:** Dictionary lookup is O(1) and clearly expresses one-to-one mapping between inputs and labels.
- **Alternate approach:** Use conditional ranges for threshold-based classification or numeric modulus/rules if categories depend on ranges rather than exact values.

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 python code to 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
```

Code:

```python
import statistics
def statistical_operations(tuple_num):
    if len(tuple_num) == 0:
        return "The tuple is empty."

    minimum = min(tuple_num)
    maximum = max(tuple_num)
    mean = statistics.mean(tuple_num)
    median = statistics.median(tuple_num)
    try:
        mode = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        mode = "No unique mode found"
    variance = statistics.variance(tuple_num)
    std_deviation = statistics.stdev(tuple_num)

    return {
        "Minimum": minimum,
        "Maximum": maximum,
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Variance": variance,
        "Standard Deviation": std_deviation
    }

# Example usage:
numbers = (12, 23, 22, 33, 42, 50, 23, 23)
results = statistical_operations(numbers)
print(results)
```

Output:

```
/task3.py"
{'Minimum': 12, 'Maximum': 50, 'Mean': 28.5, 'Median': 23.0, 'Mode': 23, 'Variance': 152.85714285714286, 'Standard Deviation': 12.3635408705250
32}
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py
/task3.py"
```

This function uses Python’s built-in statistics module, so the results are accurate. Built-in functions make the code simple and reduce mistakes. The program is easy to read and understand. Handling the mode with exception checking prevents errors when there is no single mode.

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 initialiser, method, and object creation.

PROMPT:

```
#generate a python program to create class teacher with attributes teacher_id,name,subject,and experience and
# add a method to display teacher details
```

CODE:-

```
class Educator:
    def __init__(self, educator_id, full_name, specialization, years_of_experience):
        self.educator_id = educator_id
        self.full_name = full_name
        self.specialization = specialization
        self.years_of_experience = years_of_experience

    def show_information(self):
        print(f"Educator ID: {self.educator_id}")
        print(f"Full Name: {self.full_name}")
        print(f"Specialization: {self.specialization}")
        print(f"Years of Experience: {self.years_of_experience} years")
# Example usage
educator1 = Educator(2, "suresh singh", "Computer Science", 8)
educator1.show_information()
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task4.py"
Educator ID: 2
Full Name: suresh singh
Specialization: Computer Science
Years of Experience: 8 years
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

The Teacher class clearly shows basic object-oriented programming. It uses a constructor to set values and a method to display details. The code is simple, well-structured, and easy to understand. It follows standard Python class rules and is good for learning OOP basics.

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

o Contains exactly 10 digits

Expected Output

• A valid Python function that performs all required validations

without using any input-output examples in the prompt.

PROMPT:

```
# write a python code to function number starts with 6 or 7 or 8 or 9 and it should be
# excatly 10 digit number then print that number is vaild and indian number
# otherwise print invalid number
```

Code:

```
def valid_ind_phone_number(cont_number):
    if len(cont_number) == 10 and cont_number[0] in '6789' and cont_number.isdigit():
        print(f"{cont_number} is a valid Indian phone num.")
    else:
        print(f"{cont_number} is an invalid phone num.")
# Example usage
valid_ind_phone_number("9347332712")
```

Output:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task5.py"
9347332712 is a valid Indian phone num.
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

The function checks an Indian mobile number by verifying its length, starting digit, and whether it contains only numbers. The logic is correct and works efficiently. As no examples are provided, it follows the zero-shot prompting approach correctly.

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 optimised version (using list comprehensions).

Expected Output #7:

• Python program listing Armstrong numbers in the range.

• Optimised version with explanation.

PROMPT:

```
#generate a python program  that print armstrong number form user specific range
```

Code:

```
def check_armstrong(num):
    length = len(str(num))
    total = sum(int(digit) ** length for digit in str(num))
    return num == total
def find_armstrongs(start, end):
    results = []
    for n in range(start, end + 1):
        if check_armstrong(n):
            results.append(n)
    return results
# Example usage
start_num = int(input("Enter start range: "))
end_num = int(input("Enter end range: "))
armstrongs = find_armstrongs(start_num, end_num)
print(f"Armstrong numbers between {start_num} and {end_num} are: {armstrongs}")
```

Output:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST
Enter start range: 1
Enter end range: 100
Armstrong numbers between 1 and 100 are: [1, 2, 3, 4, 5, 6, 7, 8, 9]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST
Enter start range: 1
Enter end 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\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

The program checks an Armstrong number by comparing the number with the sum of its digits raised to the power of the total digits. Using list comprehension makes the code shorter and easier to read, while still giving correct and efficient results.

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.

• Optimised version using cycle detection with explanation.

PROMPT:

```
#generate a python program to print all happy numbers from user specific range
```

CODE:

```
def check_happy_number(number):
    encountered = set()
    while number != 1 and number not in encountered:
        encountered.add(number)
        number = sum(int(digit) ** 2 for digit in str(number))
    return number == 1

def find_happy_numbers_range(start, end):
    happy_nums_list = []
    for n in range(start, end + 1):
        if check_happy_number(n):
            happy_nums_list.append(n)
    return happy_nums_list

# Example usage
begin_range = int(input("Enter the starting range: "))
finish_range = int(input("Enter the ending range: "))
happy_numbers = find_happy_numbers_range(begin_range, finish_range)
```

OUTPUTS:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task7.py"
Enter the starting range: 1
Enter the ending range: 1000
Happy numbers between 1 and 1000 are: [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, 536, 556, 563, 565, 566, 608, 617, 622, 623, 632, 635, 637, 638, 644, 649, 653, 655, 656, 665, 671, 673, 680, 683, 694, 700, 709, 716, 7
36, 739, 748, 761, 763, 784, 790, 793, 802, 806, 818, 820, 833, 836, 847, 860, 863, 874, 881, 888, 899, 901, 904, 907, 910, 912, 913, 921, 923, 931, 932, 937, 940, 946, 964, 970,
 973, 989, 998, 1000]
```

OPTIMAL CODE:

PROMPT:

```
#GENERATE A optimal code to print all happy numbers from user specific range using function and return type
```

CODE:

```
def is_happy(num):
    seen = set()
    while num != 1 and num not in seen:
        seen.add(num)
        num = sum(int(digit) ** 2 for digit in str(num))
    return num == 1

def happy_numbers_range(start, end):
    happy_numbers = []
    for number in range(start, end + 1):
        if is_happy(number):
            happy_numbers.append(number)
    return happy_numbers

# Example usage
start_range = int(input("Enter start range: "))
end_range = int(input("Enter end range: "))
happy_nums = happy_numbers_range(start_range, end_range)
print(f"Happy numbers between {start_range} and {end_range} are: {happy_nums}")
```

OUTPUT:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task7.py"
Enter start range: 1
Enter end range: 1000
Happy numbers between 1 and 1000 are: [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, 536, 556, 563, 565, 566, 608, 617, 622, 623, 632, 635, 637, 638, 644, 649, 653, 655, 656, 665, 671, 673, 680, 683, 694, 700, 709, 716, 7
36, 739, 748, 761, 763, 784, 790, 793, 802, 806, 818, 820, 833, 836, 847, 860, 863, 874, 881, 888, 899, 901, 904, 907, 910, 912, 913, 921, 923, 931, 932, 937, 940, 946, 964, 970,
 973, 989, 998, 1000]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

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 optimised version (precompute digit factorials).

Expected Output #9:

• Python program that lists Strong Numbers.

• Optimised version with explanation.

PROMPT:

```
#generate a python function to display all strong number from user specificrange
```

Code:

```
def is_strong(num):
    factorial = lambda n: 1 if n == 0 else n * factorial(n - 1)
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
    return num == sum_of_factorials
def strong_numbers_range(start, end):
    strong_numbers_list = []
    for number in range(start, end + 1):
        if is_strong(number):
            strong_numbers_list.append(number)
    return strong_numbers_list
# Example usage
start_range = int(input("Enter start range: "))
end_range = int(input("Enter end range: "))
print(strong_numbers_range(start_range, end_range))
print(f"Strong numbers between {start_range} and {end_range} are: {strong_numbers_range(start_range, end_range)}")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task8.py"
Enter start range: 1
Enter end range: 200
[1, 2, 145]
Strong numbers between 1 and 200 are: [1, 2, 145]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task8.py"
Enter start range: 1
Enter end range: 500
[1, 2, 145]
Strong numbers between 1 and 500 are: [1, 2, 145]
```

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.

PROMPT:-

```
# nested dictionary has student personal full_name = "dhoni" and student
# academic branch = "CSE", sgpa = 8.6.
# output is ("Rahul Sharma", "CSE", 8.6).
# nested dictionary has student personal full_name = "jadeja and student
# academic branch = "ECE", sgpa = 9.1.
# expected output should return ("Anita Verma", "ECE", 9.1)
```

Code:

```
student_info = {···

student_info2 = {
    "personal": {
        "full_name": "jadeja"
    },
    "academic": {
        "branch": "ECE",
        "sgpa": 9.1
    }
}

def get_student_details(student_dict):
    name = student_dict["personal"]["full_name"]
    branch = student_dict["academic"]["branch"]
    sgpa = student_dict["academic"]["sgpa"]
    return (name, branch, sgpa)

# Example usage
details1 = get_student_details(student_info)
details2 = get_student_details(student_info2)
print(details1)
print(details2)
```

Output:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/Task9.py"
('dhoni', 'CSE', 8.6)
('jadeja', 'ECE', 9.1)
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

The few-shot prompt uses examples of nested dictionaries and their outputs. By learning from these examples, the function correctly finds and extracts the needed values. This shows how examples help the AI understand the structure even without detailed instructions.

Task10: 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 optimised version (using divisor check only up to √n).

PROMPT:

```
#generate a  basic python code that print all perfect numbers from user specific range
```

Code:

```python
def check_perfect(n):
    if n < 2:
        return False
    total = 1
    i = 2
    while i * i <= n:
        if n % i == 0:
            total += i
            if i != n // i:
                total += n // i
        i += 1
    return total == n

def find_perfects(low, high):
    results = []
    for value in range(low, high + 1):
        if check_perfect(value):
            results.append(value)
    return results

try:
    low_bound = int(input("Enter lower bound: "))
    high_bound = int(input("Enter upper bound: "))
except ValueError:
    print("Invalid input: please enter integer values.")
else:
    perfects = find_perfects(low_bound, high_bound)
    print(perfects)
    print(f"Perfect numbers between {low_bound} and {high_bound}: {perfects}")
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                                    Python

PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task10.py"
Enter lower bound: 1
Enter upper bound: 1000
[6, 28, 496]
Perfect numbers between 1 and 1000: [6, 28, 496]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

Optimal code:

Prompt:

```
#write an optimal python code that print all perfect numbers from user specific range using function and return type
```

Code:

```python
def check_perfect(n: int) -> bool:
    if n < 2:
        return False
    total = 1
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            total += i
            if i != n // i:
                total += n // i
    return total == n
def find_perfects(low: int, high: int) -> list:
    perfect_numbers = []
    for value in range(low, high + 1):
        if check_perfect(value):
            perfect_numbers.append(value)
    return perfect_numbers
# Example usage
start_range = int(input("Enter start range: "))
end_range = int(input("Enter end range: "))
perfect_nums = find_perfects(start_range, end_range)
print(f"Perfect numbers between {start_range} and {end_range} are: {perfect_nums}")
```

Outputs:

```
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task10.py"
Enter start range: 1
Enter end range: 500
Perfect numbers between 1 and 500 are: [6, 28, 496]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING> & C:/Python314/python.exe "c:/Users/chvis/OneDrive/Desktop/AI ASSIST CODING/lab 29-1-26.py/task10.py"
Enter start range: 1
Enter end range: 1000
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\chvis\OneDrive\Desktop\AI ASSIST CODING>
```

The basic solution checks all divisors, but it is slow because it loops up to *n*. The optimized version is faster because it avoids unnecessary checks and uses better logic with list comprehension, making it suitable for large numbers.