

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	Academic Year:2025-2026
Instructor(s) Name		S Naresh Kumar	
Course Code	24CS002PC215	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R24
Date and Day of Assignment	Week3 – wednesday	Time(s)	
Duration	2 Hours	Applicable to Batches	
Assignment Number:6.1(Present assignment number)/24(Total number of assignments)			
Q.No.	Question		Expected Time to complete
	Lab 6: AI-Based Code Completion – Classes, Loops, and Conditionals Lab Objectives: <ul style="list-style-type: none"> To explore AI-powered auto-completion features for core Python constructs. To analyze how AI suggests logic for class definitions, loops, and conditionals. To evaluate the completeness and correctness of code generated by AI assistants. Lab Outcomes (LOs): After completing this lab, students will be able to: <ul style="list-style-type: none"> Use AI tools to generate and complete class definitions and methods. Understand and assess AI-suggested loops for iterative tasks. Generate conditional statements through prompt-driven suggestions. Critically evaluate AI-assisted code for correctness and clarity. 		Week3 - Monday
1	Task Description #1 (Loops – Automorphic Numbers in a Range) <ul style="list-style-type: none"> Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop. Instructions: <ul style="list-style-type: none"> Get AI-generated code to list Automorphic numbers using a for 		

loop.

- Analyze the correctness and efficiency of the generated logic.
- 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

Genrate a lists of automorphic numbers in a given range

Code

Using “for Loop”

```
import time as t
def is_automorphic(num):
    square = num * num
    num_str = str(num)
    square_str = str(square)
    return square_str.endswith(num_str)
def automorphic_numbers_in_range(start, end):
    automorphic_numbers = []
    for num in range(start, end + 1):
        if is_automorphic(num):
            automorphic_numbers.append(num)
    return automorphic_numbers
start_time = t.time()
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
automorphic_numbers = automorphic_numbers_in_range(start_range, end_range)
print(f"Automorphic numbers between {start_range} and {end_range}: {automorphic_numbers}")
end_time = t.time()
print(f"Time taken: {end_time - start_time} seconds\n")
```

	<p>Output</p> <pre> Enter the start of the range: 1 Enter the end of the range: 1000 Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625] Time taken: 9.609323501586914 seconds Using while loop: Enter the start of the range: 1 Enter the end of the range: 1000 Automorphic numbers between 1 and 1000: [1, 5, 6, 25, 76, 376, 625] Time taken: 2.9865434169769287 seconds </pre> <p>Explanation</p> <p><i>The program finds Automorphic numbers by checking whether the square of a number ends with the number itself using loop logic.</i></p>	
2	<p>Task Description #2 (Conditional Statements – Online Shopping Feedback Classification)</p> <ul style="list-style-type: none"> 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: <ul style="list-style-type: none"> Generate initial code using nested if-elif-else. Analyze correctness and readability. Ask AI to rewrite using dictionary-based or match-case structure. <p>Expected Output #2:</p> <ul style="list-style-type: none"> Feedback classification function with explanation and an alternative approach. <p>Prompt</p> <pre> Genrate a online shopping feedback system using nested if-elif-else to categorize feedback into positive, negative, and neutral based on 1-5 rating scale. </pre> <p>Code</p> <pre> def categorize_feedback(rating): if rating >= 4 and rating <= 5: return "Positive" elif rating == 3: </pre>	

	<pre> return "Neutral" elif rating >= 1 and rating <= 2: return "Negative" else: return "Invalid rating. Please provide a rating between 1 and 5." # Example usage ratings = [5, 4, 3, 2, 1, 6, 0] for rating in ratings: category = categorize_feedback(rating) print(f"Rating: {rating} - Feedback Category: {category}")</pre> <p>Output</p> <pre>Time Complexity: O(1) - The function performs a constant number of operations regardless of input size. Space Complexity: O(1) - The function uses a constant amount of space for variables</pre> <pre>Rating: 5 - Feedback Category: Positive Rating: 4 - Feedback Category: Positive Rating: 3 - Feedback Category: Neutral Rating: 2 - Feedback Category: Negative Rating: 1 - Feedback Category: Negative Rating: 6 - Feedback Category: Invalid rating. Please provide a rating between 1 and 5. Rating: 0 - Feedback Category: Invalid rating. Please provide a rating between 1 and 5.</pre> <p>Explanation</p> <p>The feedback classification uses conditional statements to correctly label ratings as Negative, Neutral, or Positive based on given values.</p>	
3	<p>Task 3: Statistical_operations</p> <p>Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:</p> <ul style="list-style-type: none">• Minimum, Maximum• Mean, Median, Mode• Variance, Standard Deviation <p>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</p>	

Code

```
import statistics

def statistical_operations(tuple_num):
    if not tuple_num:
        return {}

    nums = list(tuple_num)

    min_val = min(nums)
    max_val = max(nums)
    mean_val = statistics.mean(nums)
    median_val = statistics.median(nums)

    try:
        mode_val = statistics.mode(nums)
    except statistics.StatisticsError:
        mode_val = None

    variance_val = statistics.variance(nums)
    stdev_val = statistics.stdev(nums)

    return {
        'minimum': min_val,
        'maximum': max_val,
        'mean': mean_val,
        'median': median_val,
        'mode': mode_val,
        'variance': variance_val,
        'standard_deviation': stdev_val
    }

# Example usage
if __name__ == "__main__":
    sample_tuple = (1, 2, 3, 4, 5, 5, 6)
    results = statistical_operations(sample_tuple)
    print("Statistical Operations Results:")
    for key, value in results.items():
        print(f"{key.capitalize()}: {value}")
```

	<p>Output</p> <pre>Statistical Operations Results: Minimum: 1 Maximum: 6 Mean: 3.7142857142857144 Median: 4 Mode: 5 Variance: 3.238095238095238 Standard deviation: 1.7994708216848747</pre> <p>Explanation</p> <p><i>The function uses Python built-in statistics methods to compute minimum, maximum, mean, median, mode, variance, and standard deviation accurately.</i></p>	
4	<p>Task 4: Teacher Profile</p> <ul style="list-style-type: none">• 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. <p>Prompt</p> <pre>Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.</pre> <p>Code</p> <pre>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}")</pre>	

	<pre> print(f"Experience: {self.experience} years") # Example usage if __name__ == "__main__": teacher1 = Teacher(101, "Alice Johnson", "Mathematics", 10) teacher2 = Teacher(102, "Bob Smith", "Science", 8) print("Teacher 1 Details:") teacher1.display_details() print("\nTeacher 2 Details:") teacher2.display_details()</pre> <p>Output</p> <pre>Teacher 1 Details: Teacher ID: 101 Name: Alice Johnson Subject: Mathematics Experience: 10 years Teacher 2 Details: Teacher ID: 102 Name: Bob Smith Subject: Science Experience: 8 years</pre> <p>Explanation</p> <p><i>The Teacher class demonstrates object oriented programming by initializing attributes through a constructor and displaying details using a class method.</i></p>	
5	<p>Task #5 – Zero-Shot Prompting with Conditional Validation</p> <p>Use zero-shot prompting to instruct an AI tool to generate a function that validates an Indian mobile number.</p> <p>Requirements</p> <ul style="list-style-type: none">• The function must ensure the mobile number:<ul style="list-style-type: none">○ 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.

Prompt

create a filter out the indian mobile numbers from the given text file the condition is that the number should start with 6,7,8 or 9 and it should be of 10 digit long create some set of examples

Code

```
import re
def filter_indian_mobile_numbers(file_path):
    indian_mobile_numbers = []
    pattern = re.compile(r'\b[6-9]\d{9}\b')

    with open(file_path, 'r') as file:
        content = file.read()
        matches = pattern.findall(content)
        indian_mobile_numbers.extend(matches)

    return indian_mobile_numbers
# Example usage
if __name__ == "__main__":
    file_path = 'mobile_numbers.txt' # Replace
with your file path
    filtered_numbers =
filter_indian_mobile_numbers(file_path)
    print("Filtered Indian Mobile Numbers:")
    for number in filtered_numbers:
        print(number)
```

Output

```
• Filtered Indian Mobile Numbers:
9876543210
9876543210
8765432109
6123456789
7123456789
8123456789
9123456789
```

Explanation

using mobile numbers txt file it will filter out the all the indian numbers. in txt file i have already created a set of [numbers](#). it is filter out using condition that which i have given in range 6 - 9 starting numbers and

	contains 10 digits long . it will read txt file which i have given an filer it out all the number as per my conduction.	
6	<p>Task Description #6 (Loops – Armstrong Numbers in a Range)</p> <p>Task: Write a function using AI that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).</p> <p>Instructions:</p> <ul style="list-style-type: none"> • 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). <p>Expected Output #7:</p> <ul style="list-style-type: none"> • Python program listing Armstrong numbers in the range. • Optimized version with explanation. <p>Prompt</p> <pre>Write a function that finds all Armstrong numbers in a user-specified range (e.g., 1 to 1000).</pre> <p>Code</p> <pre>def is_armstrong(num): num_str = str(num) num_digits = len(num_str) sum_of_powers = sum(int(digit) ** num_digits for digit in num_str) return sum_of_powers == num def armstrong_numbers_in_range(start, end): armstrong_numbers = [] for num in range(start, end + 1): if is_armstrong(num): armstrong_numbers.append(num) return armstrong_numbers # Example usage if __name__ == "__main__": start_range = int(input("Enter the start of the range: ")) end_range = int(input("Enter the end of the range: ")) armstrong_numbers = armstrong_numbers_in_range(start_range, end_range)</pre>	

```

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

print("\n")
print("Recreating using while loop:")
'''regenerate the using list comprehensions'''
def armstrong_numbers_in_range_list_comp(start,
end):
    return [num for num in range(start, end + 1)
if is_armstrong(num)]
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of
the range: "))
    end_range = int(input("Enter the end of the
range: "))
    armstrong_numbers =
armstrong_numbers_in_range_list_comp(start_range,
end_range)
    print(f"Armstrong numbers between
{start_range} and {end_range} (using list
comprehension): {armstrong_numbers}")
# Analysis:
# Time Complexity: O(n * d) - where n is the
number of numbers in the range and d is the number
of digits in the largest number. Each number
requires checking each digit.
# Space Complexity: O(k) - where k is the number
of Armstrong numbers found in the

```

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]

Recreating using while loop:
Enter the start of the range: 1
Enter the end of the range: 1000
Armstrong numbers between 1 and 1000 (using list comprehension): [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]

```

Explanation

The program identifies Armstrong numbers by comparing each number

	with the sum of its digits raised to the power of total digits.	
7	<p>Task Description #7 (Loops – Happy Numbers in a Range)</p> <p>Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).</p> <p>Instructions:</p> <ul style="list-style-type: none"> 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). <p>Expected Output #8:</p> <ul style="list-style-type: none"> Python program that prints all Happy Numbers within a range. <pre> Genrate a all Happy Numbers within a user-specified range (e.g., 1 to 500) using functions. </pre> <p>Code</p> <pre> import time as t def is_happy_number(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 start_time = t.time() def happy_numbers_in_range(start, end): happy_numbers = [] for num in range(start, end + 1): if is_happy_number(num): happy_numbers.append(num) return happy_numbers # Example usage if __name__ == "__main__": start_range = int(input("Enter the start of the range: ")) end_range = int(input("Enter the end of the </pre>	

```

range: "))
    happy_numbers =
happy_numbers_in_range(start_range, end_range)
    print(f"Happy numbers between {start_range}
and {end_range}: {happy_numbers}")
end_time = t.time()
print(f"Time taken: {end_time - start_time}
seconds")

```

Output

```

Enter the end of the range: 500
Happy numbers between 1 and 500: [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 9
4, 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]
Time taken: 4.249208688735962 seconds

```

```

Recreating an optimized version:
Enter the start of the range: 1
Enter the end of the range: 500
Happy numbers between 1 and 500 (optimized): [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 8
2, 86, 91, 94, 97, 100, 103, 109, 129, 130, 133, 139, 167, 176, 188, 190, 192, 193, 203, 208, 219, 22
6, 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]
Time taken (optimized): 3.5933306217193604 seconds

```

- Optimized version using cycle detection with explanation.

```

'''regenerate an optimized version using a set to
detect cycles instead of infinite loops'''
def is_happy_number_optimized(num):
    seen = set()
    while num != 1:
        if num in seen:
            return False
        seen.add(num)
        num = sum(int(digit) ** 2 for digit in
str(num))
    return True
start_time = t.time()
def happy_numbers_in_range_optimized(start, end):
    happy_numbers = []
    for num in range(start, end + 1):
        if is_happy_number_optimized(num):
            happy_numbers.append(num)
    return happy_numbers
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of

```

	<pre>the range: ")) end_range = int(input("Enter the end of the range: ")) happy_numbers = happy_numbers_in_range_optimized(start_range, end_range) print(f"Happy numbers between {start_range} and {end_range} (optimized): {happy_numbers}") end_time = t.time() print(f"Time taken (optimized): {end_time - start_time} seconds") # Analysis: # Time Complexity: O(n * m) - where n is the number of numbers in the # range and m is the average number of iterations to determine if a number is happy. # Space Complexity: O(k) - where k is the number of happy numbers found in the # range, as they are stored in a list.</pre> <p>Explanation <i>Happy numbers are detected by repeatedly summing the squares of digits and using a set to prevent infinite loops.</i></p>	
8	<p>Task Description #8 (Loops – Strong Numbers in a Range)</p> <p>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.</p> <p>Instructions:</p> <ul style="list-style-type: none">• Use loops to extract digits and calculate factorials.• Validate with examples (1, 2, 145).• Ask AI to regenerate an optimized version (precompute digit factorials). <p>Expected Output #9:</p> <ul style="list-style-type: none">• Python program that lists Strong Numbers. <p>Prompt</p>	

Generate all strong numbers within a user-specified range (e.g., 1 to 500) using functions.

Code

```
def factorial(n):
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
start_time = t.time()
def is_strong_number(num):
    original_num = num
    sum_of_factorials = 0
    while num > 0:
        digit = num % 10
        sum_of_factorials += factorial(digit)
        num //= 10
    return sum_of_factorials == original_num
def strong_numbers_in_range(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if is_strong_number(num):
            strong_numbers.append(num)
    return strong_numbers
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of the
range: "))
    end_range = int(input("Enter the end of the range:
"))
    strong_numbers =
strong_numbers_in_range(start_range, end_range)
    print(f"Strong numbers between {start_range} and
{end_range}: {strong_numbers}")
end_time = t.time()
print(f"Time taken: {end_time - start_time} seconds")
```

Output

```
• Enter the start of the range: 1
Enter the end of the range: 1000
Strong numbers between 1 and 1000: [1, 2, 145]
Time taken: 4.145917892456055 seconds

Recreating an optimized version:
Enter the start of the range: 1
Enter the end of the range: 1000
Strong numbers between 1 and 1000 (optimized): [1, 2, 145]
Time taken (optimized): 5.042802810668945 seconds
```

- Optimized version with explanation.

```
def is_strong_number_optimized(num,
factorial_cache={}):
    original_num = num
    sum_of_factorials = 0
    while num > 0:
        digit = num % 10
        if digit not in factorial_cache:
            factorial_cache[digit] =
factorial(digit)
        sum_of_factorials +=
factorial_cache[digit]
        num //= 10
    return sum_of_factorials == original_num
start_time = t.time()
def strong_numbers_in_range_optimized(start, end):
    strong_numbers = []
    for num in range(start, end + 1):
        if is_strong_number_optimized(num):
            strong_numbers.append(num)
    return strong_numbers
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of
the range: "))
    end_range = int(input("Enter the end of the
range: "))
    strong_numbers =
strong_numbers_in_range_optimized(start_range,
end_range)
    print(f"Strong numbers between {start_range}
and {end_range} (optimized): {strong_numbers}")
```

	<pre>end_time = t.time() print(f"Time taken (optimized): {end_time - start_time} seconds")</pre> <p>Explanation</p> <p><i>The function checks whether a number equals the sum of factorials of its digits to identify Strong numbers.</i></p>	
9	<p>Task #9 – Few-Shot Prompting for Nested Dictionary Extraction</p> <p>Objective</p> <p>Use few-shot prompting (2–3 examples) to instruct the AI to create a function that parses a nested dictionary representing student information.</p> <p>Requirements</p> <ul style="list-style-type: none">• The function should extract and return:<ul style="list-style-type: none">○ Full Name○ Branch○ SGPA <p>Expected Output</p> <p>A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples</p> <p>Prompt</p> <pre>create a function that parses a nested dictionary representing student information. The function should extract and return: Full Name Branch SGPA Example Input: { "student1": { "name": {"first": "Mahesh", "last": "Ch"}, "branch": "CSE",</pre>	


```
        "SGPA": 9.1

    },

    "student2": {"name": {"first": "Rohan", "last":
"Kumar"},

        "branch": "ECE",

        "SGPA": 8.7

    }

}
```

Code

```
def extract_student_info(students_dict):

    extracted_info = []

    for student_key, student_data in
students_dict.items():

        first_name = student_data['name']['first']

        last_name = student_data['name']['last']

        full_name = f"{first_name} {last_name}"

        branch = student_data['branch']

        sgpa = student_data['SGPA']

        extracted_info.append({

            'Full Name': full_name,

            'Branch': branch,

            'SGPA': sgpa

        })

    return extracted_info

# Example usage

if __name__ == "__main__":

    students = {

        "student1": {

            "name": {"first": "Mahesh", "last": "Ch"},
```

```
        "branch": "CSE",
        "SGPA": 9.1
    },
    "student2": {
        "name": {"first": "Rohan", "last": "Kumar"},
        "branch": "ECE",
        "SGPA": 8.7
    },
    "student3": {
        "name": {"first": "Anita", "last":
"Sharma"},
        "branch": "ME",
        "SGPA": 8.9
    },
    "student4": {
        "name": {"first": "Priya", "last": "Singh"},
        "branch": "CE",
        "SGPA": 9.0
    },
    "student5": {
        "name": {"first": "Vikram", "last":
"Patel"},
        "branch": "EE",
        "SGPA": 8.5
    }
}

student_info = extract_student_info(students)

for info in student_info:
```

	<pre> print(info) # Analysis: # Time Complexity: O(n) - where n is the number of students in the dictionary. # Space Complexity: O(n) - where n is the number of students, as we store the extracted information in a list. </pre> <p>Output</p> <pre> {'Full Name': 'Mahesh Ch', 'Branch': 'CSE', 'SGPA': 9.1} {'Full Name': 'Rohan Kumar', 'Branch': 'ECE', 'SGPA': 8.7} {'Full Name': 'Anita Sharma', 'Branch': 'ME', 'SGPA': 8.9} {'Full Name': 'Priya Singh', 'Branch': 'CE', 'SGPA': 9.0} {'Full Name': 'Vikram Patel', 'Branch': 'EE', 'SGPA': 8.5} </pre> <p>Explanation</p> <p><i>The function navigates a nested dictionary structure to correctly extract student full name, branch, and SGPA.</i></p>	
10	<p>Task Description #10 (Loops – Perfect Numbers in a Range)</p> <p>Task: Generate a function using AI that displays all Perfect Numbers within a user-specified range (e.g., 1 to 1000).</p> <p>Instructions:</p> <ul style="list-style-type: none"> A Perfect Number is a positive integer equal to the sum of its proper divisors (excluding itself). <ul style="list-style-type: none"> 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}). <p>Expected Output #12:</p> <ul style="list-style-type: none"> Python program that lists Perfect Numbers in the given range. 	

Prompt

Genrate a perfect numbers within a user-specified range (e.g., 1 to 1000) using functions.

Code

```
import time as t
def is_perfect_number(num):
    if num < 2:
        return False
    sum_of_divisors = 1 # 1 is a proper divisor
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            sum_of_divisors += i
            if i != num // i:
                sum_of_divisors += num // i
    return sum_of_divisors == num
start_time = t.time()
def perfect_numbers_in_range(start, end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of
the range: "))
    end_range = int(input("Enter the end of the
range: "))
    perfect_numbers =
perfect_numbers_in_range(start_range, end_range)
    print(f"Perfect numbers between {start_range}
and {end_range}: {perfect_numbers}")
end_time = t.time()
print(f"Time taken: {end_time - start_time}
seconds")
```

Output

```
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000: [6, 28, 496]
Time taken: 5.176040172576904 seconds
```

```
Recreating an optimized version:
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000 (optimized): [6, 28, 496]
Time taken (optimized): 4.53221321105957 seconds
```

- Optimized version with explanation.

```
print("Recreating an optimized version:")
def is_perfect_number_optimized(num):
    if num < 2:
        return False
    sum_of_divisors = 1 # 1 is a proper divisor
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            sum_of_divisors += i
            if i != num // i:
                sum_of_divisors += num // i
    return sum_of_divisors == num
start_time = t.time()
def perfect_numbers_in_range_optimized(start,
end):
    perfect_numbers = []
    for num in range(start, end + 1):
        if is_perfect_number_optimized(num):
            perfect_numbers.append(num)
    return perfect_numbers
# Example usage
if __name__ == "__main__":
    start_range = int(input("Enter the start of
the range: "))
    end_range = int(input("Enter the end of the
range: "))
    perfect_numbers =
perfect_numbers_in_range_optimized(start_range,
end_range)
    print(f"Perfect numbers between {start_range}
and {end_range} (optimized): {perfect_numbers}")
end_time = t.time()
print(f"Time taken (optimized): {end_time -
```

```
start_time} seconds")
```

Explanation

Perfect numbers are identified by summing proper divisors efficiently by checking only up to the square root of the number.