

ASSIGNMENT - 06

Program Name: B. Tech

Assignment Type: Lab

Academic Year: 2025-2026

Instructor(s) Name : S Naresh Kumar

Course Title : AI Assisted Coding

Name : Aravind Reddy

Hall Ticket No : 2303a51027

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:
- Get AI-generated code to list Automorphic numbers using a for loop.
- Analyse 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.

Code :

```

3 # Task 1:
4
5
6 # generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.
7
8 def is_automorphic(num):
9     square = num * num
10    return str(square).endswith(str(num))
11
12 for i in range(1, 1001):
13    if is_automorphic(i):
14        print(f"{i} is an Automorphic number.")
15
16
17 print("\n")
18 # regenerate using a while loop and compare both implementations.
19
20 def is_automorphic(num):
21    square = num * num
22    return str(square).endswith(str(num))
23
24 i = 1
25 while i < 1001:
26    if is_automorphic(i):
27        print(f"{i} is an Automorphic number.")
28    i += 1
29
30 # By using time module compare the execution time of both implementations.
31
32 import time
33 start_for = time.time()
34 for i in range(1, 1001):
35    if is_automorphic(i):
36        pass
37 end_for = time.time()
38
39 start_while = time.time()
40 i = 1
41 while i < 1001:
42    if is_automorphic(i):
43        pass
44    i += 1
45 end_while = time.time()
46 print("For loop execution time: {end_for - start_for} seconds")
47 print("While loop execution time: {end_while - start_while} seconds")
48

```

① > OUTLINE
② > TIMELINE
③ PRETTY TYPESCRIPT ERROR

Output:

```

Open file in editor (cmd + click) G CONSOLE TERMINAL PORTS SPELL CHECKER 9
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 06.py"
● (base) → AI Assissted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 06.py"
1 is an Automorphic number.
5 is an Automorphic number.
6 is an Automorphic number.
25 is an Automorphic number.
76 is an Automorphic number.
376 is an Automorphic number.
625 is an Automorphic number.

1 is an Automorphic number.
5 is an Automorphic number.
6 is an Automorphic number.
25 is an Automorphic number.
76 is an Automorphic number.
376 is an Automorphic number.
625 is an Automorphic number.

For loop execution time: 0.0002410411834716797 seconds
While loop execution time: 0.0002779960632324219 seconds
○ (base) → AI Assissted Coding

```

Justification:

for loop being generally faster than while loop in Python due to optimisations in byte code execution.

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:**

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

Expected Output #2:

- Feedback classification function with explanation and an alternative approach.

Code:

The screenshot shows a dark-themed VS Code interface. The left sidebar has sections for Explorer, Source Control, Changes, AI Assisted Coding, and a list of assignments. The main editor area contains the following Python code:

```
55
56
57 # Task 2:
58 # write nested if-elif-else conditions to classify online shopping feedback as Positive, Neutral, or Negative based on a numerical rating (1-5).
59
60 rating = 4 # Example rating
61
62 if rating >= 4:
63     feedback = "Positive"
64 elif rating == 3:
65     feedback = "Neutral"
66 else:
67     feedback = "Negative"
68
69 print(f"The feedback is: {feedback}") # Output: The feedback is: Positive
70
71 # rewrite using dictionary-based or match-case structure.
72
73 feedback_dict = {
74     5: "Positive",
75     4: "Positive",
76     3: "Neutral",
77     2: "Negative",
78     1: "Negative"
79 }
80
81 feedback = feedback_dict.get(rating, "Invalid rating")
82 print("The feedback is: {feedback}") # Output: The feedback is: Positive
83
84 # Analyze correctness and readability.
85 # The first implementation using nested if-elif-else is straightforward and easy to understand for simple conditions.
86 # The second implementation using a dictionary provides a cleaner and more scalable approach, especially if more ratings need to be handled in the future.
87 # Both implementations are correct, but the dictionary-based approach may be more readable for larger sets of discrete values.
```

The status bar at the bottom shows: Ln 84, Col 38, Spaces: 4, UTF-8, LF, Python, 3.13.2, Go Live, Prettier.

Output:

The screenshot shows the terminal tab in VS Code with the following output:

```
PROBLEMS 9 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 9
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 06.py"
(base) ➜  AI Assisted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/My-Information/College/AI Assissted Coding/Assignments/Assignment - 06.py"
The feedback is: Positive
The feedback is: Positive
(base) ➜  AI Assisted Coding
```

Justification:

Analyse correctness and readability.

The first implementation using nested if-elif-else is straightforward and easy to understand for simple conditions.

The second implementation using a dictionary provides a cleaner and more scalable approach, especially if more ratings need to be handled in the future.

Both implementations are correct, but the dictionary-based approach may be more readable for larger sets of discrete 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.

Code :

```
92 # Task 3:
93 # Define a function named statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:
94
95 # Minimum, Maximum
96 # Mean, Median, Mode
97 # Standard Deviation, Variance
98
99
100 import statistics
101
102 def statistical_operations(tuple_num):
103     minimum = min(tuple_num)
104     maximum = max(tuple_num)
105     mean = statistics.mean(tuple_num)
106     median = statistics.median(tuple_num)
107     try:
108         mode = statistics.mode(tuple_num)
109     except statistics.StatisticsError:
110         mode = "No unique mode"
111     std_dev = statistics.stdev(tuple_num)
112     variance = statistics.variance(tuple_num)
113
114     return {
115         "Minimum": minimum,
116         "Maximum": maximum,
117         "Mean": mean,
118         "Median": median,
119         "Mode": mode,
120         "Standard Deviation": std_dev,
121         "Variance": variance
122     }
123
124 # Example usage
125 data = (1, 2, 3, 4, 5)
126 results = statistical_operations(data)
127 for key, value in results.items():
128     print(f"{key}: {value}")
129
130 # Output:
131
132 # Minimum: 1
133 # Maximum: 5
134 # Mean: 2.8333333333333335
135 # Median: 2.5
136 # Mode: 2
137 # Standard Deviation: 1.4719601443879744
138 # Variance: 2.1666666666666665
```

Output:

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop/Assignment - 06.py"
● (base) → AI Assisted Coding /usr/local/bin/python3 "/Users/aravindreddy/Desktop/Assignment - 06.py"
Minimum: 1
Maximum: 5
Mean: 2.8333333333333335
Median: 2.5
Mode: 2
Standard Deviation: 1.4719601443879744
Variance: 2.1666666666666665
○ (base) → AI Assisted Coding
```

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.

Code :

The screenshot shows the VS Code interface with the following details:

- EXPLORER**: Shows a tree view of files: AI ASSISTED CODING, vscode, Assignments, Assignment - 01.py, Assignment - 03.py, Assignment - 04.py, Assignment - 05.py, Assignment - 06.py, and Programs.
- SOURCE CONTROL: CHANGES**: Shows 144 changes.
- AI ASSISTED CODING**: Shows the current file: Assignment - 06.py.
- Program-4.py**: Another file listed in the sidebar.
- Assignment - 06.py**: The active file content is displayed in the main editor area:

```
144
145
146
147 # Task 4:
148 # Create a class Teacher with attributes teacher_id, name, subject, and experience. Add a method to display teacher details.
149
150 class Teacher:
151     def __init__(self, teacher_id, name, subject, experience):
152         self.teacher_id = teacher_id
153         self.name = name
154         self.subject = subject
155         self.experience = experience
156
157     def display_details(self):
158         return f"Teacher ID: {self.teacher_id}\n"
159         f"Name: {self.name}\n"
160         f"Subject: {self.subject}\n"
161         f"Experience: {self.experience} years"
162
163 # Example usage
164 teacher1 = Teacher(101, "Mr. Smith", "Mathematics", 10)
165 print(teacher1.display_details())
166 # Output:
167 # Teacher ID: 101
168 # Name: Mr. Smith
169 # Subject: Mathematics
170 # Experience: 10 years
171 teacher2 = Teacher(102, "Ms. Johnson", "Science", 8)
172 print(teacher2.display_details())
173 # Output:
174 # Teacher ID: 102
175 # Name: Ms. Johnson
176 # Subject: Science
177 # Experience: 8 years
```

- OUTLINE**: Shows the outline of the current file.
- TIMELINE**: Shows the timeline of changes.
- PRETTY TYPESCRIPT ERROR**: Shows a prettier typescript error.
- STATUS BAR**: Shows Ln 161, Col 56, Spaces: 4, UTF-8, LF, Python, 3.13.2, Go Live, Prettier.

Output :

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop  
● (base) → AI Assissted Coding /usr/local/bin/python  
Teacher ID: 101  
Name: Mr. Smith  
Subject: Mathematics  
Experience: 10 years  
Teacher ID: 102  
Name: Ms. Johnson  
Subject: Science  
Experience: 8 years  
○ (base) → AI Assissted Coding
```

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:**
 - 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.

Code :

The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows a tree view of files: Welcome, Program-4.py, Assignment - 06.py, AI ASSISTED CODING, .vscode, Assignments, Assignment - 01.py, Assignment - 3.5.py, Assignment - 03.py, Assignment - 04.py, Assignment - 06.py, and Programs.
- Source Control:** Changes pane shows modifications to Assignment - 06.py.
- Editor:** The main editor area contains Python code for creating a Teacher class and displaying its details. The code uses f-strings for string interpolation.
- Bottom Status Bar:** Shows file path (Assignment - 06.py), line 161, column 56, spaces: 4, encoding: UTF-8, LF, Python, Go Live, and Prettier icons.

Output:

```
/usr/local/bin/python3 "/Users/aravindreddy/Desktop
● (base) → AI Assissted Coding /usr/local/bin/python
Teacher ID: 101
Name: Mr. Smith
Subject: Mathematics
Experience: 10 years
Teacher ID: 102
Name: Ms. Johnson
Subject: Science
Experience: 8 years
○ (base) → AI Assissted Coding
```

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.

Code :

```
# Task 6:

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

# use only 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).

def is_armstrong(num):
    digits = [int(d) for d in str(num)]
    power = len(digits)
    sum_of_powers = sum(d ** power for d in digits)
    return sum_of_powers == num

armstrong_numbers = []

for i in range(1, 1001):
    if is_armstrong(i):
        armstrong_numbers.append(i)

print("Armstrong numbers between 1 and 1000 are:", armstrong_numbers)
# Output: Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
# Optimized version using list comprehensions

armstrong_numbers_optimized = [num for num in range(1, 1001) if is_armstrong(num)]
print("Optimized Armstrong numbers between 1 and 1000 are:", armstrong_numbers_optimized)
# Output: Optimized Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
```

Output:

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

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 optimised 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.

Code :

```

# Task 7:

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

# 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).

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

happy_numbers = []

for i in range(1, 501):
    if is_happy(i):
        happy_numbers.append(i)
print("Happy numbers between 1 and 500 are:", happy_numbers)

# Output: Happy numbers between 1 and 500 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, 191, 193, 203, 208, 219, 226, 230, 236, 239, 241, 250, 259, 268, 277, 286, 295, 304, 313, 322, 331, 340, 349, 358, 367, 376, 379, 383, 386, 391, 392, 397, 404, 409, 414, 423, 432, 441, 450, 459, 468, 477, 486, 495, 496]

# Optimized version using a set to detect cycles

def is_happy_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

happy_numbers_optimized = [num for num in range(1, 501) if is_happy_optimized(num)]
print("Optimized Happy numbers between 1 and 500 are:", happy_numbers_optimized)

# Output: Optimized Happy numbers between 1 and 500 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, 191, 193, 203, 208, 219, 226, 230, 236, 239, 241, 250, 259, 268, 277, 286, 295, 304, 313, 322, 331, 340, 349, 358, 367, 376, 379, 383, 386, 391, 392, 397, 404, 409, 414, 423, 432, 441, 450, 459, 468, 477, 486, 495, 496]

```

Output:

```

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, 241, 250, 259, 268, 277, 286, 295, 304, 313, 322, 331, 340, 349, 358, 367, 376, 379, 383, 386, 391, 392, 397, 404, 409, 414, 423, 432, 441, 450, 459, 468, 477, 486, 495, 496]
e: [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, 241, 250, 259, 268, 277, 286, 295, 304, 313, 322, 331, 340, 349, 358, 367, 376, 379, 383, 386, 391, 392, 397, 404, 409, 414, 423, 432, 441, 450, 459, 468, 477, 486, 495, 496]

```

Justification :

The optimised version improves cycle detection by using a set to track previously seen numbers, preventing infinite loops and enhancing efficiency.

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.

Code :

```

# Task 8:
...
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).
...

import math

def is_strong(num):
    sum_of_factorials = 0
    for digit in str(num):
        sum_of_factorials += math.factorial(int(digit))
    return sum_of_factorials == num

strong_numbers = []

for i in range(1, 1001):
    if is_strong(i):
        strong_numbers.append(i)

print("Strong numbers between 1 and 1000 are:", strong_numbers)
# Output: Strong numbers between 1 and 1000 are: [1, 2, 145, 40585]
# Optimized version with precomputed digit factorials
digit_factorials = {str(i): math.factorial(i) for i in range(10)}

def is_strong_optimized(num):
    sum_of_factorials = sum(digit_factorials[digit] for digit in str(num))
    return sum_of_factorials == num
strong_numbers_optimized = [num for num in range(1, 1001) if is_strong_optimized(num)]
print("Optimized Strong numbers between 1 and 1000 are:", strong_numbers_optimized)
# Output: Optimized Strong numbers between 1 and 1000 are: [1, 2, 145, 40585]
# The optimized version reduces redundant calculations by precomputing the factorials of digits 0-9, improving efficiency.

```

Output :

```

Strong numbers between 1 and 1000 are: [1, 2, 145]
Optimized Strong numbers between 1 and 1000 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:**
 - Full Name
 - Branch
 - SGPA

Expected Output

A reusable Python function that correctly navigates and extracts values

from nested dictionaries based on the provided examples

Code :

```

create a function that parses a nested dictionary representing student information.
Requirements :
• The function should extract and return:
    Full Name, Branch, SGPA

Example 1:
{
    "name": "John Doe",
    "age": 20,
    "branch": "Computer Science",
    "academic": {
        "sgpa": 8.5,
        "year": 3
    }
}

Example 2:
{
    "name": "Jane Smith",
    "age": 22,
    "academic": {
        "sgpa": 9.1,
        "year": 4
    }
}
...

def extract_student_info(student_dict):
    full_name = student_dict.get("name", "N/A")
    branch = student_dict.get("branch", "N/A")
    sgpa = student_dict.get("academic", {}).get("sgpa", "N/A")

    return {
        "Full Name": full_name,
        "Branch": branch,
        "SGPA": sgpa
    }

# Example usage
student_info = {
    "name": "John Doe",
    "age": 20,
    "branch": "Computer Science",
    "academic": {
        "sgpa": 8.5,
        "year": 3
    }
}

extracted_info = extract_student_info(student_info)
print(extracted_info)

```

Output :

```
{'Full Name': 'John Doe', 'Branch': 'Computer Science', 'SGPA': 8.5}
```

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).
 - 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}).

Expected Output #12:

- Python program that lists Perfect Numbers in the given range.
- Optimized version with explanation.

Code :

```
# Task 10:

...
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  $\sqrt{n}$ )
...

import math

def is_perfect(num):
    if num < 2:
        return False
    sum_of_divisors = 1 # 1 is a proper divisor of all integers > 1
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            sum_of_divisors += i
            if i != num // i:
                sum_of_divisors += num // i
    return sum_of_divisors == num
perfect_numbers = []
for i in range(1, 1001):
    if is_perfect(i):
        perfect_numbers.append(i)
print("Perfect numbers between 1 and 1000 are:", perfect_numbers)
# Output: Perfect numbers between 1 and 1000 are: [6, 28
# Optimized version using divisor check only up to  $\sqrt{n}$ 
def is_perfect_optimized(num):
    if num < 2:
        return False
    sum_of_divisors = 1
    for i in range(2, int(math.sqrt(num)) + 1):
        if num % i == 0:
            sum_of_divisors += i
            if i != num // i:
                sum_of_divisors += num // i
    return sum_of_divisors == num
perfect_numbers_optimized = [num for num in range(1, 1001) if is_perfect_optimized(num)]
print("Optimized Perfect numbers between 1 and 1000 are:", perfect_numbers_optimized)
# Output: Optimized Perfect numbers between 1 and 1000 are: [6, 28, 496]
# The optimized version improves efficiency by limiting the divisor check to the square root of the number, reducing the number of iterations needed
# Optimized version using divisor check only up to  $\sqrt{n}$ 
```

Output :

```
Perfect numbers between 1 and 1000 are: [6, 28, 496]
Optimized Perfect numbers between 1 and 1000 are: [6, 28, 496]
```

Justification :

The optimized version improves efficiency by limiting the divisor check to the square root of the number, reducing the number of iterations needed to find proper divisors., 496]

Optimized version using divisor check only up to \sqrt{n}