AI ASSISTED CODING

ASSIGNMENT-6.3

Name: B.Bhargava Chary

H.No: 2303A51747

Batch-24

Task Description #1 (Loops – Automorphic Numbers in a Range)

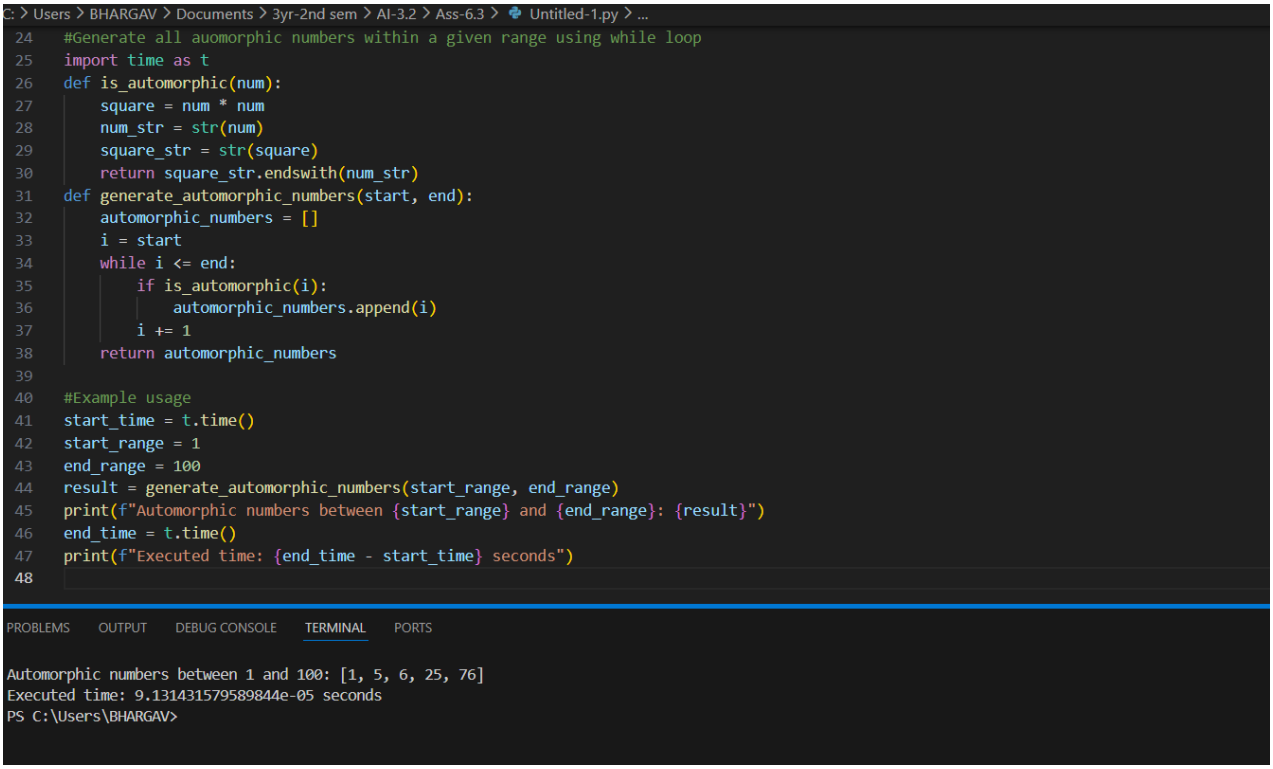
• Task: Prompt AI to generate a function that displays all Automorphic numbers between 1 and 1000 using a for loop.

• Instructions:

o Get AI-generated code to list Automorphic numbers using a for loop. o Analyze the correctness and efficiency of the generated logic. o Ask AI to regenerate using a while loop and compare both implementations.

Expected Output #1:

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



1. 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.

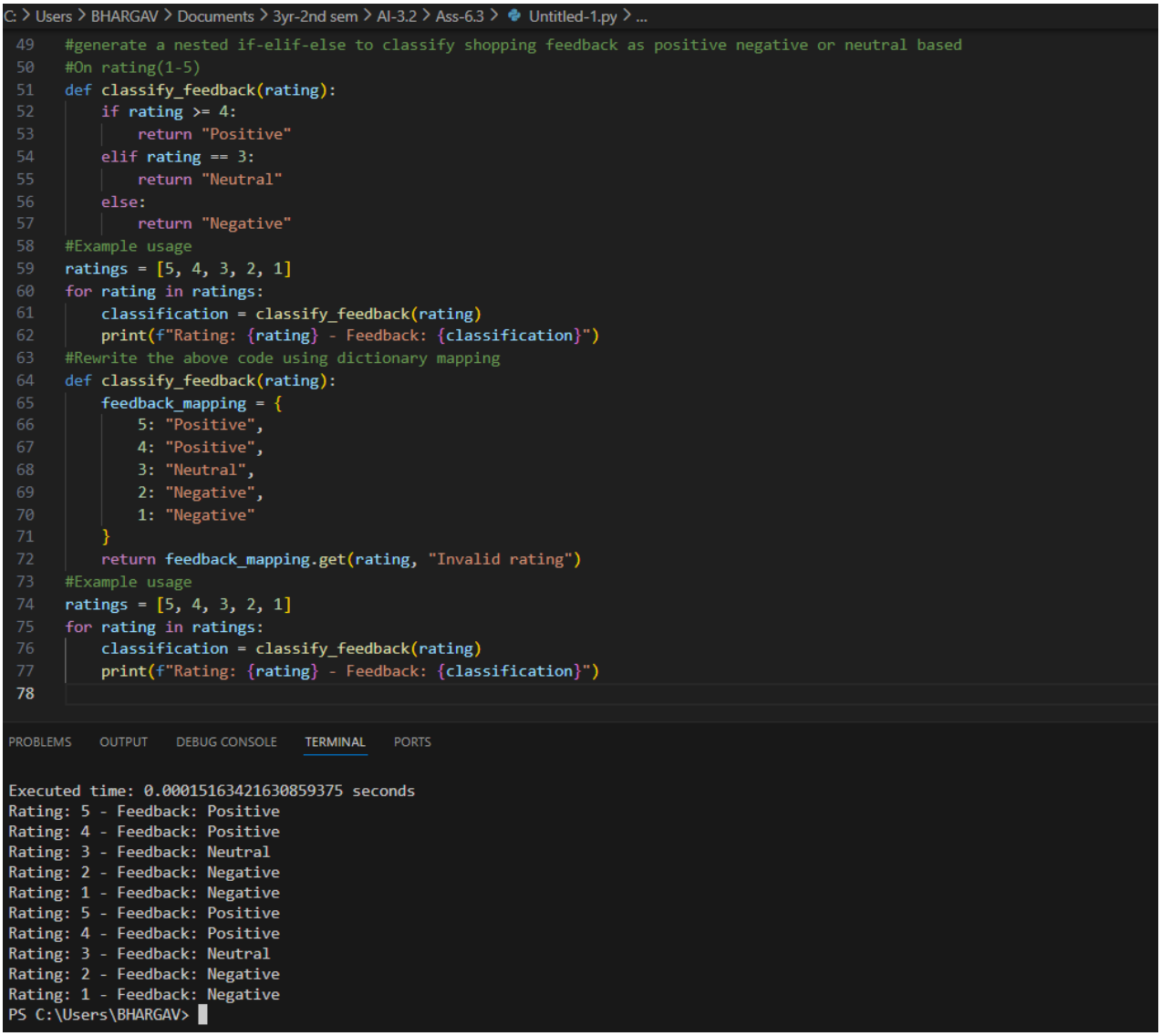
Task Description #2 (Conditional Statements – Online Shopping Feedback Classification)

• 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 dictionarybased or match-case structure.

Expected Output #2:

• Feedback classification function with explanation and an alternative approach.



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

C: > Users > BHARGAV > Documents > 3yr-2nd sem > AI-3.2 > Ass-6.3 > Untitled-1.py > ...

```python
#genearte a function named statistical_operations that takes  list of numbers as input and calculate
#minimum,maximum,mean,median,mode,standard deviation,variance of the list
import statistics as stats
def statistical_operations(numbers):
    minimum = min(numbers)
    maximum = max(numbers)
    mean = stats.mean(numbers)
    median = stats.median(numbers)
    mode = stats.mode(numbers)
    std_dev = stats.stdev(numbers)
    variance = stats.variance(numbers)

    return {
        "Minimum": minimum,
        "Maximum": maximum,
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Standard Deviation": std_dev,
        "Variance": variance
    }
#Example usage
data = [10,20,20,30,40,50]
results = statistical_operations(data)
for key, value in results.items():
    print(f"{key}: {value}")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Minimum: 10
Maximum: 50
Mean: 28.333333333333332
Median: 25.0
Mode: 20
Standard Deviation: 14.719601443879744
Variance: 216.66666666666666
PS C:\Users\BHARGAV>
```

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.

```python
#Create  class teacher with attributes teacher_id, name, subject,
# and experience.Add a method to display teacher details.
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}")
        print(f"Experience: {self.experience} years")
#Example usage
teacher =Teacher(1, "John Doe", "Mathematics", 10)
teacher.display_details()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                Python

PS C:\Users\BHARGAV> & C:/Users/BHARGAV/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/BHARGAV/Documents/3yr-2nd sem/AI-3.2/Ass-6.3/Untitled-1.py"
Teacher ID: 1
Name: John Doe
Subject: Mathematics
Experience: 10 years
PS C:\Users\BHARGAV>
```

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

mucxz66y76ge2 q

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.

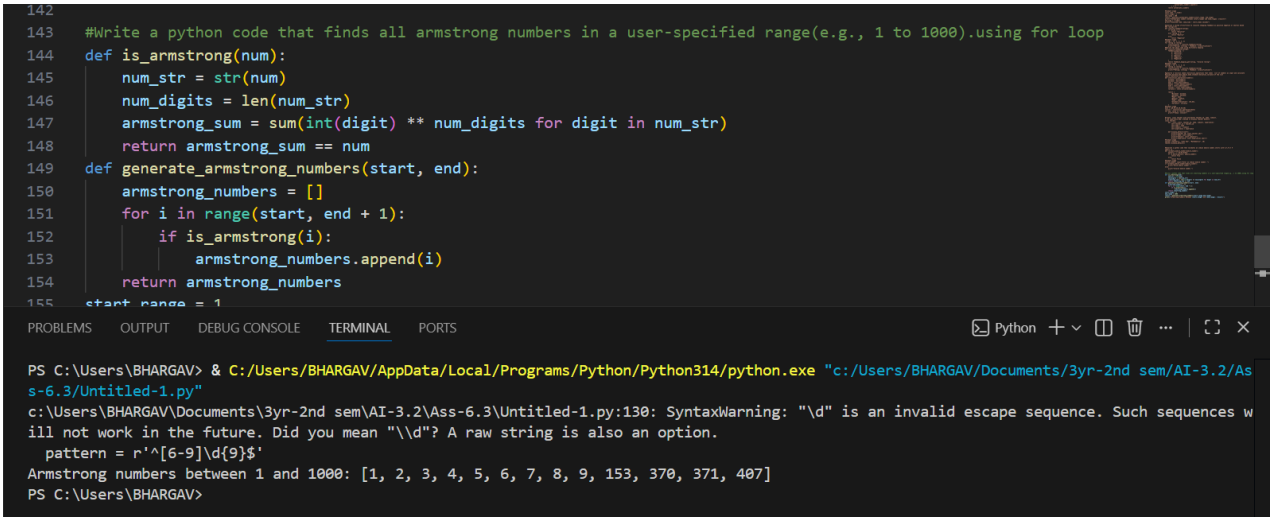
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).

• 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.

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)

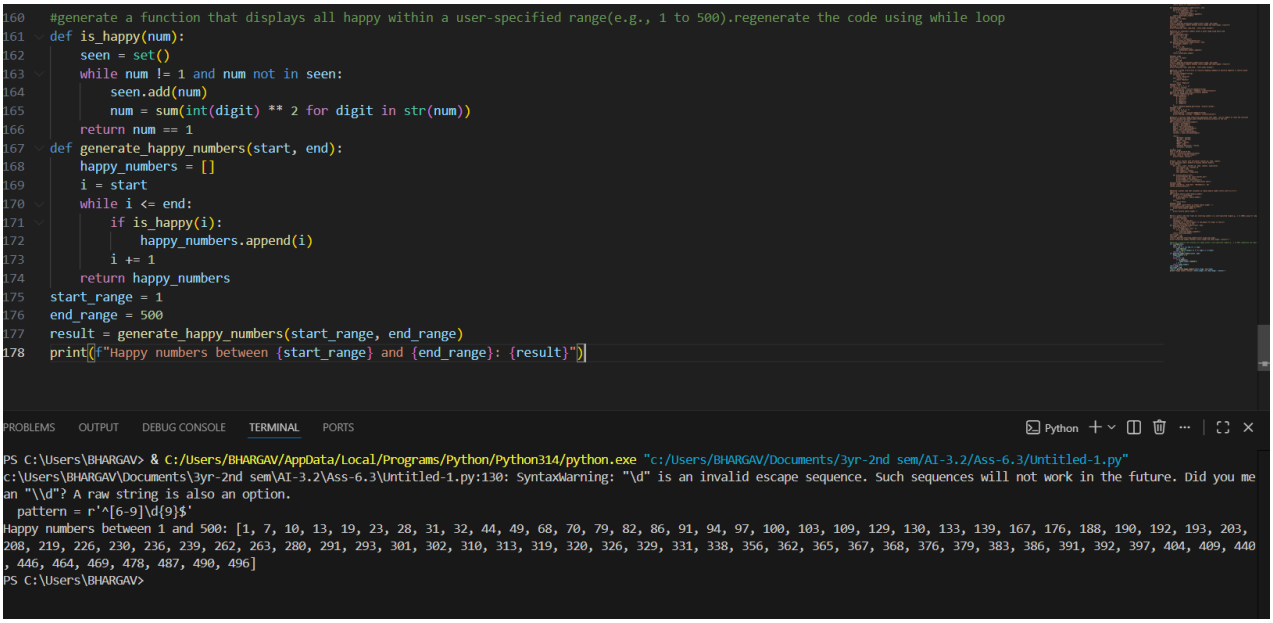
Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).

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.



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.

```python
#generate a function displays all strong numbers(sum of factorial of digits of digiyts equals the number,
#e.g.,145=1!+4!+5!)within a given range.with 1,2,145
import math;
def is_strong_number(num):
    sum_of_factionals=sum(math.factorial(int(digit))for digit in str(num))
    return sum_of_factionals==num
def generate_strong_numbers(start,end):
    strong_numbers=[]
    for i in range(start,end+1):
        if is_strong_number(i):
            strong_numbers.append(i)
    return strong_numbers
start_range=1
end_range=1000
result=generate_strong_numbers(start_range,end_range)
print(f"Strong numbers between {start_range} and {end_range}: {result}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                   Python

PS C:\Users\BHARGAV> & C:/Users/BHARGAV/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/BHARGAV/Documents/3yr-2nd sem/AI-3.2/As
s-6.3/Untitled-1.py"
c:\Users\BHARGAV\Documents\3yr-2nd sem\AI-3.2\Ass-6.3\Untitled-1.py:130: SyntaxWarning: "\d" is an invalid escape sequence. Such sequences w
ill not work in the future. Did you mean "\\d"? A raw string is also an option.
  pattern = r'^[6-9]\d{9}$'
Strong numbers between 1 and 1000: [1, 2, 145]
PS C:\Users\BHARGAV>
```

the

```
PS D:\AI> & "C:/Users/sande/miniconda3/sr univ/python.exe" d:/AI/Assignment-6.3.py
d:\AI\Assignment-6.3.py:106: 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

```python
#generate a python code that parses a nested dictionary representing student information\
#the dictionry contains fu;l name branch SGPA
def parse_student_info(student_info):
    for student_id, details in student_info.items():
        full_name = details.get("full_name", "N/A")
        branch = details.get("branch", "N/A")
        sgpa = details.get("SGPA", "N/A")
        print(f"Student ID: {student_id}")
        print(f"Full Name: {full_name}")
        print(f"Branch: {branch}")
        print(f"SGPA: {sgpa}")
        print("-" * 20)
#Example usage
students = {
    "001": {"full_name": "Alice Smith", "branch": "Computer Science", "SGPA": 8.5},
    "002": {"full_name": "Bob Johnson", "branch": "Mechanical Engineering", "SGPA": 7.8},
    "003": {"full_name": "Charlie Brown", "branch": "Electrical Engineering", "SGPA": 9.2}
}
parse_student_info(students)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Full Name: Alice Smith
Branch: Computer Science
SGPA: 8.5
--------------------
Student ID: 002
Full Name: Bob Johnson
Branch: Mechanical Engineering
SGPA: 7.8
--------------------
Student ID: 003
Full Name: Charlie Brown
Branch: Electrical Engineering
SGPA: 9.2
```

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

C: > Users > BHARGAV > Documents > 3yr-2nd sem > AI-3.2 > Ass-6.3 > Untitled-1.py > ...

```python
#generate a python code displays all perfect numbers within a user-specified range(e.g., 1 to 1000).using for loop
def is_perfect(num):
    if num < 2:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num
def generate_perfect_numbers(start, end):
    perfect_numbers = []
    for i in range(start, end + 1):
        if is_perfect(i):
            perfect_numbers.append(i)
    return perfect_numbers
start_range = 1
end_range = 1000
result = generate_perfect_numbers(start_range, end_range)
print(f"Perfect numbers between {start_range} and {end_range}: {result}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

```
PS C:\Users\BHARGAV> & C:/Users/BHARGAV/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/BHARGAV/Documents/3yr-2nd sem/AI-3.2/Ass-6.3/Untitled-1.py"
c:\Users\BHARGAV\Documents\3yr-2nd sem\AI-3.2\Ass-6.3\Untitled-1.py:130: SyntaxWarning: "\d" is an invalid escape sequence. Such sequences will not work in the future. Did you mean "\\d"? A raw string is also an option.
  pattern = r'^[6-9]\d{9}$'
Perfect numbers between 1 and 1000: [6, 28, 496]
PS C:\Users\BHARGAV>
```