AI Assisted Coding

Assignment 6.3

Name: P. Hemanth kumar

HT No: 2303A52143

Batch: 32

Prompts:

1.'''

Write a python code that display all automorphic numbers between 1 to 1000

write using for loop

calculate how much time taken to execute the code

calculate the end time and print the time taken to execute the code

'''

2. '''

write a python code to classify the feedback as Positive,Negative or Neutral using nested if elif else based on rating provided by user from 1 to 5

write the time taken to execute the code

rewrite the code using dictionary based approach

rewrite the code using dictionary based approach

'''

3. '''

write a python code that performs Minimum, Maximum,Mean,Median,Mode,Variance,Standard Deviation on a tuple numbers

'''

4. #write a python code to Create a class Teacher with attributes teacher_id, name, subject and experience

#Add a method to display teacher details

5. #write a python code to generate a function that validates Indian Mobile Number that number should start with 6,7,8 or 9 and should be of 10 digits

6. # write a python code that finds all Armstrong numbers in a user specified range

#Use for loop and digit power logic and validate correctness by checking known armstrong numbers

#give optimized version of the code

7. #write a python code that displays all happy numbers between sepcified range

#write using loop and function and validate correctness by checking known happy numbers

#give optimized version of the code

8. #write a python code to display all strong numbers within a specified range

#Use loops to extract digits and calculate factorials, validate correctness by checking known strong numbers

#give optimized version of the code using pre commute digit factorials

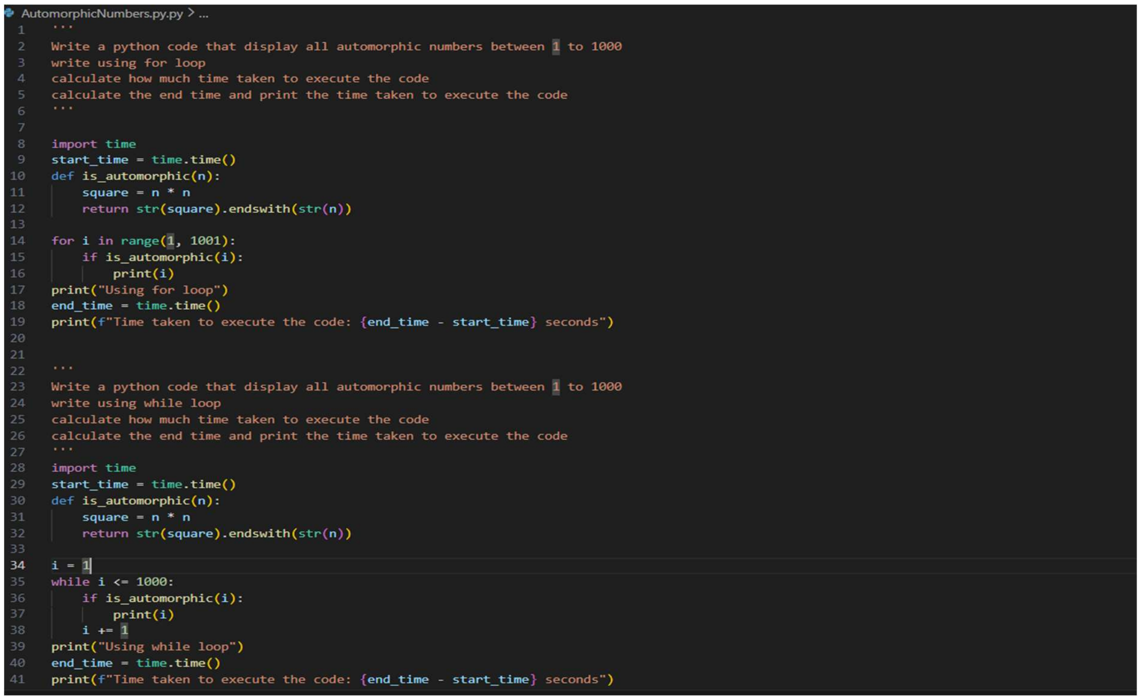
9. #write a python code using few shot promptcreate a function that parses a nested dictionary representing student information

#The function should extract and return fullname,branch and SGPA

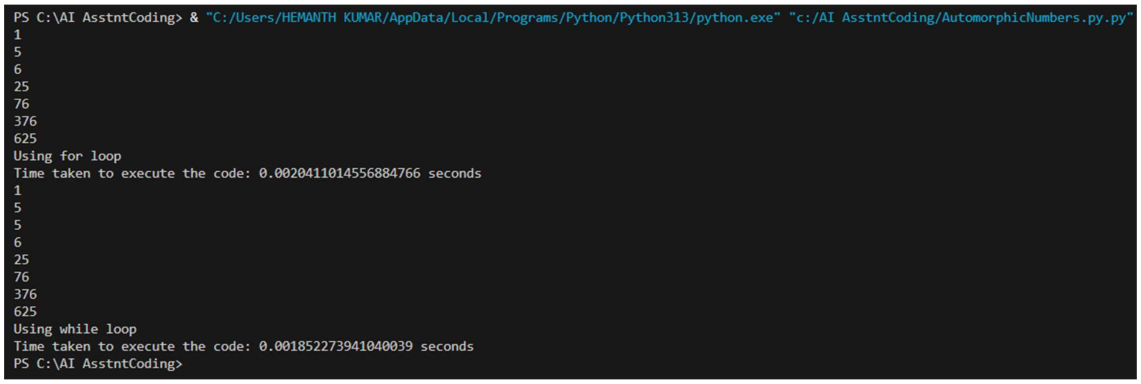
10. #write a python code to display all perfect numbers within a specified range using for loop to find divisors

#give optimized version of the code using divisor check only upto square root of the number

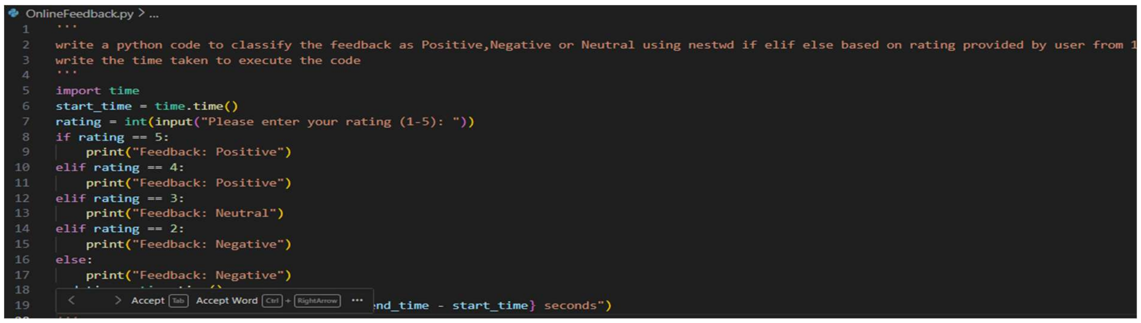
Code:



```python
'''
Write a python code that display all automorphic numbers between 1 to 1000
write using for loop
calculate how much time taken to execute the code
calculate the end time and print the time taken to execute the code
'''

import time
start_time = time.time()
def is_automorphic(n):
    square = n * n
    return str(square).endswith(str(n))

for i in range(1, 1001):
    if is_automorphic(i):
        print(i)
print("Using for loop")
end_time = time.time()
print(f"Time taken to execute the code: {end_time - start_time} seconds")


'''
Write a python code that display all automorphic numbers between 1 to 1000
write using while loop
calculate how much time taken to execute the code
calculate the end time and print the time taken to execute the code
'''
import time
start_time = time.time()
def is_automorphic(n):
    square = n * n
    return str(square).endswith(str(n))

i = 1
while i <= 1000:
    if is_automorphic(i):
        print(i)
    i += 1
print("Using while loop")
end_time = time.time()
print(f"Time taken to execute the code: {end_time - start_time} seconds")
```

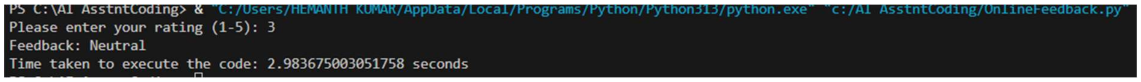


```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/AutomorphicNumbers.py.py"
1
5
6
25
76
376
625
Using for loop
Time taken to execute the code: 0.0020411014556884766 seconds
1
5
5
6
25
76
376
625
Using while loop
Time taken to execute the code: 0.001852273941040039 seconds
PS C:\AI AsstntCoding>
```

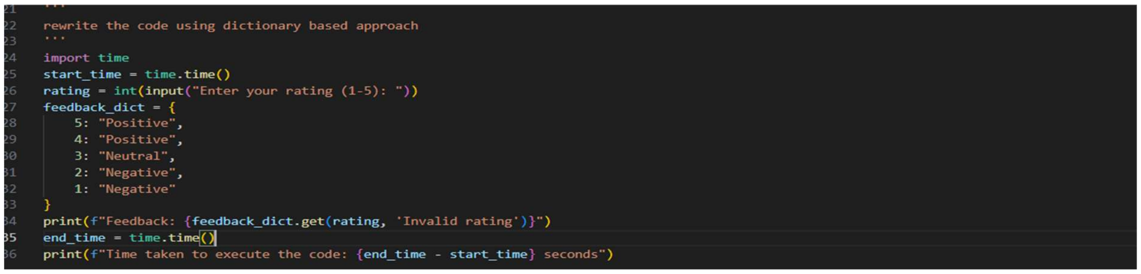


OnlineFeedback.py > ...

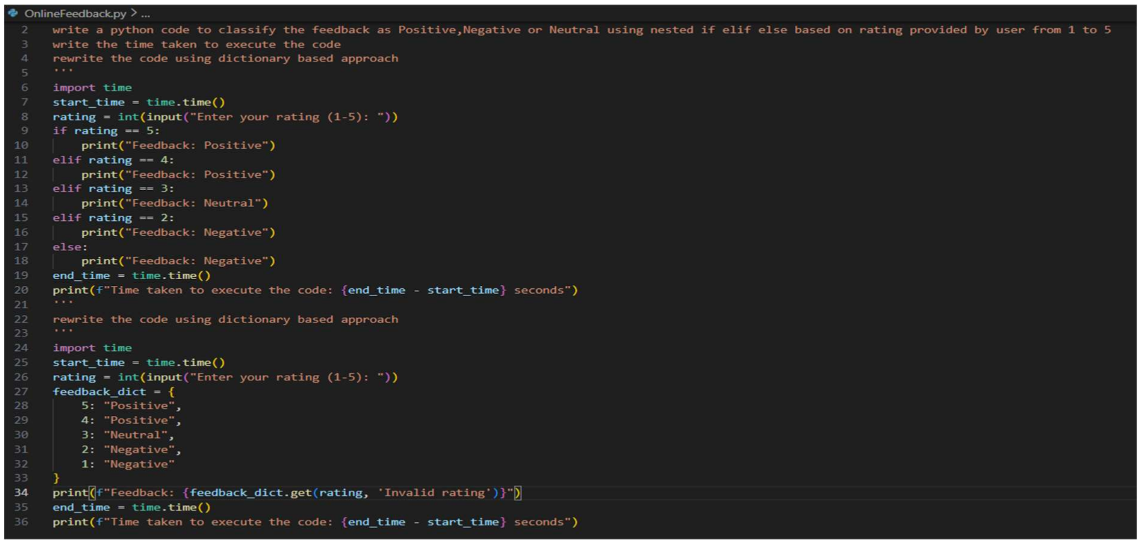
```
'''
write a python code to classify the feedback as Positive,Negative or Neutral using nestwd if elif else based on rating provided by user from 1
write the time taken to execute the code
'''
import time
start_time = time.time()
rating = int(input("Please enter your rating (1-5): "))
if rating == 5:
    print("Feedback: Positive")
elif rating == 4:
    print("Feedback: Positive")
elif rating == 3:
    print("Feedback: Neutral")
elif rating == 2:
    print("Feedback: Negative")
else:
    print("Feedback: Negative")

<  >  Accept Tab  Accept Word Ctrl + RightArrow  ···  nd_time - start_time} seconds")
```



```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/OnlineFeedback.py"
Please enter your rating (1-5): 3
Feedback: Neutral
Time taken to execute the code: 2.983675003051758 seconds
```

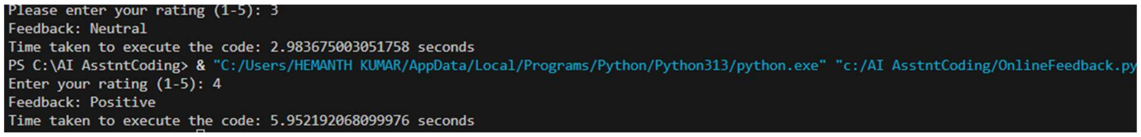


```
'''
rewrite the code using dictionary based approach
'''
import time
start_time = time.time()
rating = int(input("Enter your rating (1-5): "))
feedback_dict = {
    5: "Positive",
    4: "Positive",
    3: "Neutral",
    2: "Negative",
    1: "Negative"
}
print(f"Feedback: {feedback_dict.get(rating, 'Invalid rating')}")
end_time = time.time()
print(f"Time taken to execute the code: {end_time - start_time} seconds")
```

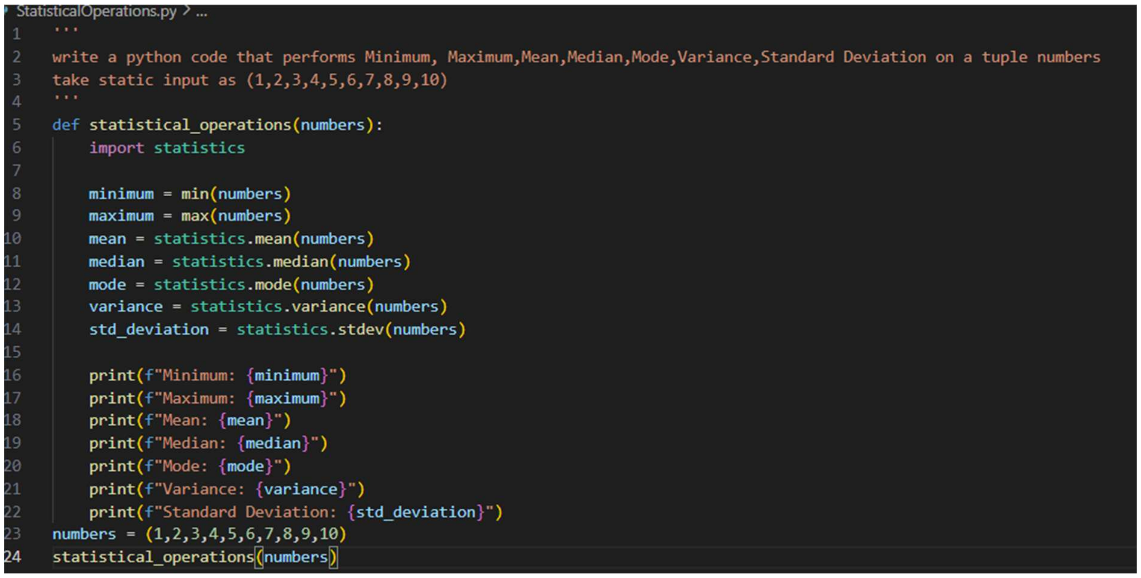


OnlineFeedback.py > ...

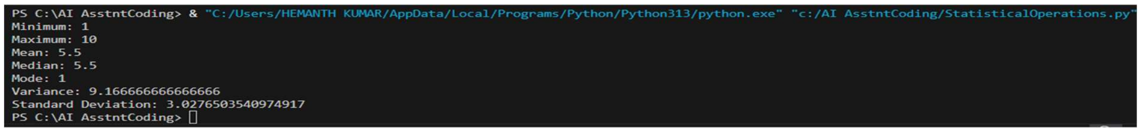
```
write a python code to classify the feedback as Positive,Negative or Neutral using nested if elif else based on rating provided by user from 1 to 5
write the time taken to execute the code
rewrite the code using dictionary based approach
'''
import time
start_time = time.time()
rating = int(input("Enter your rating (1-5): "))
if rating == 5:
    print("Feedback: Positive")
elif rating == 4:
    print("Feedback: Positive")
elif rating == 3:
    print("Feedback: Neutral")
elif rating == 2:
    print("Feedback: Negative")
else:
    print("Feedback: Negative")
end_time = time.time()
print(f"Time taken to execute the code: {end_time - start_time} seconds")
'''
rewrite the code using dictionary based approach
'''
import time
start_time = time.time()
rating = int(input("Enter your rating (1-5): "))
feedback_dict = {
    5: "Positive",
    4: "Positive",
    3: "Neutral",
    2: "Negative",
    1: "Negative"
}
print(f"Feedback: {feedback_dict.get(rating, 'Invalid rating')}")
end_time = time.time()
print(f"Time taken to execute the code: {end_time - start_time} seconds")
```



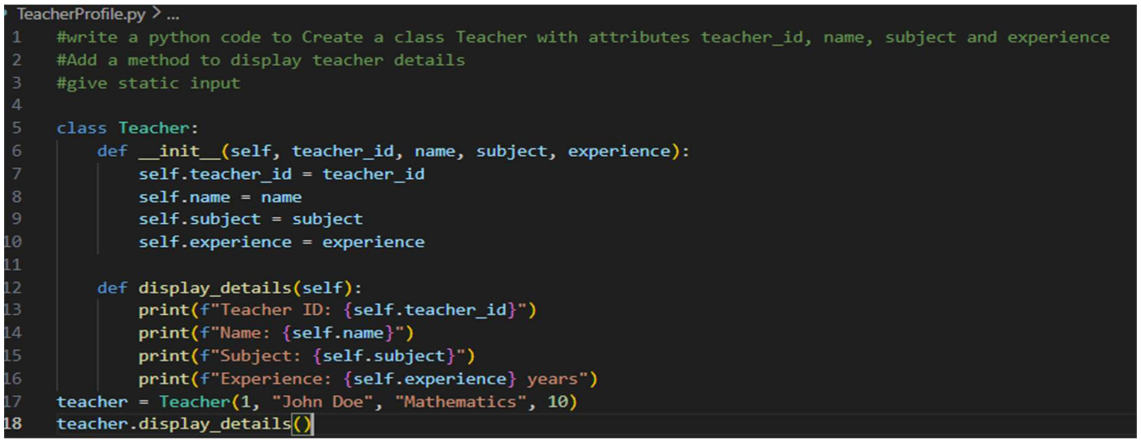
```
Please enter your rating (1-5): 3
Feedback: Neutral
Time taken to execute the code: 2.983675003051758 seconds
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/OnlineFeedback.py
Enter your rating (1-5): 4
Feedback: Positive
Time taken to execute the code: 5.952192068099976 seconds
```



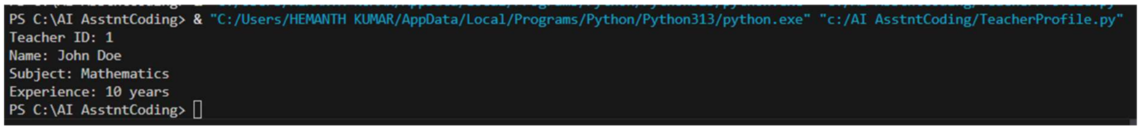
```python
'''
write a python code that performs Minimum, Maximum,Mean,Median,Mode,Variance,Standard Deviation on a tuple numbers
take static input as (1,2,3,4,5,6,7,8,9,10)
'''
def statistical_operations(numbers):
    import statistics

    minimum = min(numbers)
    maximum = max(numbers)
    mean = statistics.mean(numbers)
    median = statistics.median(numbers)
    mode = statistics.mode(numbers)
    variance = statistics.variance(numbers)
    std_deviation = statistics.stdev(numbers)

    print(f"Minimum: {minimum}")
    print(f"Maximum: {maximum}")
    print(f"Mean: {mean}")
    print(f"Median: {median}")
    print(f"Mode: {mode}")
    print(f"Variance: {variance}")
    print(f"Standard Deviation: {std_deviation}")
numbers = (1,2,3,4,5,6,7,8,9,10)
statistical_operations(numbers)
```



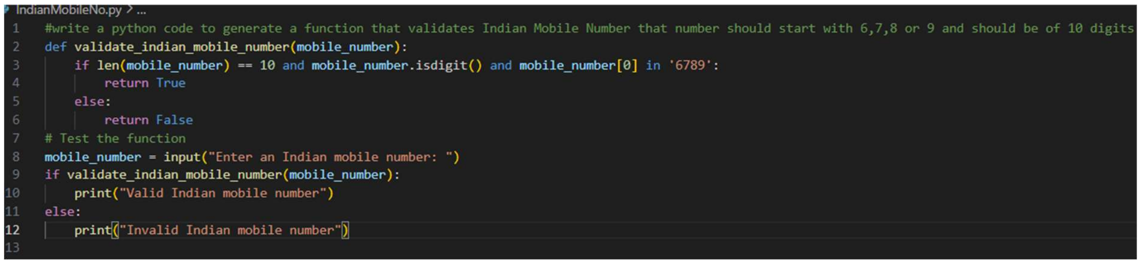
```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/StatisticalOperations.py"
Minimum: 1
Maximum: 10
Mean: 5.5
Median: 5.5
Mode: 1
Variance: 9.166666666666666
Standard Deviation: 3.0276503540974917
PS C:\AI AsstntCoding> 
```



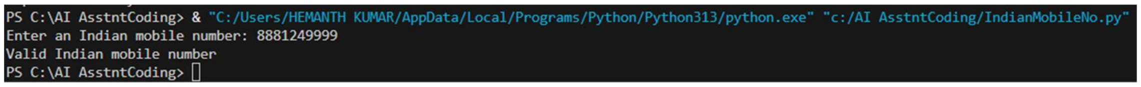
```python
#write a python code to Create a class Teacher with attributes teacher_id, name, subject and experience
#Add a method to display teacher details
#give static input

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")
teacher = Teacher(1, "John Doe", "Mathematics", 10)
teacher.display_details()
```



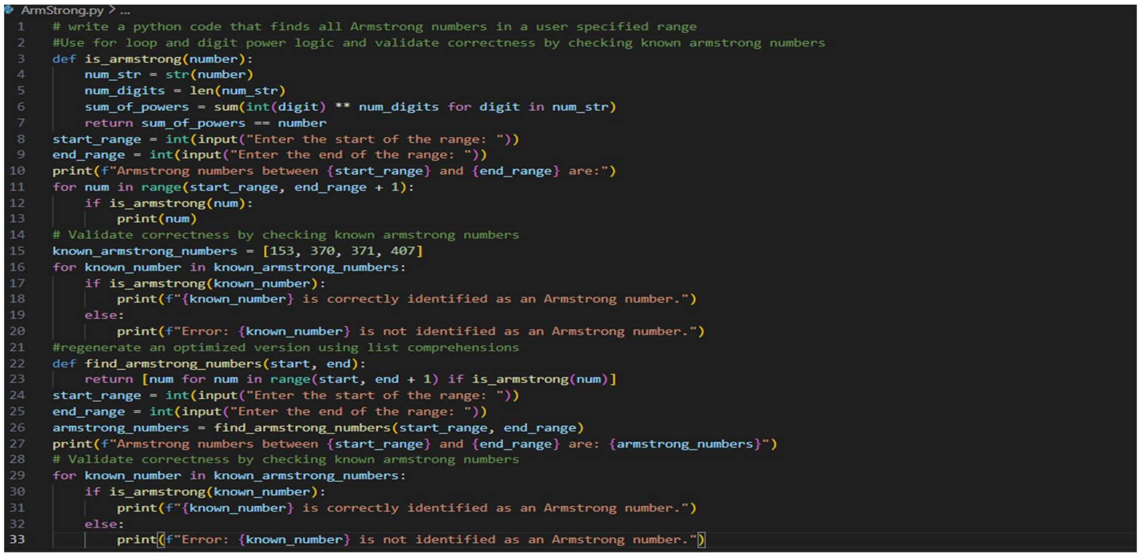
```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/TeacherProfile.py"
Teacher ID: 1
Name: John Doe
Subject: Mathematics
Experience: 10 years
PS C:\AI AsstntCoding> 
```



```python
#write a python code to generate a function that validates Indian Mobile Number that number should start with 6,7,8 or 9 and should be of 10 digits
def validate_indian_mobile_number(mobile_number):
    if len(mobile_number) == 10 and mobile_number.isdigit() and mobile_number[0] in '6789':
        return True
    else:
        return False
# Test the function
mobile_number = input("Enter an Indian mobile number: ")
if validate_indian_mobile_number(mobile_number):
    print("Valid Indian mobile number")
else:
    print("Invalid Indian mobile number")
```

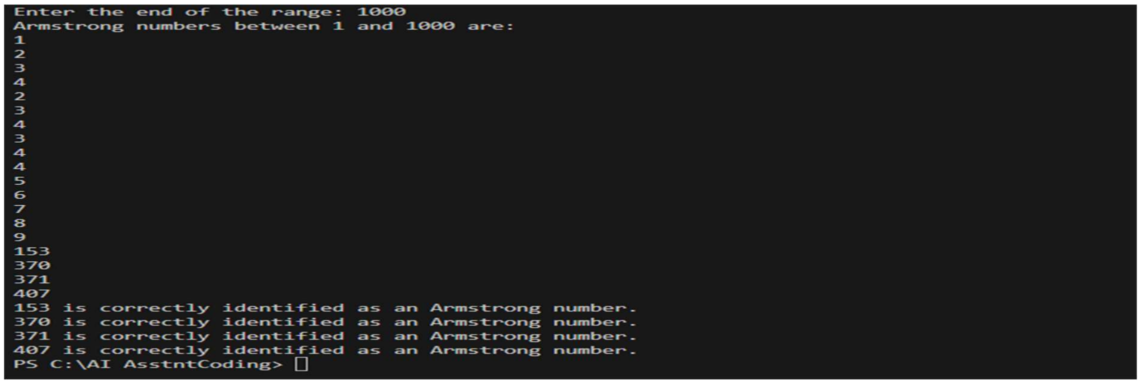


```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/IndianMobileNo.py"
Enter an Indian mobile number: 8881249999
Valid Indian mobile number
PS C:\AI AsstntCoding> 
```

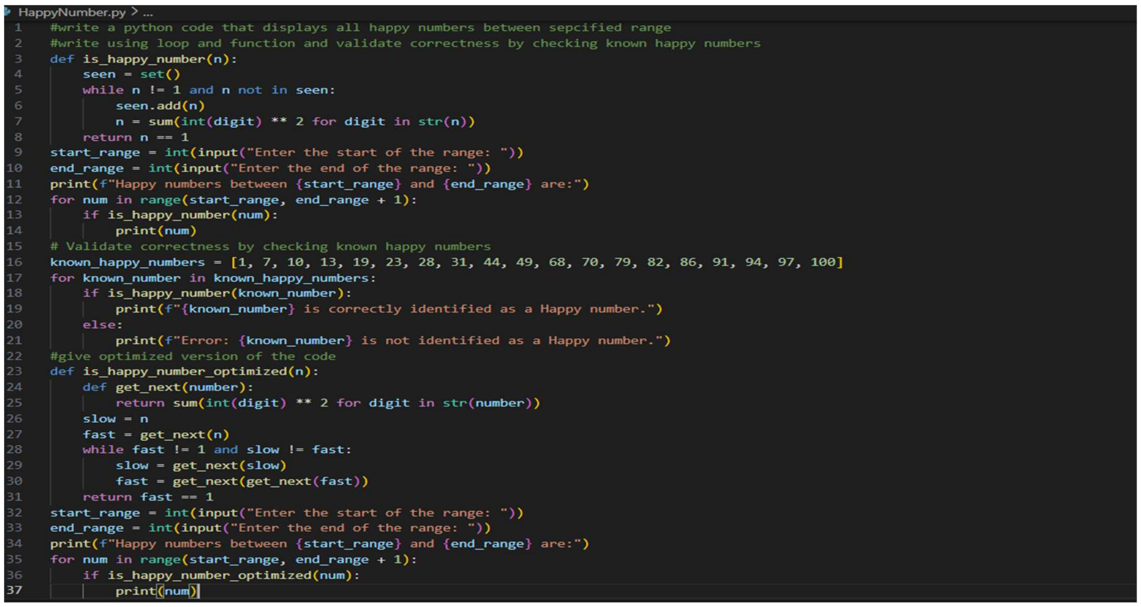


ArmStrong.py > ...

```python
# write a python code that finds all Armstrong numbers in a user specified range
#Use for loop and digit power logic and validate correctness by checking known armstrong numbers
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
    return sum_of_powers == number
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Armstrong numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_armstrong(num):
        print(num)
# Validate correctness by checking known armstrong numbers
known_armstrong_numbers = [153, 370, 371, 407]
for known_number in known_armstrong_numbers:
    if is_armstrong(known_number):
        print(f"{known_number} is correctly identified as an Armstrong number.")
    else:
        print(f"Error: {known_number} is not identified as an Armstrong number.")
#regenerate an optimized version using list comprehensions
def find_armstrong_numbers(start, end):
    return [num for num in range(start, end + 1) if is_armstrong(num)]
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
armstrong_numbers = find_armstrong_numbers(start_range, end_range)
print(f"Armstrong numbers between {start_range} and {end_range} are: {armstrong_numbers}")
# Validate correctness by checking known armstrong numbers
for known_number in known_armstrong_numbers:
    if is_armstrong(known_number):
        print(f"{known_number} is correctly identified as an Armstrong number.")
    else:
        print(f"Error: {known_number} is not identified as an Armstrong number.")
```

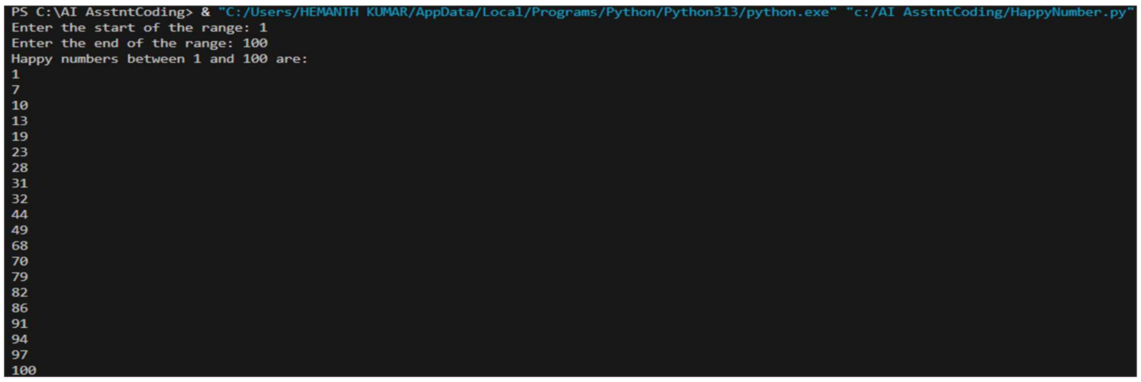


```
Enter the end of the range: 1000
Armstrong numbers between 1 and 1000 are:
1
2
3
4
2
3
4
3
4
4
5
6
7
8
9
153
370
371
407
153 is correctly identified as an Armstrong number.
370 is correctly identified as an Armstrong number.
371 is correctly identified as an Armstrong number.
407 is correctly identified as an Armstrong number.
PS C:\AI AsstntCoding> 
```

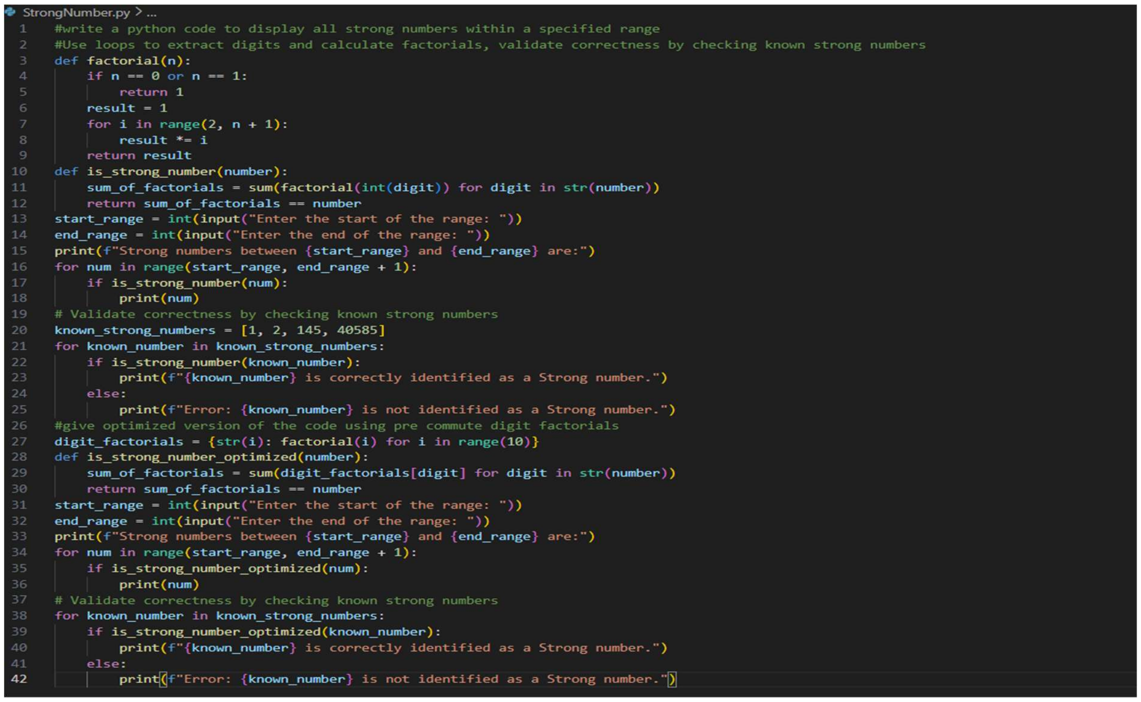


HappyNumber.py > ...

```python
#write a python code that displays all happy numbers between sepcified range
#write using loop and function and validate correctness by checking known happy numbers
def is_happy_number(n):
    seen = set()
    while n != 1 and n not in seen:
        seen.add(n)
        n = sum(int(digit) ** 2 for digit in str(n))
    return n == 1
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Happy numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_happy_number(num):
        print(num)
# Validate correctness by checking known happy numbers
known_happy_numbers = [1, 7, 10, 13, 19, 23, 28, 31, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
for known_number in known_happy_numbers:
    if is_happy_number(known_number):
        print(f"{known_number} is correctly identified as a Happy number.")
    else:
        print(f"Error: {known_number} is not identified as a Happy number.")
#give optimized version of the code
def is_happy_number_optimized(n):
    def get_next(number):
        return sum(int(digit) ** 2 for digit in str(number))
    slow = n
    fast = get_next(n)
    while fast != 1 and slow != fast:
        slow = get_next(slow)
        fast = get_next(get_next(fast))
    return fast == 1
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Happy numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_happy_number_optimized(num):
        print(num)
```

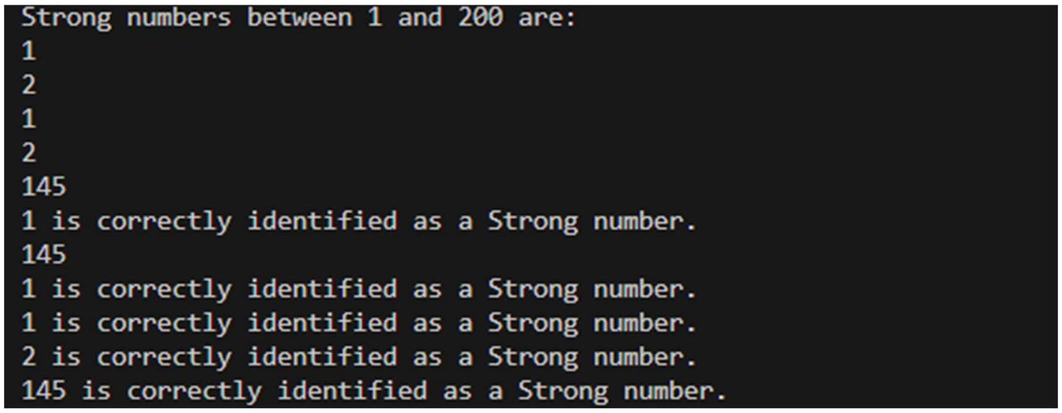


```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/HappyNumber.py"
Enter the start of the range: 1
Enter the end of the range: 100
Happy numbers between 1 and 100 are:
1
7
10
13
19
23
28
31
32
44
49
68
70
79
82
86
91
94
97
100
```

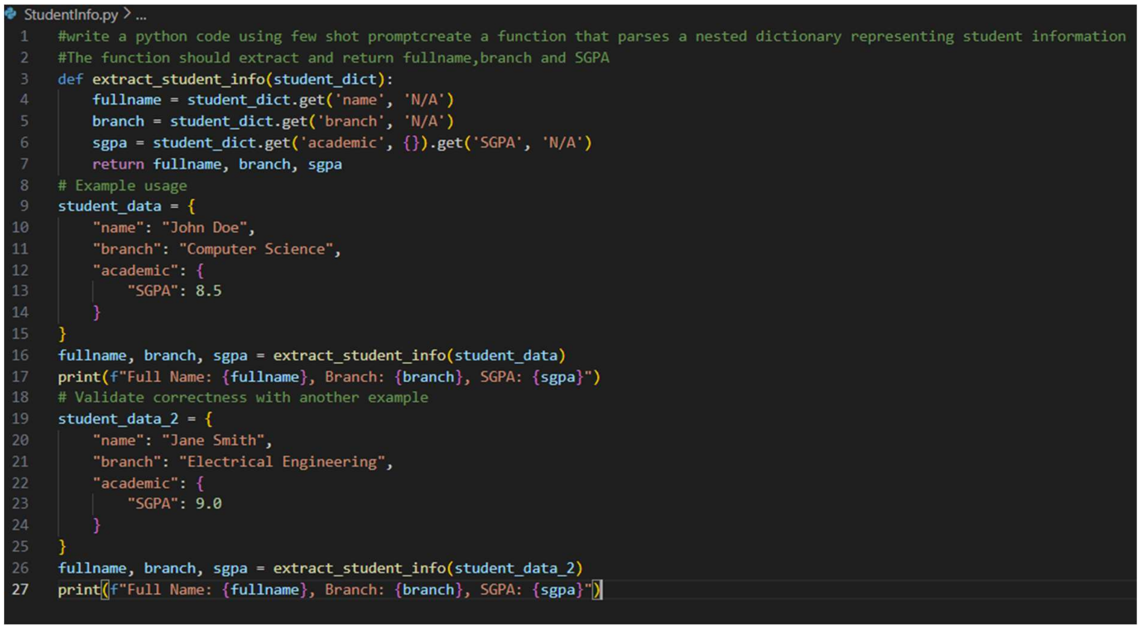


StrongNumber.py > ...

```python
#write a python code to display all strong numbers within a specified range
#Use loops to extract digits and calculate factorials, validate correctness by checking known strong numbers
def factorial(n):
    if n == 0 or n == 1:
        return 1
    result = 1
    for i in range(2, n + 1):
        result *= i
    return result
def is_strong_number(number):
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(number))
    return sum_of_factorials == number
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Strong numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_strong_number(num):
        print(num)
# Validate correctness by checking known strong numbers
known_strong_numbers = [1, 2, 145, 40585]
for known_number in known_strong_numbers:
    if is_strong_number(known_number):
        print(f"{known_number} is correctly identified as a Strong number.")
    else:
        print(f"Error: {known_number} is not identified as a Strong number.")
#give optimized version of the code using pre commute digit factorials
digit_factorials = {str(i): factorial(i) for i in range(10)}
def is_strong_number_optimized(number):
    sum_of_factorials = sum(digit_factorials[digit] for digit in str(number))
    return sum_of_factorials == number
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Strong numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_strong_number_optimized(num):
        print(num)
# Validate correctness by checking known strong numbers
for known_number in known_strong_numbers:
    if is_strong_number_optimized(known_number):
        print(f"{known_number} is correctly identified as a Strong number.")
    else:
        print(f"Error: {known_number} is not identified as a Strong number.")
```

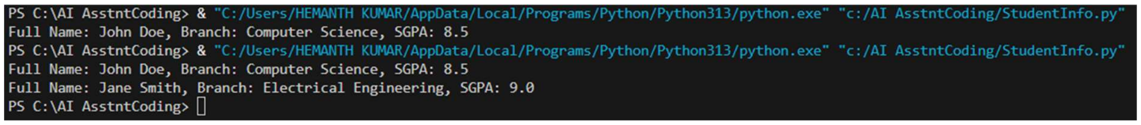


```
Strong numbers between 1 and 200 are:
1
2
1
2
145
1 is correctly identified as a Strong number.
145
1 is correctly identified as a Strong number.
1 is correctly identified as a Strong number.
2 is correctly identified as a Strong number.
145 is correctly identified as a Strong number.
```

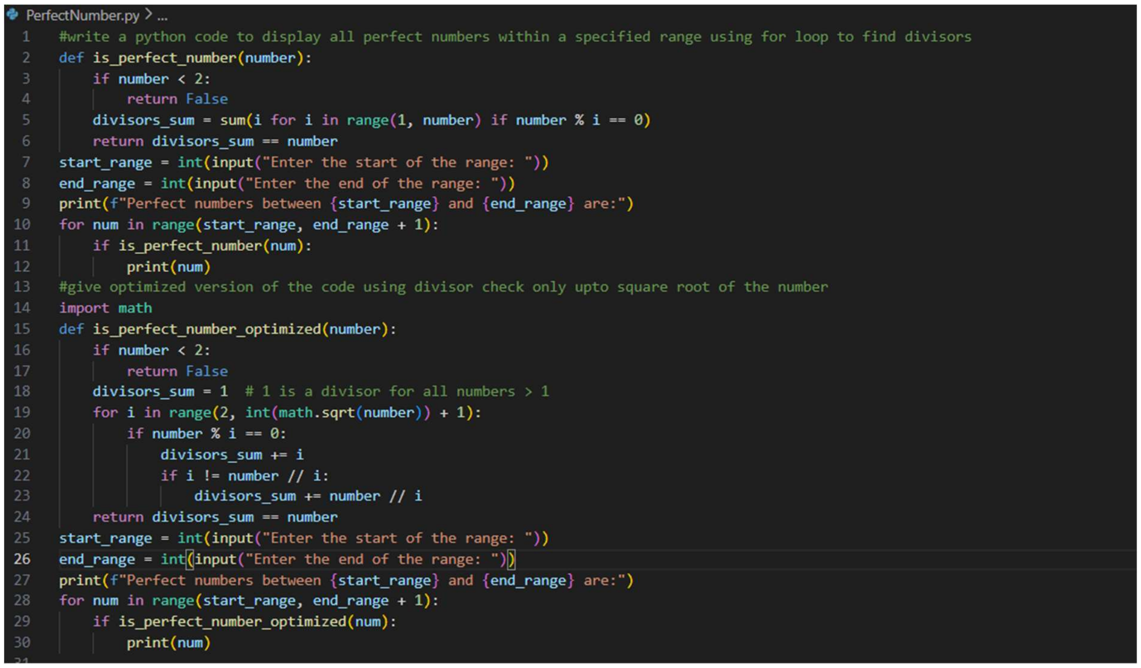


StudentInfo.py > ...

```python
#write a python code using few shot promptcreate a function that parses a nested dictionary representing student information
#The function should extract and return fullname,branch and SGPA
def extract_student_info(student_dict):
    fullname = student_dict.get('name', 'N/A')
    branch = student_dict.get('branch', 'N/A')
    sgpa = student_dict.get('academic', {}).get('SGPA', 'N/A')
    return fullname, branch, sgpa
# Example usage
student_data = {
    "name": "John Doe",
    "branch": "Computer Science",
    "academic": {
        "SGPA": 8.5
    }
}
fullname, branch, sgpa = extract_student_info(student_data)
print(f"Full Name: {fullname}, Branch: {branch}, SGPA: {sgpa}")
# Validate correctness with another example
student_data_2 = {
    "name": "Jane Smith",
    "branch": "Electrical Engineering",
    "academic": {
        "SGPA": 9.0
    }
}
fullname, branch, sgpa = extract_student_info(student_data_2)
print(f"Full Name: {fullname}, Branch: {branch}, SGPA: {sgpa}")
```

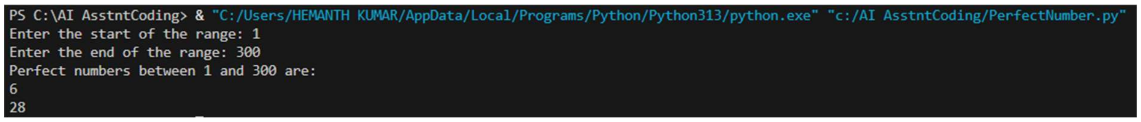


```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/StudentInfo.py"
Full Name: John Doe, Branch: Computer Science, SGPA: 8.5
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/StudentInfo.py"
Full Name: John Doe, Branch: Computer Science, SGPA: 8.5
Full Name: Jane Smith, Branch: Electrical Engineering, SGPA: 9.0
PS C:\AI AsstntCoding> 
```



PerfectNumber.py > ...

```python
#write a python code to display all perfect numbers within a specified range using for loop to find divisors
def is_perfect_number(number):
    if number < 2:
        return False
    divisors_sum = sum(i for i in range(1, number) if number % i == 0)
    return divisors_sum == number
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Perfect numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_perfect_number(num):
        print(num)
#give optimized version of the code using divisor check only upto square root of the number
import math
def is_perfect_number_optimized(number):
    if number < 2:
        return False
    divisors_sum = 1  # 1 is a divisor for all numbers > 1
    for i in range(2, int(math.sqrt(number)) + 1):
        if number % i == 0:
            divisors_sum += i
            if i != number // i:
                divisors_sum += number // i
    return divisors_sum == number
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
print(f"Perfect numbers between {start_range} and {end_range} are:")
for num in range(start_range, end_range + 1):
    if is_perfect_number_optimized(num):
        print(num)
```



```
PS C:\AI AsstntCoding> & "C:/Users/HEMANTH KUMAR/AppData/Local/Programs/Python/Python313/python.exe" "c:/AI AsstntCoding/PerfectNumber.py"
Enter the start of the range: 1
Enter the end of the range: 300
Perfect numbers between 1 and 300 are:
6
28
```

Explanations:

1. The program finds Automorphic numbers by checking whether the square of a number ends with the number itself using loop logic.
2. The feedback classification uses conditional statements to correctly label ratings as Negative, Neutral, or Positive based on given values.
3. The function uses Python built-in statistics methods to compute minimum, maximum, mean, median, mode, variance, and standard deviation accurately.
4. The Teacher class demonstrates object oriented programming by initializing attributes through a constructor and displaying details using a class method.
5. The function validates an Indian mobile number by checking that it has exactly ten digits and starts with six, seven, eight, or nine.
6. The program identifies Armstrong numbers by comparing each number with the sum of its digits raised to the power of total digits.
7. Happy numbers are detected by repeatedly summing the squares of digits and using a set to prevent infinite loops.
8. The function checks whether a number equals the sum of factorials of its digits to identify Strong numbers.
9. The function navigates a nested dictionary structure to correctly extract student full name, branch, and SGPA.
10. Perfect numbers are identified by summing proper divisors efficiently by checking only up to the square root of the number.