ASSIGNMENT-6.1

Hall Ticket: 2303A51630

Name: K. Nithin Kumar

Batch:23

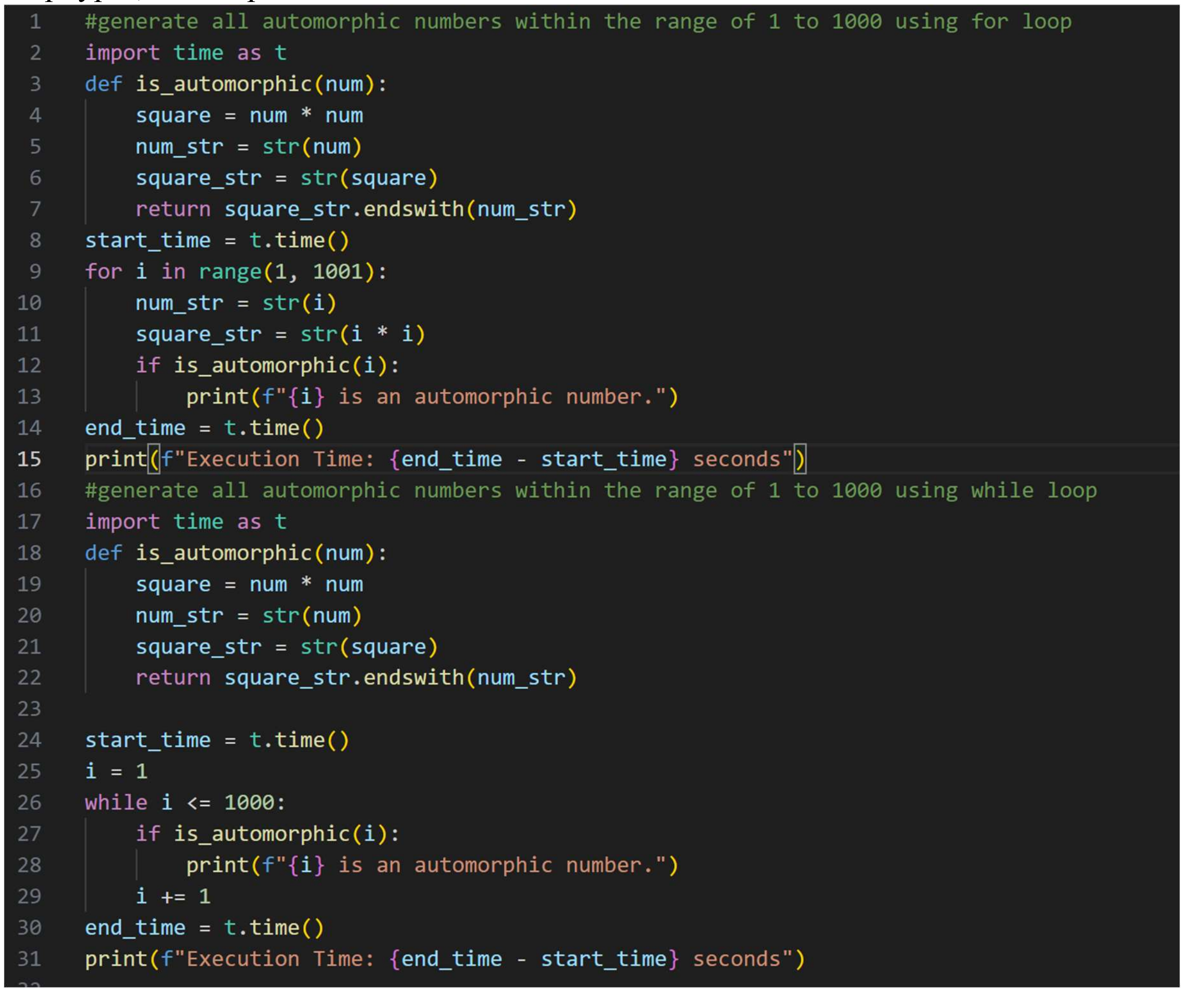
Task Description

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:

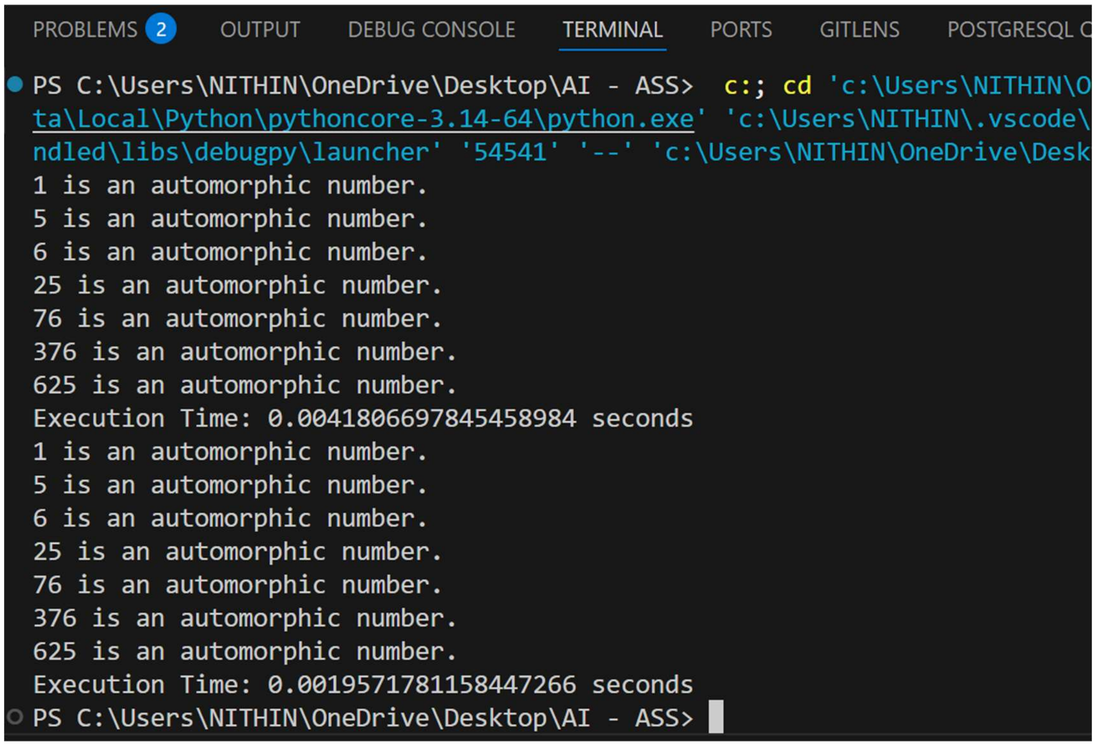
Get AI-generated code to list Automorphic numbers using a for loop. o Analyse the correctness and anciency of the generated logic. o Ask AI to regenerate using a while loop and compare both implementations.

Expected Output #1: Correct implementation that lists Automorphic numbers using both loop types, with explanation



```python
#generate all automorphic numbers within the range of 1 to 1000 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)
start_time = t.time()
for i in range(1, 1001):
    num_str = str(i)
    square_str = str(i * i)
    if is_automorphic(i):
        print(f"{i} is an automorphic number.")
end_time = t.time()
print(f"Execution Time: {end_time - start_time} seconds")
#generate all automorphic numbers within the range of 1 to 1000 using while 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)

start_time = t.time()
i = 1
while i <= 1000:
    if is_automorphic(i):
        print(f"{i} is an automorphic number.")
    i += 1
end_time = t.time()
print(f"Execution Time: {end_time - start_time} seconds")
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\O
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\
ndled\libs\debugpy\launcher' '54541' '--' 'c:\Users\NITHIN\OneDrive\Desk
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.
Execution Time: 0.0041806697845458984 seconds
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.
Execution Time: 0.0019571781158447266 seconds
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

2.Task Description

(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:

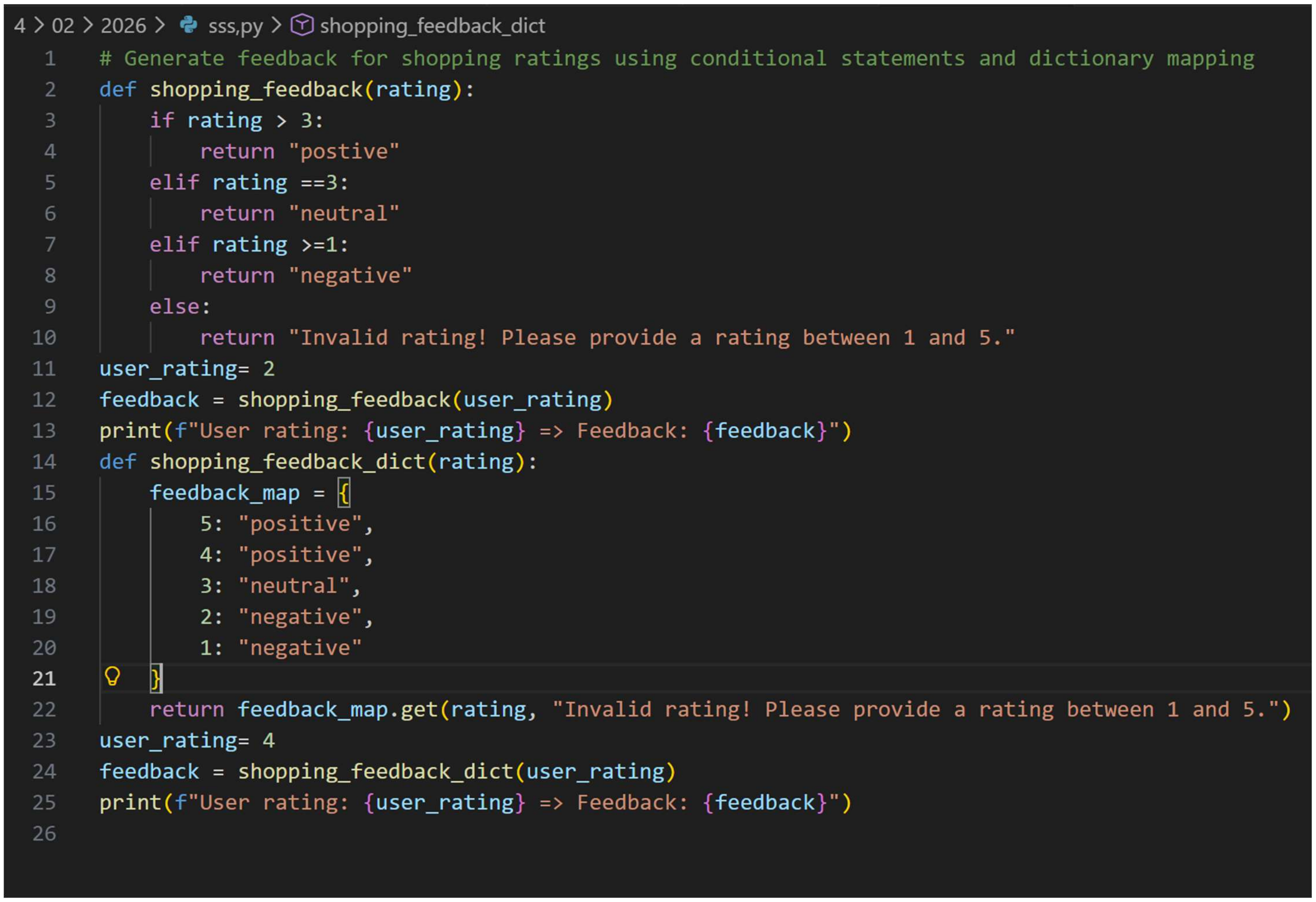
o Generate initial code using nested if-elif-else.

o Analyze correctness and readability.

o Ask AI to rewrite using dictionary-based or match-case structure.

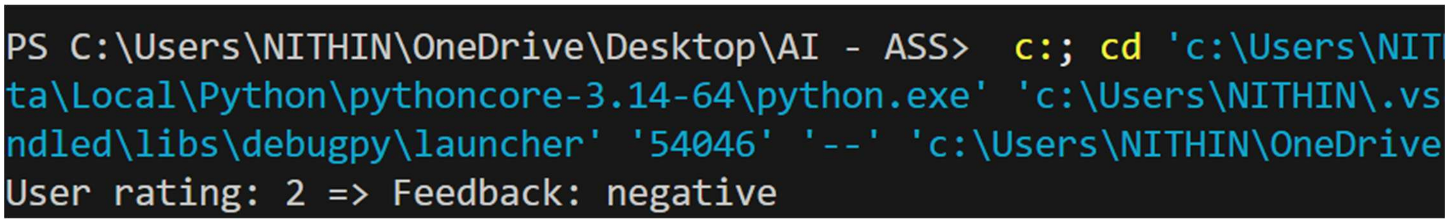
Expected Output #2:

• Feedback classification function with explanation and an alternative approach.



```
4 > 02 > 2026 > sss.py > shopping_feedback_dict
# Generate feedback for shopping ratings using conditional statements and dictionary mapping
def shopping_feedback(rating):
    if rating > 3:
        return "postive"
    elif rating ==3:
        return "neutral"
    elif rating >=1:
        return "negative"
    else:
        return "Invalid rating! Please provide a rating between 1 and 5."
user_rating= 2
feedback = shopping_feedback(user_rating)
print(f"User rating: {user_rating} => Feedback: {feedback}")
def shopping_feedback_dict(rating):
    feedback_map = {
        5: "positive",
        4: "positive",
        3: "neutral",
        2: "negative",
        1: "negative"
    }
    return feedback_map.get(rating, "Invalid rating! Please provide a rating between 1 and 5.")
user_rating= 4
feedback = shopping_feedback_dict(user_rating)
print(f"User rating: {user_rating} => Feedback: {feedback}")
```

Output:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NIT
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vs
ndled\libs\debugpy\launcher' '54046' '--' 'c:\Users\NITHIN\OneDrive
User rating: 2 => Feedback: negative
```

Task 3: Statistical_operations

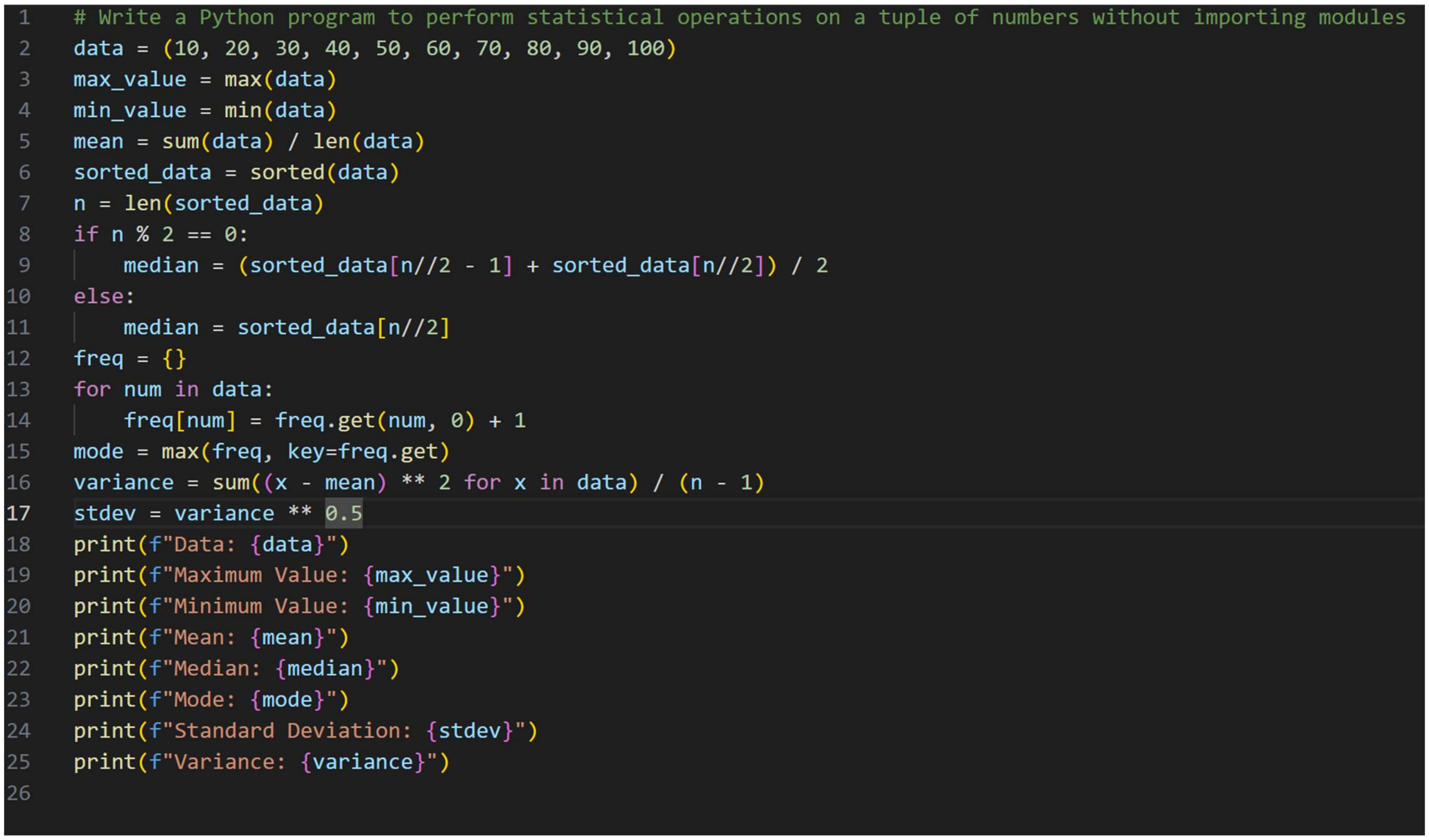
Define a function named statistical operations(tuple Num) that performs the following statistical operations on a tuple of numbers:

• Minimum, Maximum

• Mean, Median, Mode

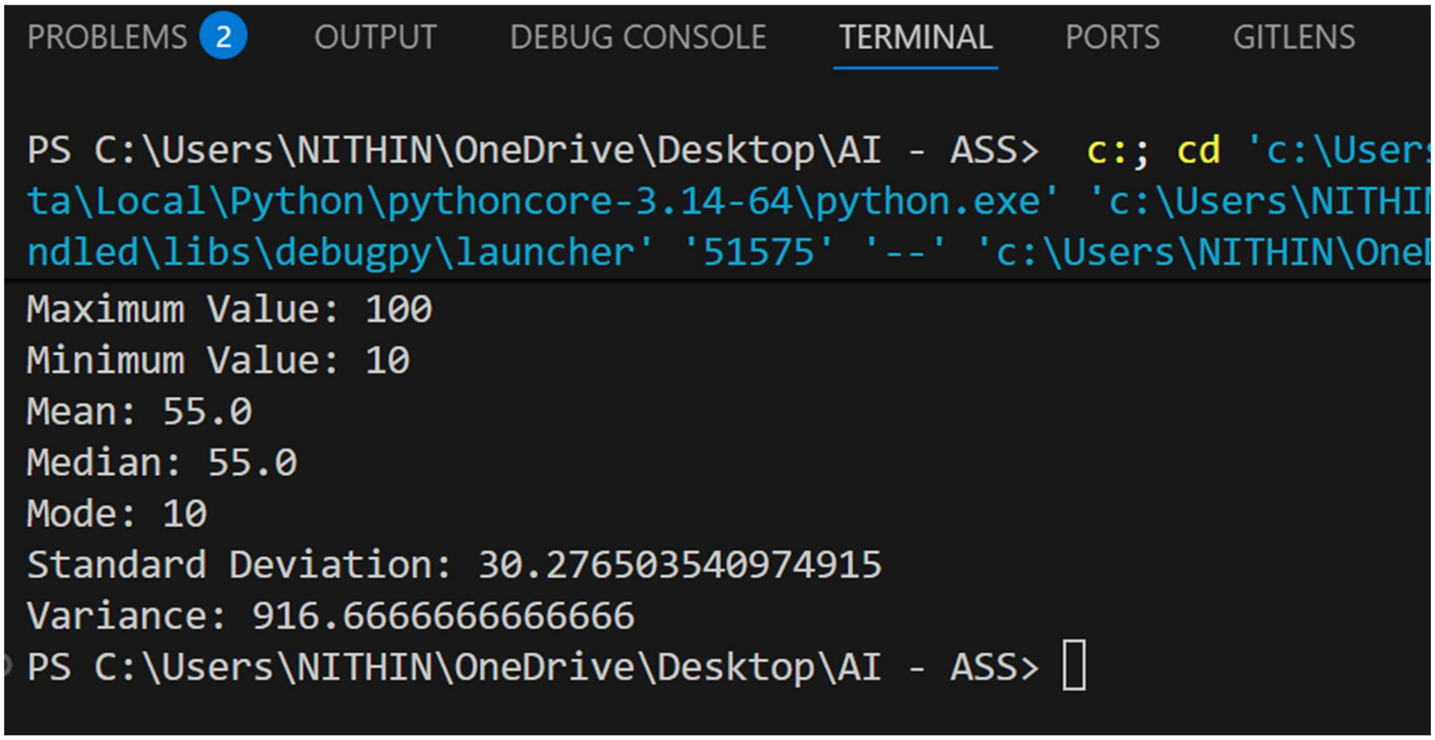
• Variance, Standard Deviation

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



```python
# Write a Python program to perform statistical operations on a tuple of numbers without importing modules
data = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
max_value = max(data)
min_value = min(data)
mean = sum(data) / len(data)
sorted_data = sorted(data)
n = len(sorted_data)
if n % 2 == 0:
    median = (sorted_data[n//2 - 1] + sorted_data[n//2]) / 2
else:
    median = sorted_data[n//2]
freq = {}
for num in data:
    freq[num] = freq.get(num, 0) + 1
mode = max(freq, key=freq.get)
variance = sum((x - mean) ** 2 for x in data) / (n - 1)
stdev = variance ** 0.5
print(f"Data: {data}")
print(f"Maximum Value: {max_value}")
print(f"Minimum Value: {min_value}")
print(f"Mean: {mean}")
print(f"Median: {median}")
print(f"Mode: {mode}")
print(f"Standard Deviation: {stdev}")
print(f"Variance: {variance}")
```

OUTPUT:



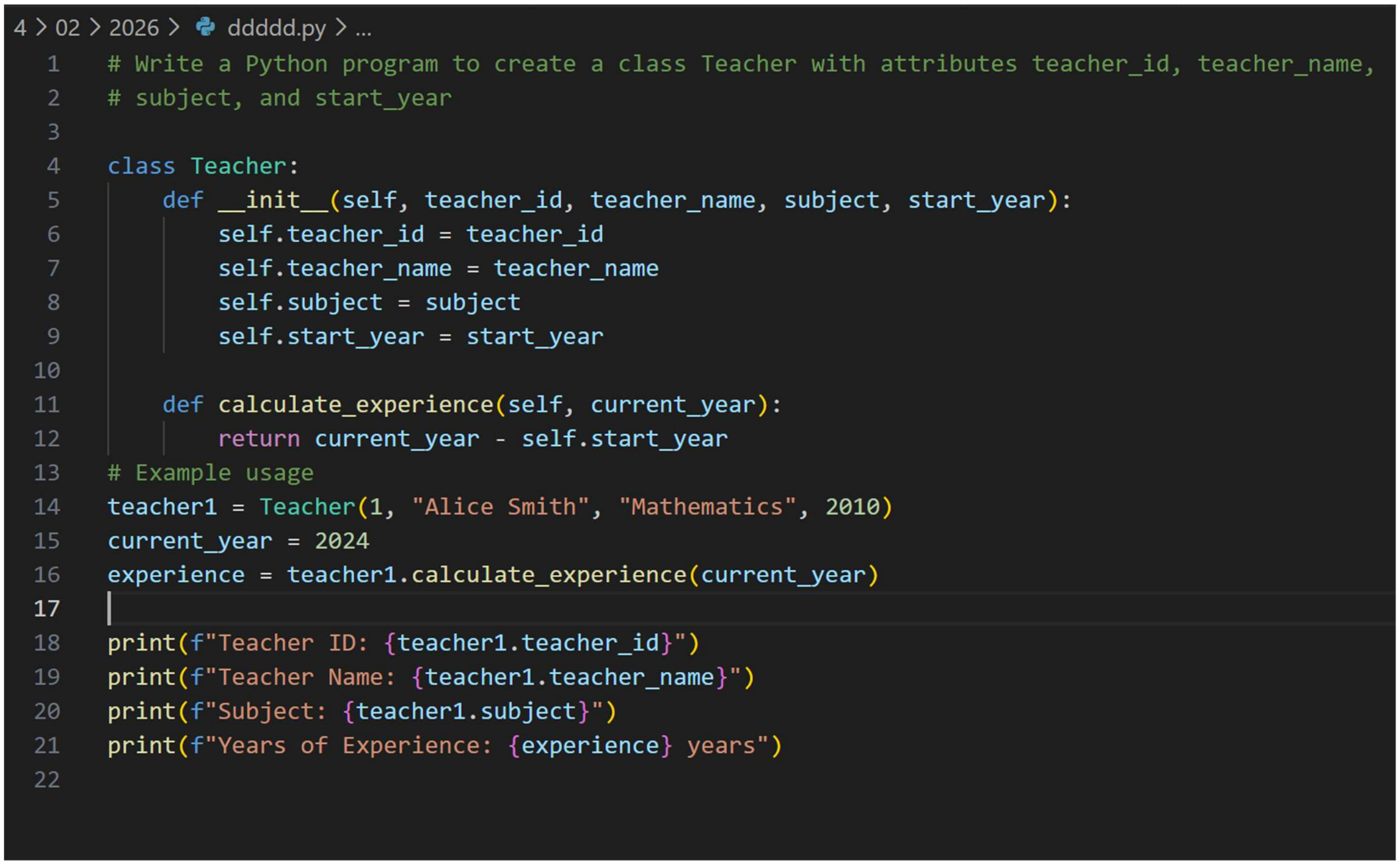
```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHI
ndled\libs\debugpy\launcher' '51575' '--' 'c:\Users\NITHIN\One
Maximum Value: 100
Minimum Value: 10
Mean: 55.0
Median: 55.0
Mode: 10
Standard Deviation: 30.276503540974915
Variance: 916.6666666666666
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS> 
```

Task 4: Teacher Profile

• Prompt: Create a class Teacher with attributes teacher_id, name, subject, and experience.

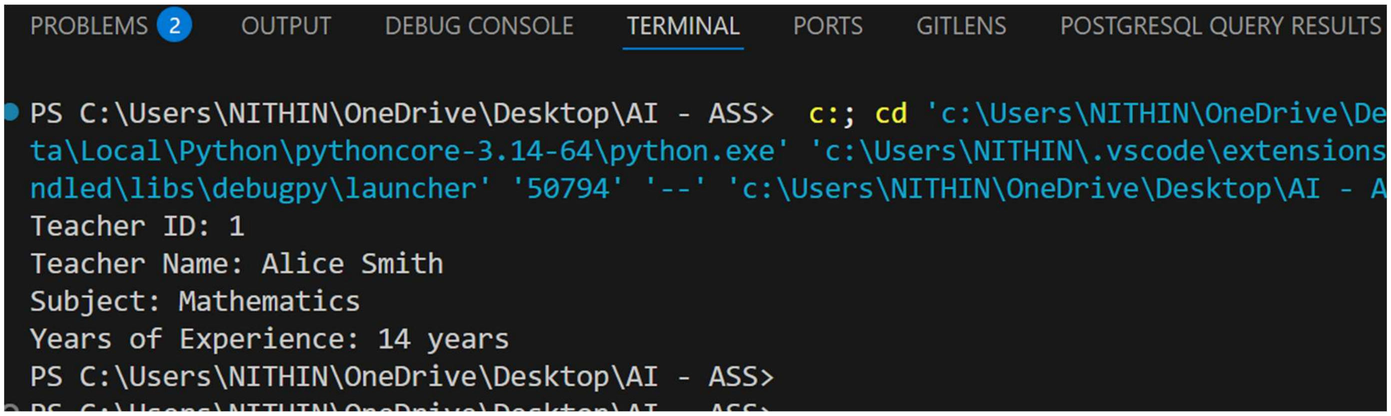
Add a method to display teacher details.

• Expected Output: Class with initializer, method, and object creation.



```
4 > 02 > 2026 > ddddd.py > ...
# Write a Python program to create a class Teacher with attributes teacher_id, teacher_name,
# subject, and start_year

class Teacher:
    def __init__(self, teacher_id, teacher_name, subject, start_year):
        self.teacher_id = teacher_id
        self.teacher_name = teacher_name
        self.subject = subject
        self.start_year = start_year

    def calculate_experience(self, current_year):
        return current_year - self.start_year
# Example usage
teacher1 = Teacher(1, "Alice Smith", "Mathematics", 2010)
current_year = 2024
experience = teacher1.calculate_experience(current_year)

print(f"Teacher ID: {teacher1.teacher_id}")
print(f"Teacher Name: {teacher1.teacher_name}")
print(f"Subject: {teacher1.subject}")
print(f"Years of Experience: {experience} years")

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\De
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\extensions
ndled\libs\debugpy\launcher' '50794' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI - A
Teacher ID: 1
Teacher Name: Alice Smith
Subject: Mathematics
Years of Experience: 14 years
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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:

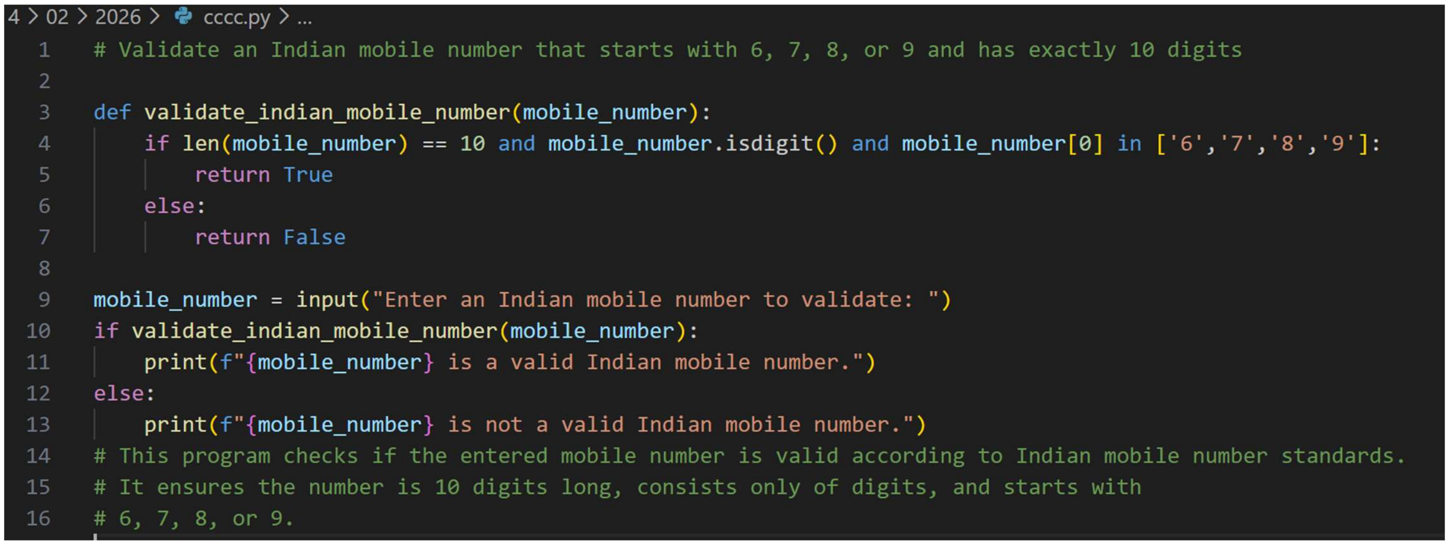
o Starts with 6, 7, 8, or 9 o Contains

exactly 10 digits

Expected Output

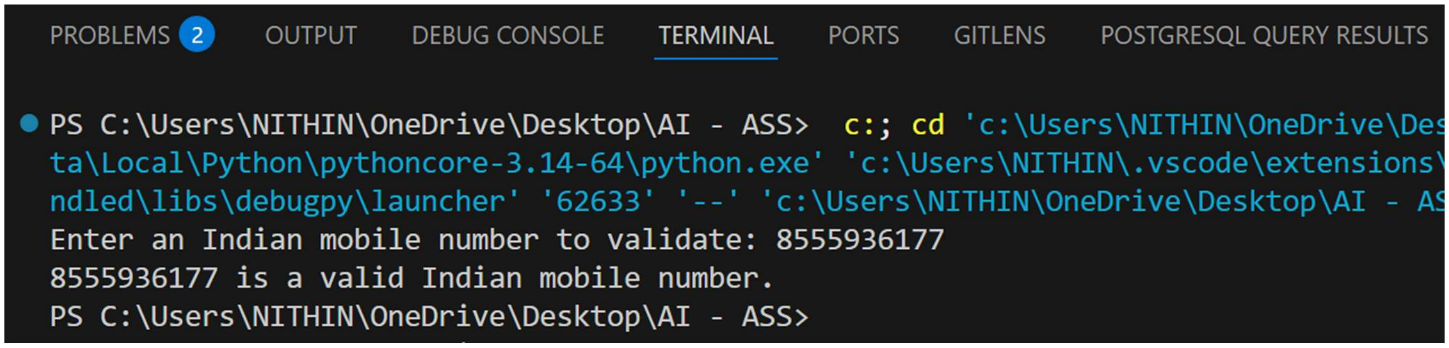
• A valid Python function that performs all required validations without using any input-

output examples in the prompt.



```python
# Validate an Indian mobile number that starts with 6, 7, 8, or 9 and has exactly 10 digits

def validate_indian_mobile_number(mobile_number):
    if len(mobile_number) == 10 and mobile_number.isdigit() and mobile_number[0] in ['6','7','8','9']:
        return True
    else:
        return False

mobile_number = input("Enter an Indian mobile number to validate: ")
if validate_indian_mobile_number(mobile_number):
    print(f"{mobile_number} is a valid Indian mobile number.")
else:
    print(f"{mobile_number} is not a valid Indian mobile number.")
# This program checks if the entered mobile number is valid according to Indian mobile number standards.
# It ensures the number is 10 digits long, consists only of digits, and starts with
# 6, 7, 8, or 9.
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\Des
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\extensions\
ndled\libs\debugpy\launcher' '62633' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI - AS
Enter an Indian mobile number to validate: 8555936177
8555936177 is a valid Indian mobile number.
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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.

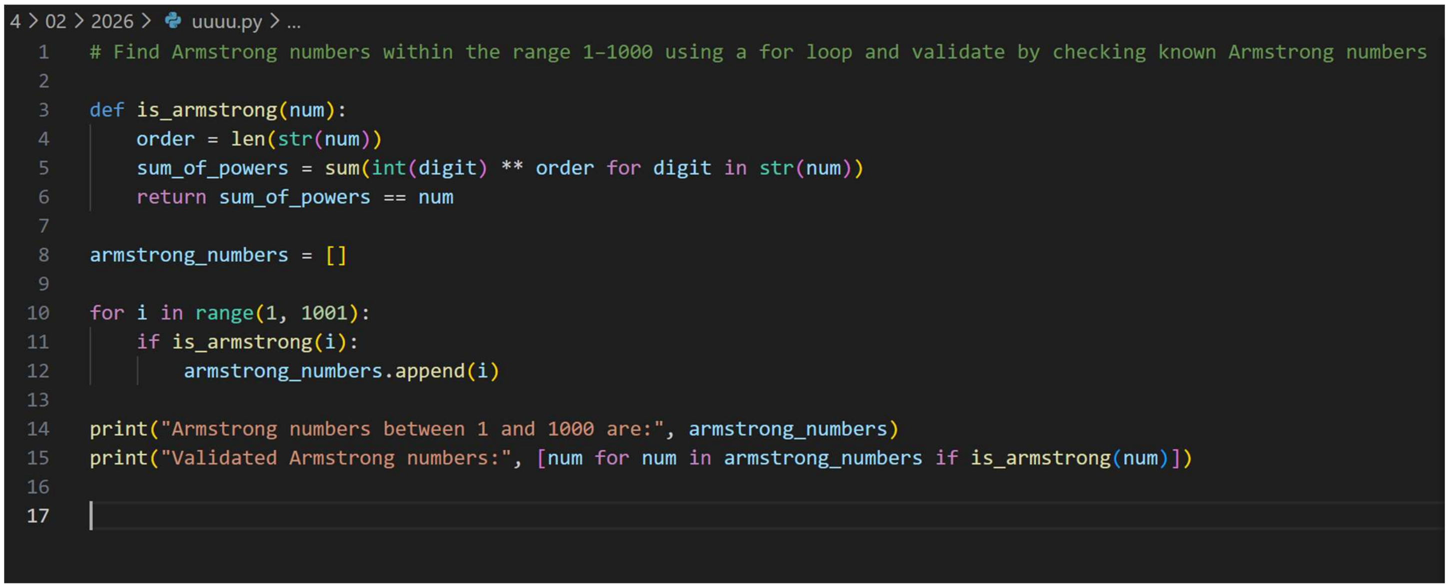
• Validate correctness by checking known Armstrong numbers (153, 370, etc.).

• Ask AI to regenerate an optimized version (using list comprehensions).

Expected Output #7:

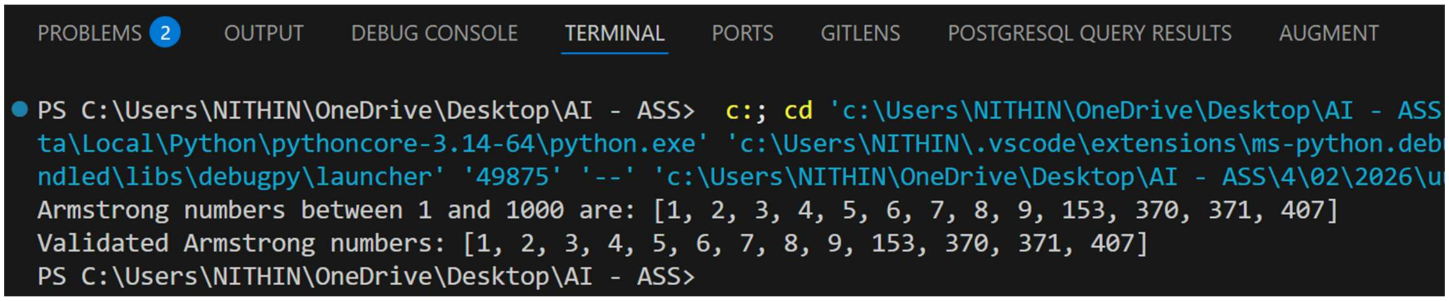
• Python program listing Armstrong numbers in the range.

• Optimized version with explanation.



```python
# Find Armstrong numbers within the range 1-1000 using a for loop and validate by checking known Armstrong numbers

def is_armstrong(num):
    order = len(str(num))
    sum_of_powers = sum(int(digit) ** order for digit in str(num))
    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)
print("Validated Armstrong numbers:", [num for num in armstrong_numbers if is_armstrong(num)])

```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\Desktop\AI - ASS
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\extensions\ms-python.deb
ndled\libs\debugpy\launcher' '49875' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI - ASS\4\02\2026\u
Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
Validated Armstrong numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

Task Description #7 (Loops – Happy Numbers in a Range)

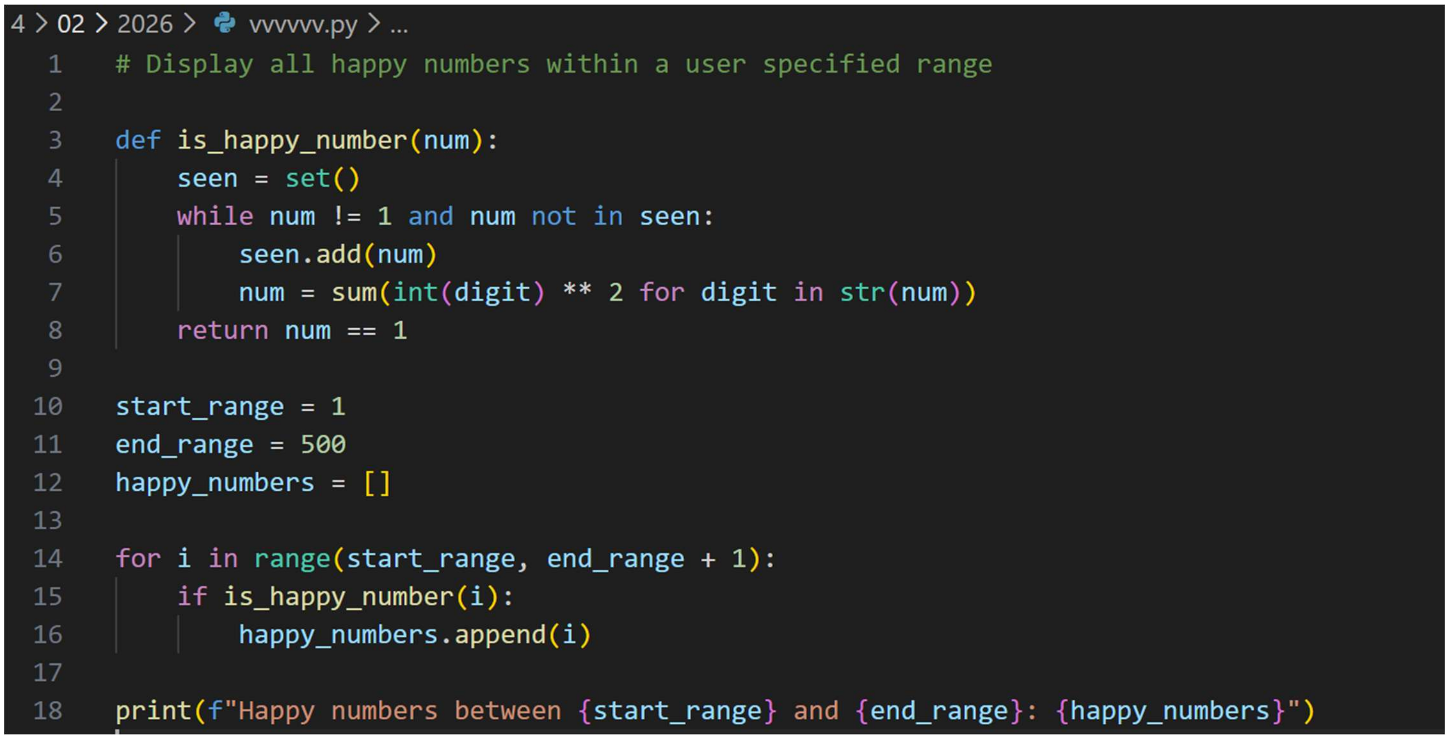
Task: Generate a function using AI that displays all Happy Numbers within a user-specified range (e.g., 1 to 500).

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:

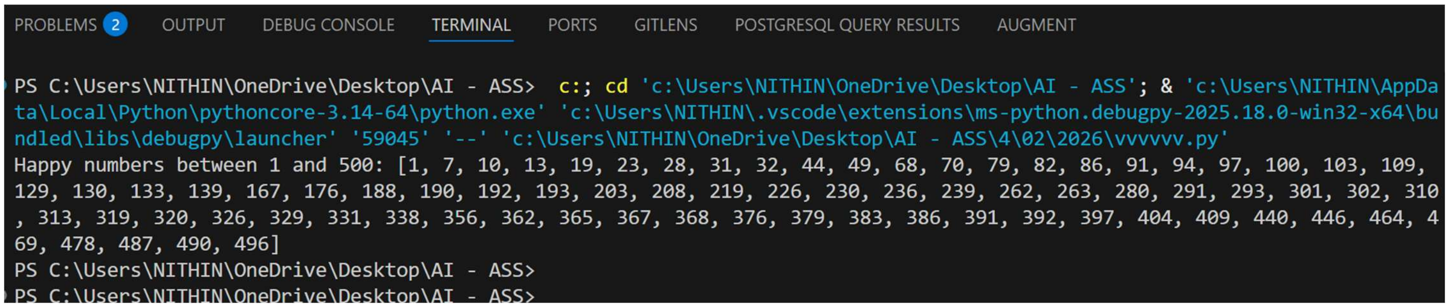
- Python program that prints all Happy Numbers within a range.
- Optimized version using cycle detection with explanation.



4 > 02 > 2026 > vvvvvv.py > ...

```python
# Display all happy numbers within a user specified range

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_range = 1
end_range = 500
happy_numbers = []

for i in range(start_range, end_range + 1):
    if is_happy_number(i):
        happy_numbers.append(i)

print(f"Happy numbers between {start_range} and {end_range}: {happy_numbers}")
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneDrive\Desktop\AI - ASS'; & 'c:\Users\NITHIN\AppDa
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bu
ndled\libs\debugpy\launcher' '59045' '--' 'c:\Users\NITHIN\OneDrive\Desktop\AI - ASS\4\02\2026\vvvvvv.py'
Happy numbers between 1 and 500: [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, 4
69, 478, 487, 490, 496]
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

Task Description #8 (Loops – Strong Numbers in a Range)

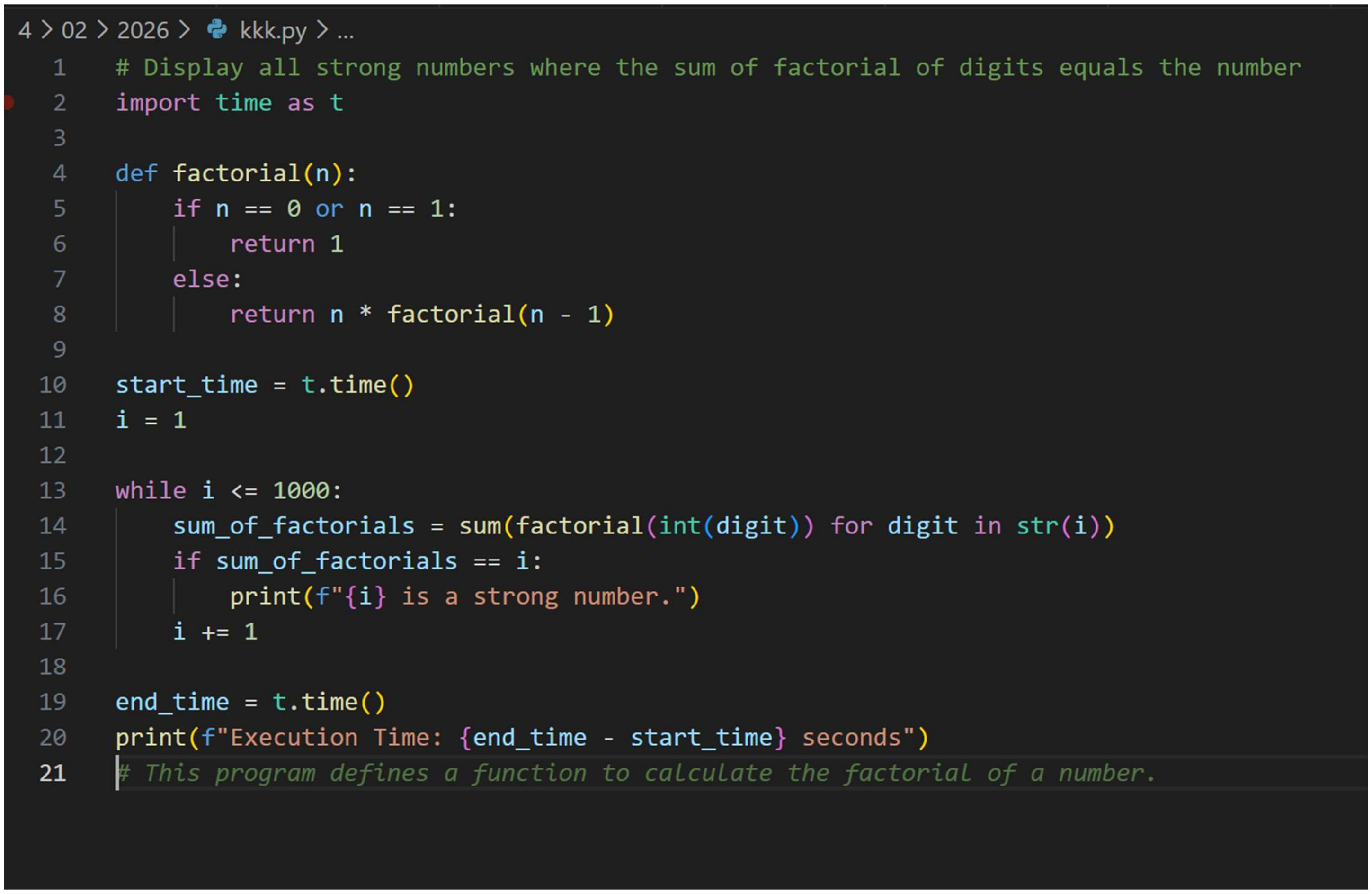
Task: Generate a function using AI that displays all Strong Numbers (sum of factorial of digits equals the number, e.g., 145 = 1!+4!+5!) within a given range.

Instructions:

• Use loops to extract digits and calculate factorials.

• Validate with examples (1, 2, 145).

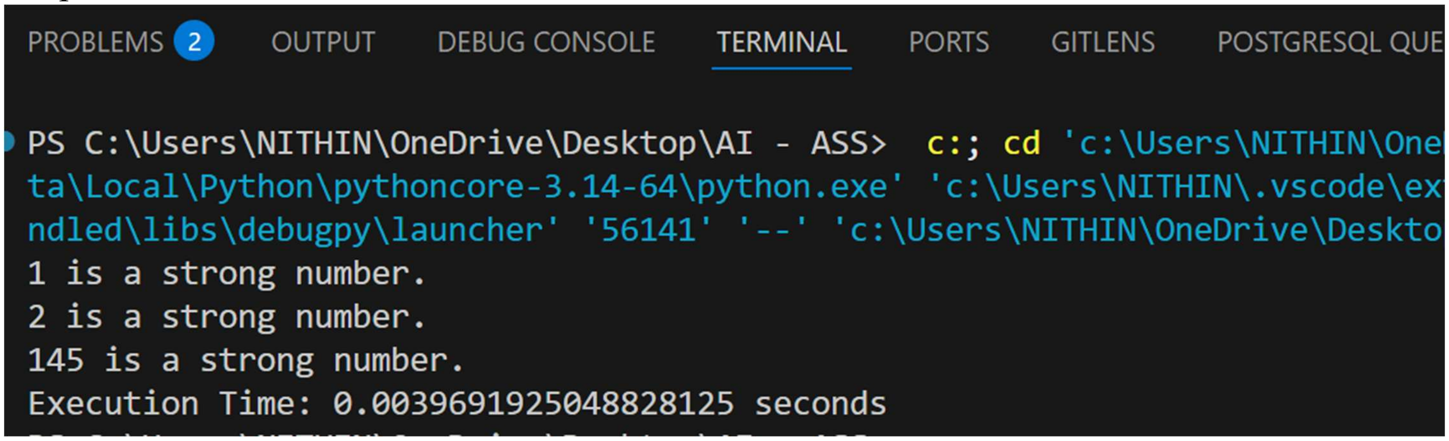
• Ask AI to regenerate an optimized version (precompute digit factorials).



4 > 02 > 2026 > kkk.py > ...

```python
# Display all strong numbers where the sum of factorial of digits equals the number
import time as t

def factorial(n):
    if n == 0 or n == 1:
        return 1
    else:
        return n * factorial(n - 1)

start_time = t.time()
i = 1

while i <= 1000:
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(i))
    if sum_of_factorials == i:
        print(f"{i} is a strong number.")
    i += 1

end_time = t.time()
print(f"Execution Time: {end_time - start_time} seconds")
# This program defines a function to calculate the factorial of a number.
```

Output:



PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS POSTGRESQL QUE

```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\One
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\ex
ndled\libs\debugpy\launcher' '56141' '--' 'c:\Users\NITHIN\OneDrive\Deskto
1 is a strong number.
2 is a strong number.
145 is a strong number.
Execution Time: 0.0039691925048828125 seconds
```

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:

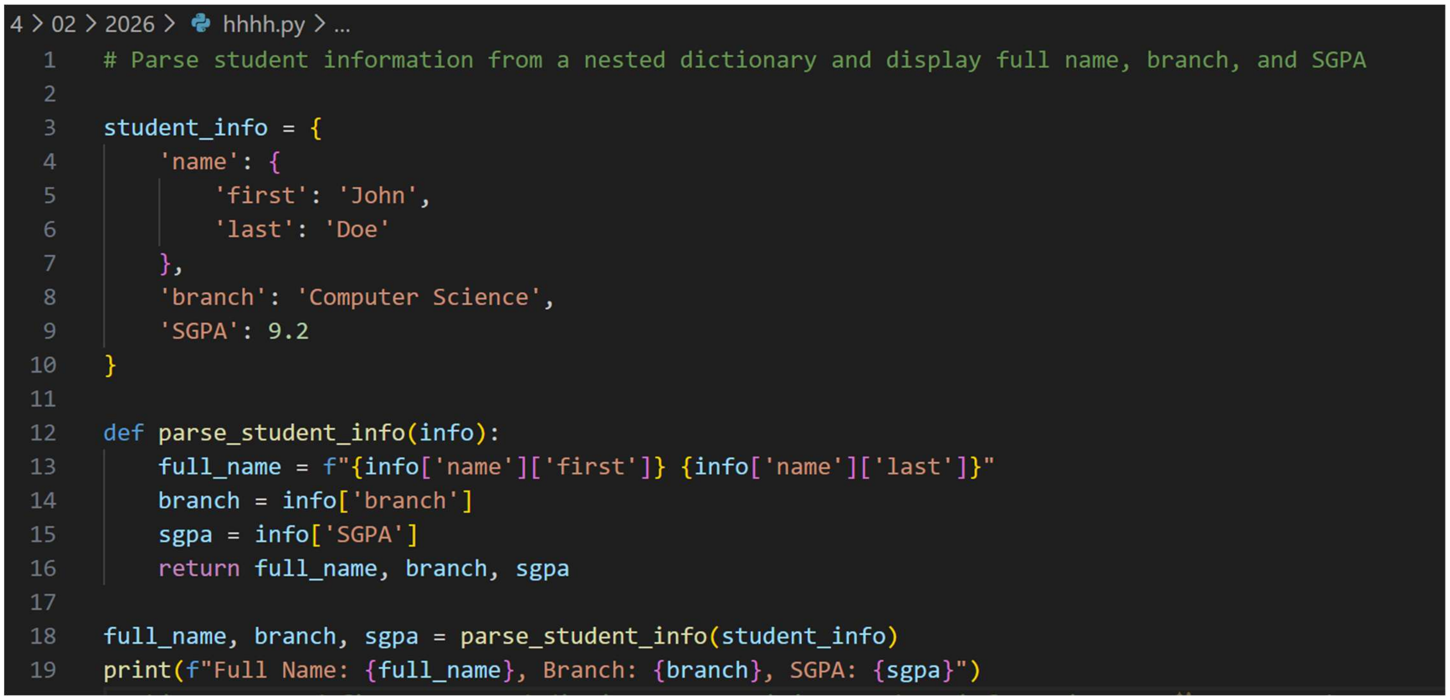
o Full Name o Branch o

SGPA

Expected Output

A reusable Python function that correctly navigates and extracts values from nested

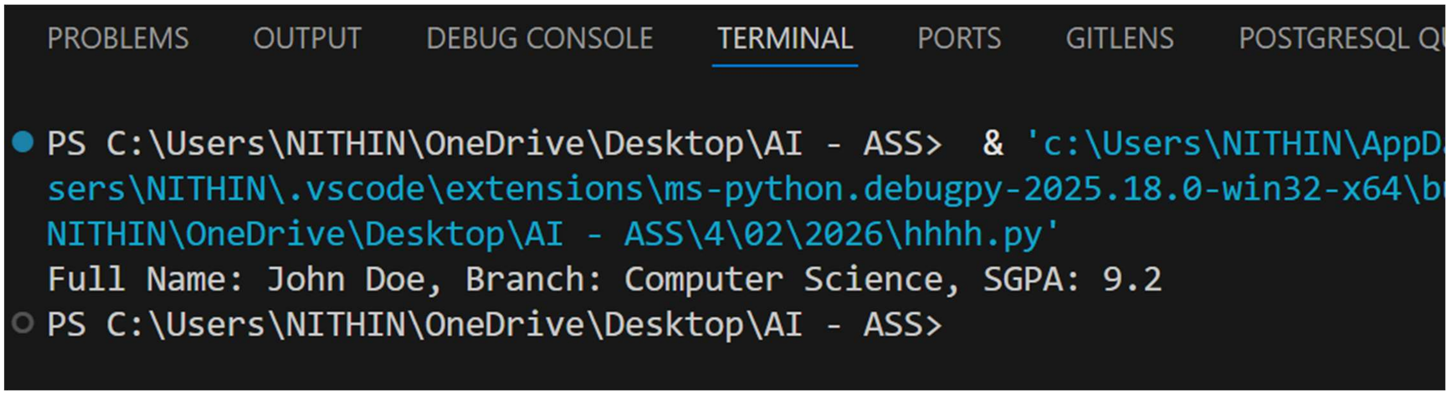
dictionaries based on the provided examples



4 > 02 > 2026 > hhhh.py > ...

```python
# Parse student information from a nested dictionary and display full name, branch, and SGPA

student_info = {
    'name': {
        'first': 'John',
        'last': 'Doe'
    },
    'branch': 'Computer Science',
    'SGPA': 9.2
}

def parse_student_info(info):
    full_name = f"{info['name']['first']} {info['name']['last']}"
    branch = info['branch']
    sgpa = info['SGPA']
    return full_name, branch, sgpa

full_name, branch, sgpa = parse_student_info(student_info)
print(f"Full Name: {full_name}, Branch: {branch}, SGPA: {sgpa}")
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  & 'c:\Users\NITHIN\AppDa
sers\NITHIN\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bu
NITHIN\OneDrive\Desktop\AI - ASS\4\02\2026\hhhh.py'
Full Name: John Doe, Branch: Computer Science, SGPA: 9.2
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```

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.

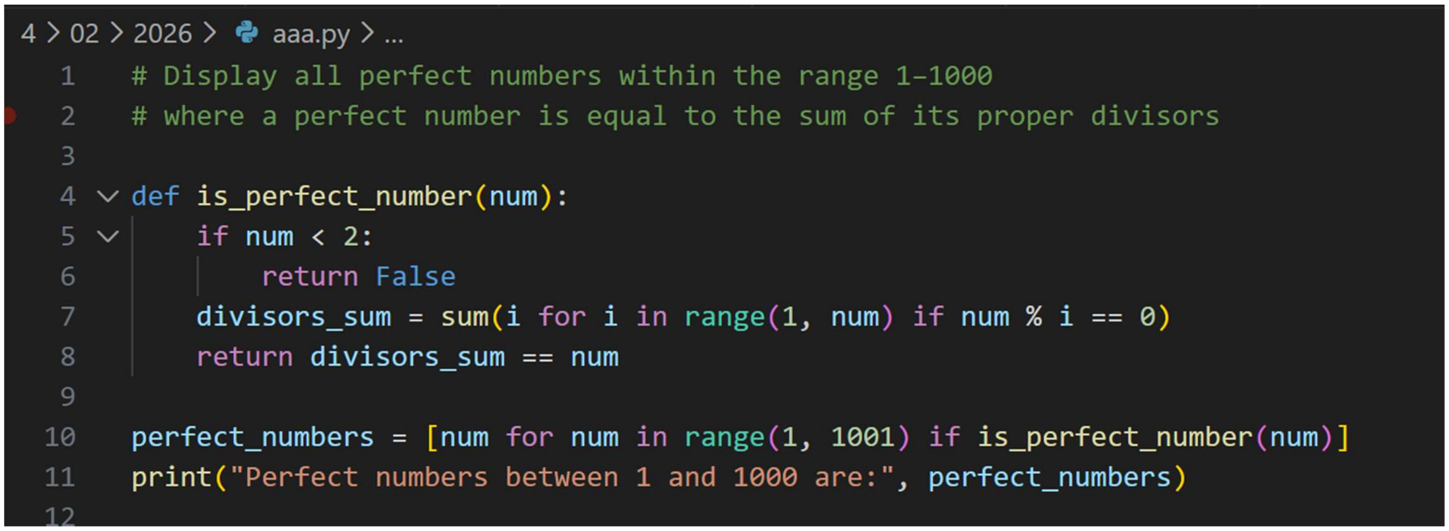
• Validate correctness with known Perfect Numbers (6, 28, 496…).

• Ask AI to regenerate an optimized version (using divisor check only up to √n).

Expected Output #12:

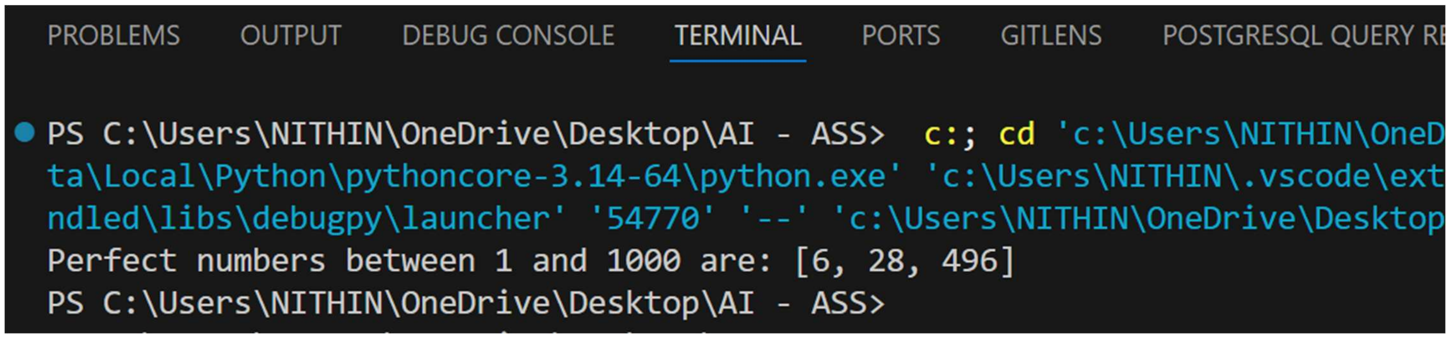
• Python program that lists Perfect Numbers in the given range.

• Optimized version with explanation.



```
4 > 02 > 2026 > aaa.py > ...
# Display all perfect numbers within the range 1-1000
# where a perfect number is equal to the sum of its proper divisors

def is_perfect_number(num):
    if num < 2:
        return False
    divisors_sum = sum(i for i in range(1, num) if num % i == 0)
    return divisors_sum == num

perfect_numbers = [num for num in range(1, 1001) if is_perfect_number(num)]
print("Perfect numbers between 1 and 1000 are:", perfect_numbers)
```

OUTPUT:



```
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>  c:; cd 'c:\Users\NITHIN\OneD
ta\Local\Python\pythoncore-3.14-64\python.exe' 'c:\Users\NITHIN\.vscode\ext
ndled\libs\debugpy\launcher' '54770' '--' 'c:\Users\NITHIN\OneDrive\Desktop
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\NITHIN\OneDrive\Desktop\AI - ASS>
```