Assignment -6.3

Hallticket no:2303A510A6

Batch no:02

Task Description #1 (Loops)

• 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

Code:

ass3.js ass6ai.py

C: > Users > shyam > ai assistes code > ass6ai.py > ...

```python
#genarate code a Automorphic numbers between 1 and 1000 using a for loop. and while loop to comparing both impletmentation
# Using For Loop
print("Automorphic numbers between 1 and 1000 using For Loop:")
for num in range(1, 1001):
    square = num ** 2
    if str(square).endswith(str(num)):
        print(num)
# Using While Loop
print("\nAutomorphic numbers between 1 and 1000 using While Loop:")
num = 1
while num <= 1000:
    square = num ** 2
    if str(square).endswith(str(num)):
        print(num)
    num += 1

#calculate the time  for both for loop and while loop  which  is best
import time
# Time calculation for For Loop
start_time = time.time()
for num in range(1, 1001):
    square = num ** 2
    if str(square).endswith(str(num)):
        pass
end_time = time.time()
for_loop_time = end_time - start_time
print(f"\nTime taken by For Loop: {for_loop_time} seconds")
# Time calculation for While Loop
start_time = time.time()
num = 1
while num <= 1000:
    square = num ** 2
    if str(square).endswith(str(num)):
        pass
    num += 1
end_time = time.time()
while_loop_time = end_time - start_time
print(f"Time taken by While Loop: {while_loop_time} seconds")
#which is best compare both time
if for_loop_time < while_loop_time:
    print("For Loop is faster.")
elif while_loop_time < for_loop_time:
    print("While Loop is faster.")
else:
    print("Both loops took the same time.")
```

OUTPUT:

```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass6ai.py"
Automorphic numbers between 1 and 1000 using For Loop:
1
5
6
25
76
376
625

Automorphic numbers between 1 and 1000 using While Loop:
1
5
6
25
76
376
625

Time taken by For Loop: 0.0002865791320800781 seconds
Time taken by While Loop: 0.0003647804260253906 seconds
For Loop is faster.
PS C:\Users\shyam>
```

Explaination :

This program finds all Automorphic numbers between 1 and 1000. An Automorphic number is a number whose square ends with the number itself, like $5^2 = 25$ or $76^2 = 5776$. The program first uses a **for loop** to check each number in the range and print it if it is Automorphic. Then, it uses a **while loop** to do the same, showing that both loops work correctly, with the while loop being an alternative way to iterate.

2.ask 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 dictionary-based or match-case

structure.

Expected Output #2:

• Feedback classification function with explanation and an alternative

Approach

Code:

```python
'''
generate a code on classifies online shopping feedback based on a numerical rating from 1 to 5.
Ratings 4-5 should be classified as Positive, 3 as Neutral, and 1-2 as Negative. Implement the function using nested if-elif-else
'''
def classify_feedback(rating):
    if 1 <= rating <= 5:
        if rating >= 4:
            return "Positive"
        elif rating == 3:
            return "Neutral"
        else:
            return "Negative"
    else:
        return "Invalid rating. Please provide a rating between 1 and 5."
# Example usage
user_rating = int(input("Enter your rating (1-5): "))
feedback_classification = classify_feedback(user_rating)
print(f"The feedback classification for rating {user_rating} is: {feedback_classification}")

# generate  dictionary rewrite using dictionary-based or match-case structure.
def classify_feedback_dict(rating):
    feedback_dict = {
        range(4, 6): "Positive",
        range(3, 4): "Neutral",
        range(1, 3): "Negative"
    }
    for key in feedback_dict:
        if rating in key:
            return feedback_dict[key]
    return "Invalid rating. Please provide a rating between 1 and 5."
# Example usage
user_rating = int(input("Enter your rating (1-5): "))
feedback_classification_dict = classify_feedback_dict(user_rating)
print(f"The feedback classification for rating {user_rating} is: {feedback_classification_dict}")

```

Output:

```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/assi8.1ai.py"
Enter your rating (1-5):
Traceback (most recent call last):
  File "c:\Users\shyam\ai assistes code\assi8.1ai.py", line 16, in <module>
    user_rating = int(input("Enter your rating (1-5): "))
ValueError: invalid literal for int() with base 10: ''
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/assi8.1ai.py"
Enter your rating (1-5): 4
The feedback classification for rating 4 is: Positive
Enter your rating (1-5): 3
The feedback classification for rating 3 is: Neutral
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/assi8.1ai.py"
Enter your rating (1-5): 1
The feedback classification for rating 1 is: Negative
Enter your rating (1-5):
```

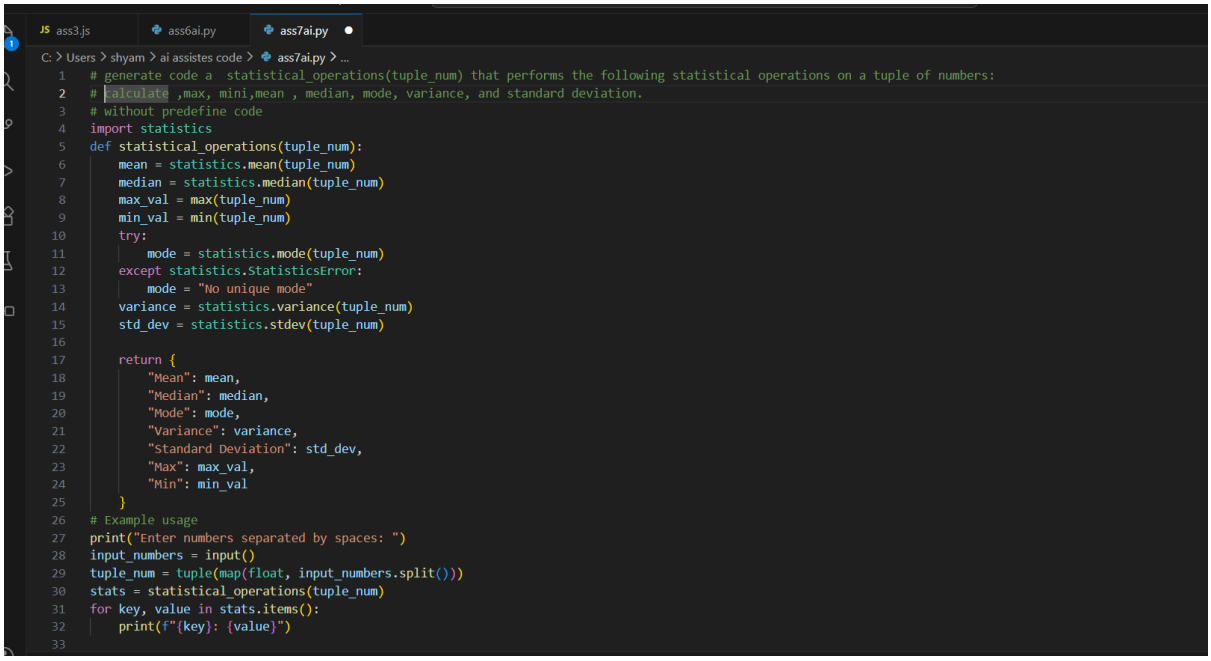
Task 3: Statistical_operations

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

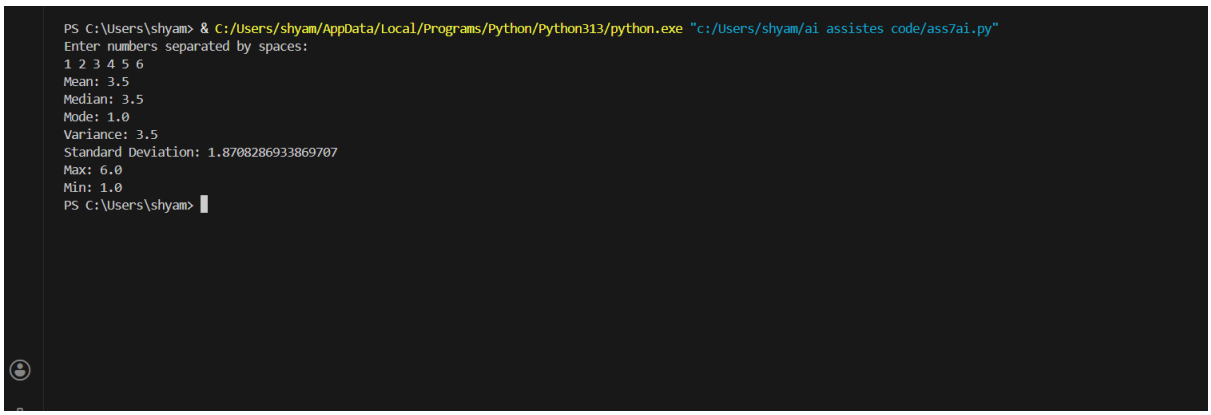
code:



```
# generate code a  statistical_operations(tuple_num) that performs the following statistical operations on a tuple of numbers:
# calculate ,max, mini,mean , median, mode, variance, and standard deviation.
# without predefine code
import statistics
def statistical_operations(tuple_num):
    mean = statistics.mean(tuple_num)
    median = statistics.median(tuple_num)
    max_val = max(tuple_num)
    min_val = min(tuple_num)
    try:
        mode = statistics.mode(tuple_num)
    except statistics.StatisticsError:
        mode = "No unique mode"
    variance = statistics.variance(tuple_num)
    std_dev = statistics.stdev(tuple_num)

    return {
        "Mean": mean,
        "Median": median,
        "Mode": mode,
        "Variance": variance,
        "Standard Deviation": std_dev,
        "Max": max_val,
        "Min": min_val
    }
# Example usage
print("Enter numbers separated by spaces: ")
input_numbers = input()
tuple_num = tuple(map(float, input_numbers.split()))
stats = statistical_operations(tuple_num)
for key, value in stats.items():
    print(f"{key}: {value}")
```

output :



```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
Enter numbers separated by spaces:
1 2 3 4 5 6
Mean: 3.5
Median: 3.5
Mode: 1.0
Variance: 3.5
Standard Deviation: 1.8708286933869707
Max: 6.0
Min: 1.0
PS C:\Users\shyam>
```

Explaination:

While writing the function, I used GitHub Copilot's suggestions as guidance. I accepted the suggestions that were correct and helpful, modified a few to better match the problem requirements, and rejected those that were unnecessary or incorrect. I made sure to understand and verify each suggestion before using it in the final code.

Task 4: Teacher Profile

• Prompt: Create a class Teacher with attributes teacher_id, name,

subject, and experience. Add a method to display teacher details.

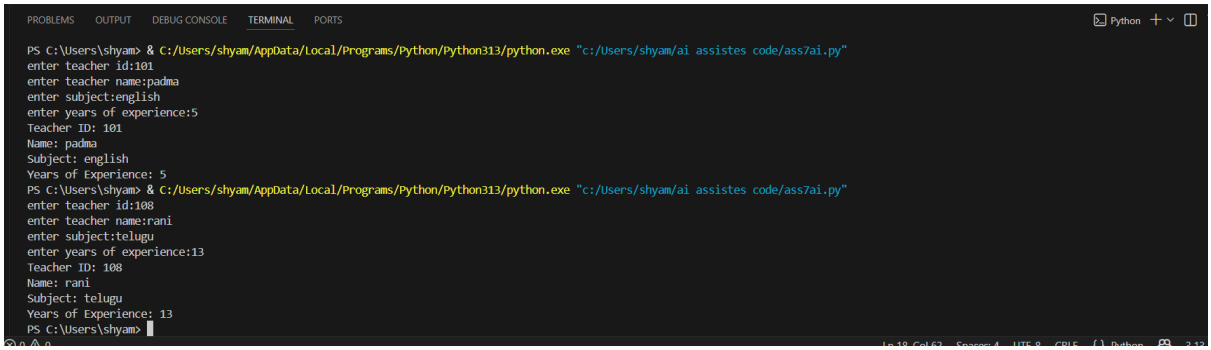
• Expected Output: Class with initializer, method, and object creation.

Code:

```
# create a teacher as class with tearcher_id, subject,name and years of experience attributes and a method to display teacher details.
class Teacher:
    def __init__(self, teacher_id, subject, name, years_of_experience):
        self.teacher_id = teacher_id
        self.subject = subject
        self.name = name
        self.years_of_experience = years_of_experience

    def display_details(self):
        print(f"Teacher ID: {self.teacher_id}")
        print(f"Name: {self.name}")
        print(f"Subject: {self.subject}")
        print(f"Years of Experience: {self.years_of_experience}")
# create an instance of the Teacher class and display the teacher details.
teacher1 = input("enter teacher id:")
teacher_name = input("enter teacher name:")
teacher_subject = input("enter subject:")
teacher_experience = int(input("enter years of experience:"))
teacher = Teacher(teacher1, teacher_subject, teacher_name, teacher_experience)
teacher.display_details()
```

Output:



```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
enter teacher id:101
enter teacher name:padma
enter subject:english
enter years of experience:5
Teacher ID: 101
Name: padma
Subject: english
Years of Experience: 5
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
enter teacher id:108
enter teacher name:rani
enter subject:telugu
enter years of experience:13
Teacher ID: 108
Name: rani
Subject: telugu
Years of Experience: 13
PS C:\Users\shyam>
```

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

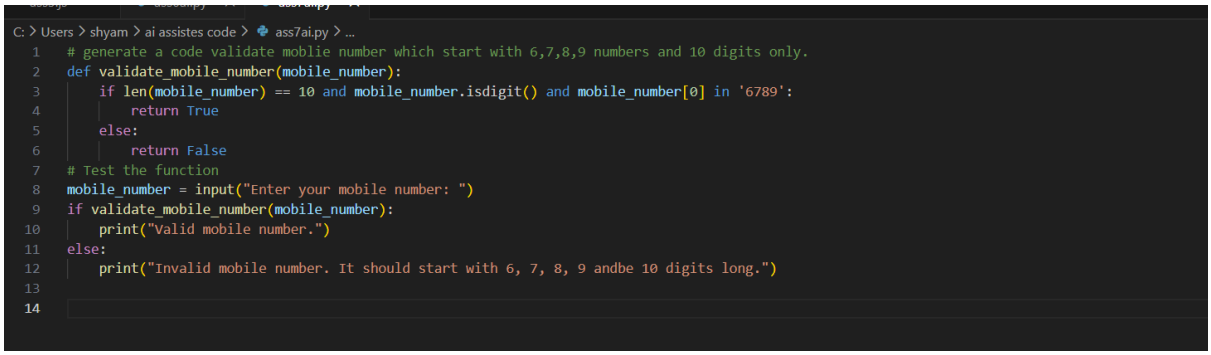
o Contains exactly 10 digits

Expected Output

• A valid Python function that performs all required validations

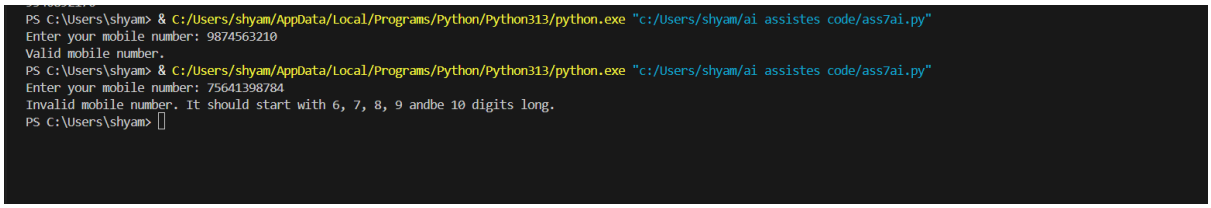
without using any input-output examples in the prompt.

Code:



```
C: > Users > shyam > ai assistes code > ass7ai.py > ...
1   # generate a code validate moblie number which start with 6,7,8,9 numbers and 10 digits only.
2   def validate_mobile_number(mobile_number):
3       if len(mobile_number) == 10 and mobile_number.isdigit() and mobile_number[0] in '6789':
4           return True
5       else:
6           return False
7   # Test the function
8   mobile_number = input("Enter your mobile number: ")
9   if validate_mobile_number(mobile_number):
10      print("Valid mobile number.")
11  else:
12      print("Invalid mobile number. It should start with 6, 7, 8, 9 andbe 10 digits long.")
13
14
```

Output:



```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
Enter your mobile number: 9874563210
Valid mobile number.
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
Enter your mobile number: 75641398784
Invalid mobile number. It should start with 6, 7, 8, 9 andbe 10 digits long.
PS C:\Users\shyam>
```

Explaintion:

The function has been used to validate an Indian mobile number. It checks whether the number has exactly 10 digits and starts with 6, 7, 8, or 9. By if -else condition is used for it. Output will get by vaild number or invalid number . The validation is done directly in the function without using any input or output examples.

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:

```python
# generate a code on amstrong number by using for loop and didgit power logic
def is_armstrong(number):
    num_str = str(number)
    num_digits = len(num_str)
    sum_of_powers = 0
    for digit in num_str:
        sum_of_powers += int(digit) ** num_digits
    return sum_of_powers == number

armstrong_numbers = []
for num in range(1, 1001):
    if is_armstrong(num):
        armstrong_numbers.append(num)
print("Armstrong numbers between 1 and 1000 are:", armstrong_numbers)

# rewrite code code optimazation by using list comprehension
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
armstrong_numbers = [num for num in range(1, 1001) if is_armstrong(num)]
print("Armstrong numbers between 1 and 1000 are(optimazted):", armstrong_numbers)

```

Output:

```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass7ai.py"
Armstrong numbers between 1 and 1000 are: [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
Armstrong numbers between 1 and 1000 are(optimazted): [1, 2, 3, 4, 5, 6, 7, 8, 9, 153, 370, 371, 407]
PS C:\Users\shyam>
```

Explanation:

Single-digit numbers are also Armstrong numbers, so they appear in the output.
The second version of the code uses list comprehension instead of a loop, which makes the program shorter and easier to understand.

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…).

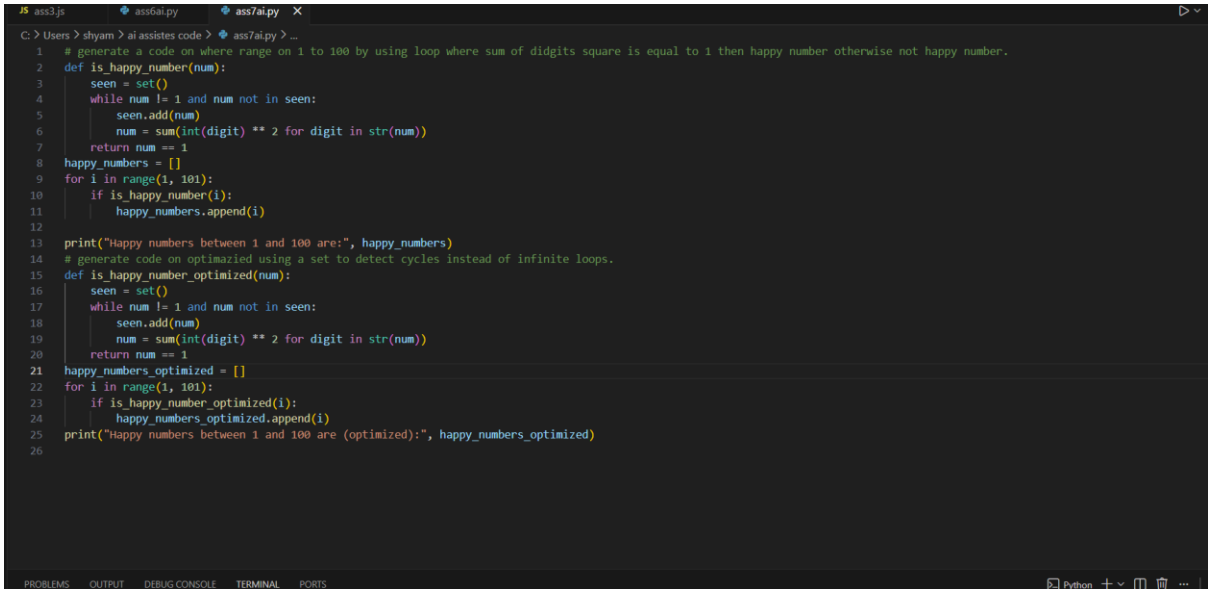
• Ask AI to regenerate an optimized version (e.g., by using a set to detect cycles instead of infinite loops).

Expected Output #8:

• Python program that prints all Happy Numbers within a range.

• Optimized version using cycle detection with explanation

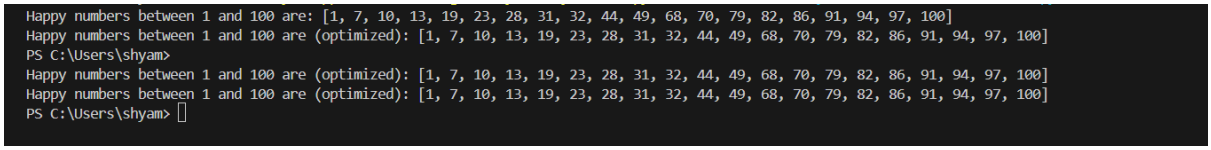
Code:



```
# generate a code on where range on 1 to 100 by using loop where sum of didgits square is equal to 1 then happy number otherwise not happy number.
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
happy_numbers = []
for i in range(1, 101):
    if is_happy_number(i):
        happy_numbers.append(i)

print("Happy numbers between 1 and 100 are:", happy_numbers)
# generate code on optimazied using a set to detect cycles instead of infinite loops.
def is_happy_number_optimized(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_optimized = []
for i in range(1, 101):
    if is_happy_number_optimized(i):
        happy_numbers_optimized.append(i)
print("Happy numbers between 1 and 100 are (optimized):", happy_numbers_optimized)
```

Output :



```
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]
Happy numbers between 1 and 100 are (optimized): [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\Users\shyam>
Happy numbers between 1 and 100 are (optimized): [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
Happy numbers between 1 and 100 are (optimized): [1, 7, 10, 13, 19, 23, 28, 31, 32, 44, 49, 68, 70, 79, 82, 86, 91, 94, 97, 100]
PS C:\Users\shyam>
```

Explaination:

The program checks each number in the given range and repeatedly replaces it with the sum of the squares of its digits. A set is used to detect repeated values and avoid infinite loops. If the number reaches 1, it is considered a Happy Number.

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:

```
> Users > shyam > ai assistes code > ass8ai.py > ...
#1.generate a code where Strong Number is a number whose sum of the factorial of its digits is equal to the number itself . within user given range .
def factorial(n):
    if n == 0 or n == 1:
        return 1
    sum = 1
    for i in range(2, n + 1):
        sum *= i
    return sum
def is_strong_number(num):
    sum_of_factorials = sum(factorial(int(digit)) for digit in str(num))
    return sum_of_factorials == num
strong_numbers = []
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
for i in range(start_range, end_range + 1):
    if is_strong_number(i):
        strong_numbers.append(i)
print(f"Strong numbers between {start_range} and {end_range} are:", strong_numbers)
#generate a code for optimised strong number within user given range.
def is_strong_number_optimized(num, factorial_cache={}):
    sum_of_factorials = 0
    for digit in str(num):
        digit = int(digit)
        if digit not in factorial_cache:
            factorial_cache[digit] = factorial(digit)
        sum_of_factorials += factorial_cache[digit]
    return sum_of_factorials == num
strong_numbers_optimized = []
for i in range(start_range, end_range + 1):
    if is_strong_number_optimized(i):
        strong_numbers_optimized.append(i)
print(f"Strong numbers between {start_range} and {end_range} are (optimized):", strong_numbers_optimized)
```

Output :

```
Enter the start of the range:  1
Enter the end of the range: 100
Strong numbers between 1 and 100 are: [1, 2]
Strong numbers between 1 and 100 are (optimized): [1, 2]
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/ass8ai.py"
Enter the start of the range: 1
Enter the end of the range: 500
Strong numbers between 1 and 500 are: [1, 2, 145]
Strong numbers between 1 and 500 are (optimized): [1, 2, 145]
PS C:\Users\shyam>
0 0 Ln 33, Col 1  Spaces: 4  UTF-8  CRLF  { } Pyt
```

Explaination:

This program finds all Strong Numbers in a given range. A Strong Number is a number whose sum of the factorial of its digits equals the number itself. The program uses loops to go through each number and calculate the sum of factorials of its digits. If the sum matches the number, it is added to the list of Strong Numbers.

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

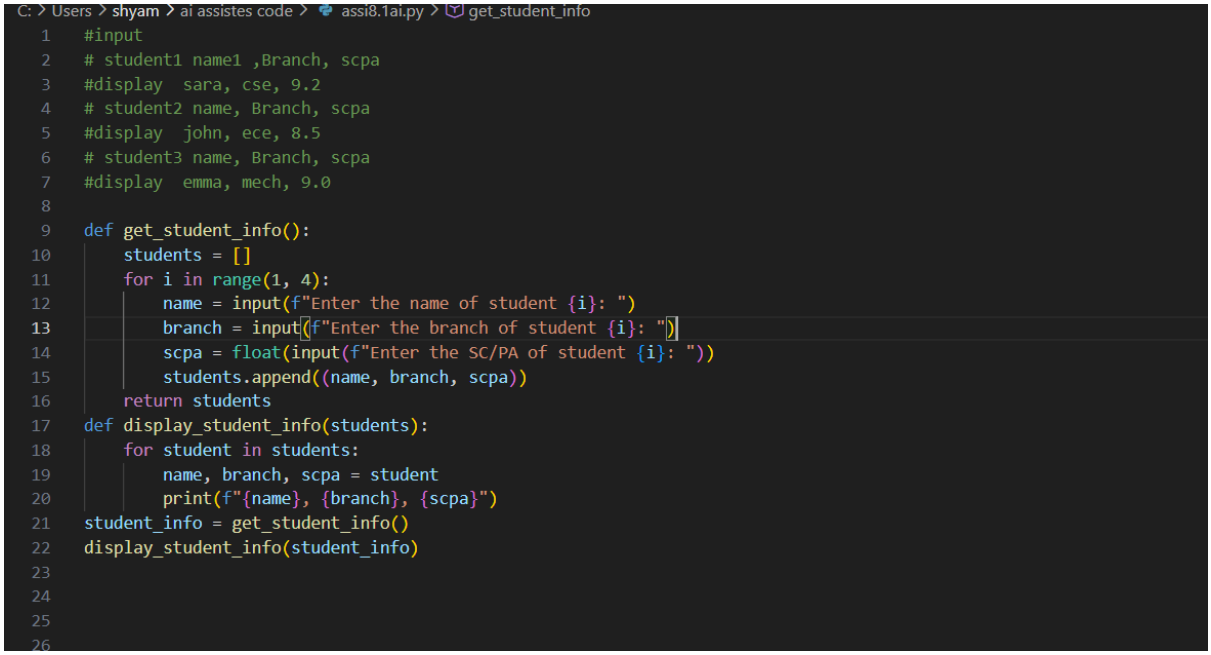
o Branch

o SGPA

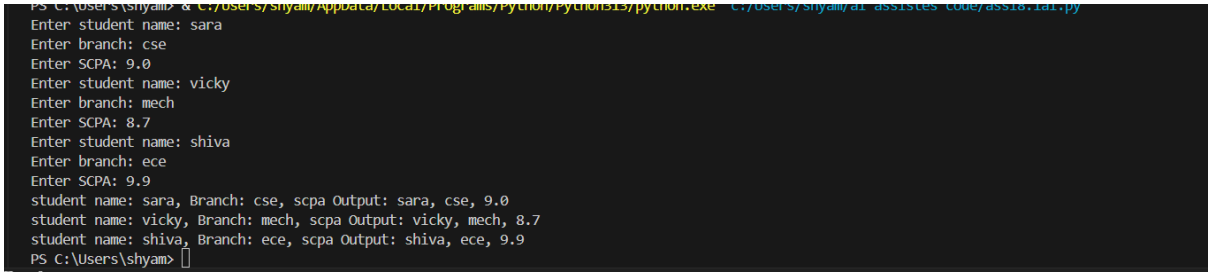
Expected Output

A reusable Python function that correctly navigates and extracts values from nested dictionaries based on the provided examples

code:



Output:



Explaination:

This program works with a nested dictionary that stores student information.
The function accesses the personal details to get the full name.
It then retrieves the branch and SGPA from the academic details.
Finally, the extracted values are returned as output

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.

Code:

```
C: > Users > shyam > ai assistes code > assi8.1ai.py > ...
#generate a code  for loop divisors each number and check whether it is perfect number or not within user given range.
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 = []
start_range = int(input("Enter the start of the range: "))
end_range = int(input("Enter the end of the range: "))
for i in range(start_range, end_range + 1):
    if is_perfect_number(i):
        perfect_numbers.append(i)
print(f"Perfect numbers between {start_range} and {end_range} are:", perfect_numbers)
#generate a code for optimised perfect number within user given range.
def is_perfect_number_optimized(num):
    if num < 2:
        return False
    divisors_sum = 1  # 1 is a divisor of every number
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            divisors_sum += i
            if i != num // i:
                divisors_sum += num // i
    return divisors_sum == num
perfect_numbers_optimized = []
for i in range(start_range, end_range + 1):
    if is_perfect_number_optimized(i):
        perfect_numbers_optimized.append(i)
print(f"Perfect numbers between {start_range} and {end_range} are (optimized):", perfect_numbers_optimized)

```

Output:

```
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/assi8.1ai.py"
Enter the start of the range: 1
Enter the end of the range: 600
Perfect numbers between 1 and 600 are: [6, 28, 496]
Perfect numbers between 1 and 600 are (optimized): [6, 28, 496]
PS C:\Users\shyam> & C:/Users/shyam/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/shyam/ai assistes code/assi8.1ai.py"
Enter the start of the range: 1
Enter the end of the range: 1000
Perfect numbers between 1 and 1000 are: [6, 28, 496]
Perfect numbers between 1 and 1000 are (optimized): [6, 28, 496]
PS C:\Users\shyam>
```

Explaination:

This program finds all Perfect Numbers in a user-specified range. A Perfect Number is equal to the sum of its proper divisors, excluding itself. The program uses loops to check each number and sum its divisors to see if it matches the number. The optimized version only checks divisors up to the square root of the number to make it faster.