

AI Assisted Coding

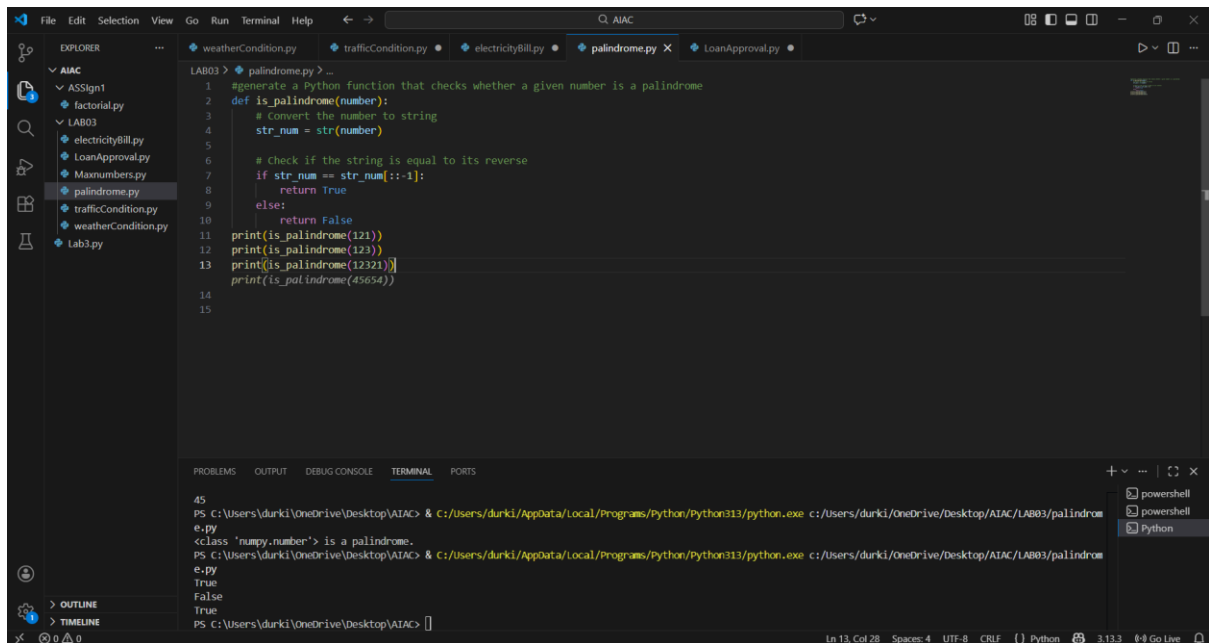
Assignment 3.1

Name:D.Shruthi

HT NO: 2303A52407

Batch:31

Question 1: Zero-Shot Prompting (Palindrome Number Program)

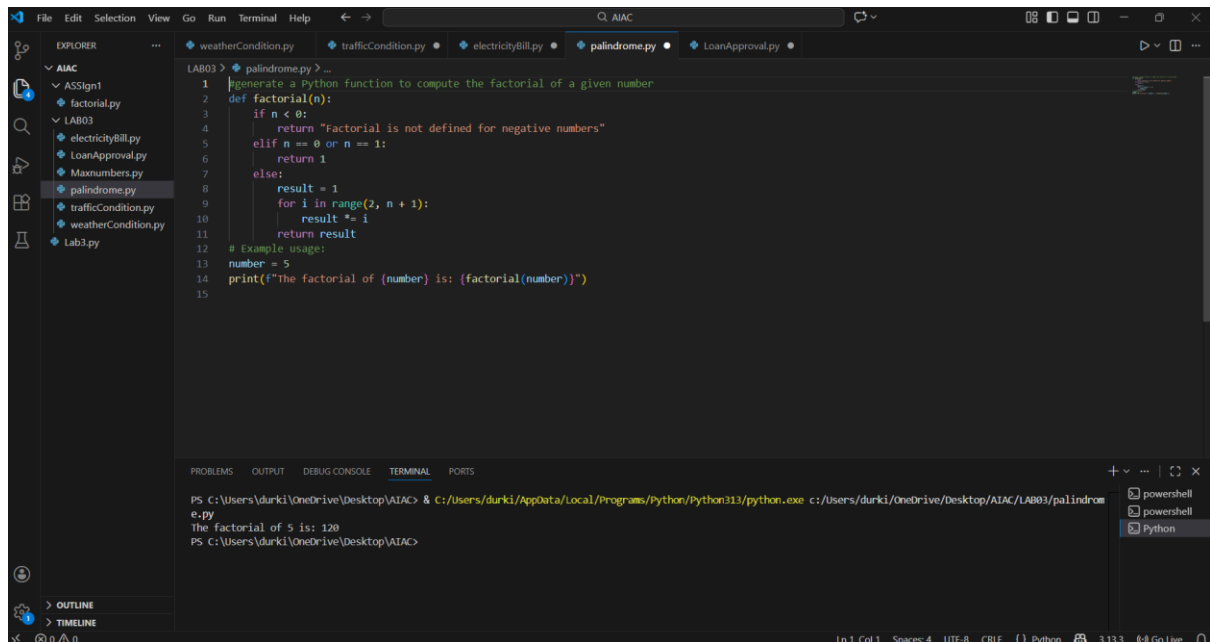


```
LAB03 > palindrom.py > _
1 generate a python function that checks whether a given number is a palindrome
2 def is_palindrome(number):
3     # Convert the number to string
4     str_num = str(number)
5
6     # Check if the string is equal to its reverse
7     if str_num == str_num[::-1]:
8         return True
9     else:
10        return False
11
12 print(is_palindrome(121))
13 print(is_palindrome(123))
14 print(is_palindrome(12321))
15 print(is_palindrome(45654))
16
```

```
45
PS C:\Users\durki\OneDrive\Desktop\AIAC> & c:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrom
e.py
<class 'numpy.number'> is a palindrome.
PS C:\Users\durki\OneDrive\Desktop\AIAC> & c:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrom
e.py
True
False
True
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

In zero-shot prompting, no examples are given to the AI. The AI generates basic code to check whether a number is a palindrome. It works for normal inputs but does not handle edge cases. The solution is simple but limited.

Question 2: One-Shot Prompting (Factorial Calculation)



The screenshot shows a Visual Studio Code editor with a file explorer on the left. The file explorer shows a project named 'AIAC' with a subfolder 'LAB03'. Inside 'LAB03', there are several Python files: 'factorial.py', 'electricityBill.py', 'LoanApproval.py', 'MaxNumbers.py', 'palindrome.py', 'trafficCondition.py', and 'weatherCondition.py'. The 'palindrome.py' file is selected and its content is displayed in the editor. The code is a Python function to compute the factorial of a given number. It includes a docstring, a function definition, a check for negative numbers, a base case, and a recursive case. An example usage is provided at the bottom. The terminal at the bottom shows the command to run the script and the output: 'The factorial of 5 is: 120'.

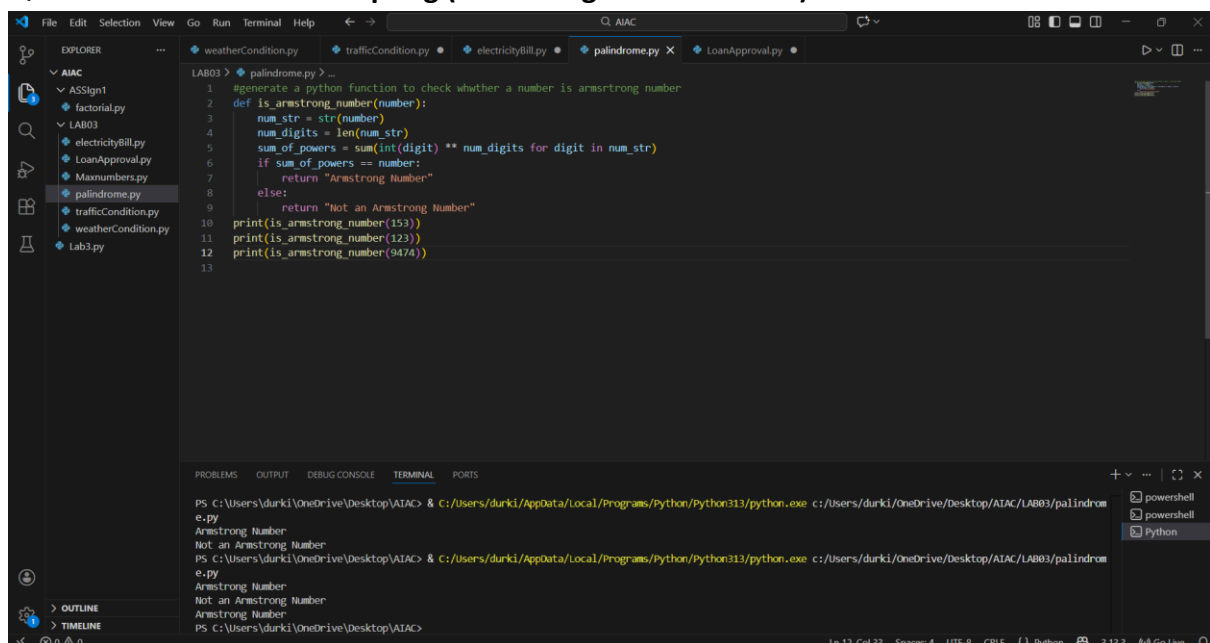
```
LAB03 > palindrome.py > ...
1 generate a python function to compute the factorial of a given number
2 def factorial(n):
3     if n < 0:
4         return "Factorial is not defined for negative numbers"
5     elif n == 0 or n == 1:
6         return 1
7     else:
8         result = 1
9         for i in range(2, n + 1):
10            result *= i
11        return result
12 # Example usage:
13 number = 5
14 print(f"The factorial of {number} is: {factorial(number)}")
15
```

```
PS C:\Users\durki\OneDrive\Desktop\AIAC> & c:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrome.py
The factorial of 5 is: 120
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

Comparison:

One-shot prompting gives better results than zero-shot. The code is more clear and readable. Providing an example helps the AI understand the problem better.

Question 3: Few-Shot Prompting (Armstrong Number Check)



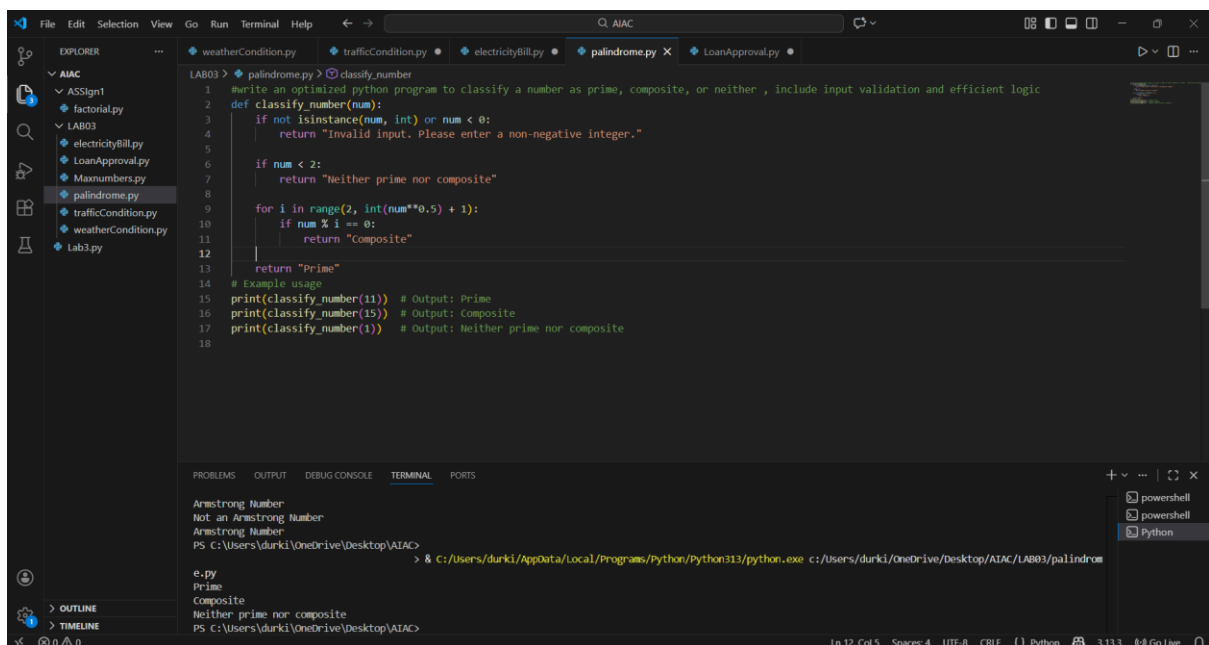
The screenshot shows a Visual Studio Code editor with a file explorer on the left. The file explorer shows a project named 'AIAC' with a subfolder 'LAB03'. Inside 'LAB03', there are several Python files: 'factorial.py', 'electricityBill.py', 'LoanApproval.py', 'MaxNumbers.py', 'palindrome.py', 'trafficCondition.py', and 'weatherCondition.py'. The 'palindrome.py' file is selected and its content is displayed in the editor. The code is a Python function to check whether a number is an Armstrong number. It includes a docstring, a function definition, a conversion to string, a calculation of the sum of powers, and a return statement. Three example usages are provided at the bottom. The terminal at the bottom shows the command to run the script and the output: 'Armstrong Number', 'Not an Armstrong Number', and 'Armstrong Number'.

```
LAB03 > palindrome.py > ...
1 generate a python function to check whether a number is armstrong number
2 def is_armstrong_number(number):
3     num_str = str(number)
4     num_digits = len(num_str)
5     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
6     if sum_of_powers == number:
7         return "Armstrong Number"
8     else:
9         return "Not an Armstrong Number"
10 print(is_armstrong_number(153))
11 print(is_armstrong_number(123))
12 print(is_armstrong_number(9474))
13
```

```
PS C:\Users\durki\OneDrive\Desktop\AIAC> & c:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrome.py
Armstrong Number
Not an Armstrong Number
Armstrong Number
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

Few-shot prompting uses multiple examples. This helps the AI generate accurate and well-structured code. The logic works correctly for valid inputs. Few-shot prompting gives better results.

Question 4: Context-Managed Prompting (Optimized Number Classification)

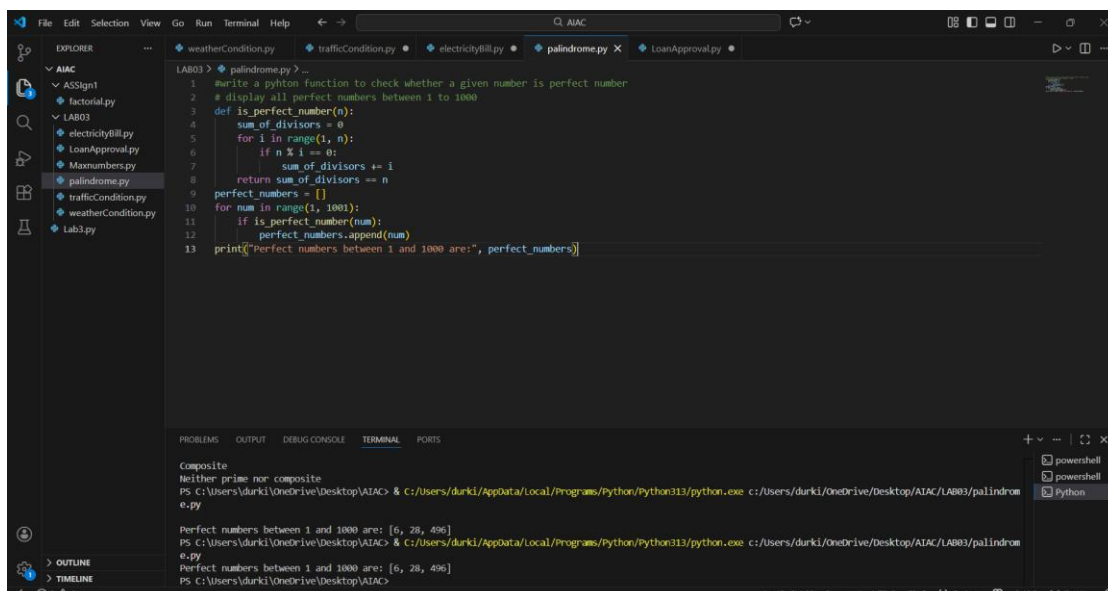


```
LAB03 > palindrome.py > classify_number
1 #write an optimized python program to classify a number as prime, composite, or neither , include input validation and efficient logic
2 def classify_number(num):
3     if not isinstance(num, int) or num < 0:
4         return "Invalid input. Please enter a non-negative integer."
5
6     if num < 2:
7         return "Neither prime nor composite"
8
9     for i in range(2, int(num**0.5) + 1):
10        if num % i == 0:
11            return "Composite"
12
13    return "Prime"
14
15 # Example usage
16 print(classify_number(11)) # Output: Prime
17 print(classify_number(15)) # Output: Composite
18 print(classify_number(1)) # Output: Neither prime nor composite
```

```
Armstrong Number
Not an Armstrong Number
PS C:\Users\durki\OneDrive\Desktop\AIAC>
> & C:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrom
e.py
Prime
Composite
Neither prime nor composite
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

The program checks whether the input is valid and handles invalid values properly. Efficient logic is used by checking divisibility only up to the square root of the number. This approach gives faster and more accurate results. Compared to earlier prompting methods, this produces clearer and better output.

Question5:Zero-Shot Prompting (Perfect Number Check)



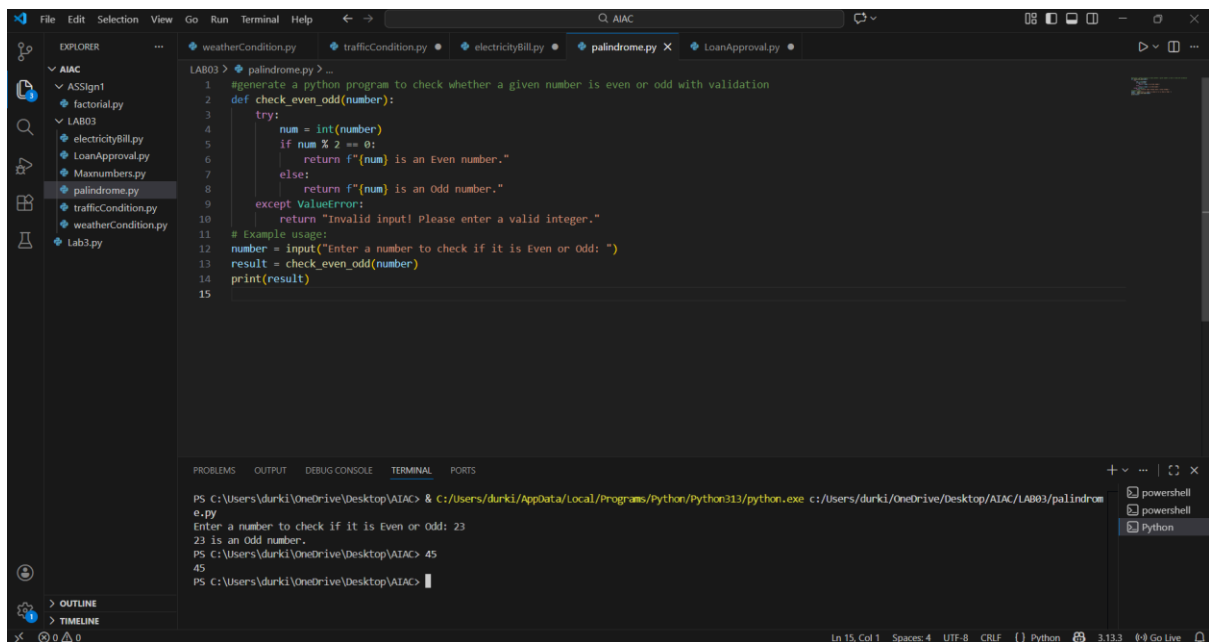
```
LAB03 > palindrome.py >
1 #write a python function to check whether a given number is perfect number
2 # display all perfect numbers between 1 to 1000
3 def is_perfect_number(n):
4     sum_of_divisors = 0
5     for i in range(1, n):
6         if n % i == 0:
7             sum_of_divisors += i
8     return sum_of_divisors == n
9 perfect_numbers = []
10 for num in range(1, 1001):
11     if is_perfect_number(num):
12         perfect_numbers.append(num)
13 print("Perfect numbers between 1 and 1000 are:", perfect_numbers)
```

```
Composite
Neither prime nor composite
PS C:\Users\durki\OneDrive\Desktop\AIAC> & C:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrom
e.py
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\durki\OneDrive\Desktop\AIAC> & C:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrom
e.py
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

The program checks whether a number is a perfect number by adding all its divisors.
If the sum of the divisors equals the number, it is marked as perfect.

The program tests numbers from 1 to 1000 and prints all perfect numbers.
The logic works correctly but is not efficient because it checks all values up to the number.

Question6:Few-Shot Prompting (Even or Odd Classification with Validation)



The screenshot shows a Visual Studio Code editor with a file explorer on the left containing a project named 'AIAC' with subfolders 'ASSIGN1' and 'LAB03'. The 'LAB03' folder is expanded, showing several Python files, including 'palindrome.py' which is currently open in the editor. The code in 'palindrome.py' is a Python function 'check_even_odd' that takes a number and returns a string indicating if it's even or odd, with error handling for non-integer inputs. Below the editor, a terminal window shows the command to run the script and the resulting output for inputs 23 and 45.

```
LAB03 > python palindrome.py
1 # generate a python program to check whether a given number is even or odd with validation
2 def check_even_odd(number):
3     try:
4         num = int(number)
5         if num % 2 == 0:
6             return f"{num} is an Even number."
7         else:
8             return f"{num} is an Odd number."
9     except ValueError:
10        return "Invalid input! Please enter a valid integer."
11 # Example usage:
12 number = input("Enter a number to check if it is Even or Odd: ")
13 result = check_even_odd(number)
14 print(result)
15
```

```
PS C:\Users\durki\OneDrive\Desktop\AIAC> & C:\Users\durki\AppData\Local\Programs\Python\Python313\python.exe c:\Users\durki\OneDrive\Desktop\AIAC\LAB03\palindrom
e.py
Enter a number to check if it is Even or Odd: 23
23 is an Odd number.
PS C:\Users\durki\OneDrive\Desktop\AIAC> 45
45
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

The given examples help the AI clearly understand the expected output format.
The program correctly handles even, odd, and zero values.
Negative numbers are also classified correctly.
Non-integer inputs are identified as invalid, improving output clarity