

AI Assisted Coding

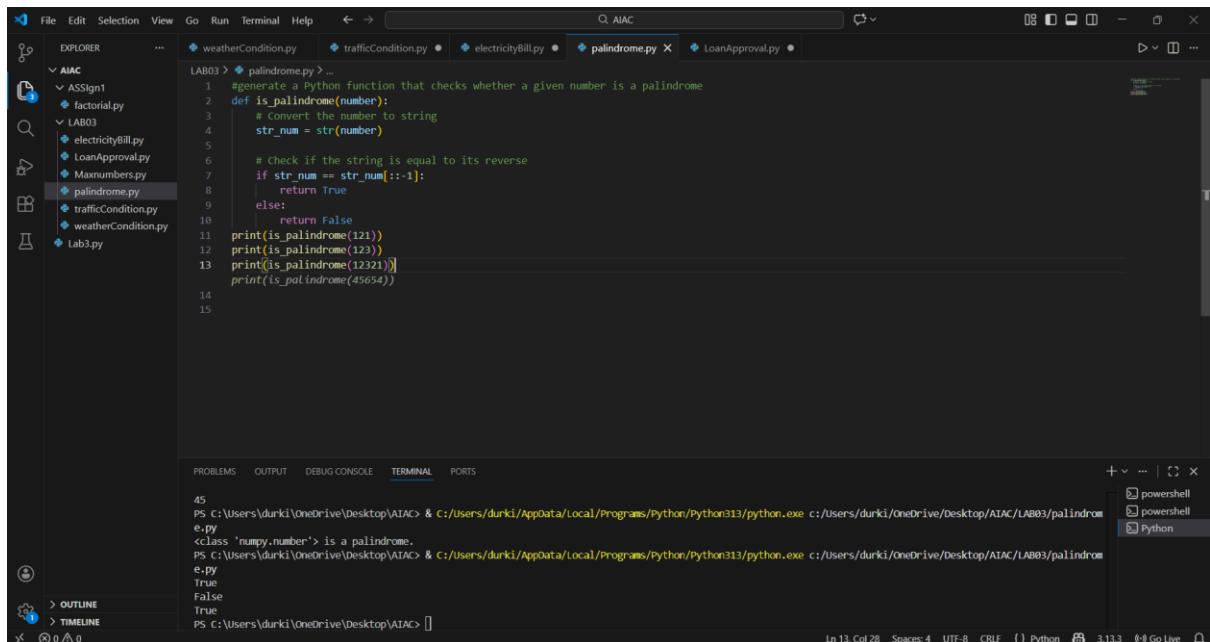
Assignment 3.1

Name:D.Shruthi

HT NO: 2303A52407

Batch:31

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



```
File Edit Selection View Go Run Terminal Help <- > AIAC
EXPLORER ... weatherCondition.py trafficCondition.py electricityBill.py palindrome.py LoanApproval.py
AIAC
ASSign1 factorial.py
LAB03
electricityBill.py
LoanApproval.py
Maxnumbers.py
palindrome.py
trafficCondition.py
weatherCondition.py
Lab3.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 print(is_palindrome(121))
12 print(is_palindrome(123))
13 print(is_palindrome(12321))
14 print(is_palindrome(45654))
15

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ v ... | C X
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/palindrome.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/palindrome.py
True
False
True
PS C:\Users\durki\OneDrive\Desktop\AIAC> []
Ln 13, Col 28 Spaces:4 UFT-8 CRLF {} Python 3.13.3 Go Live
```

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 the Visual Studio Code interface with the "AIAC" extension active. In the Explorer sidebar, there are several Python files: ASign1/factorial.py, LAB03/electricityBill.py, LAB03/LoanApproval.py, LAB03/Maxnumbers.py, LAB03/palindrome.py, and Lab3.py. The current file is palindrome.py. The code completion window is open over the code, showing a function definition:

```
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
13 # Example usage:
14 number = 5
15 print(f"The factorial of {number} is: {factorial(number)}")
```

The terminal below shows the output of running the script:

```
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 the Visual Studio Code interface with the "AIAC" extension active. In the Explorer sidebar, there are several Python files: ASign1/factorial.py, LAB03/electricityBill.py, LAB03/LoanApproval.py, LAB03/Maxnumbers.py, LAB03/palindrome.py, and Lab3.py. The current file is palindrome.py. The code completion window is open over the code, showing a function definition:

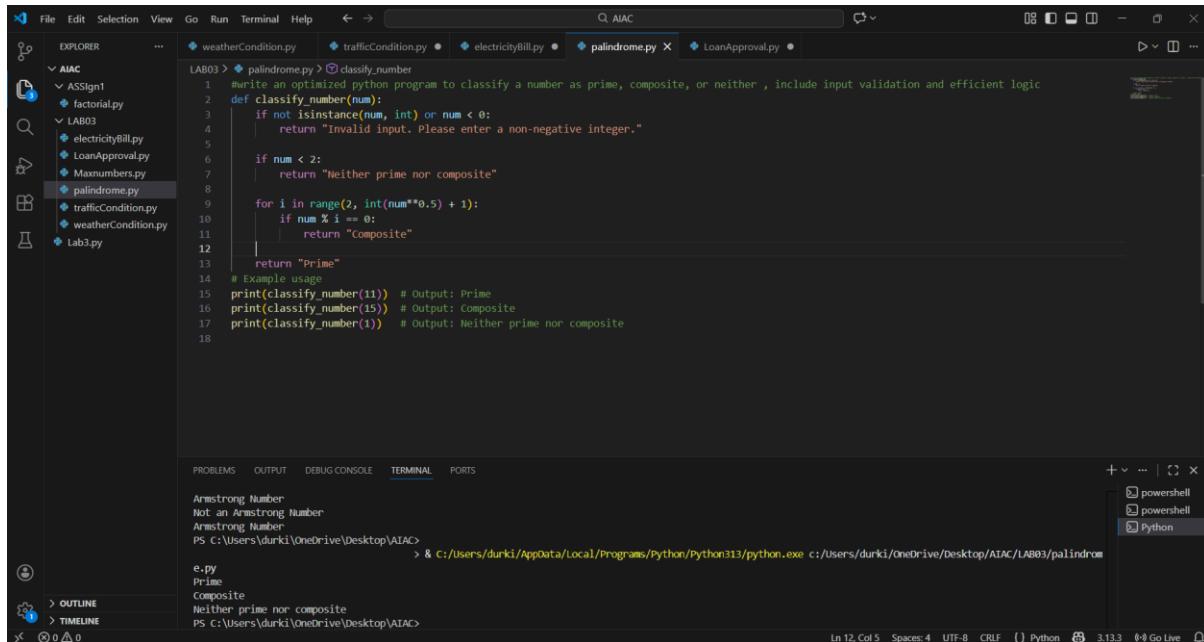
```
1 #generate a python function to check whether a number is armstrong number
2 def is_armstrong_number(number):
3     num_str = str(number)
4     num_digits = len(num_str)
5     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
6     if sum_of_powers == number:
7         return "Armstrong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

The terminal below shows the output of running the script:

```
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
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)



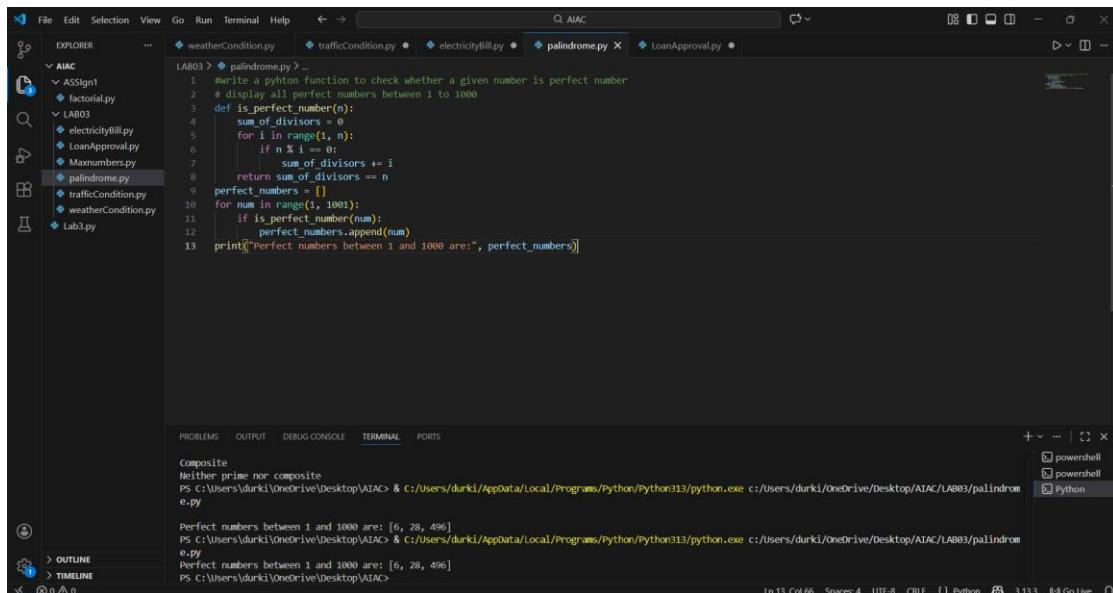
```
LAB03 > palindromes.py > classify_number
1 #write an optimized python program to classify a number as prime, composite, or neither , include input validation and efficient logic
2 def classify_number(num):
3     if not isinstance(num, int) or num < 0:
4         return "Invalid input. Please enter 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 print(classify_number(11)) # Output: Prime
16 print(classify_number(19)) # Output: Composite
17 print(classify_number(1)) # Output: Neither prime nor composite
```

The terminal output shows the program running and outputting the classification for various numbers:

```
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
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.

Question 5: Zero-Shot Prompting (Perfect Number Check)



```
LAB03 > palindromes.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)
```

The terminal output shows the program running and printing the perfect numbers between 1 and 1000:

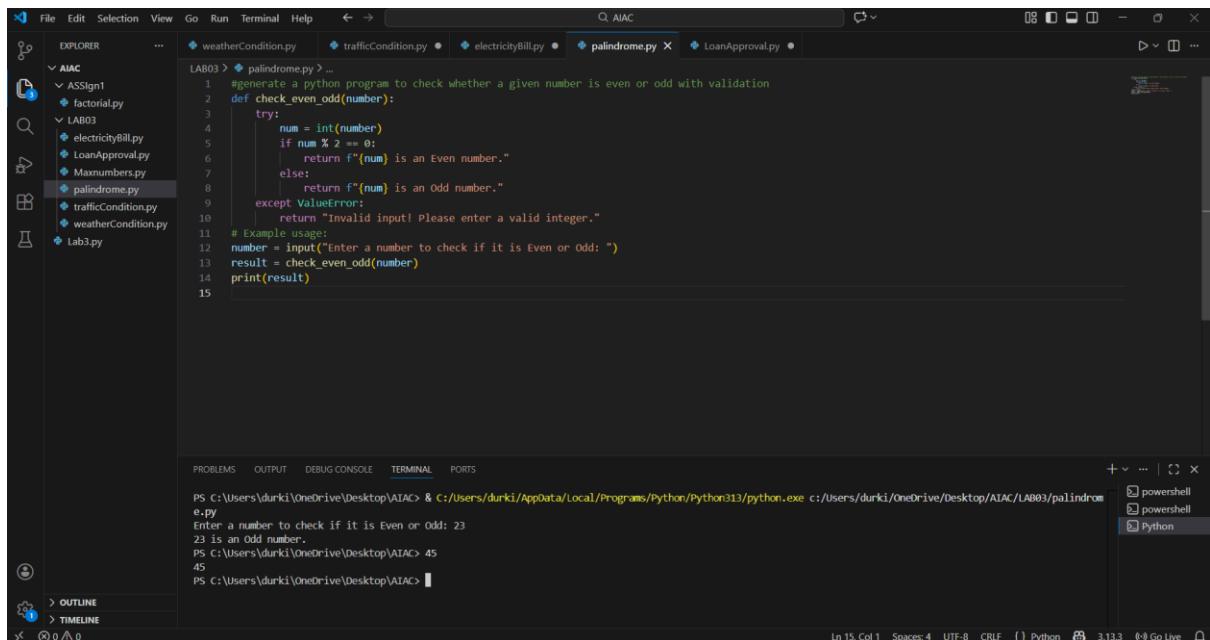
```
PS C:\Users\durki\OneDrive\Desktop\AIAC> & c:/Users/durki/AppData/Local/Programs/Python/Python313/python.exe c:/Users/durki/OneDrive/Desktop/AIAC/LAB03/palindrome.py
e.py
Perfect numbers between 1 and 1000 are: [6, 28, 496]
PS C:\Users\durki\OneDrive\Desktop\AIAC>
```

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)



A screenshot of the Visual Studio Code (VS Code) interface. The top menu bar includes File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar. The left sidebar shows a file tree under the 'EXPLORER' tab, with folders AIAC, ASSign1, LAB03, and Lab3, and files factorial.py, electricityBill.py, LoanApproval.py, Maxnumbers.py, palindrome.py, trafficCondition.py, weatherCondition.py, and Lab3.py. The main code editor window displays a Python script named 'palindrome.py'. The script contains the following code:

```
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)
```

The bottom right corner of the code editor shows the status bar with 'Ln 15, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.3 Go Live'. Below the code editor is a terminal window showing command-line interactions:

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

The bottom right corner of the terminal window also shows the status bar with 'Ln 15, Col 1 Spaces: 4 UTF-8 CRLF { } Python 3.13.3 Go Live'.

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