

## Lab Assignment – 3.1

Hall Ticket No.: 2303A510D9

Name – O.Karthik Reddy

Batch – 29

Question 1: Zero-Shot Prompting (Palindrome Number Program) Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a palindrome.

### Task:

- Record the AI-generated code.
  - Test the code with multiple inputs.
  - Identify any logical errors or missing edge-case handling.

## CODE & OUTPUT

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure under "RUN AND DEBUG" containing "aac3.py".
- Run View:** Displays the command "Run and Debug" and instructions to "To customize Run and Debug, open a folder and create a launch.json file". It also includes a link to "Show automatic Python configurations".
- Code Editor:** The active file is "aac3.py", which contains two functions: "is\_palindrome\_number" and "factorial".
- Terminal:** Shows the output of running the script. The script calculates the factorial of 11 (39916800) and checks if various numbers are Armstrong, perfect, or even.
- Bottom Status Bar:** Shows "Spaces: 4" and "UTF-8".

## Question 2: One-Shot Prompting (Factorial Calculation)

Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Example:

Input: 5 → Output: 120 Task:

- Compare the generated code with a zero-shot solution.
- Examine improvements in clarity and correctness.

## CODE & OUTPUT

The screenshot shows the VS Code interface with the following details:

- File Explorer:** Shows files: aac1.py, aac3.1.py, and tr.py.
- Run View:** Set to "RUN". It includes a "Run and Debug" section with instructions to customize Run and Debug settings or use a terminal. It also has a "Show automatic Python configurations" button.
- Code Editor:** Displays two Python scripts:
  - aac3.1.py:** A one-shot prompt for calculating factorial.

```
C:\> Users > boora > Downloads > aac3.1.py > ...
11     print(f'{number} is not a palindrome number.')
12
13
14
15 # One-Shot Prompting Factorial Calculation
16 def factorial(n):
17     """Calculate the factorial of a number."""
18     if n == 0 or n == 1:
19         return 1
20     else:
21         return n * factorial(n - 1)
22 if __name__ == "__main__":
23     num = int(input("Enter a number to calculate its factorial: "))
24     result = factorial(num)
25     print(f"The factorial of {num} is {result}.")
```
  - aac1.py:** A few-shot prompt for Armstrong number check.

```
PS C:\Users\boora\Downloads> & 'c:\Users\boora\AppData\Local\Programs\Python\Python311\python.exe' 'c:\Users\boora\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\debugpy\launcher' '52521' '--' 'C:\Users\boora\Downloads\aac3.1.py'
Enter a number: 16
16 is an Armstrong number.
Enter a number to calculate its factorial: 11
The factorial of 11 is 39916800.
Enter a number: 28
28 is not an Armstrong number.
Enter a number: 77
The number 77 is positive.
Enter a number: 16
16 is not a perfect number.
Enter an integer: 28
The number 28 is even.
PS C:\Users\boora\Downloads>
```
- Terminal:** Shows the execution of the Python script and its output.
- Bottom Status Bar:** Shows "Spaces 4" and "Python 3.11.5".

Question 3: Few-Shot Prompting (Armstrong Number Check) Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python function to check whether a given number is an Armstrong number.

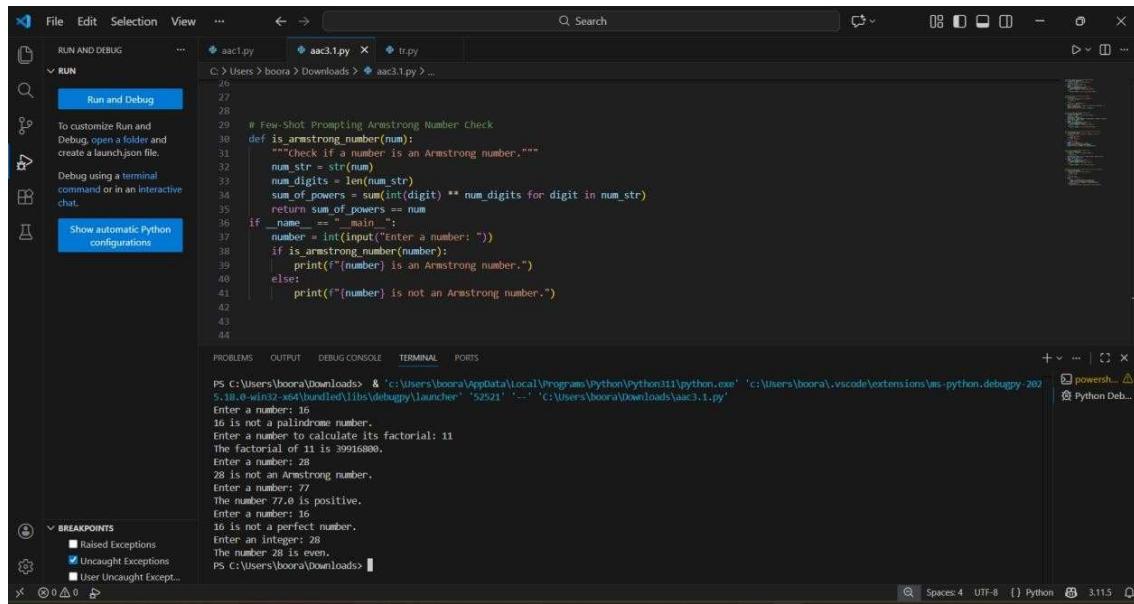
Examples:

- Input: 153 → Output: Armstrong Number

- Input: 370 → Output: Armstrong Number
  - Input: 123 → Output: Not an Armstrong Number
- Task:

- Analyze how multiple examples influence code structure and accuracy.
- Test the function with boundary values and invalid inputs.

## CODE & OUTPUT



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files `aac1.py`, `aac3.1.py` (the current file), and `tr.py`.
- Run View:** Displays the configuration for running the script. It includes options to customize run and debug settings, open a folder, and create a launch.json file. A button for "Show automatic Python configurations" is also present.
- Code Editor:** Contains the Python code for checking if a number is Armstrong. The code uses string manipulation and loops to calculate the sum of powers of digits.
- Terminal:** Shows the command-line output of the script. It prompts for a number, calculates its factorial, checks if it's a palindrome, and prints whether it's Armstrong or not.
- Breakpoints:** A sidebar showing breakpoints for the current file, with options to raise exceptions, uncaught exceptions, and user uncaught exceptions.

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

Design a context-managed prompt with clear instructions and constraints to generate an optimized Python program that classifies a number as prime, composite, or neither.

Task:

- Ensure proper input validation.
- Optimize the logic for efficiency.
- Compare the output with earlier prompting strategies.

## CODE & OUTPUT

The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows files `aac1.py`, `aac3.1.py`, and `tr.py`.
- Run and Debug View:** A tooltip indicates "Run and Debug" with instructions to "Customize Run and Debug, open a folder and create a launch.json file." It also has a link to "Show automatic Python configurations".
- Code Editor:** Displays Python code for classifying numbers and checking for perfect numbers.
- Terminal:** Shows the command line output for running `aac3.1.py`. The user enters various numbers (16, 11, 28) and receives responses about their properties (not palindromes, factorial calculations, Armstrong numbers, etc.).
- Breakpoints:** A sidebar shows breakpoints for "Raised Exceptions", "Uncalled Exceptions" (which is checked), and "User Uncalled Except...".

**Question 5: Zero-Shot Prompting (Perfect Number Check)** Write a zero-shot prompt (without providing any examples) to generate a Python function that checks whether a given number is a perfect number.

Task:

- Record the AI-generated code.
- Test the program with multiple inputs.
- Identify any missing conditions or inefficiencies in the logic.

## CODE & OUTPUT

The screenshot shows the Visual Studio Code interface with the 'aac3.1.py' file open in the editor. The terminal at the bottom displays the execution of the script, which performs various number checks like perfect numbers, palindromes, factorials, Armstrong numbers, and even/odd validation. The terminal output is as follows:

```
PS C:\Users\boora\Downloads> 8 "c:\Users\boora\AppData\Local\Programs\Python\Python311\python.exe" "c:\Users\boora\vscode\extensions\ms-python.vscode-pydebug-2025.11.0-win32-x64\bin\led11ts\debug\Launcher" "52521" "<>" "C:\Users\boora\Downloads\aac3.1.py"
Enter a number: 16
16 is not a palindrome number.
Enter a number to calculate its factorial: 11
The factorial of 11 is 39916800.
Enter a number: 28
28 is not an Armstrong number.
Enter a number: 77
The number 77.0 is positive.
Enter a number: 16
16 is not a perfect number.
Enter an integer: 28
The number 28 is even.
PS C:\Users\boora\Downloads>
```

### Question 6: Few-Shot Prompting (Even or Odd Classification with Validation)

Write a few-shot prompt by providing multiple input-output examples to guide the AI in generating a Python program that determines whether a given number is even or odd, including proper input validation.

Examples:

- Input: 8 → Output: Even
- Input: 15 → Output: Odd
- Input: 0 → Output: Even Task:
- Analyze how examples improve input handling and output clarity.
- Test the program with negative numbers and non-integer inputs.

## CODE & OUTPUT

The screenshot shows a Microsoft Visual Studio Code interface with the following details:

- File Explorer:** Shows a folder structure under "RUN AND DEBUG" and "RUN".
- Search Bar:** Contains the text "Search".
- Run and Debug Bar:** Contains icons for Run, Stop, and Refresh.
- Code Editor:** Displays a Python script named "aacl.py". The code defines a function "classify\_even\_odd" that takes a number as input and returns "even" if it's divisible by 2, or "odd" otherwise. It also includes a try-except block to handle non-integer inputs.
- Terminal:** Shows the output of running the script "aacl.py". The user enters "16" and "28" as input. The script outputs the classification ("even" for 16, "not a palindrome" for 16, "factorial of 11 is 39916800" for 28), and checks if the numbers are Armstrong and perfect.
- Problems:** Shows no errors or warnings.
- Output:** Shows the standard output of the script.
- Breakpoints:** Shows breakpoints set at various lines of code, with "Raised Exceptions" checked.
- Status Bar:** Shows "Spaces: 4", "UTF-8", "Python", and "3.11.5".