

ASSIGNMENT -3.1

BATCH-30

ROLL-NO:2303A51932

NAME-SRAVANI TASK-1:

ZERO-SHOT PROMPTING (PALINDROME NUMBER PROGRAM) PROMPT:

Write a Python function that checks whether a given number is a palindrome. The function should return True if it is a palindrome and False otherwise.

CODE:

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

- File Explorer:** Shows a project structure with files like `index.js`, `AI ASS Coding.ipynb`, and several Python files (`ai 3.1.py`, `ai 2.5.py`, etc.).
- Code Editor:** Displays the following Python code:

```
#Now write a Python function that checks whether a given number is a Armstrong number.

def is_armstrong(number):
    total=0
    temp=number
    digits=len(str(number))
    while temp>0:
        digit=temp%10
        total+=digit**digits
        temp//=10
    if total==number:
        return 'Armstrong number'
    else:
        return 'Not an Armstrong number'

# Example usage
print(is_armstrong(153)) # Should return 'Armstrong number'
print(is_armstrong(123)) # Should return 'Not an Armstrong number'
print(is_armstrong(9474)) # Should return 'Armstrong number'
```
- Terminal:** Shows the output of the code execution:

```
Not an Armstrong number
Armstrong number
Prime
Composite
Neither prime nor composite
Neither prime nor composite
```
- Status Bar:** Shows the current file is `ai 3.1.py`, line 71, column 1, with Python selected as the language.

OBSERVATION: -

The model is given only the explanation of the question -Any example or detailed explanation is not given -Answer is accurate but not specific with negative and nonintegers values

TASK-2:

ONE-SHOT PROMPTING (FACTORIAL CALCULATION) PROMPT:

write a python function that compute the factorial of given number. The function should return the result.

Example:

Input:5

Output:120 CODE:

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

- File Explorer:** Shows a tree view of files and folders. In the "DEVOPS EXP 2,3,4" folder, "ai 3.1.py" is selected.
- Code Editor:** Displays Python code for Armstrong number detection. The code includes imports, function definitions, and print statements to test the function.
- Terminal:** Shows the command "python -u "c:\Users\heman\OneDrive\Desktop\Devops\exp 2,3,4\ai 1932\ai 3.1.py"" being run, with the output "True" and "False".
- Status Bar:** Shows file statistics like "Ln 15, Col 1" and encoding information like "UTF-8".

OBSERVATION:

Clear understanding of the output better choice of logic-stack overflow, recursion complexity Correct handling of base case Improve code simplicity TASK-3:

FEW-SHOT PROMPTING (ARMSTRONG NUMBER CHECK)

Prompt:

Example 1:

Input: 153

Output: Armstrong Number Example

2:

Input: 370

Output: Armstrong Number

Example 3:

Input: 123

Output: Not an Armstrong Number Now write a Python function that checks whether a given number is an Armstrong number. The function should return an appropriate result. CODE:

OBSERVATION:

Clear output formatting. structured way Correct logic selection Easy understanding of code Exact Appropriate answer Optimized and customized solution

TASK-4:

CONTEXT-MANAGED PROMPTING (OPTIMIZED NUMBER CLASSIFICATION) PROMPT:

You are writing a Python program for number classification.

Requirements: -

Accept only integer input - Handle invalid and negative inputs properly - Classify the number as Prime, Composite, or Neither - Optimize the logic for efficiency (avoid unnecessary checks) - Return clear and user-friendly messages - Write clean and readable Python code Generate the program accordingly.

CODE:

The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows files like `index.js`, `AI ASS Coding.ipynb`, and several `Untitled` files.
- Code Editor:** Displays a Python script named `ai 3.1.py`. The code defines a function `is_armstrong` that takes a number as input and returns a string indicating if it's an Armstrong number or not. It includes comments and example usage with print statements.
- Terminal:** Shows the output of running the script with different inputs: 153, 123, and 9474. The output indicates that 153 and 123 are Armstrong numbers, while 9474 is not.
- Bottom Status Bar:** Shows the current file is `ai 3.1.py`, the line is 71, column 1, and the character encoding is UTF-8.

```
#Now write a Python function that checks whether a given number is an Armstrong number.

Zencoder
def is_armstrong(number):
    total=0
    temp=number
    digits=len(str(number))
    while temp>0:
        digit=temp%10
        total+=digit**digits
        temp//=10
    if total==number:
        return 'Armstrong number'
    else:
        return 'Not an Armstrong number'

# Example usage
print(is_armstrong(153)) # Should return 'Armstrong number'
print(is_armstrong(123)) # Should return 'Not an Armstrong number'
print(is_armstrong(9474)) # Should return 'Armstrong number'
```

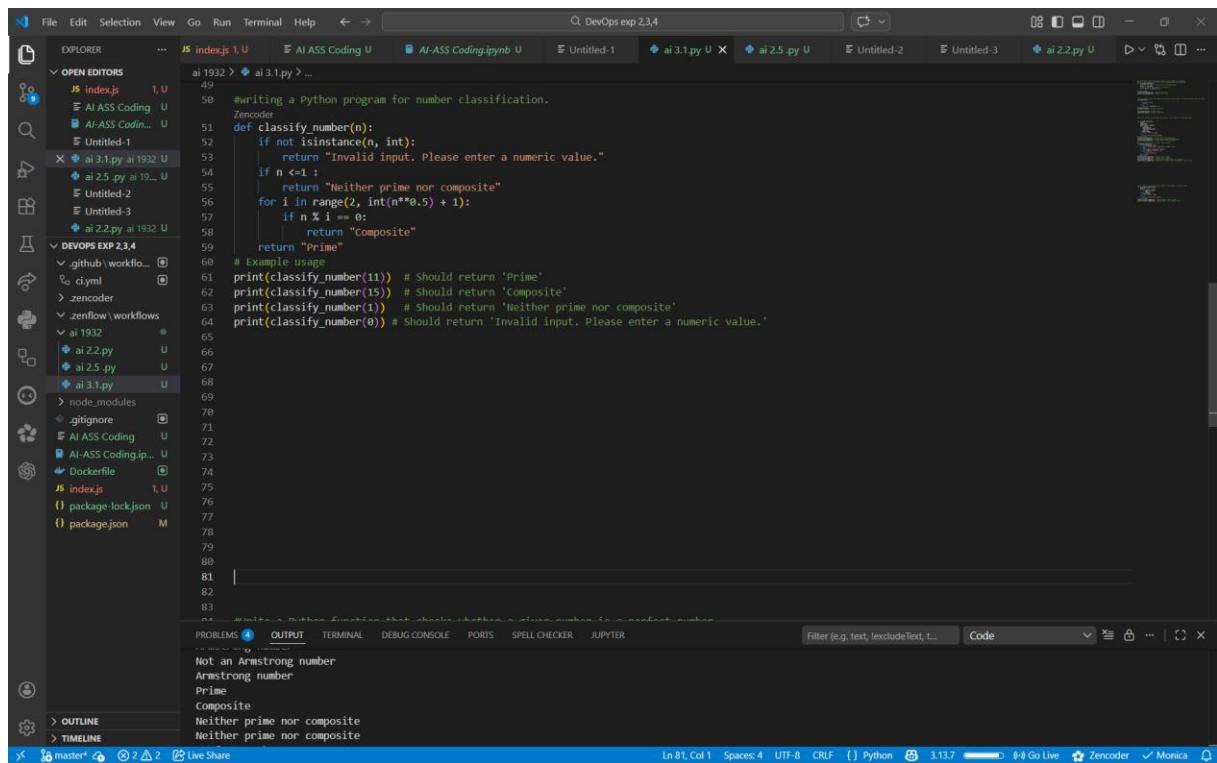
OBSERVATION:

The role is defined Constraints are clearly stated Efficiency and validation of the code but the inputs should be specified more clearly mentioned TASK-5:

ZERO-SHOT PROMPTING (PERFECT NUMBER CHECK) VALIDATION) PROMPT:

Write a Python function that checks whether a given number is a perfect number. The function should return an appropriate result.

CODE:



A screenshot of a code editor (VS Code) showing a Python file named `ai 3.1.py`. The file contains the following code:

```
49 # Writing a Python program for number classification.
50 # Zencoder
51 def classify_number(n):
52     if not isinstance(n, int):
53         return "Invalid input. Please enter a numeric value."
54     if n <= 1:
55         return "Neither prime nor composite"
56     for i in range(2, int(n**0.5) + 1):
57         if n % i == 0:
58             return "Composite"
59     return "Prime"
60 # Example usage
61 print(classify_number(11)) # Should return 'Prime'
62 print(classify_number(15)) # Should return 'Composite'
63 print(classify_number(1)) # Should return 'Neither prime nor composite'
64 print(classify_number(0)) # Should return 'Invalid input. Please enter a numeric value.'
```

The code defines a function `classify_number` that takes an integer `n` and returns a string indicating whether it is Prime, Composite, or Neither prime nor composite. It includes example usage at the bottom.

OBSERVATION:

No input validation – if negative or float any. Inefficient for large input Did not specify input constraints No edge case handing seen TASK-6:

FEW-SHOT PROMPTING (EVEN OR ODD CLASSIFICATION WITH VALIDATION) PROMPT:

Example 1:

Input: 8 Output:

Even Example 2:

Input: 15 Output:

Odd Example 3:

Input: 0

Output: Even Now write a Python program that determines whether a given number is Even or Odd. The program should include proper input validation and return clear messages.

CODE:

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

- File Explorer:** Shows a file tree with several files: `index.js`, `AI ASS Coding.ipynb`, `ai 3.1.py`, `ai 2.5.py`, `ai 2.2.py`, `ai 3.1.py` (selected), `ai 2.2.py`, `ai 2.5.py`, `ai 3.1.py`, `node_modules`, `.gitignore`, `AI ASS Coding.ipynb`, `Dockerfile`, `index.js`, `package-lock.json`, and `package.json`.
- Code Editor:** Displays the `ai 3.1.py` file content:

```
def is_perfect_number(number):
    total=0
    for i in range(1, number):
        if number % i == 0:
            total+=i
    if total==number:
        return 'Perfect number'
    else:
        return 'Not a perfect number'

# Example usage
print(is_perfect_number(6)) # Should return 'Perfect number'
print(is_perfect_number(10)) # Should return 'Not a perfect number'
```
- Terminal:** Shows the output of the code execution:

```
Not an Armstrong number
Armstrong number
Prime
Composite
Neither prime nor composite
Neither prime nor composite
Perfect number
Not a perfect number
```
- Status Bar:** Shows the status bar with information like "ln 79, Col 69", "Spaces: 4", "UTF-8", "CR/LF", "Python", "3.13.7", "Go Live", "Zencoder", and "Monica".

OBSERVATION:

Negative integer are handled correctly Program safely rejected non integer inputs
Improve input handling Clear and consistent output