Assignment-3.1

Name: B.Shravya

Hall ticket No:2303A51492

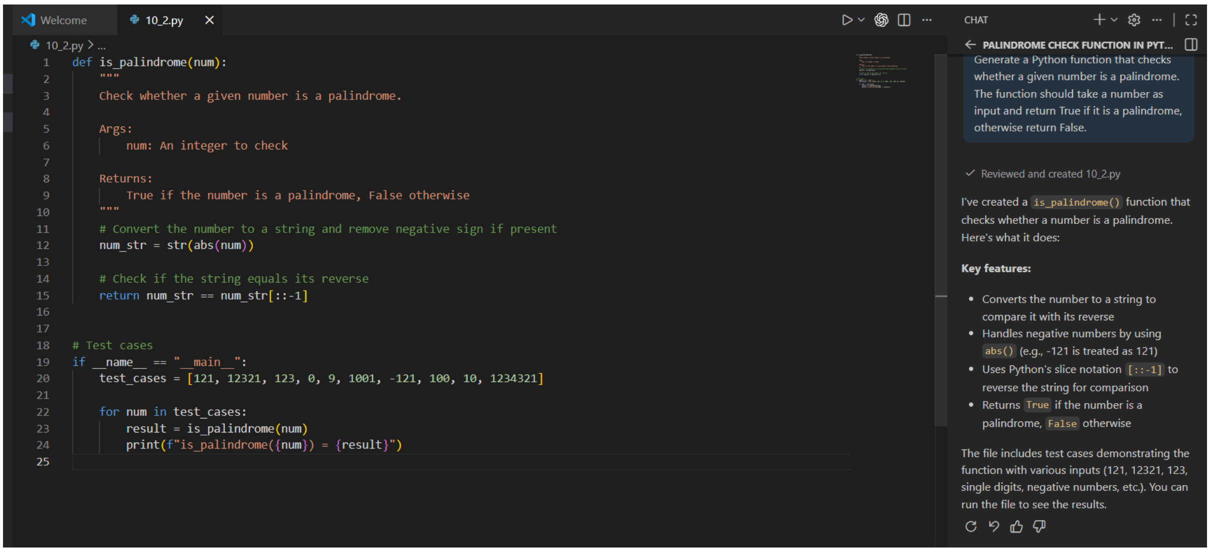
Batch:08

Lab Experiment: Prompt Engineering – Improving Prompts and Context Management

Question 1 – Zero-Shot Prompting

Prompt Used

Generate a Python function that checks whether a given number is a palindrome. The function should take a number as input and return True if it is a palindrome, otherwise return False.



```python
def is_palindrome(num):
    """
    Check whether a given number is a palindrome.

    Args:
        num: An integer to check

    Returns:
        True if the number is a palindrome, False otherwise
    """
    # Convert the number to a string and remove negative sign if present
    num_str = str(abs(num))

    # Check if the string equals its reverse
    return num_str == num_str[::-1]


# Test cases
if __name__ == "__main__":
    test_cases = [121, 12321, 123, 0, 9, 1001, -121, 100, 10, 1234321]

    for num in test_cases:
        result = is_palindrome(num)
        print(f"is_palindrome({num}) = {result}")
```

Generate a Python function that checks whether a given number is a palindrome. The function should take a number as input and return True if it is a palindrome, otherwise return False.

✓ Reviewed and created 10_2.py

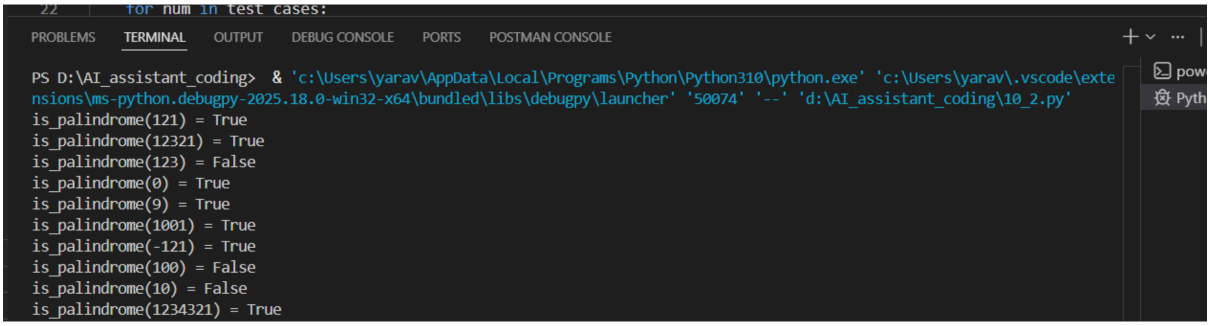
I've created a `is_palindrome()` function that checks whether a number is a palindrome. Here's what it does:

Key features:

- Converts the number to a string to compare it with its reverse
- Handles negative numbers by using `abs()` (e.g., -121 is treated as 121)
- Uses Python's slice notation `[::-1]` to reverse the string for comparison
- Returns `True` if the number is a palindrome, `False` otherwise

The file includes test cases demonstrating the function with various inputs (121, 12321, 123, single digits, negative numbers, etc.). You can run the file to see the results.

Output:



```
PS D:\AI_assistant_coding>  & 'c:\Users\yarav\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\yarav\.vscode\exte
nsions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50074' '--' 'd:\AI_assistant_coding\10_2.py'
is_palindrome(121) = True
is_palindrome(12321) = True
is_palindrome(123) = False
is_palindrome(0) = True
is_palindrome(9) = True
is_palindrome(1001) = True
is_palindrome(-121) = True
is_palindrome(100) = False
is_palindrome(10) = False
is_palindrome(1234321) = True
```

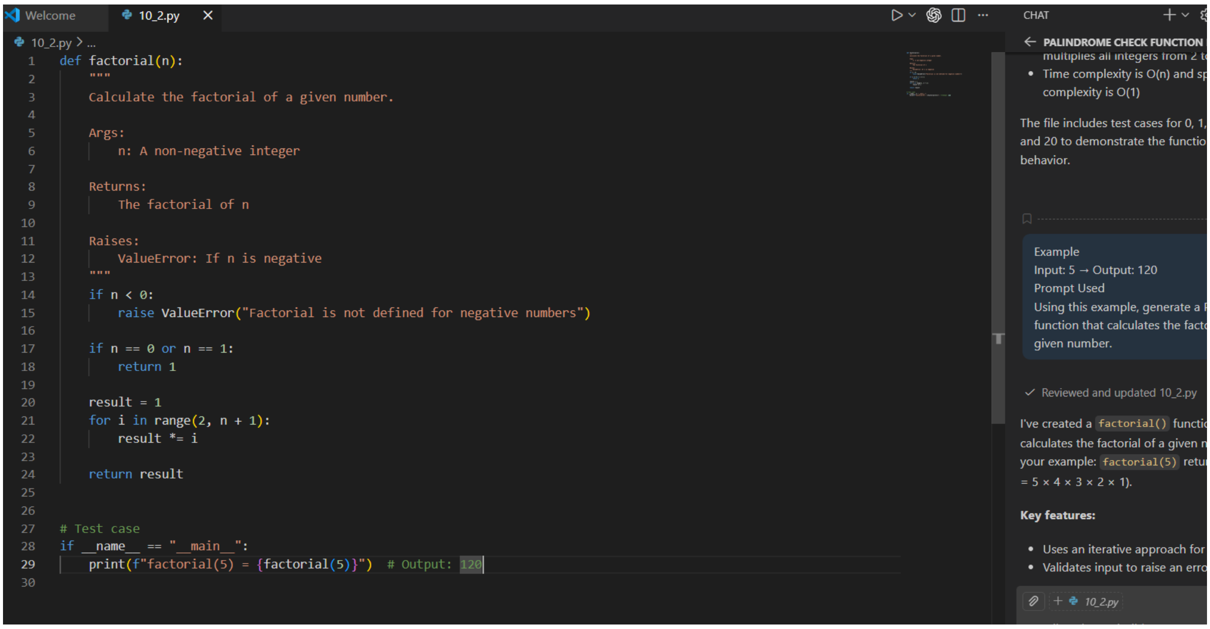
Question 2 – One-Shot Prompting

Example

Input: 5 → Output: 120

Prompt Used

Using this example, generate a Python function that calculates the factorial of a given number.



```python
def factorial(n):
    """
    Calculate the factorial of a given number.

    Args:
        n: A non-negative integer

    Returns:
        The factorial of n

    Raises:
        ValueError: If n is negative
    """
    if n < 0:
        raise ValueError("Factorial is not defined for negative numbers")

    if n == 0 or n == 1:
        return 1

    result = 1
    for i in range(2, n + 1):
        result *= i

    return result


# Test case
if __name__ == "__main__":
    print(f"factorial(5) = {factorial(5)}")  # Output: 120
```

multiplies all integers from 2 t
- Time complexity is O(n) and sp
complexity is O(1)

The file includes test cases for 0, 1, and 20 to demonstrate the functio behavior.

Example
Input: 5 → Output: 120
Prompt Used
Using this example, generate a P function that calculates the facto given number.

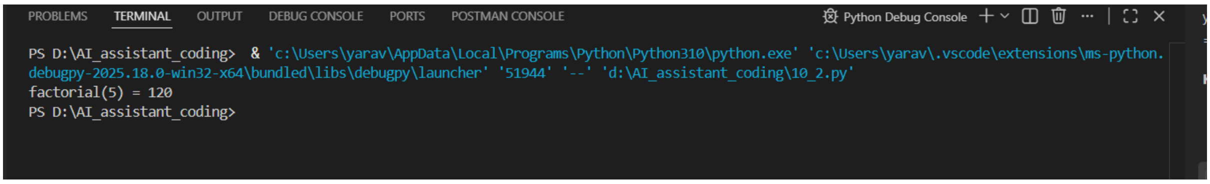
✓ Reviewed and updated 10_2.py

I've created a `factorial()` functio calculates the factorial of a given n your example: `factorial(5)` retu = 5 × 4 × 3 × 2 × 1).

Key features:

- Uses an iterative approach for
- Validates input to raise an erro

Output:



```
PS D:\AI_assistant_coding>  & 'c:\Users\yarav\AppData\Local\Programs\Python\Python310\python.exe' 'c:\Users\yarav\.vscode\extensions\ms-python.
debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '51944' '--' 'd:\AI_assistant_coding\10_2.py'
factorial(5) = 120
PS D:\AI_assistant_coding>
```

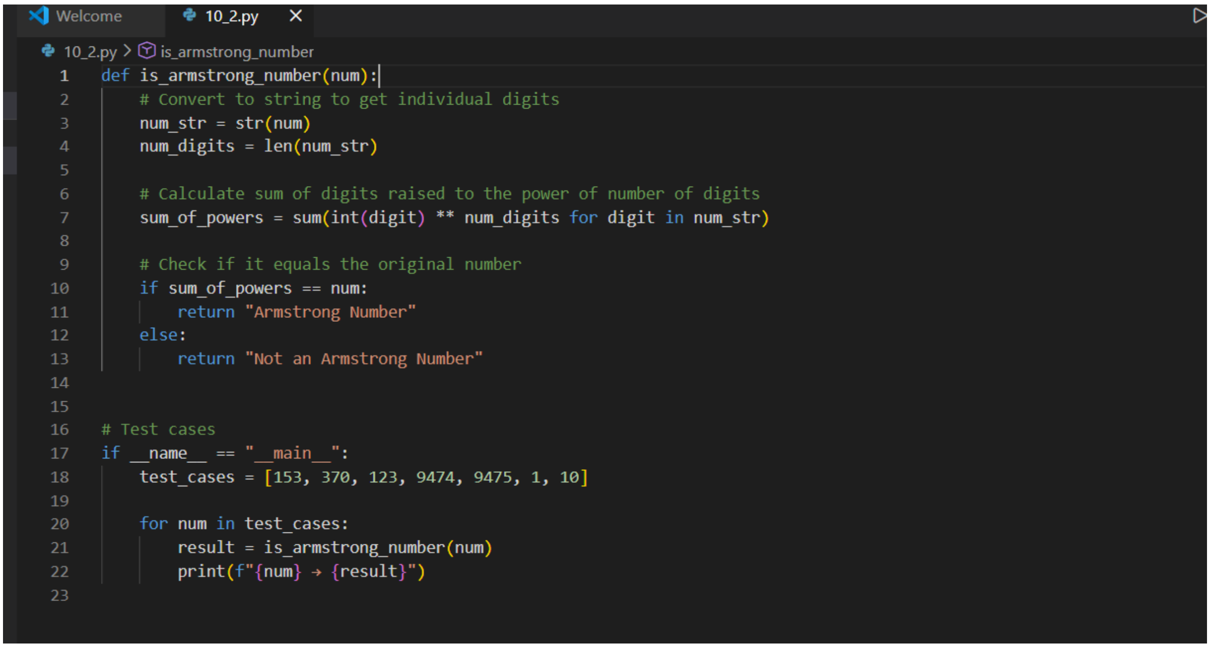
Question 3 – Few-Shot Prompting

Examples

153 → Armstrong Number

370 → Armstrong Number

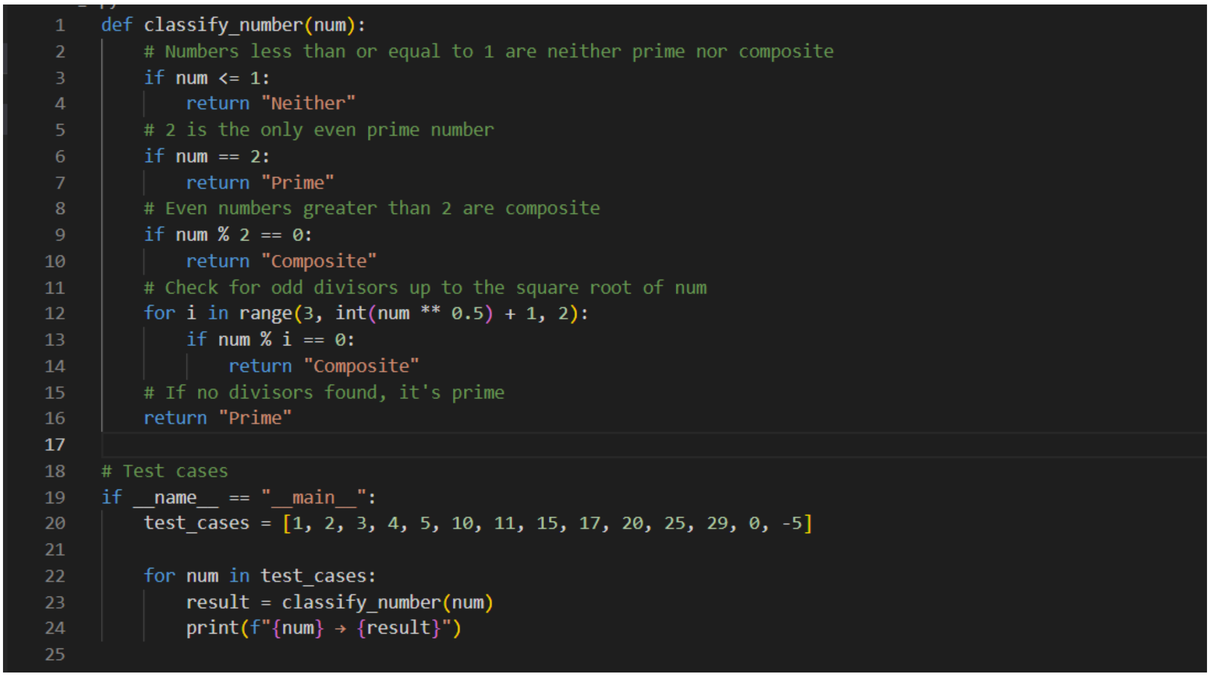
123 → Not an Armstrong Number



```python
def is_armstrong_number(num):
    # Convert to string to get individual digits
    num_str = str(num)
    num_digits = len(num_str)

    # Calculate sum of digits raised to the power of number of digits
    sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)

    # Check if it equals the original number
    if sum_of_powers == num:
        return "Armstrong Number"
    else:
        return "Not an Armstrong Number"


# Test cases
if __name__ == "__main__":
    test_cases = [153, 370, 123, 9474, 9475, 1, 10]

    for num in test_cases:
        result = is_armstrong_number(num)
        print(f"{num} → {result}")

```

Question 4 – Context-Managed Prompting

Classify number as Prime, Composite, or Neither



```python
def classify_number(num):
    # Numbers less than or equal to 1 are neither prime nor composite
    if num <= 1:
        return "Neither"
    # 2 is the only even prime number
    if num == 2:
        return "Prime"
    # Even numbers greater than 2 are composite
    if num % 2 == 0:
        return "Composite"
    # Check for odd divisors up to the square root of num
    for i in range(3, int(num ** 0.5) + 1, 2):
        if num % i == 0:
            return "Composite"
    # If no divisors found, it's prime
    return "Prime"

# Test cases
if __name__ == "__main__":
    test_cases = [1, 2, 3, 4, 5, 10, 11, 15, 17, 20, 25, 29, 0, -5]

    for num in test_cases:
        result = classify_number(num)
        print(f"{num} → {result}")

```

Conclusion

Few-shot and context-managed prompts produce more accurate and optimized code. Examples improve validation, formatting, and logical structure.