

Assignment-3.1

Name: Y. Poojitha

Hall ticket No:2303A51499

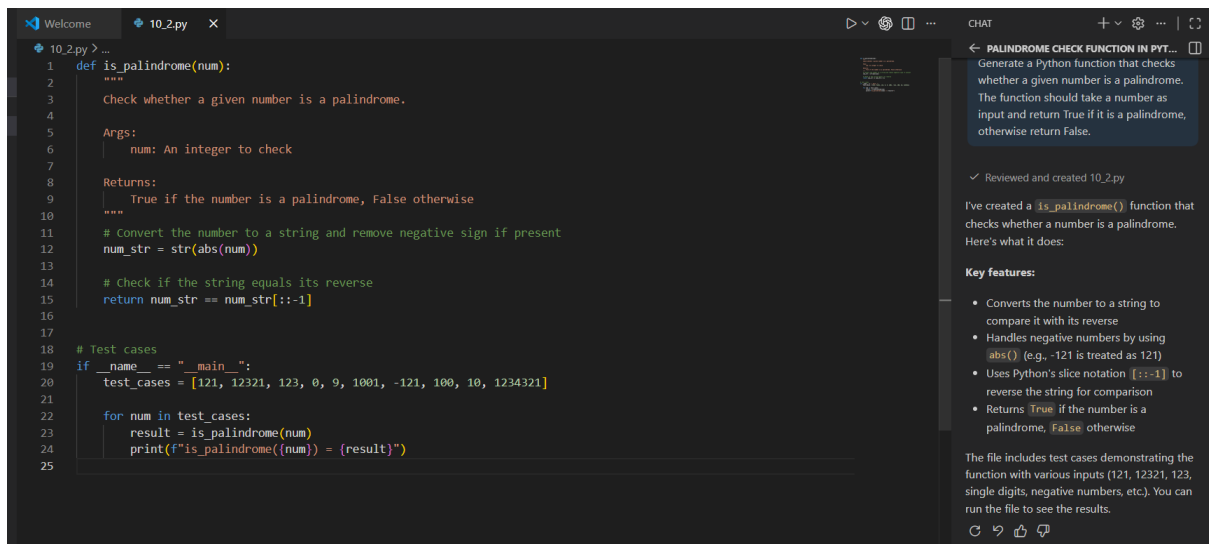
Batch:08

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.



The screenshot shows a VS Code editor with a file named `10_2.py` open. The code defines a function `is_palindrome(num)` that checks if a number is a palindrome. It includes docstrings, arguments, and test cases. The chat window on the right shows a prompt asking for a Python function to check for palindromes, and a response providing the function and its key features.

```
1 def is_palindrome(num):
2     """
3     Check whether a given number is a palindrome.
4
5     Args:
6         num: An integer to check
7
8     Returns:
9         True if the number is a palindrome, False otherwise
10    """
11    # Convert the number to a string and remove negative sign if present
12    num_str = str(abs(num))
13
14    # Check if the string equals its reverse
15    return num_str == num_str[::-1]
16
17
18 # Test cases
19 if __name__ == "__main__":
20     test_cases = [121, 12321, 123, 0, 9, 1001, -121, 100, 10, 1234321]
21
22     for num in test_cases:
23         result = is_palindrome(num)
24         print(f"is_palindrome({num}) = {result}")
25
```

PALINDROME CHECK FUNCTION IN PYTHON

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

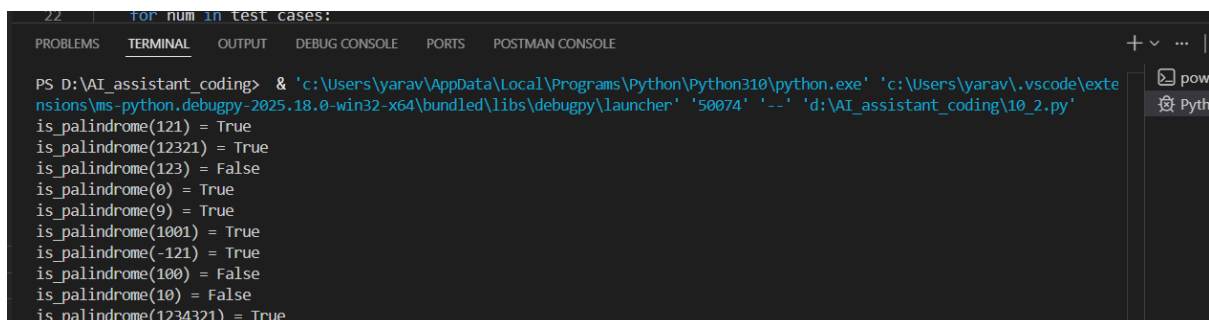
I've created a `is_palindrome()` function that checks whether a number is a palindrome. Here's what it does:

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:



The screenshot shows a terminal window with the output of the Python script. The output displays the results of the `is_palindrome` function for each test case.

```
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' '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
```

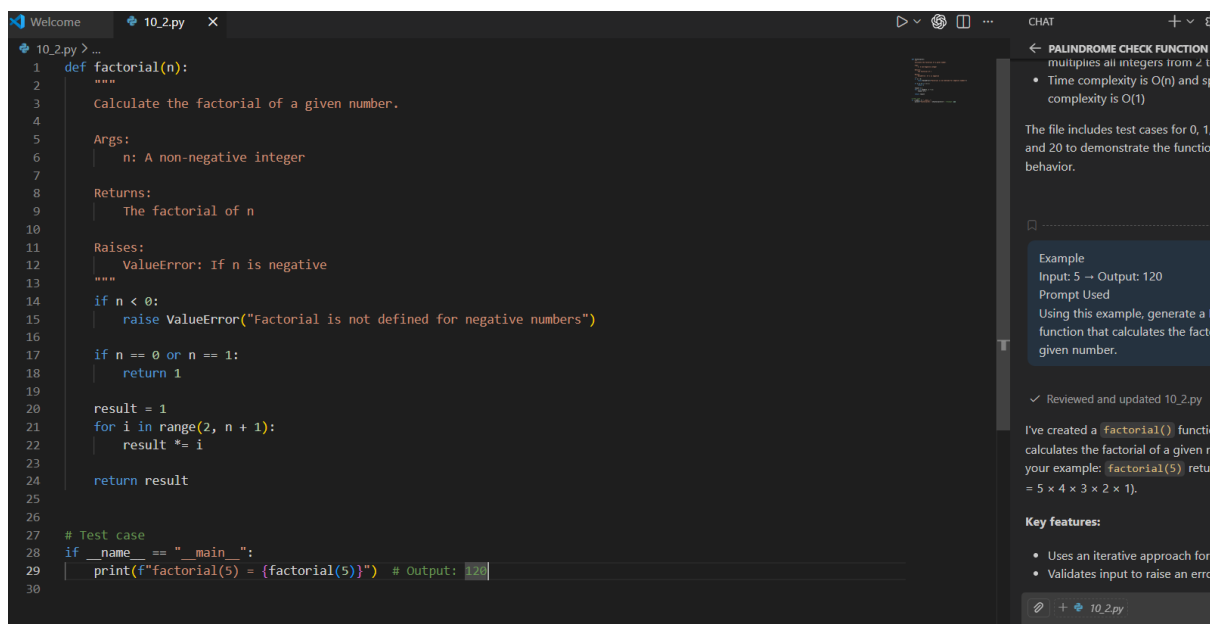
Question 2 – One-Shot Prompting

Example

Input: 5 → Output: 120

Prompt Used

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



The screenshot shows a VS Code editor with a file named `10_2.py`. The code defines a `factorial(n)` function that calculates the factorial of a given number `n`. The function includes docstrings for description, arguments, returns, and raises, as well as a test case for `factorial(5)` which outputs `120`. To the right, a chat window is open with a prompt: "Using this example, generate a Python function that calculates the factorial of a given number." The chat response shows the AI has reviewed and updated the file, and provides a summary of the function's key features: it uses an iterative approach and validates input to raise an error for negative numbers.

```
1 def factorial(n):
2     """
3     Calculate the factorial of a given number.
4
5     Args:
6         n: A non-negative integer
7
8     Returns:
9         The factorial of n
10
11     Raises:
12         ValueError: If n is negative
13     """
14     if n < 0:
15         raise ValueError("Factorial is not defined for negative numbers")
16
17     if n == 0 or n == 1:
18         return 1
19
20     result = 1
21     for i in range(2, n + 1):
22         result *= i
23
24     return result
25
26
27 # Test case
28 if __name__ == "__main__":
29     print(f"factorial(5) = {factorial(5)}") # Output: 120
30
```

CHAT

← PALINDROME CHECK FUNCTION
multiplies all integers from 2 to n.
• Time complexity is O(n) and space complexity is O(1)

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

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

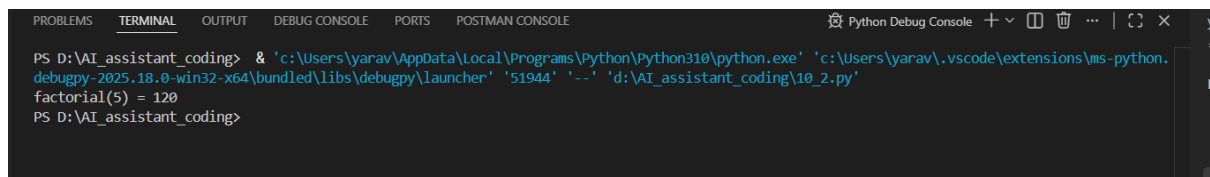
✓ Reviewed and updated 10_2.py

I've created a `factorial()` function that calculates the factorial of a given number. For your example: `factorial(5)` returns `5 × 4 × 3 × 2 × 1`.

Key features:

- Uses an iterative approach for calculation
- Validates input to raise an error for negative numbers

Output:



The screenshot shows a terminal window with the command `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\bundle\libs\debugpy\launcher' '51944' '--' 'd:\AI_assistant_coding\10_2.py'`. The output shows `factorial(5) = 120`.

```
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\bundle\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

```
Welcome 10_2.py X
10_2.py > is_armstrong_number
1 def is_armstrong_number(num):
2     # Convert to string to get individual digits
3     num_str = str(num)
4     num_digits = len(num_str)
5
6     # Calculate sum of digits raised to the power of number of digits
7     sum_of_powers = sum(int(digit) ** num_digits for digit in num_str)
8
9     # Check if it equals the original number
10    if sum_of_powers == num:
11        return "Armstrong Number"
12    else:
13        return "Not an Armstrong Number"
14
15
16 # Test cases
17 if __name__ == "__main__":
18     test_cases = [153, 370, 123, 9474, 9475, 1, 10]
19
20     for num in test_cases:
21         result = is_armstrong_number(num)
22         print(f"{num} → {result}")
23
```

Question 4 – Context-Managed Prompting

Classify number as Prime, Composite, or Neither

```
1 def classify_number(num):
2     # Numbers less than or equal to 1 are neither prime nor composite
3     if num <= 1:
4         return "Neither"
5     # 2 is the only even prime number
6     if num == 2:
7         return "Prime"
8     # Even numbers greater than 2 are composite
9     if num % 2 == 0:
10        return "Composite"
11    # Check for odd divisors up to the square root of num
12    for i in range(3, int(num ** 0.5) + 1, 2):
13        if num % i == 0:
14            return "Composite"
15    # If no divisors found, it's prime
16    return "Prime"
17
18 # Test cases
19 if __name__ == "__main__":
20     test_cases = [1, 2, 3, 4, 5, 10, 11, 15, 17, 20, 25, 29, 0, -5]
21
22     for num in test_cases:
23         result = classify_number(num)
24         print(f"{num} → {result}")
25
```

Conclusion

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