SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE	DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech	**Assignment Type: Lab**	**Academic Year:**2025-2026

Course Coordinator Name	Dr. Rishabh Mittal

Instructor(s) Name	
	Mr. S Naresh Kumar
	Ms. B. Swathi
	Dr. Sasanko Shekhar Gantayat
	Mr. Md Sallauddin
	Dr. Mathivanan
	Mr. Y Srikanth
	Ms. N Shilpa
	Dr. Rishabh Mittal (Coordinator)
	Dr. R. Prashant Kumar
	Mr. Ankushavali MD
	Mr. B Viswanath
	Ms. Sujitha Reddy
	Ms. A. Anitha
	Ms. M.Madhuri
	Ms. Katherashala Swetha
	Ms. Velpula sumalatha
	Mr. Bingi Raju

CourseCode	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/II	Regulation	R23
Date and Day of Assignment	**Week2 – Monday**	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches

Assignment Number:3.1(Present assignment number)/**24**(Total number of assignments)

Q.No.	Question	Expected Time to complete
1	**Lab Experiment: Prompt Engineering – Improving Prompts and**	Week2 - Monday

Context Management	
Lab Objectives	
1. To understand and apply different prompt engineering techniques for generating Python programs using AI-assisted tools.	
2. To analyze the impact of context and examples on the accuracy and efficiency of AI-generated code.	
3. To develop and refine real-world Python applications through iterative prompt improvement.	
Lab Outcomes	
1. Students will be able to design effective prompts to generate correct and optimized Python code.	
2. Students will be able to compare and evaluate AI-generated solutions produced using different prompting strategies.	
3. Students will be able to implement and document real-world Python applications using AI-assisted coding tools.	
Experiment – Prompt Engineering Techniques **Task Description** Design and refine prompts using different prompting strategies to generate Python programs for basic computational problems.	
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.	

Zero-Shot Prompt:

Write a Python function that takes an integer as input and returns True
if the number is a palindrome and False otherwise. The function should
not convert the number to a string and should handle edge cases
correctly.

Code:

```python
1    def is_palindrome(num):
2        if num < 0:
3            return False
4
5        original = num
6        reversed_num = 0
7
8        while num > 0:
9            digit = num % 10
10           reversed_num = reversed_num * 10 + digit
11           num //= 10
12
13       return original == reversed_num
14
15   test_cases = [
16       121,          # palindrome
17       12321,        # palindrome
18       123,          # not palindrome
19       10,           # not palindrome
20   ]
```

Output:

```
OUTPUT    PROBLEMS    PORTS    SPELL CHECKER    DEBUG CONSOLE    TERMINAL
● PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> python -u "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py"
  121 True
  12321 True
  123 False
  10 False
⬟ PS C:\Users\shaik\Downloads\Projects-source_code\python-PD>
```

```
14
15    is_palindrome(12.21)    # breaks logic
16    is_palindrome("121")    # TypeError
17    is_palindrome(010)      # invalid in Python, but conceptually ambiguous
```

OUTPUT   PROBLEMS   PORTS   SPELL CHECKER   DEBUG CONSOLE   **TERMINAL**

```
PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> python -u "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py"
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 17
    is_palindrome(010)  # invalid in Python, but conceptually ambiguous
                   ^
SyntaxError: leading zeros in decimal integer literals are not permitted; use an 0o prefix for octal integers
PS C:\Users\shaik\Downloads\Projects-source_code\python-PD>
```

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.

```python
basic.py > ...
1    def factorial(n):
2        if n == 0:
3            return 1
4        return n * factorial(n - 1)
5
```

```
basic.py > factorial
 1    def factorial(n):
 2        """
 3        Computes the factorial of a non-negative integer n.|
 4        Example: factorial(5) -> 120
 5        """
 6        if not isinstance(n, int):
 7            raise TypeError("Input must be an integer")
 8        if n < 0:
 9            raise ValueError("Factorial is not defined for negative numbers")
10
11        result = 1
12        for i in range(1, n + 1):
13            result *= i
14        return result
15
```

Comparison: Zero-Shot vs One-Shot

Aspect	Zero-Shot	One-Shot
Correctness	Partial	Robust
Handles negative input	No	Yes
Handles invalid types	No	Yes
Risk of runtime errors	High	Low
Readability	Basic	Clear & documented
Scalability	Poor (recursion)	Better (iterative)

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.

```python
basic.py > ...
1    def is_armstrong(number):
2        """
3        Checks whether a given number is an Armstrong number.
4        """
5        if not isinstance(number, int):
6            raise TypeError("Input must be an integer")
7
8        if number < 0:
9            return "Not an Armstrong Number"
10
11        digits = str(number)
12        power = len(digits)
13        total = sum(int(digit) ** power for digit in digits)
14
15        if total == number:
16            return "Armstrong Number"
17        else:
18            return "Not an Armstrong Number"
19
20
21   print(is_armstrong(0))        # Armstrong Number
22   print(is_armstrong(1))        # Armstrong Number
23   print(is_armstrong(9))        # Armstrong Number
24   print(is_armstrong(9474))     # Armstrong Number (4-digit case)
25
26
27   print(is_armstrong(-153))     # Not an Armstrong Number
28   print(is_armstrong(10))       # Not an Armstrong Number
29   print(is_armstrong(153.0))    # TypeError
30   print(is_armstrong("153"))    # TypeError
31
```

```
PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> python -u "c:\Users\shaik\Downloads\Projects-source_code\python-PD\bas
Armstrong Number
Armstrong Number
Armstrong Number
Armstrong Number
Not an Armstrong Number
Not an Armstrong Number
Traceback (most recent call last):
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 29, in <module>
    print(is_armstrong(153.0))   # TypeError
          ^^^^^^^^^^^^^^^^^^
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 6, in is_armstrong
    raise TypeError("Input must be an integer")
Armstrong Number
Armstrong Number
Armstrong Number
Not an Armstrong Number
Not an Armstrong Number
Traceback (most recent call last):
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 29, in <module>
    print(is_armstrong(153.0))   # TypeError
          ^^^^^^^^^^^^^^^^^^
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 6, in is_armstrong
    raise TypeError("Input must be an integer")
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 29, in <module>
    print(is_armstrong(153.0))   # TypeError
          ^^^^^^^^^^^^^^^^^^
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 6, in is_armstrong
    raise TypeError("Input must be an integer")
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 6, in is_armstrong
    raise TypeError("Input must be an integer")
    raise TypeError("Input must be an integer")
TypeError: Input must be an integer
PS C:\Users\shaik\Downloads\Projects-source_code\python-PD>
```

(Optional Extension)

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.

Context-Managed Prompt:

You are a Python developer writing production-quality code.

Task: Write an **optimized Python program** that classifies a given number as:

- **Prime**
- **Composite**

- **Neither prime nor composite**

Constraints:

- Accept **only integers** as valid input

- Handle **negative numbers, 0, and 1 correctly**

- Optimize for performance (avoid unnecessary iterations)

- Return clear, human-readable output

- Code must be readable, efficient, and safe for large inputs

```python
basic.py > ...
1  def classify_number(n):
2      """
3      Classifies a number as Prime, Composite, or Neither.
4      """
5
6      if not isinstance(n, int):
7          raise TypeError("Input must be an integer")
8
9      if n <= 1:
10         return "Neither prime nor composite"
11
12     if n == 2:
13         return "Prime"
14
15     if n % 2 == 0:
16         return "Composite"
17
18     limit = int(n ** 0.5) + 1
19     for i in range(3, limit, 2):
20         if n % i == 0:
21             return "Composite"
22
23     return "Prime"
24  def fibonacci(n):
```

```python
basic.py > is_prime
1  def is_prime(n):
2      for i in range(2, n):
3          if n % i == 0:
4              return False
5      return True
6
```

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.

```python
# basic.py > ...
1    def is_perfect(n):
2        total = 0
3        for i in range(1, n):
4            if n % i == 0:
5                total += i
6        return total == n
7
8    print(is_perfect(6))      # Expected: True
9    print(is_perfect(28))     # Expected: True
10   print(is_perfect(12))     # Expected: False
11   print(is_perfect(1))      # Expected: False
12   print(is_perfect(0))      # Expected: False
13   print(is_perfect(-6))     # Expected: False
14
```

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

```python
basic.py > ...
1   def classify_even_odd(n):
2       """
3       Determines whether a number is Even or Odd.
4       """
5
6       if not isinstance(n, int):
7           raise TypeError("Input must be an integer")
8
9       if n % 2 == 0:
10          return "Even"
11      else:
12          return "Odd"
13
14  print(classify_even_odd(-4))    # Even
15  print(classify_even_odd(-7))    # Odd
16
17  print(classify_even_odd(3.5))      # TypeError
18  print(classify_even_odd("10"))     # TypeError
19  print(classify_even_odd(True))     # TypeError
20
```

```
OUTPUT   PROBLEMS   PORTS   SPELL CHECKER   DEBUG CONSOLE   TERMINAL

PS C:\Users\shaik\Downloads\Projects-source_code\python-PD> python -u "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.
⊙ Even
  Odd
Traceback (most recent call last):
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 17, in <module>
    print(classify_even_odd(3.5))      # TypeError
          ^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\shaik\Downloads\Projects-source_code\python-PD\basic.py", line 7, in classify_even_odd
    raise TypeError("Input must be an integer")
TypeError: Input must be an integer
PS C:\Users\shaik\Downloads\Projects-source_code\python-PD>
```

.