

AI Assisted Coding

Lab Assignment – 3.1

Name: Y.Manideep

Hall Ticket Number:

2303A510F8

Question 1: Zero-Shot Prompting – Palindrome Number Program

Prompt Used: Write a Python function to check whether a given number is a palindrome.

AI-Generated Code:

```
def is_palindrome(num):  
    temp = num  
    rev = 0  
    while num > 0:  
        digit = num % 10  
        rev = rev * 10 + digit  
        num //= 10  
    return temp == rev
```

Testing (Input / Output):

Input	Output
121	True
123	False
0	True
-121	False

Observations: Works for positive integers. No validation for non-integers or negatives.

Conclusion: Zero-shot prompting gives basic but incomplete logic.

Question 2: One-Shot Prompting – Factorial Calculation

Prompt Used: Example: 5 → 120

AI-Generated Code:

```
def factorial(n):  
    if n < 0:  
        return "Invalid input"  
    result = 1  
    for i in range(1, n + 1):  
        result *= i  
    return result
```

Testing (Input / Output):

Input	Output
5	120
0	1
1	1
-3	Invalid input

Observations: Negative input handled, logic is clearer.

Conclusion: One-shot prompting improves correctness.

Question 3: Few-Shot Prompting – Armstrong Number Check

Prompt Used: 153 → Armstrong, 370 → Armstrong, 123 → Not Armstrong

AI-Generated Code:

```
def is_armstrong(num):
```

```

if num < 0:
    return "Invalid input"
digits = str(num)
power = len(digits)
total = 0
for d in digits:
    total += int(d) ** power
return "Armstrong Number" if total == num else "Not an Armstrong Number"

```

Testing (Input / Output):

Input	Output
153	Armstrong
370	Armstrong
123	Not Armstrong
0	Armstrong
-10	Invalid

Observations: Examples helped correct logic and structure.

Conclusion: Few-shot prompting improves accuracy.

Question 4: Context-Managed Prompting – Number Classification

Prompt Used: Classify number as Prime, Composite or Neither.

AI-Generated Code:

```

def classify_number(n):
    if not isinstance(n, int):
        return "Invalid input"
    if n <= 1:
        return "Neither Prime nor Composite"
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            return "Composite"
    return "Prime"

```

Testing (Input / Output):

Input	Output
2	Prime
9	Composite
1	Neither
-5	Neither
7	Prime

Observations: Efficient sqrt logic used.

Conclusion: Context-managed prompting gives optimized code.

Question 5: Zero-Shot Prompting – Perfect Number Check

Prompt Used: Write a Python function to check perfect number.

AI-Generated Code:

```

def is_perfect(num):

```

```

if num <= 0:
    return False
total = 0
for i in range(1, num):
    if num % i == 0:
        total += i
return total == num

```

Testing (Input / Output):

Input	Output
6	True
28	True
12	False
1	False

Observations: Correct but inefficient for large inputs.

Conclusion: Zero-shot lacks optimization.

Question 6: Few-Shot Prompting – Even or Odd with Validation

Prompt Used: 8 → Even, 15 → Odd, 0 → Even

AI-Generated Code:

```

def even_or_odd(n):
    if not isinstance(n, int):
        return "Invalid input"
    return "Even" if n % 2 == 0 else "Odd"

```

Testing (Input / Output):

Input	Output
8	Even
15	Odd
0	Even
-4	Even
3.5	Invalid

Observations: Input validation and clarity improved.

Conclusion: Few-shot prompting improves output consistency.

Overall Conclusion

Prompt engineering strongly influences AI-generated code quality. Zero-shot is basic, one-shot improves safety, few-shot improves accuracy, and context-managed prompting produces optimized solutions.