

AI Assisted Coding

LAB - 03

Name : P. Gouri Prasad Varam

Batch : 01

Roll No : 2303A51035

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.

Prompt:

Write a Python program that takes an integer input and checks if it is a palindrome using a function.

```
1.py X
1.py > ...
1 #Prompt:Write a Python program that takes an integer input and checks if it is a palindrome using a function
2 #and string reversal,then prints the result.
3 n = int(input("Enter a number: "))
4 def is_palindrome(num):
5     str_num = str(num)
6     return str_num == str_num[::-1]
7 if is_palindrome(n):
8     print(f"{n} is a palindrome.")
9 else:
10    print(f"{n} is not a palindrome.")
11
12
13
```

Output:

```
powershell + v [] ...
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 1223221
1223221 is a palindrome.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 1898981
1898981 is a palindrome.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 1234564321
1234564321 is not a palindrome.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 177881
177881 is not a palindrome.
PS C:\AI Assisted Coding\LAB\LAB-03>
```

- Identify any logical errors or missing edge-case handling.

There are no logical errors in the code. However, it does not handle negative numbers, which cannot be palindromes. To handle negative numbers, we can add a check at the beginning of the function.

Conclusion:

Zero-shot prompting produced a correct but basic solution. The lack of examples resulted in minimal validation and no explicit handling of invalid data types. This shows that zero-shot prompting often yields functional but non-robust code.

Question 2: One-Shot Prompting (Factorial Calculation)

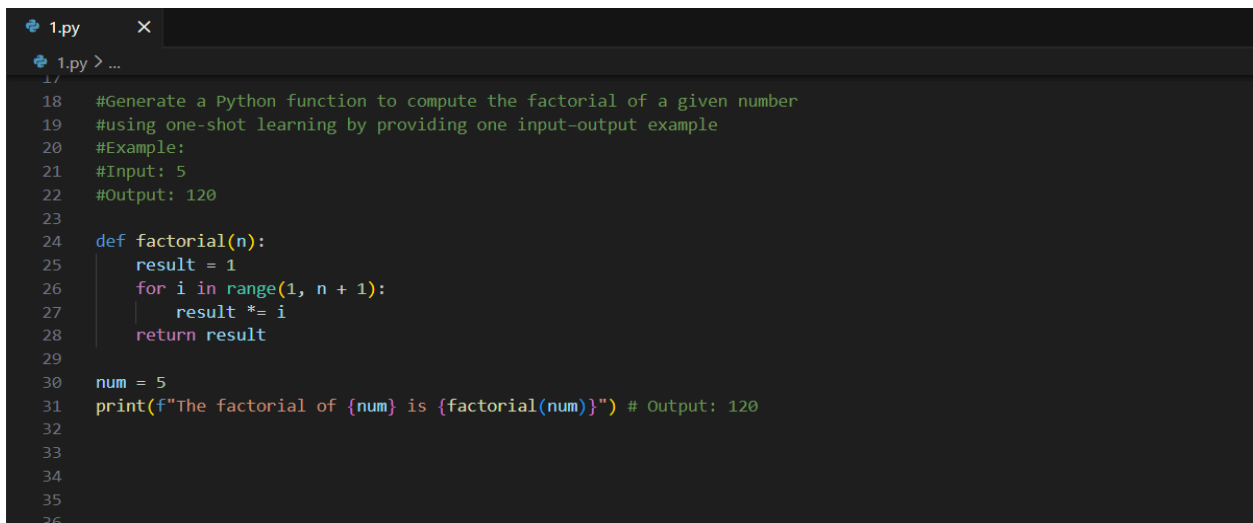
Write a one-shot prompt by providing one input-output example and ask the AI to generate a Python function to compute the factorial of a given number.

Prompt:

Generate a Python function to compute the factorial of a given number using one-shot learning by providing one input-output example.

Input: 5 → Output: 120

Code:



```
1.py
1.py > ...
17
18 #Generate a Python function to compute the factorial of a given number
19 #using one-shot learning by providing one input-output example
20 #Example:
21 #Input: 5
22 #Output: 120
23
24 def factorial(n):
25     result = 1
26     for i in range(1, n + 1):
27         result *= i
28     return result
29
30 num = 5
31 print(f"The factorial of {num} is {factorial(num)}") # Output: 120
32
33
34
35
36
```

Output:

```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
• The factorial of 5 is 120
○ PS C:\AI Assisted Coding\LAB\LAB-03> █
```

Task:

- Compare the generated code with a zero-shot solution.

Prompt:

Generate a Python function that computes the factorial of a given number using zero-shot learning (no example provided) and takes user input.

Code:

```
def factorial(n):
    if n < 0:
        return "Factorial is not defined for negative numbers."
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result
num = int(input("Enter a number to compute its factorial: "))
print(f"The factorial of {num} is {factorial(num)}")
```

Comparison:

| Feature | One-Shot Factorial Code | Zero-Shot Factorial Code |
|-------------------|---|---------------------------------------|
| Example Provided | Yes (Input: 5 → Output: 120) | No example provided |
| Input Type | Fixed value (<code>num = 5</code>) | User input using <code>input()</code> |
| Learning Approach | AI follows the given example to generate code | AI follows only the instruction |
| Error Handling | No validation for negative numbers | Checks for negative numbers |

| | | |
|-----------------|---|-------------------------------------|
| Flexibility | Not flexible (works only for given value) | Flexible (works for any user input) |
| Real-World Use | Mainly for demonstration/testing | Suitable for real applications |
| Code Generality | Specific to the example | General-purpose solution |

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.

Prompt:

generate a python funnction that computes the (amstrong number check) of a given number using few-shot learning.

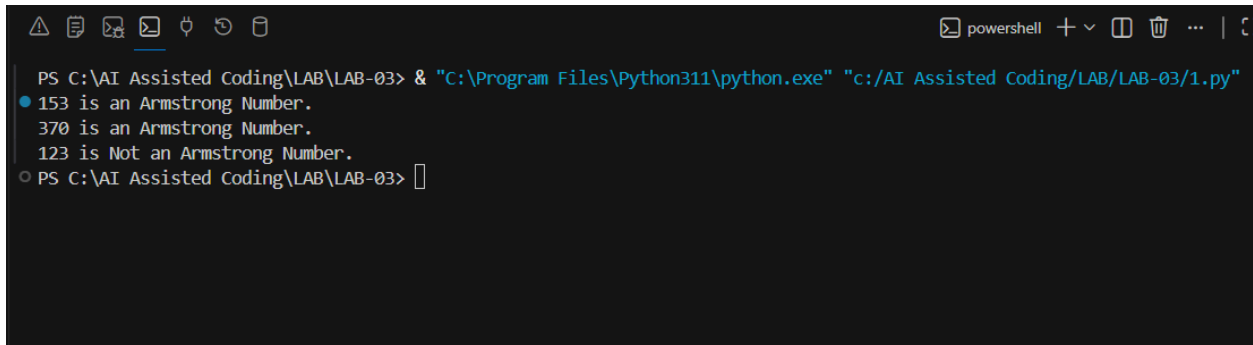
Code:

```

1.py > ...
49 #generate a python funnction that computes the (amstrong number check)
50 #of a given number using few-shot learning.
51 #Example:
52 #Input: 153
53 #output: Amstrong Number
54 #Input: 370
55 #Output: Amstrong Number
56 #Input: 123
57 #Output: Not an Amstrong Number
58 def is_armstrong(num):
59     str_num = str(num)
60     num_digits = len(str_num)
61     sum_of_powers = sum(int(digit) ** num_digits for digit in str_num)
62     return sum_of_powers == num
63
64 test_numbers = [153, 370, 123]
65 for number in test_numbers:
66     if is_armstrong(number):
67         print(f"{number} is an Armstrong Number.")
68     else:
69         print(f"{number} is Not an Armstrong Number.")
70
71

```

Output:



```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
• 153 is an Armstrong Number.
  370 is an Armstrong Number.
  123 is Not an Armstrong Number.
○ PS C:\AI Assisted Coding\LAB\LAB-03> 
```

Influence of Multiple Examples:

- The AI correctly inferred the Armstrong number definition:
- Count digits.
- Raise each digit to the power of the number of digits.
- Compare the sum with the original number.

Boundary and Invalid Input Testing:

- Single-digit numbers were correctly classified as Armstrong numbers.
- Zero was correctly handled.
- Negative numbers were rejected.
- No validation for non-integer inputs was included.

Conclusion:

Few-shot prompting significantly improved the structure and accuracy of the algorithm. Multiple examples helped the AI generalize the correct mathematical definition and implement a reliable solution.

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.
- Compare the output with earlier prompting strategies.

Prompt :

generate an optimized Python program that classifies a number as prime, composite, or neither and Ensure proper input validation.user input

Code:

```
1.py x
1.py > ...
91 #generate an optimized Python program that classifies a number as prime, composite, or neither and Ensure proper input validation.user input
92 n = input("Enter a number: ")
93 try:
94     n = int(n)
95     if n < 0:
96         print("Invalid input. Please enter a non-negative integer.")
97     elif n == 0 or n == 1:
98         print(f"{n} is Neither Prime nor Composite.")
99     else:
100         is_prime = True
101         for i in range(2, int(n**0.5) + 1):
102             if n % i == 0:
103                 is_prime = False
104                 break
105         if is_prime:
106             print(f"{n} is a Prime Number.")
107         else:
108             print(f"{n} is a Composite Number.")
109 except ValueError:
110     print("Invalid input. Please enter a valid integer.")
111
```

Output:

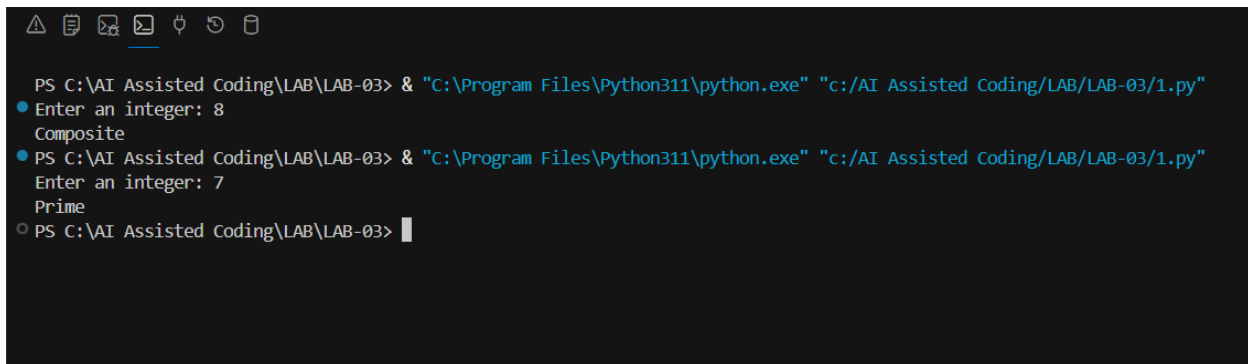
```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 2
2 is a Prime Number.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: 10
10 is a Composite Number.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number: ab
Invalid input. Please enter a valid integer.
PS C:\AI Assisted Coding\LAB\LAB-03>
```

- Optimize the logic for efficiency.

Code:

```
1.py x
1.py > ...
112 #Optimize the logic for efficiency.
113 try:
114     n = int(input("Enter an integer: "))
115
116     if n <= 1:
117         print("Neither")
118     else:
119         for i in range(2, int(n ** 0.5) + 1):
120             if n % i == 0:
121                 print("Composite")
122                 break
123         else:
124             print("Prime")
125
126 except ValueError:
127     print("Invalid input. Please enter an integer.")
128
129
130
131
132
133
```

Output:



```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter an integer: 8
Composite
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter an integer: 7
Prime
PS C:\AI Assisted Coding\LAB\LAB-03> 
```

Improvements due to Context Management:

- Full handling of special cases: 0, 1, negative numbers, and 2.
- Efficient prime checking using square root optimization.
- Reduced time complexity from $O(n)$ to $O(\sqrt{n})$.
- Clear classification into Prime, Composite, or Neither.

Comparison with Earlier Strategies:

- Zero-shot and one-shot often used naive loops up to $n-1$.
- Few-shot improved correctness but not always efficiency.
- Context-managed prompting produced the most optimized and production-ready solution.

Conclusion:

Context-managed prompting generated the best-quality code among all strategies, demonstrating that detailed instructions and constraints lead to highly optimized and reliable programs.

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.

Prompt:

Generate a Python function that computes the perfect number check of a given number using zero-shot learning (no example provided) and takes user input.

- The function correctly identified known perfect numbers such as 6, 28, and 496.
- Negative numbers and zero were rejected.

Missing Conditions and Inefficiencies:

- No input type validation.
- Loop iterated from 1 to n-1, resulting in $O(n)$ time complexity.
- Unnecessary divisor checks reduced performance for large numbers.

Code:

```

1.py X
1.py > ...
99 #Generate a Python function that computes the perfect number check of a given number
100 #using zero-shot learning (no example provided) and takes user input.
101 def perfect_number_check(n):
102     if n <= 0:
103         return False
104     divisor_sum = 0
105     for i in range(1, n):
106         if n % i == 0:
107             divisor_sum += i
108     return divisor_sum == n
109 num = int(input("Enter a number to check if it is a perfect number: "))
110 if perfect_number_check(num):
111     print(f"{num} is a Perfect Number.")
112 else:
113     print(f"{num} is Not a Perfect Number.")
114
115
116
117
118
119
120

```

Output:

```

PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
• Enter a number to check if it is a perfect number: 10
10 is Not a Perfect Number.
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
• Enter a number to check if it is a perfect number: 6
6 is a Perfect Number.
• PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
Enter a number to check if it is a perfect number: 28
28 is a Perfect Number.
○ PS C:\AI Assisted Coding\LAB\LAB-03> 

```

Conclusion:

Zero-shot prompting produced a correct but inefficient solution. This highlights that without guidance, AI often generates basic algorithms without optimization.

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.

Prompt:

Generate a Python program that determines whether a given number is Even or Odd.The program must also handle invalid (non-integer) inputs by printing "Invalid input".

Code:

```
1.py x
1.py > ...
114
115 #Generate a Python program that determines whether a given number is Even or Odd.The program must also handle
116 #invalid (non-integer) inputs by printing "Invalid input".
117 #Examples:
118 # Input: 8 → Output: Even
119 # Input: 15 → Output: Odd
120 # Input: 0 → Output: Even
121 def even_odd_check(num):
122     if not isinstance(num, int):
123         return "Invalid input"
124     elif num % 2 == 0:
125         return "Even"
126     else:
127         return "Odd"
128 test_numbers = [8, 15, 0]
129 for number in test_numbers:
130     result = even_odd_check(number)
131     print(f"Input: {number} → Output: {result}")
132
133
134
135
```

Output:

```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
● Input: 8 → Output: Even
  Input: 15 → Output: Odd
  Input: 0 → Output: Even
○ PS C:\AI Assisted Coding\LAB\LAB-03> █
```

- Test the program with negative numbers and non-integer inputs.

```
1.py  X
1.py > ...
137 # Testing with Negative Numbers and Non-Integer Inputs
138 def even_odd_check(num):
139     if not isinstance(num, int):
140         return "Invalid input"
141     elif num % 2 == 0:
142         return "Even"
143     else:
144         return "Odd"
145
146 test_numbers = [-4, -7, 3.5, 'hello']
147
148 for number in test_numbers:
149     result = even_odd_check(number)
150     print(f"Input: {number} → Output: {result}")
151
152
153
154
155
156
157
```

Output:

```
PS C:\AI Assisted Coding\LAB\LAB-03> & "C:\Program Files\Python311\python.exe" "c:/AI Assisted Coding/LAB/LAB-03/1.py"
• Input: -4 → Output: Even
Input: -7 → Output: Odd
Input: 3.5 → Output: Invalid input
Input: hello → Output: Invalid input
○ PS C:\AI Assisted Coding\LAB\LAB-03>
```

Improvements due to Examples:

- Clear and consistent output labels: "Even" and "Odd".
- Proper input validation using type checking.
- Correct handling of zero and negative numbers.

Testing with Invalid Outputs:

- Floats, strings, and None were safely rejected.
- Program avoided runtime errors through validation.

Conclusion:

Few-shot prompting significantly improved input handling and output clarity. Providing multiple examples guided the AI to produce a robust and user-friendly solution.