

Hall ticket: 2303A51927

Batch: 30

Assignment: 4.2

Task 1: Zero-shot: Prompt AI with only the instruction.

Prompt:

Generate a zero shot prompt to determine whether the given number is prime or not.

Code and Output:

The screenshot shows a Microsoft VS Code interface with the following details:

- File Explorer:** Shows a file named "Lab 4.2.py" in the current workspace.
- Code Editor:** Displays the following Python code:

```
1  # Generate a zero-shot prompt to determine whether the given number is prime or not.
2  def is_prime(n):
3      if n < 2:
4          return False
5      for i in range(2, int(n**0.5) + 1):
6          if n % i == 0:
7              return False
8      return True
9
10 number = 29
11 response = "Prime" if is_prime(number) else "Not Prime"
12 print("Response:", response)
13
14 # Output:
15 # Response: Prime
16
```
- Terminal:** Shows the output of running the script with Python 3.1:

```
3\python.exe "c:/Users/Lenovo/Downloads/Lab 4.2.py"
Response: Prime
PS C:\Users\Lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Lenovo\AppData\Local\Programs\Python\Python31
3\python.exe "c:/Users/Lenovo/Downloads/Lab 4.2.py"
Response: Prime
```

Justification:

This task represents zero-shot prompting because the model was provided only with a clear instruction and no sample inputs or outputs. Based on its existing knowledge, the AI independently generated the logic to check whether a number is prime. The function properly handles edge cases such as numbers less than 2 and efficiently checks factors only up to the square root of the number. This highlights that zero-shot prompting works well for simple, clearly defined problems. It also shows the AI's capability to apply common algorithms correctly without needing examples or additional guidance.

Task 2: One-shot: Provide one example:

Prompt:

Generate a function that calculates the sum of elements in a list.

EX: Input: [1 2 3 4] -> output:10

Code and Output:

File Edit Selection View Go Run ... < > Q AI Assistant

Lab 4.2.py X

C: > Users > Lenovo > Downloads > Lab 4.2.py > ...

```
1 # Generate a function that calculates the sum of elements in a list.
2 # EX: Input: [1, 2, 3, 4] -> output: 10
3 def sum_of_elements(lst):
4     return sum(lst)
5
6 # Example usage:
7 numbers = [1, 2, 3, 4]
8 result = sum_of_elements(numbers)
9 print("Sum of elements:", result) # Output: Sum of elements: 10
10
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

powershell + ⌂ ⌂ ... | ⌂ x

poses. If you want to re-enable it, run 'Import-Module PSReadLine'.

```
PS C:\Users\Lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Lenovo\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Lenovo/Downloads/Lab 4.2.py"
Sum of elements: 10
```

Justification:

In this case, one sample input and output were given to help the AI understand the requirement. This example clarified what the function was expected to do and minimized confusion. The resulting function accurately computes the total of all elements in a list. One-shot prompting helped improve correctness and ensured the output followed the demonstrated format. This shows that even a single example can effectively guide proper code generation.

Task 3: Few-shot: Give 2–3 examples

Prompt:

Generate function that extracts digits from an alphanumeric string

Ex:1 Input: "a1b2c3d4" -> output: "1234"

Ex:2 Input: "abc" -> output: ""

Ex:3 Input: "12345" -> output: "12345"

Code and Output:

The screenshot shows the VS Code interface with a Python file named 'Lab 4.2.py' open. The code defines a function 'extract_digits' that takes a string 's' and returns a string containing only the digits from 's'. It uses a list comprehension with the filter method to achieve this. The code then demonstrates the function with three examples: 'a1b2c3d4', 'abc', and '12345', printing the results to the terminal.

```
1 # Generate function that extracts digits from an alphanumeric string
2 # Ex:1 Input: "a1b2c3d4" -> output: "1234"
3 # Ex:2 Input: "abc" -> output: ""
4 # Ex:3 Input: "12345" -> output: "12345"
5
6 def extract_digits(s):
7     return ''.join(filter(str.isdigit, s))
8
9 # Example usage:
10 input_str1 = "a1b2c3d4"
11 output1 = extract_digits(input_str1)
12 print("Extracted digits from '{}': {}".format(input_str1, output1)) #
13
14 input_str2 = "abc"
15 output2 = extract_digits(input_str2)
16 print("Extracted digits from '{}': {}".format(input_str2, output2)) #
17
18 input_str3 = "12345"
19 output3 = extract_digits(input_str3)
20 print("Extracted digits from '{}': {}".format(input_str3, output3)) #
21
22 # Output:
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

PS C:\Users\Lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Lenovo\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Lenovo/Downloads/Lab 4.2.py"
Extracted digits from 'a1b2c3d4': 1234
Extracted digits from 'abc':
Extracted digits from '12345': 12345

Justification:

Several examples were provided, covering cases with mixed characters, only alphabets, and only numeric values. These examples helped the AI clearly understand the task as well as handle edge situations. The generated function correctly extracts digits from any given alphanumeric string. Few-shot prompting enhanced accuracy, clarity, and reliability. This shows how providing multiple examples improves the model's ability to recognize patterns effectively.

Task 4: zero-shot vs few-shot prompting

Prompt:

Generate a function that counts the number of vowels in a string.

Code and OUTPUT:

The screenshot shows the Microsoft VS Code interface with a dark theme. A Python file named 'Lab 4.2.py' is open in the editor. The code defines a function 'count_vowels' that counts the number of vowels in a given string, including both lowercase and uppercase vowels. It includes examples for strings like 'Hello World', 'Python', and 'AEIOU'. The terminal below the editor shows the command 'python Lab 4.2.py' being run, followed by the output: 'Number of vowels in 'Hello World': 3' and 'Number of vowels in 'Python': 1'.

```
1 # Generate a function that counts the number of vowels in a string.
2 # Ex: Input: "Hello World" -> output: 3
3 # Ex: Input: "Python" -> output: 1
4 # Ex: Input: "AEIOU" -> output: 5
5
6 def count_vowels(s):
7     vowels = 'aeiouAEIOU'
8     return sum(1 for char in s if char in vowels)
9
10 # Example usage:
11 input_str = "Hello World"
12 vowel_count = count_vowels(input_str)
13 print("Number of vowels in '{}': {}".format(input_str, vowel_count)) # Output: Number of vowels in 'Hello World': 3
14
15 # Output: Number of vowels in 'Hello World': 3
16
17 # Example usage:
18 input_str2 = "Python"
19 vowel_count2 = count_vowels(input_str2)
20 print("Number of vowels in '{}': {}".format(input_str2, vowel_count2)) # Output: Number of vowels in 'Python': 1
21
22 # Output: Number of vowels in 'Python': 1
```

PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS

PS C:\Users\Lenovo\AppData\Local\Programs\Microsoft VS Code> & C:\Users\Lenovo\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/Lenovo/Downloads/Lab 4.2.py"
Number of vowels in 'Hello World': 3
Number of vowels in 'Python': 1

Justification:

This task follows **few-shot prompting** because multiple example inputs and outputs were provided before generating the function. These examples clearly demonstrate how vowels should be counted for different types of strings, including mixed-case letters and all-uppercase vowels. The examples helped the AI understand the expected logic and output format. The generated function correctly counts vowels in each case. This shows that few-shot prompting improves clarity, accuracy, and consistency by guiding the AI with relevant examples.

Prompt:

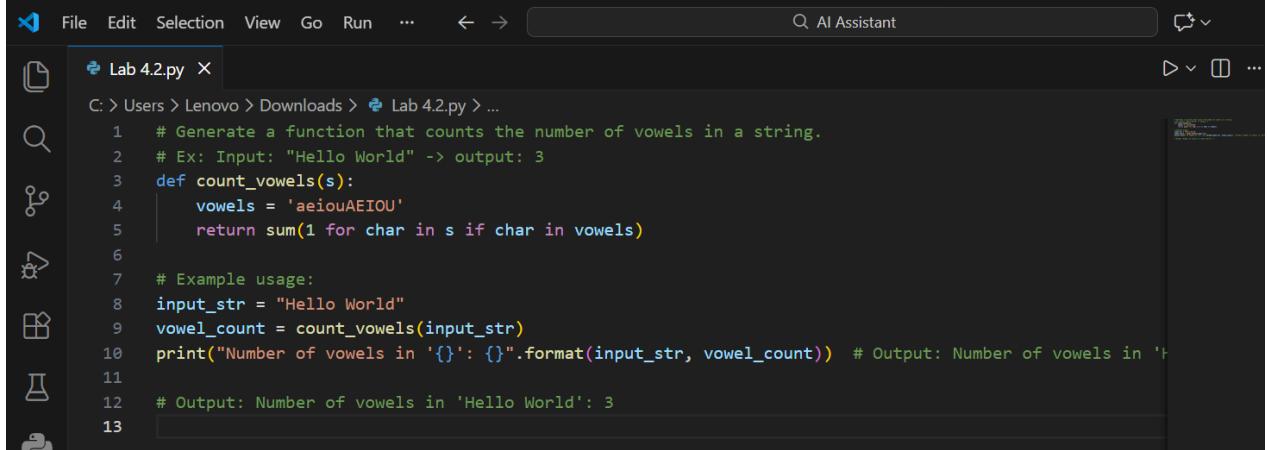
Generate a function that counts the number of vowels in a string.

Ex: Input: "Hello World" -> output: 3

Ex: Input: "Python" -> output: 1

Ex: Input: "AEIOU" -> output: 5

Code and OUTPUT:



The screenshot shows a code editor window with a dark theme. The title bar says "Lab 4.2.py". The code in the editor is as follows:

```
C: > Users > Lenovo > Downloads > Lab 4.2.py > ...
1 # Generate a function that counts the number of vowels in a string.
2 # Ex: Input: "Hello World" -> output: 3
3 def count_vowels(s):
4     vowels = 'aeiouAEIOU'
5     return sum(1 for char in s if char in vowels)
6
7 # Example usage:
8 input_str = "Hello World"
9 vowel_count = count_vowels(input_str)
10 print("Number of vowels in '{}': {}".format(input_str, vowel_count)) # Output: Number of vowels in 'Hello World': 3
11
12 # Output: Number of vowels in 'Hello World': 3
13
```

Comparison Table:

# Comparison Table for both zero shot and few shot prompting for counting the number of vowels in a string, just a table not code		
Aspect	Zero Shot Prompting	Few Shot Prompting
Definition	Directly asks the model to count vowels without examples	Provides examples of input-output pairs for counting vowels
Examples Provided	None	Multiple examples demonstrating the task
Clarity of Task	May be less clear, relies on model's understanding	
More explicit due to examples provided		
Accuracy	May vary, depends on model's prior knowledge	Generally higher accuracy due
Contextual Understanding	Relies on model's general knowledge	Enhanced by specific examples provided
Response Time	Potentially faster as no examples need to be processed	May take slightly longer due to processing examples
Use Case	Quick tasks where examples are not feasible	Tasks requiring higher accuracy and clarity
Flexibility	More flexible, can handle a wider range of inputs	More rigid, may perform best on inputs similar to examples
Adaptability	Adapts based on model's training data	Adapts based on provided examples
Complexity Handling	May struggle with complex inputs	Better at handling complexity due to examples
Training Requirement	No additional training needed	Requires careful selection of examples for training
Cost	Lower cost due to fewer tokens used	Higher cost due to additional tokens for examples
Scalability	Easier to scale for various tasks	More effort needed to create examples for each task
User Effort	Minimal effort, just a prompt	More effort to create and select examples

Justification:

With zero-shot prompting, the AI relies solely on its general understanding of vowels without any guidance. In contrast, few-shot prompting includes examples that demonstrate how vowels should be counted across different inputs. This comparison shows that few-shot prompting offers clearer direction and more consistent results. Providing examples helps minimize confusion and increases the accuracy of the output. It emphasizes how examples play a key role in effectively guiding AI behavior.

Task 5: Use few-shot prompting with 3 sample inputs

Prompt:

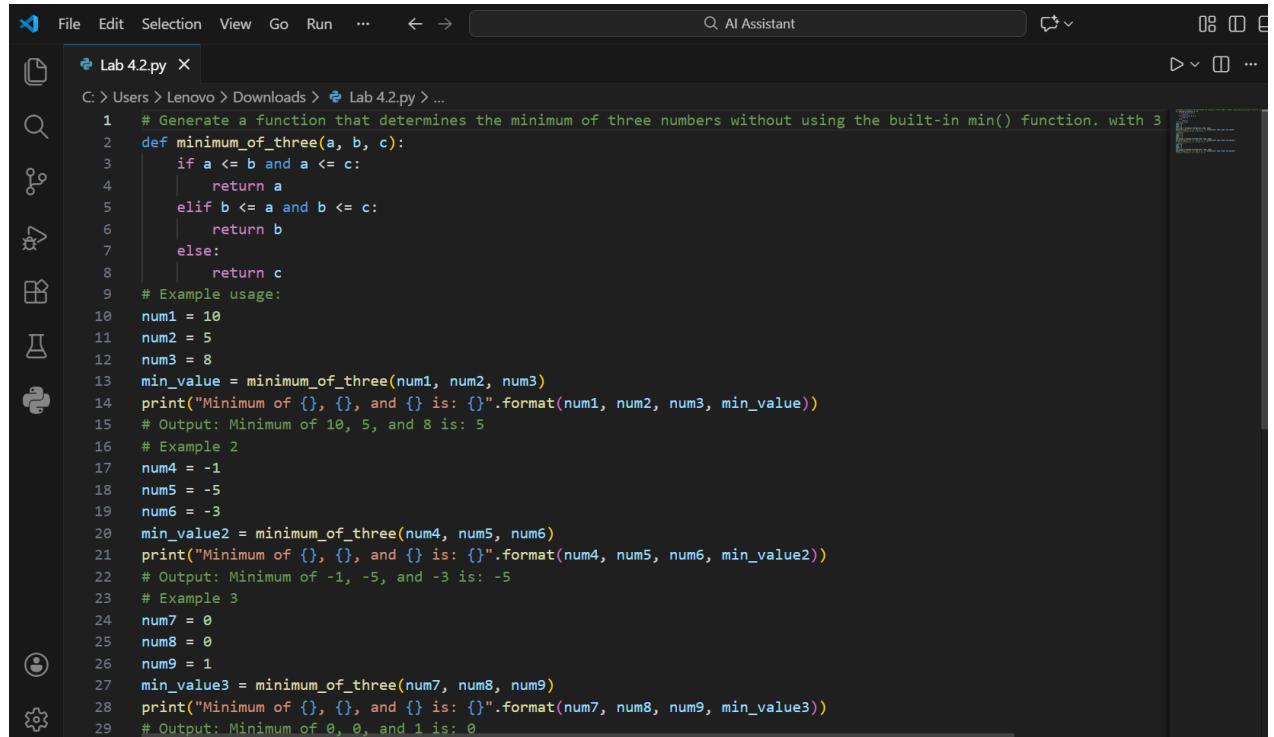
Generate a function that determines the minimum of three numbers without using the built-in min() function. with 3 or more examples.

Ex 1: Input: (10, 5, 8) -> output: 5

Ex 2: Input: (-1, -5, -3) -> output: -5

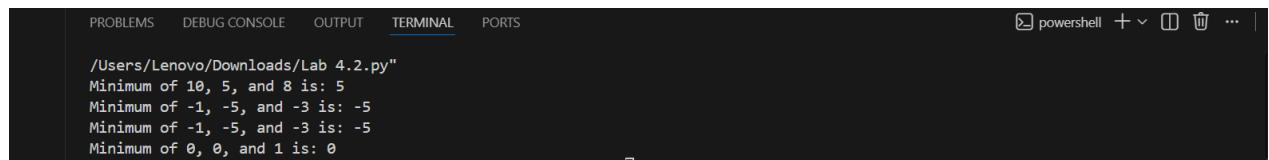
Ex 3: Input: (0, 0, 1) -> output: 0

Code:



```
Lab 4.2.py
C: > Users > Lenovo > Downloads > Lab 4.2.py > ...
1 # Generate a function that determines the minimum of three numbers without using the built-in min() function. with 3 or more examples.
2 def minimum_of_three(a, b, c):
3     if a <= b and a <= c:
4         return a
5     elif b <= a and b <= c:
6         return b
7     else:
8         return c
9 # Example usage:
10 num1 = 10
11 num2 = 5
12 num3 = 8
13 min_value = minimum_of_three(num1, num2, num3)
14 print("Minimum of {}, {} , and {} is: {}".format(num1, num2, num3, min_value))
15 # Output: Minimum of 10, 5, and 8 is: 5
16 # Example 2
17 num4 = -1
18 num5 = -5
19 num6 = -3
20 min_value2 = minimum_of_three(num4, num5, num6)
21 print("Minimum of {}, {} , and {} is: {}".format(num4, num5, num6, min_value2))
22 # Output: Minimum of -1, -5, and -3 is: -5
23 # Example 3
24 num7 = 0
25 num8 = 0
26 num9 = 1
27 min_value3 = minimum_of_three(num7, num8, num9)
28 print("Minimum of {}, {} , and {} is: {}".format(num7, num8, num9, min_value3))
29 # Output: Minimum of 0, 0, and 1 is: 0
```

Output:



```
PROBLEMS DEBUG CONSOLE OUTPUT TERMINAL PORTS
powershell + ⌂ ⌄ ⌅ ... |
```

```
/Users/Lenovo/Downloads/Lab 4.2.py"
Minimum of 10, 5, and 8 is: 5
Minimum of -1, -5, and -3 is: -5
Minimum of -1, -5, and -3 is: -5
Minimum of 0, 0, and 1 is: 0
```

Justification:

Three sample cases were given, including positive values, negative values, and equal numbers. These examples helped the AI understand the comparison logic clearly. The generated function accurately finds the minimum value without using the built-in `min()` function. Few-shot prompting guided the use of correct conditional statements and proper handling of different scenarios. This shows that providing examples improves the accuracy of decision-based code generation.