

Assignment:4.2

Task01-Zero-shot: Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime

The image shows a code editor on the left and an AI assistant interface on the right. The code editor displays a Python function `is_prime(n)` that checks for primality. The function includes docstrings for its purpose, arguments, and return values, as well as test cases. The AI assistant on the right provides a detailed explanation of the function's logic, including a list of test cases and their expected outputs.

```
1 def is_prime(n):
2     """
3     Determine whether a given number is prime.
4
5     Args:
6         n: An integer to check for primality
7
8     Returns:
9         bool: True if n is prime, False otherwise
10    """
11    if n < 2:
12        return False
13
14    if n == 2:
15        return True
16
17    if n % 2 == 0:
18        return False
19
20    for i in range(3, int(n**0.5) + 1, 2):
21        if n % i == 0:
22            return False
23
24    return True
25
26
27 # Test cases
28 if __name__ == "__main__":
29     test_numbers = [1, 2, 3, 4, 5, 10, 17, 20, 97, 100]
30     for num in test_numbers:
31         print(f"{num}: {is_prime(num)}")
32
```

PYTHON FUNCTION TO CHECK FOR PRIME NUMBERS

Zero-shot: Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime and expected output as A basic Python function to check if a number is prime, demonstrating correct logical conditions without relying on examples or additional context.

✓ Read Python file

Now I'll write a prime-checking function to the file:

✓ Created Task1.py

Done! I've written a prime-checking function with the following logic:

1. **Numbers < 2:** Returns `False` (not prime by definition)
2. **Number = 2:** Returns `True` (the only even prime)
3. **Even numbers:** Returns `False` (divisible by 2)
4. **Odd numbers:** Checks divisibility by odd numbers from 3 up to \sqrt{n} (efficient approach since any factor larger than \sqrt{n} would have a corresponding factor smaller than \sqrt{n})

The function includes test cases demonstrating its correctness across various inputs including edge cases (1, 2), primes (3, 5, 17, 97), and composites (4, 10, 20, 100).

Claude Haiku 4.5 • 1x

> 1 file changed +32 -1

Keep Undo

+ Task1.py

Describe what to build next

Agent Auto

The image shows a VS Code editor with a Python file named `Task1.py`. The code defines a function `is_prime(n)` that checks if a number is prime. The function includes comments for its purpose, arguments, and return values. The logic checks for numbers less than 2, even numbers (except 2), and divisibility by odd numbers up to the square root of `n`.

```
1 def is_prime(n):
2     """
3     Determine whether a given number is prime.
4
5     Args:
6         n: An integer to check for primality
7
8     Returns:
9         bool: True if n is prime, False otherwise
10    """
11    if n < 2:
12        return False
13
14    if n == 2:
15        return True
16
17    if n % 2 == 0:
18        return False
19
20    for i in range(3, int(n**0.5) + 1, 2):
21        if n % i == 0:
22            return False
23
```

The terminal shows the execution of the script, which prints the results of `is_prime` for various inputs:

```
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task1.py
1: False
2: True
3: True
4: False
5: True
10: False
17: True
20: False
97: True
100: False
PS C:\Users\tloke\OneDrive\AI>
```

On the right, a chat window displays a Zero-shot AI prompt: "Prompt AI with only the instruction. Write a Python function to determine whether a given number is prime and expected output as A basic Python function to check if a number is prime, demonstrating correct logical conditions without relying on examples or additional context". Below the prompt, there are instructions to read the Python file and write a prime-checking function. A list of test cases is provided: 1. Numbers < 2: Returns False; 2. Number = 2: Returns True; 3. Even numbers: Returns False; 4. Odd numbers: Checks divisibility by odd numbers from 3 up to sqrt(n).

Task02-One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a function that calculates the sum of elements in a list.

The image shows a VS Code editor window with a Python script named `Task2.py`. The script defines a function `sum_list(elements)` that calculates the sum of elements in a list. The script includes a docstring, arguments, and returns. It also includes a test case: `# Example from specification` where `sum_list([1, 2, 3, 4])` is called and the result is printed.

```
1 def sum_list(elements):
2     """
3     Calculate the sum of elements in a list.
4
5     Args:
6     |     elements: A list of numeric values
7
8     Returns:
9     |     int or float: The sum of all elements in the list
10    """
11    total = 0
12    for element in elements:
13        total += element
14    return total
15
16
17 # Example from specification
18 if __name__ == "__main__":
19     # Input: [1, 2, 3, 4], Output: 10
20     result = sum_list([1, 2, 3, 4])
21     print(f"Input: [1, 2, 3, 4]")
22     print(f"Output: {result}")
23
```

The terminal output shows the execution of the script:

```
17: True
20: False
97: True
100: False
PS C:\Users\tloke\OneDrive\AI> & C:/Users/tloke/AppData/Local/Python/pythoncore-3.14-64/python.exe c:/Users/tloke/OneDrive/AI/Task2.py
Input: [1, 2, 3, 4]
Output: 10

sum_list([5, 10, 15]): 30
sum_list([]): 0
sum_list([42]): 42
PS C:\Users\tloke\OneDrive\AI>
```

On the right side, there is a chat interface with a message from the AI:

One-shot: Provide one example: Input: [1, 2, 3, 4], Output: 10 to help AI generate a function that calculates the sum of elements in a list, and expected output as A correct conversion function guided by the single example.

Below the chat message, there is a section titled "Created and read Python file" with the text: "Done! I've created a `sum_list()` function that calculates the sum of elements in a list. The function:

- 1. Iterates through each element in the input list
- 2. Accumulates the total by adding each element
- 3. Returns the final sum

The example [1, 2, 3, 4] → 10 is demonstrated, along with additional test cases showing the function handles edge cases like empty lists and single elements correctly.

At the bottom, there is a section titled "Describe what to build next" with a dropdown menu showing "Agent v" and "Auto v" and a button with a right arrow.

Task3-Give 2–3 examples to create a function that extracts digits from an alphanumeric string

The screenshot shows a code editor with the following content:

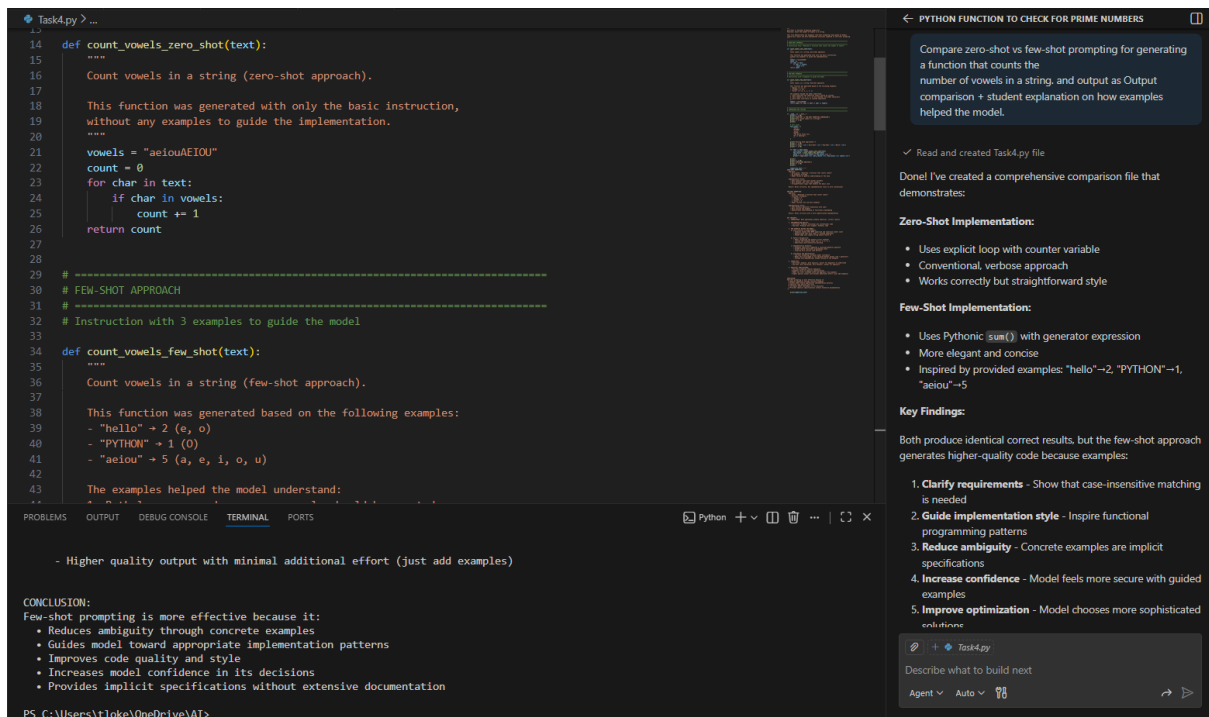
```

Task3.py > ...
1 def extract_digits(alphanumeric_string):
2     """
3     Extract all digits from an alphanumeric string.
4
5     Args:
6         alphanumeric_string: A string containing letters, digits, and possibly spaces
7
8     Returns:
9         str: A string containing only the digits found in the input
10    """
11    return ''.join(char for char in alphanumeric_string if char.isdigit())
12
13
14 # Few-shot examples
15 if __name__ == "__main__":
16     # Example 1
17     input1 = "abc123def"
18     output1 = extract_digits(input1)
19     print(f"Example 1:")
20     print(f"Input: '{input1}'")
21     print(f"Output: '{output1}'")
22
23
24 Example 2:
25 Input: 'hello456world789'
26 Output: '456789'
27
28 Example 3:
29 Input: 'xyz'
30 Output: ''
31
32 Additional test cases:
33 extract_digits('test123abc456'): '123456'
34 extract_digits('12345'): '12345'
35 extract_digits('a1b2c3d4'): '1234'

```

Task4-Compare zero-shot vs few-shot prompting for generating a function that

counts the number of vowels in a string.



```
14 def count_vowels_zero_shot(text):
15     """
16     Count vowels in a string (zero-shot approach).
17
18     This function was generated with only the basic instruction,
19     without any examples to guide the implementation.
20     """
21     vowels = "aeiouAEIOU"
22     count = 0
23     for char in text:
24         if char in vowels:
25             count += 1
26     return count
27
28
29 # FEW-SHOT APPROACH
30 # Instruction with 3 examples to guide the model
31
32 def count_vowels_few_shot(text):
33     """
34     Count vowels in a string (few-shot approach).
35
36     This function was generated based on the following examples:
37     - "hello" → 2 (e, o)
38     - "PYTHON" → 1 (O)
39     - "aeiou" → 5 (a, e, i, o, u)
40
41     The examples helped the model understand:
42
43     - Higher quality output with minimal additional effort (just add examples)
44
45     CONCLUSION:
46     Few-shot prompting is more effective because it:
47     • Reduces ambiguity through concrete examples
48     • Guides model toward appropriate implementation patterns
49     • Improves code quality and style
50     • Increases model confidence in its decisions
51     • Provides implicit specifications without extensive documentation
```

PYTHON FUNCTION TO CHECK FOR PRIME NUMBERS

Compare zero-shot vs few-shot prompting for generating a function that counts the number of vowels in a string, and output as Output comparison + student explanation on how examples helped the model.

✓ Read and created Task4.py file

Done! I've created a comprehensive comparison file that demonstrates:

Zero-Shot Implementation:

- Uses explicit loop with counter variable
- Conventional, verbose approach
- Works correctly but straightforward style

Few-Shot Implementation:

- Uses Pythonic `sum()` with generator expression
- More elegant and concise
- Inspired by provided examples: "hello"→2, "PYTHON"→1, "aeiou"→5

Key Findings:

Both produce identical correct results, but the few-shot approach generates higher-quality code because examples:

1. **Clarify requirements** - Show that case-insensitive matching is needed
2. **Guide implementation style** - Inspire functional programming patterns
3. **Reduce ambiguity** - Concrete examples are implicit specifications
4. **Increase confidence** - Model feels more secure with guided examples
5. **Improve optimization** - Model chooses more sophisticated solutions

Describe what to build next

Agent Auto

Explanation: In zero-shot prompting, the model generates the function using its general understanding of programming and vowels. Since no examples are given, the model decides on its own how to handle cases like uppercase letters.

In few-shot prompting, the examples clearly show what the expected input and output look like. From these examples, the model learns important rules such as:

- Uppercase vowels should be counted
- Strings without vowels should return 0

Because of this guidance, the model produces a more accurate and user-aligned function. Examples act like hints, reducing ambiguity and improving correctness.

Task5-Use few-shot prompting with 3 sample inputs to generate a function that determines the minimum of three numbers without using the built-in min() function.

