

LAB ASSIGNMENT 3.1: Improving Prompts and Context Management

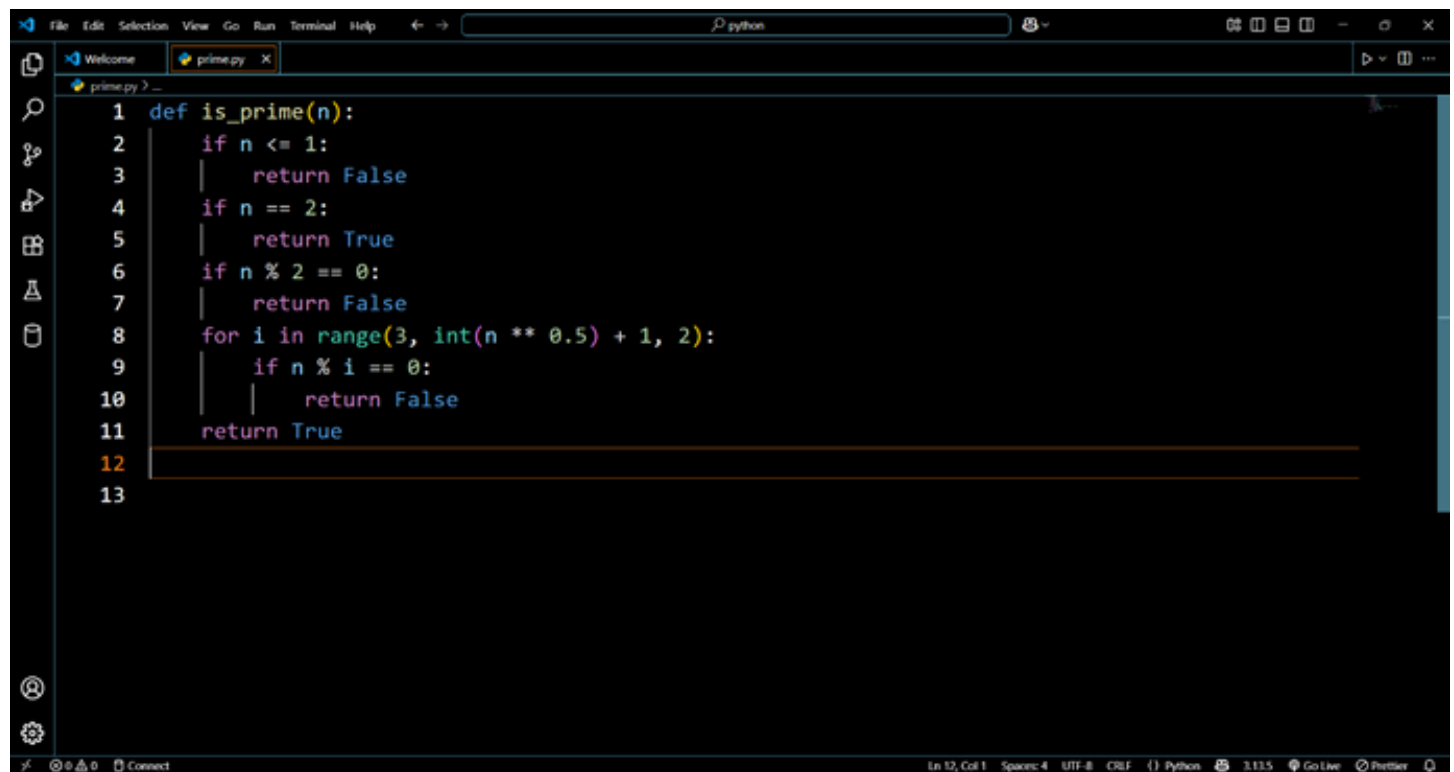
TASK 1: To explore how prompt design and context influence AI-generated outputs and to learn techniques to improve AI responses.

TOOL: GIT-HUB COPILOT VS CODE

ZERO-SHOT:

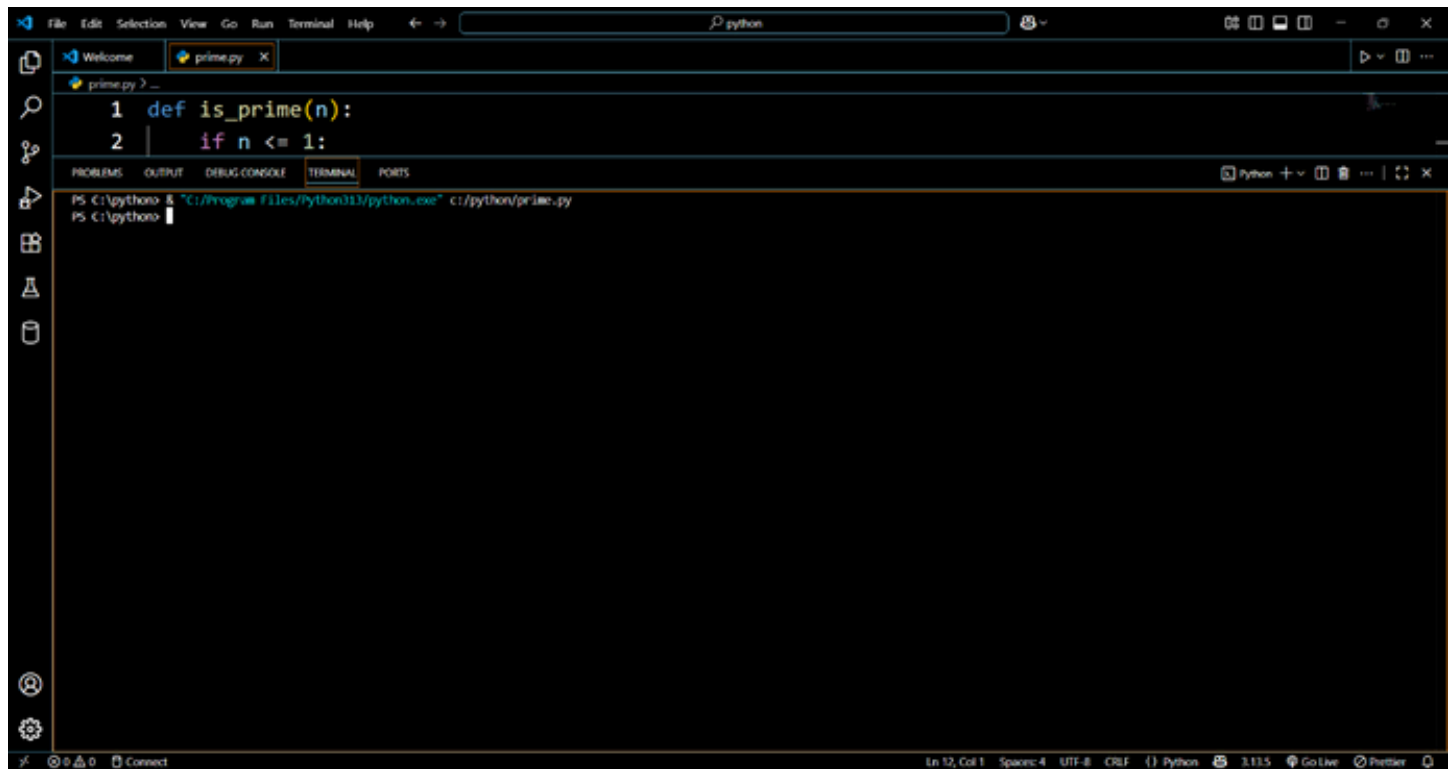
PROMPT: write a python function to check whether a number is prime.

CODE:

A screenshot of the Visual Studio Code editor interface. The main window displays a Python file named 'prime.py'. The code defines a function 'is_prime(n)' that checks if a number is prime. The function returns False for n <= 1, True for n == 2, False for even numbers (n % 2 == 0), and then iterates from 3 to the square root of n to check for divisors. The status bar at the bottom indicates the file is at line 12, column 1, with 4 spaces, UTF-8 encoding, CRLF line endings, and Python 3.11.5. The Go Line and PyLint extensions are also visible.

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13
```

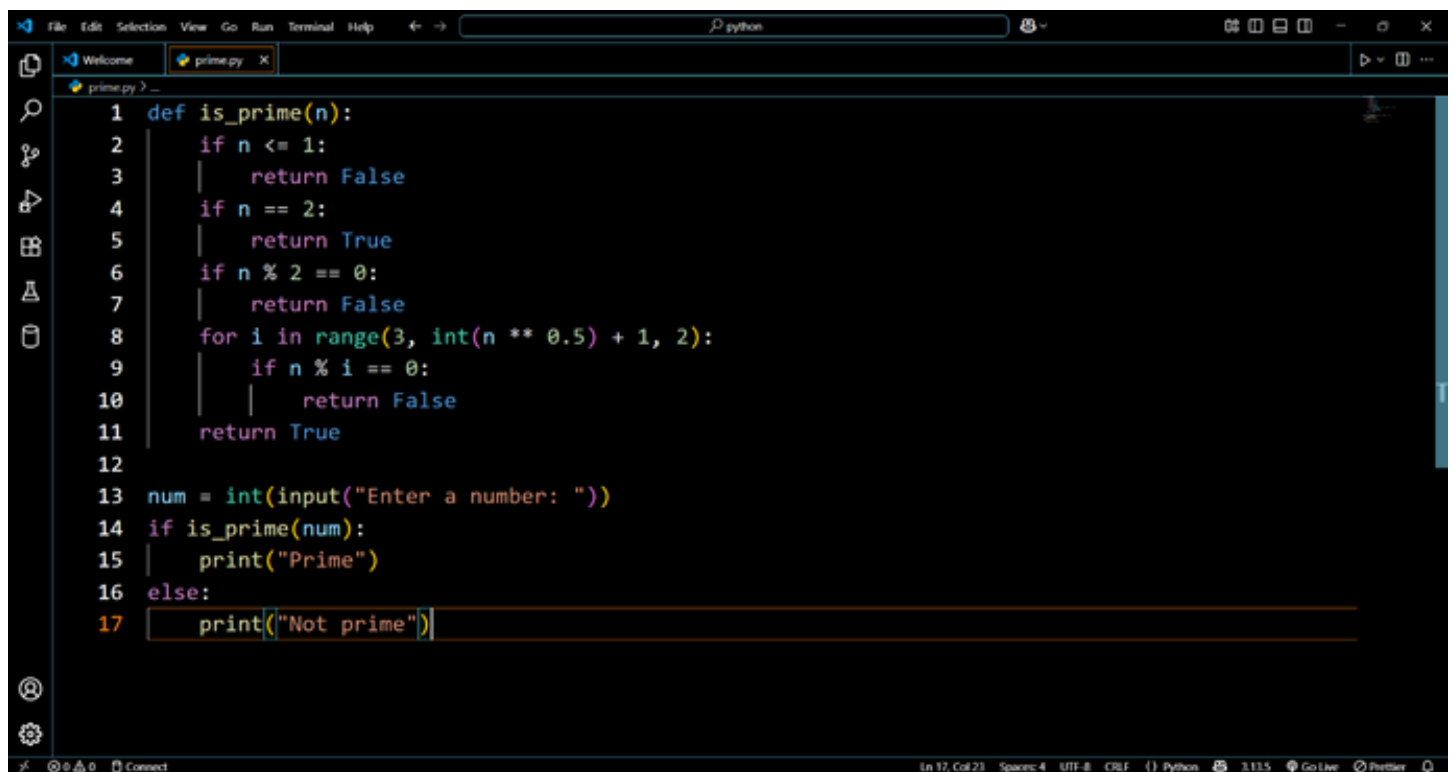
OUTPUT: no example provided, since, its does not call the function and take any input it's just a zero shot code.



```
File Edit Selection View Go Run Terminal Help python
prime.py x
prime.py > _
1 def is_prime(n):
2     if n <= 1:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - Python 3.11.5 Go Live Prettier
PS C:\python> & "C:/Program Files/Python311/python.exe" c:/python/prime.py
PS C:\python>
```

ONE-SHOT:

CODE:



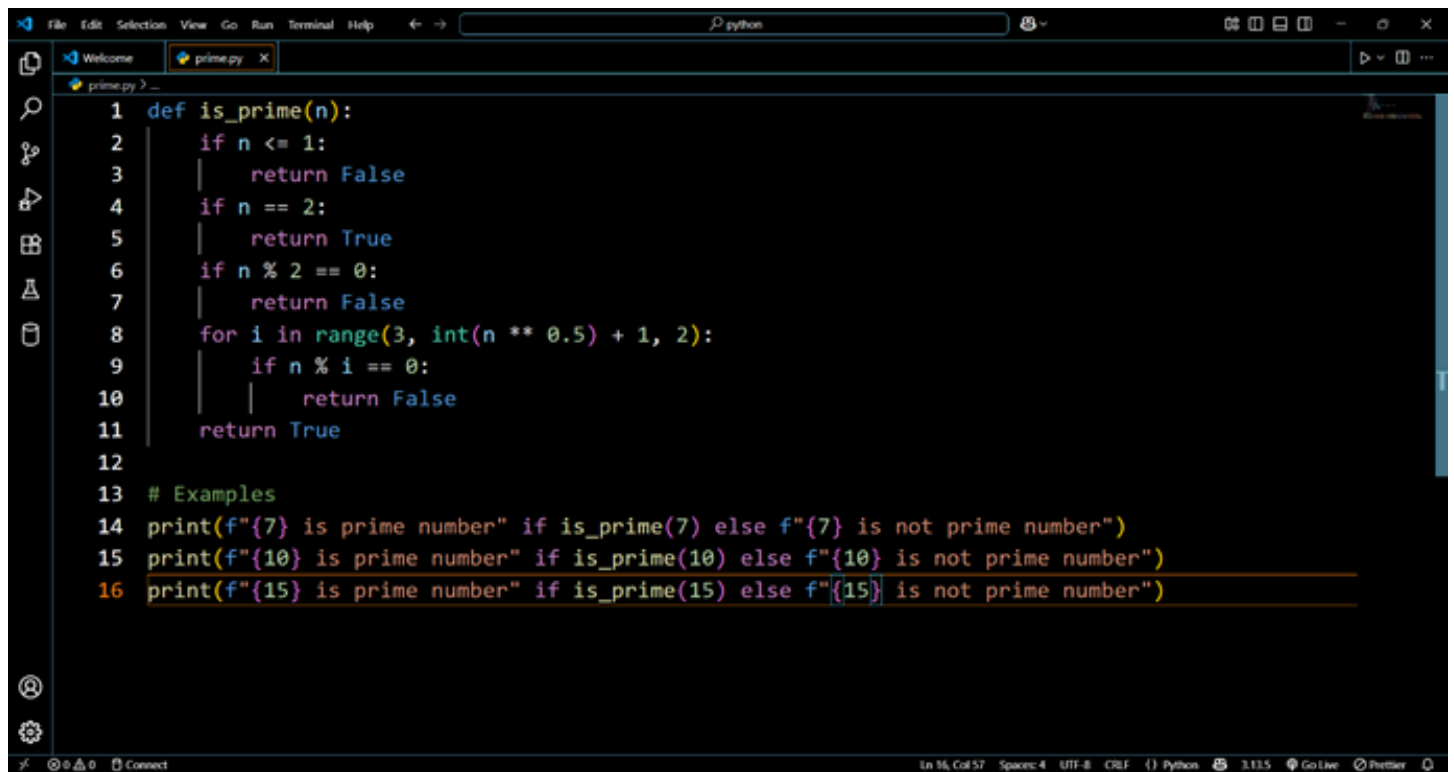
```
File Edit Selection View Go Run Terminal Help python
prime.py x
prime.py > _
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13 num = int(input("Enter a number: "))
14 if is_prime(num):
15     print("Prime")
16 else:
17     print("Not prime")
```

OUTPUT:



FEW-SHOT:

CODE:



OUTPUT:

```
File Edit Selection View Go Run Terminal Help python
Welcome prime.py x
prime.py > ...
1 def is_prime(n):
2     if n <= 1:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - - - - -
PS C:\python> & "C:/Program Files/Python311/python.exe" c:/python/prime.py
7 is prime number
10 is not prime number
15 is not prime number
PS C:\python>
In 16, Col 57 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live Prettier
```

CONTEXT-MANAGED:

CODE:

```
File Edit Selection View Go Run Terminal Help python
Welcome prime.py x
prime.py > is prime
1 def is_prime(n: int) -> bool:
2     """Check if a number is prime.
3     Args:
4     | n (int): Number to check.
5     Returns:
6     | bool: True if n is prime, False otherwise.
7     """
8     if n <= 1:
9         return False
10    if n == 2:
11        return True
12    if n % 2 == 0:
13        return False
14    sqrt_n = int(n ** 0.5) + 1
15    for i in range(3, sqrt_n, 2):
16        if n % i == 0:
17            return False
18    return True
19 if __name__ == "__main__":
20     test_numbers = [7, 10, 15, 11, 23, 29, 1, 0, 4, 9, -5]
21     for num in test_numbers:
22         result = "is prime number" if is_prime(num) else "is not prime number"
23         print(f"{num} {result}")
In 8, Col 15 Spaces: 4 UTF-8 CRLF Python 3.11.5 Go Live Prettier
```

OUTPUT:



```
PS C:\python> python -u "c:\python\prime.py"
7 is prime number
10 is not prime number
15 is not prime number
11 is prime number
23 is prime number
29 is prime number
1 is not prime number
0 is not prime number
4 is not prime number
9 is not prime number
-5 is not prime number
PS C:\python>
```

TOOL: GEMINI

Zero-Shot:

I used idle to display the code got from Gemini to run the code

Prompt: write a python code to check whether a number is prime or not.

Here, the code is given with function is_prime_basic

This code has no output

```
*gemini_zero shot.py - C:/Users/haasini/gemini_zero shot.py (3.13.5)*
File Edit Format Run Options Window Help
def is_prime_basic(n):
    # Prime numbers must be greater than 1
    if n <= 1:
        return False

    # Check for divisibility from 2 up to n-1
    for i in range(2, n):
        if n % i == 0:
            return False # Found a divisor, so it's not prime

    return True # No divisors were found, so it's prime
```

One-shot: Here, the code uses function again and checks a number whether its prime or not.

Prompt: write a python function to check a number is prime or not give one example usage.

```
*gemini_zero shot.py - C:/Users/haasini/gemini_zero shot.py (3.13.5)*
File Edit Format Run Options Window Help
import math

def is_prime(n):
    # Prime numbers must be greater than 1.
    if n <= 1:
        return False

    # Check for divisibility from 2 up to the square root of n.
    # We only need to check up to the square root because if a number 'n' has a
    # divisor greater than its square root, it must also have a divisor
    # smaller than its square root.
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            # If n is divisible by any number in this range, it's not prime.
            return False

    # If no divisors were found, the number is prime.
    return True

# --- Example Usage ---
number_to_check = 17
if is_prime(number_to_check):
    print(f"{number_to_check} is a prime number.")
else:
    print(f"{number_to_check} is not a prime number.")
```

Output:



```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>> ===== RESTART: C:/Users/haasini/gemini_zero shot.py =====
17 is a prime number.
>>>
```

Few-Shot:

Prompt: write a python function to check whether a number is prime or not example usages any 3 or 4.



```
*gemini_zero shot.py - C:/Users/haasini/gemini_zero shot.py (3.13.5)*
File Edit Format Run Options Window Help

import math
def is_prime(n):
    # Prime numbers must be greater than 1.
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            # If n is divisible by any number in this range, it's not prime.
            return False

    # If no divisors were found, the number is prime.
    return True

# --- Example Usage ---

# Example 1: A prime number
number_to_check_1 = 17
if is_prime(number_to_check_1):
    print(f"{number_to_check_1} is a prime number.")
else:
    print(f"{number_to_check_1} is not a prime number.")

# Example 2: A composite number
number_to_check_2 = 25
if is_prime(number_to_check_2):
    print(f"{number_to_check_2} is a prime number.")
else:
    print(f"{number_to_check_2} is not a prime number.")

# Example 3: The number 2 (the only even prime)
number_to_check_3 = 2
if is_prime(number_to_check_3):
    print(f"{number_to_check_3} is a prime number.")
else:
    print(f"{number_to_check_3} is not a prime number.")

# Example 4: A number less than 2
number_to_check_4 = 1
if is_prime(number_to_check_4):
    print(f"{number_to_check_4} is a prime number.")
else:
    print(f"{number_to_check_4} is not a prime number.")
```

Output:




```
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.

>>> ===== RESTART: C:/Users/haasini/gemini_zero shot.py =====
17 is a prime number.
25 is not a prime number.
2 is a prime number.
1 is not a prime number.
>>>
```

CONTEXT-MANAGED

PROMPT: Give the python code to find a number is prime or not in more efficient and optimized way.

Code:



```
*gemini_zero shot.py - C:/Users/haasini/gemini_zero shot.py (3.13.5)*
File Edit Format Run Options Window Help

import math
def is_prime(n):
    if n <= 1:
        return False
    if n <= 3:
        return True

    # Numbers divisible by 2 or 3 are not prime.
    if n % 2 == 0 or n % 3 == 0:
        return False

    # Check for divisors starting from 5.
    # All primes greater than 3 can be expressed in the form 6k ± 1.
    # So, we can check for divisibility by i and i + 2, where i starts at 5
    # and increments by 6.
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6

    # If no divisors were found, the number is prime.
    return True

# --- Example Usage ---

# Example 1: A prime number
number_to_check_1 = 17
if is_prime(number_to_check_1):
    print(f"{number_to_check_1} is a prime number.")
else:
    print(f"{number_to_check_1} is not a prime number.")

# Example 2: A composite number
number_to_check_2 = 25
if is_prime(number_to_check_2):
    print(f"{number_to_check_2} is a prime number.")
else:
    print(f"{number_to_check_2} is not a prime number.")
```




```
IDLE Shell 3.13.5
File Edit Shell Debug Options Window Help
Python 3.13.5 (tags/v3.13.5:6cb20a2, Jun 11 2025, 16:15:46) [MSC v.1943 64 bit (AMD64)] on win32
Enter "help" below or click "Help" above for more information.
>>>
===== RESTART: C:/Users/haasini/gemini_zero/shot.py =====
17 is a prime number.
25 is not a prime number.
>>>
```

1. Handles Small Numbers Quickly

python

Copy code

if $n \leq 1$:

 return False

if $n \leq 3$:

 return True

- Any number ≤ 1 is not prime — instantly returns False.
- 2 and 3 are prime — instantly returns True. Benefit: Saves time by avoiding unnecessary loops for small numbers.

2. Eliminates Even and Multiple-of-3 Numbers Early

python

Copy code

if $n \% 2 == 0$ or $n \% 3 == 0$:

 return False

- Checks divisibility by 2 and 3 right away.
- If a number is divisible by either, it's not prime. Benefit: This removes ~66% of numbers instantly without looping.

3. Uses $6k \pm 1$ Optimization

python

Copy code

```
i = 5
```

```
while i * i <= n:
```

```
    if n % i == 0 or n % (i + 2) == 0:
```

```
        return False
```

```
    i += 6
```

- Mathematical fact: All primes greater than 3 are of the form $6k \pm 1$.
- Instead of checking every number, it only checks numbers that *could possibly* be prime (skipping all even numbers and multiples of 3).
- Increments by 6 to jump over numbers that are definitely not prime.

Benefit: Reduces the number of checks from n down to roughly $n / 3$, and then further to only $\sim\sqrt{n}$ candidates.

4. Square Root Limit

python

Copy code

```
while i * i <= n:
```

- Only checks divisors up to \sqrt{n} .
- If a number has no divisor less than or equal to \sqrt{n} , it's prime.

Benefit: Turns the complexity from $O(n)$ to $O(\sqrt{n})$, which is much faster for large inputs.

5. Clear Structure & Example Usage

- Code is split into logical blocks (base cases, divisibility by 2 & 3, main loop).
- Example usage is included, so testing is quick.

TOOL: GOOGLE COLAB(GEMINI):

PROMPT: WRITE A PYTHON FUNCTION TO CHECK WHETHER A NUMBER IS PRIME OR NOT

ZERO-SHOT:

The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: <https://colab.research.google.com/drive/1D41fuQdKflpPnaSfrXCYLmjvgmwEHGF>. The notebook title is 'Untitled4.ipynb'. The menu bar includes File, Edit, View, Insert, Runtime, Tools, and Help. The toolbar shows a search icon, 'Commands', '+ Code', '+ Text', 'Run all', and a 'Connect' dropdown. On the left sidebar, there are icons for file explorer, search, expand/collapse, and a folder icon. The main code cell contains a Python function definition for 'is_prime' with a docstring and an example usage block. The status bar at the bottom shows 'Variables', 'Terminal', and a diamond icon.

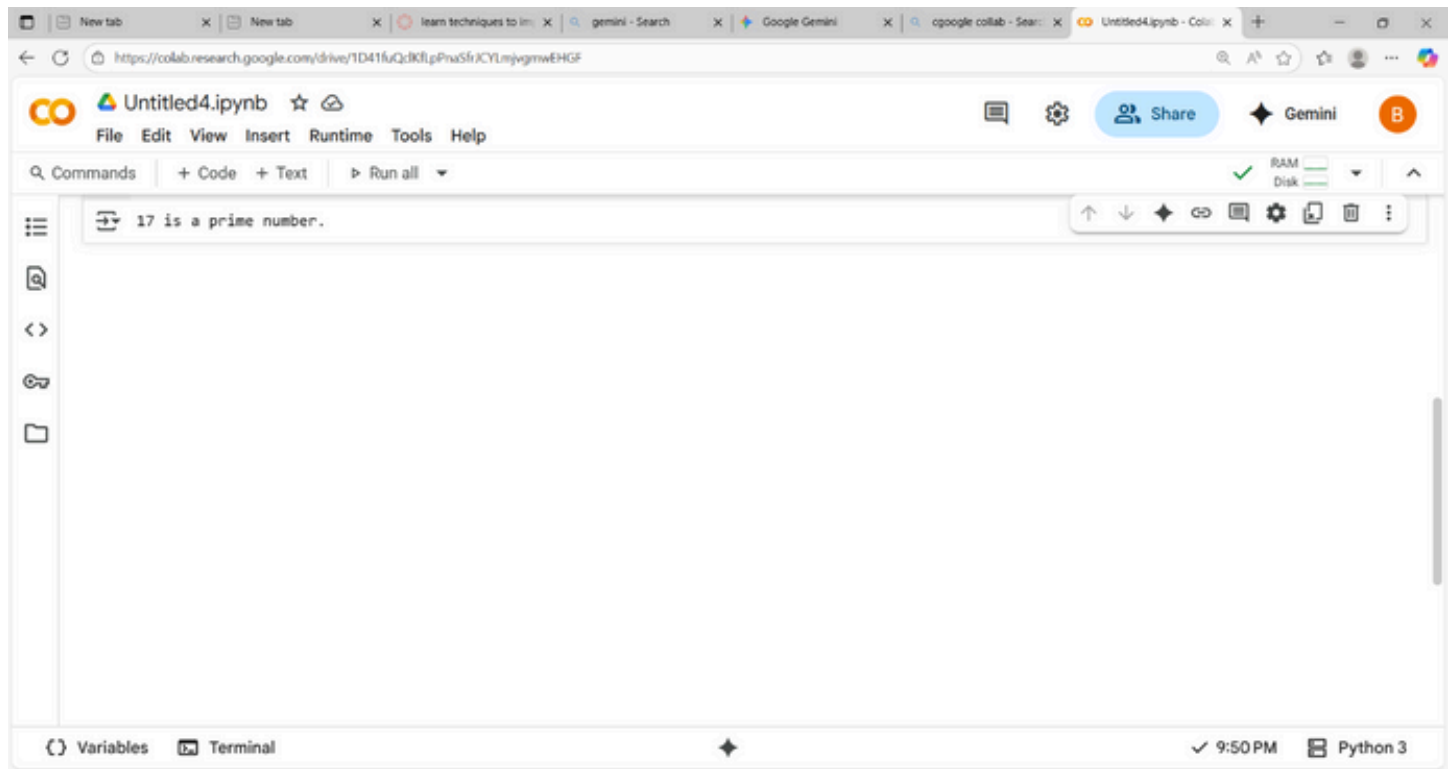
```
def is_prime(num):  
    """  
    Checks if a number is prime.  
  
    Args:  
        num: The number to check.  
  
    Returns:  
        True if the number is prime, False otherwise.  
    """  
    if num <= 1:  
        return False  
    for i in range(2, int(num**0.5) + 1):  
        if num % i == 0:  
            return False  
    return True  
  
# Example usage:  
number_to_check = 17  
if is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")
```

ONE-SHOT:

CODE:

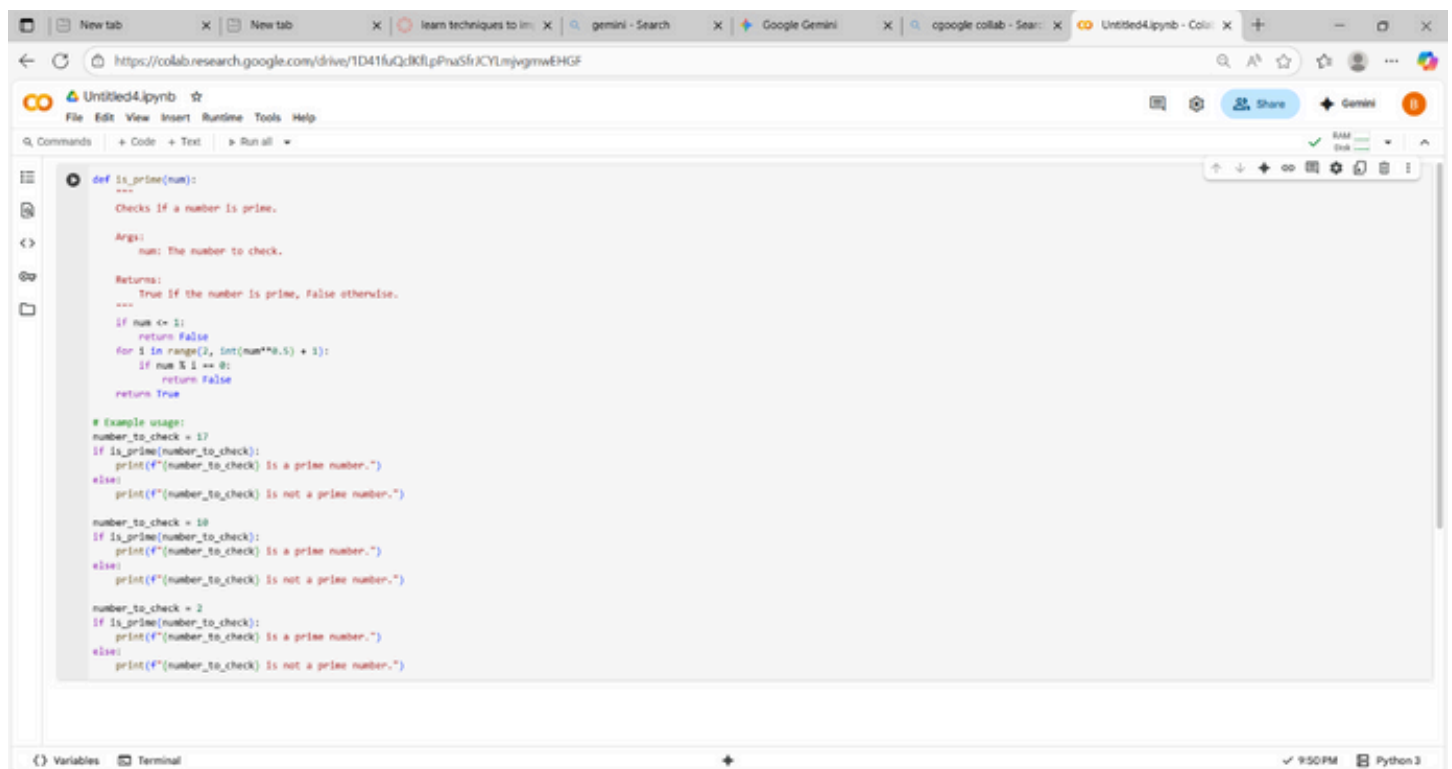
This screenshot is identical to the one above, showing the same Google Colab notebook interface and code. The status bar at the bottom now includes '9:49 PM' and 'Python 3' on the right side.

OUTPUT:

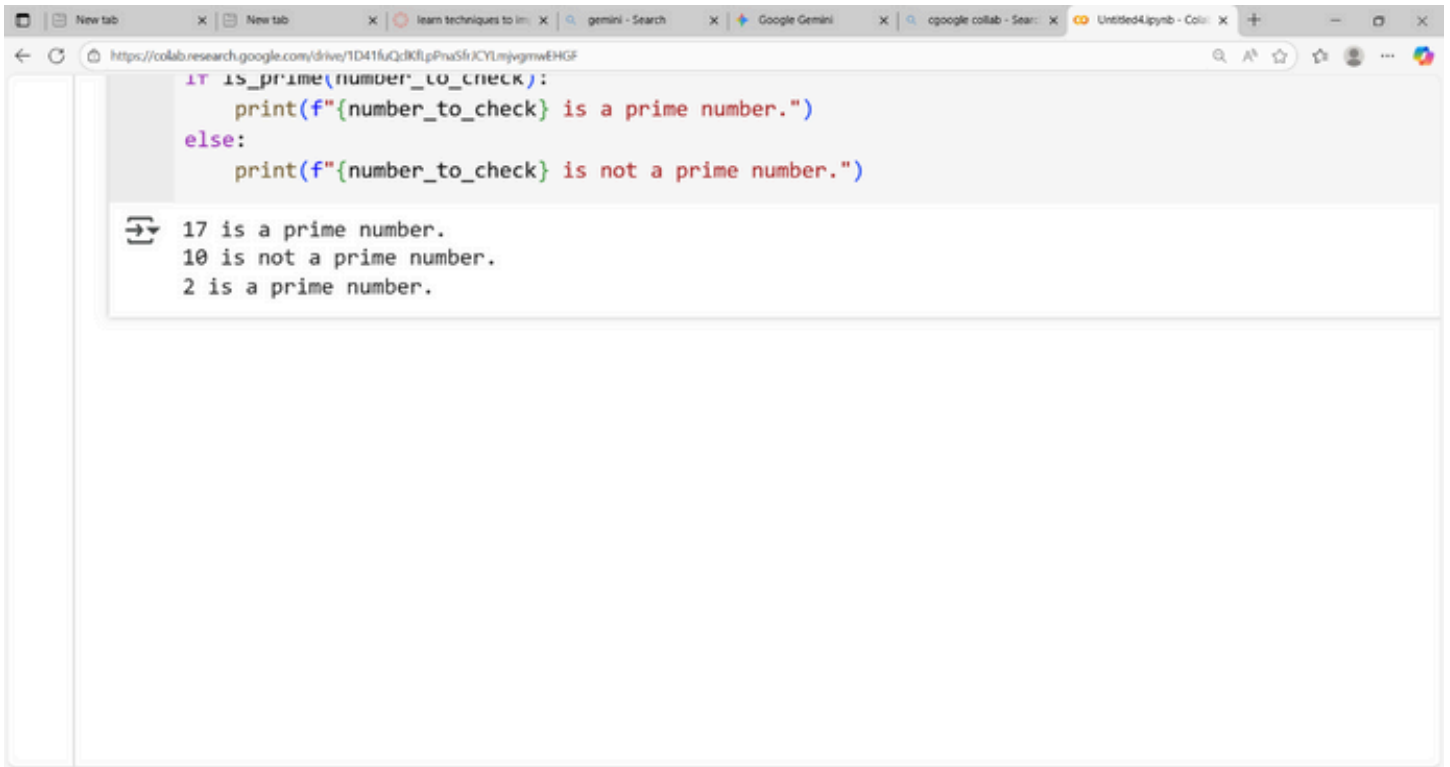


FEW-SHOT:

CODE:



OUTPUT:



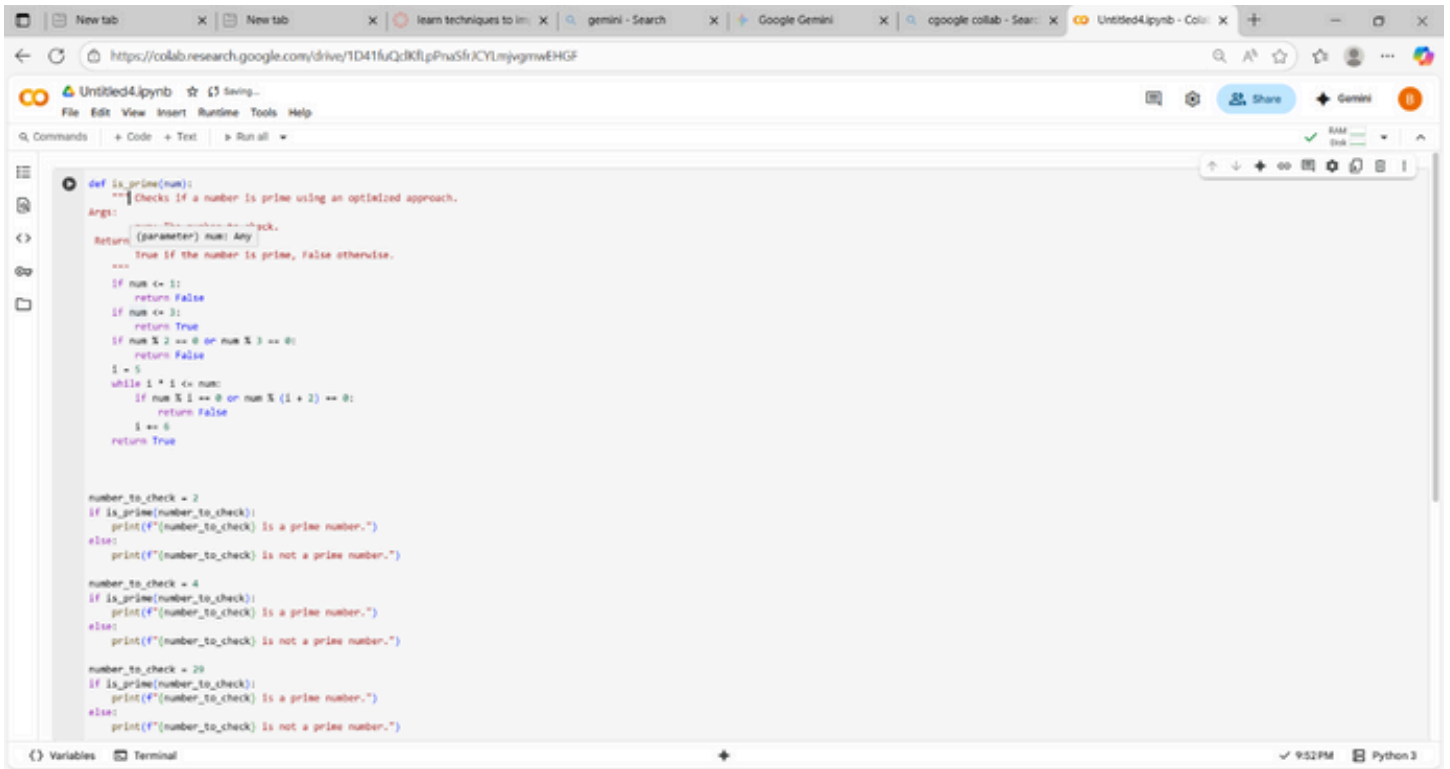
The screenshot shows a Google Colab notebook interface. The top toolbar includes buttons for 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main code cell contains a Python function `is_prime` and three test calls. The function `is_prime` takes a parameter `number_to_check` and prints whether it is a prime number. The test calls are `is_prime(17)`, `is_prime(10)`, and `is_prime(2)`. The output cell shows the results of these calls: '17 is a prime number.', '10 is not a prime number.', and '2 is a prime number.'

```
def is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")  
  
is_prime(17)  
is_prime(10)  
is_prime(2)
```

17 is a prime number.
10 is not a prime number.
2 is a prime number.

CONTEXT-MANAGED : MORE OPTIMIZED AND MANAGED CODE.

CODE:

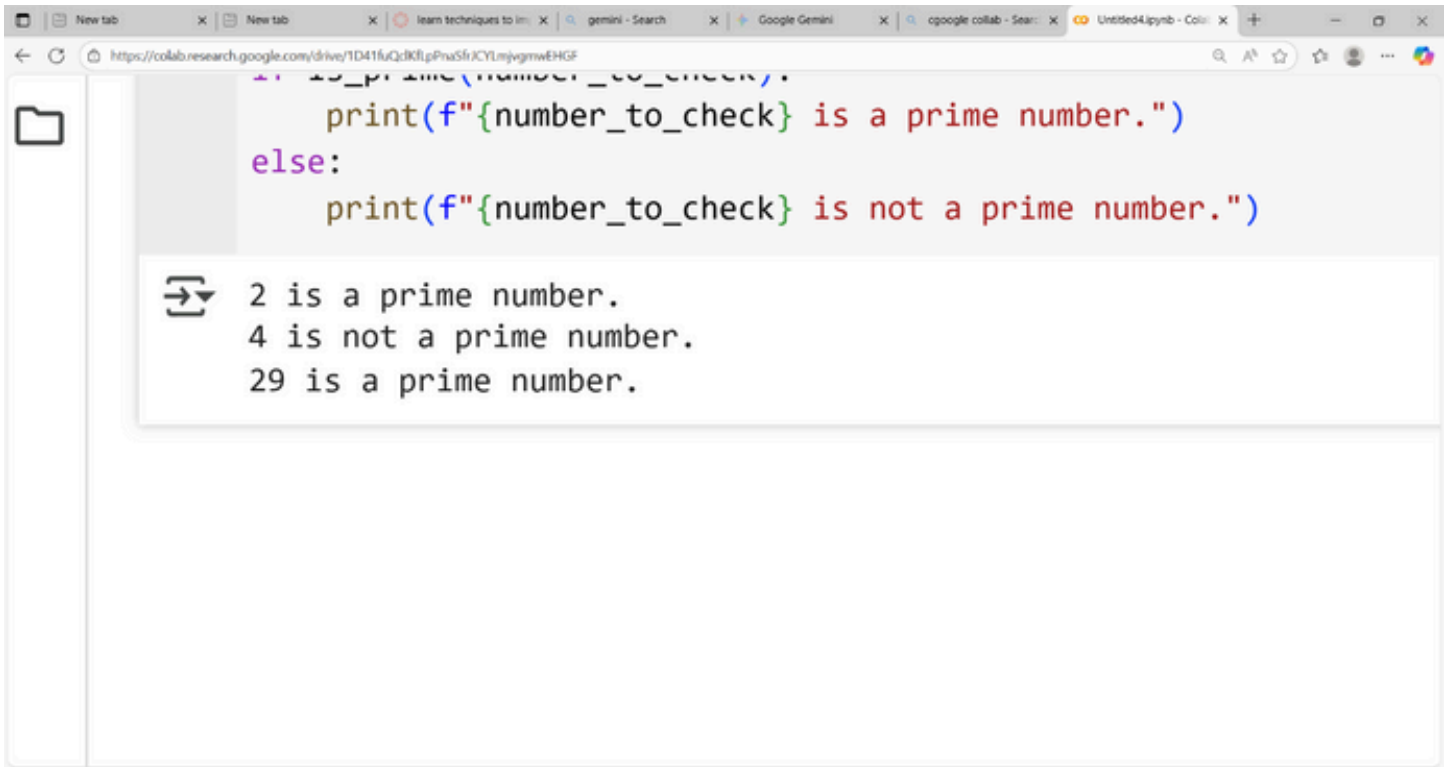


The screenshot shows a Google Colab notebook interface. The top toolbar includes buttons for 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The main code cell contains a Python function `is_prime` and three test calls. The function `is_prime` takes a parameter `num` and returns a boolean value. The test calls are `is_prime(2)`, `is_prime(4)`, and `is_prime(20)`. The output cell shows the results of these calls: '2 is a prime number.', '4 is not a prime number.', and '20 is not a prime number.'

```
def is_prime(num):  
    """Checks if a number is prime using an optimized approach.  
    Args:  
        (parameter) num: Any  
    Return:  
        True if the number is prime, False otherwise.  
    """  
    if num <= 1:  
        return False  
    if num <= 3:  
        return True  
    if num % 2 == 0 or num % 3 == 0:  
        return False  
    i = 5  
    while i * i <= num:  
        if num % i == 0 or num % (i + 2) == 0:  
            return False  
        i += 6  
    return True  
  
number_to_check = 2  
if is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")  
  
number_to_check = 4  
if is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")  
  
number_to_check = 20  
if is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")
```

2 is a prime number.
4 is not a prime number.
20 is not a prime number.

OUTPUT:



```
def is_prime(number_to_check):  
    print(f"{number_to_check} is a prime number.")  
else:  
    print(f"{number_to_check} is not a prime number.")
```

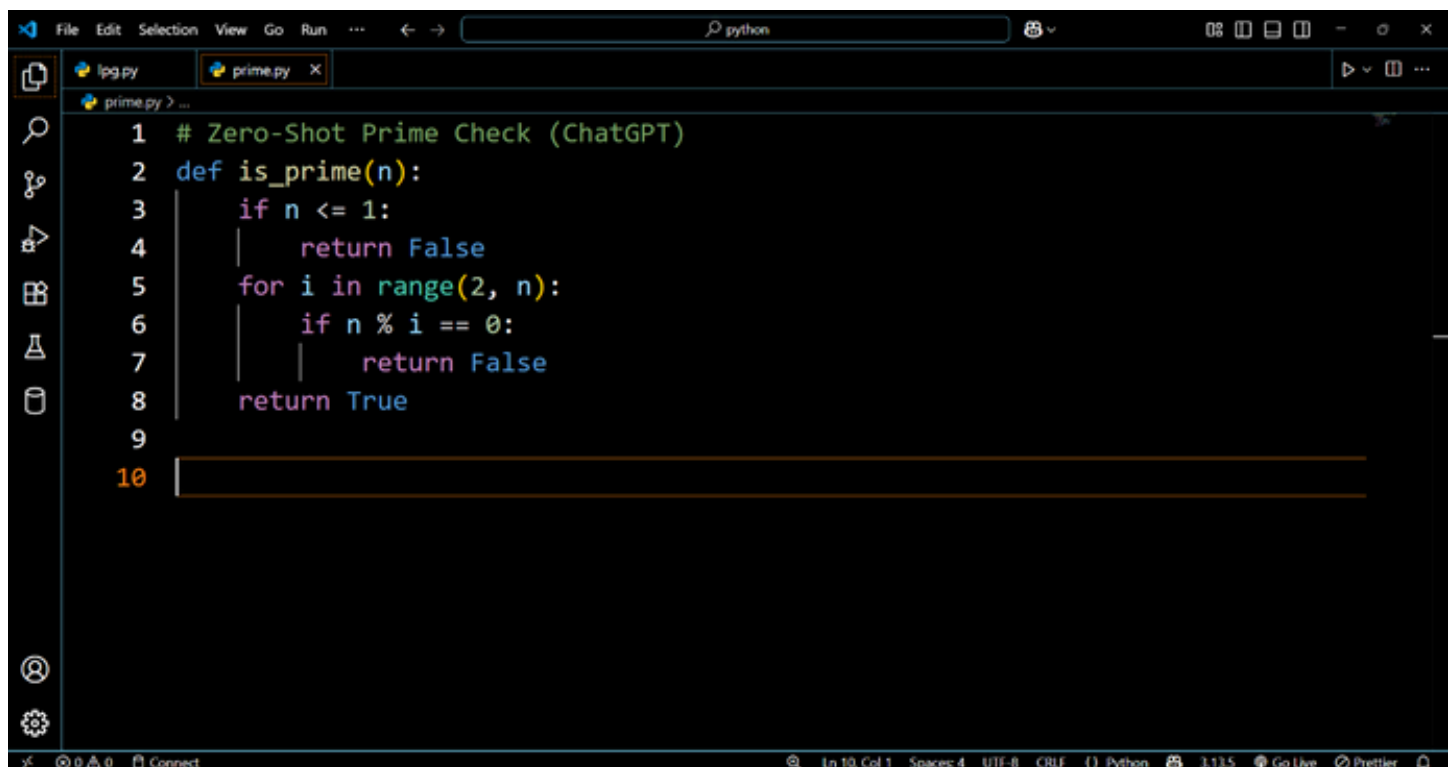
2 is a prime number.
4 is not a prime number.
29 is a prime number.

TOOL: CHATGPT

ZERO-SHOT:

PROMPT: WRITE A PYTHON FUNCTION TO CHECK A NUMBER PRIME OR NOT

CODE:

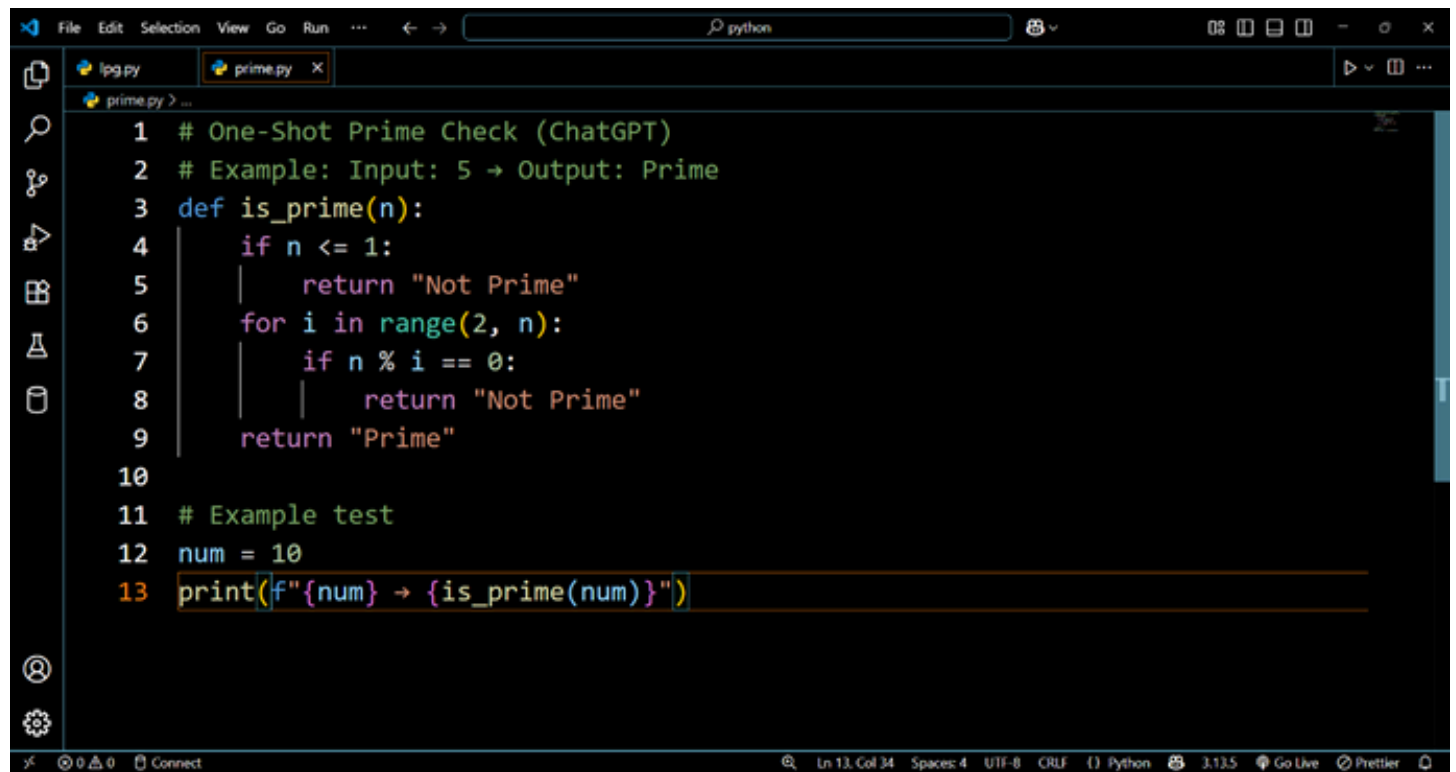


```
1 # Zero-Shot Prime Check (ChatGPT)  
2 def is_prime(n):  
3     if n <= 1:  
4         return False  
5     for i in range(2, n):  
6         if n % i == 0:  
7             return False  
8     return True  
9  
10
```

ONE SHOT

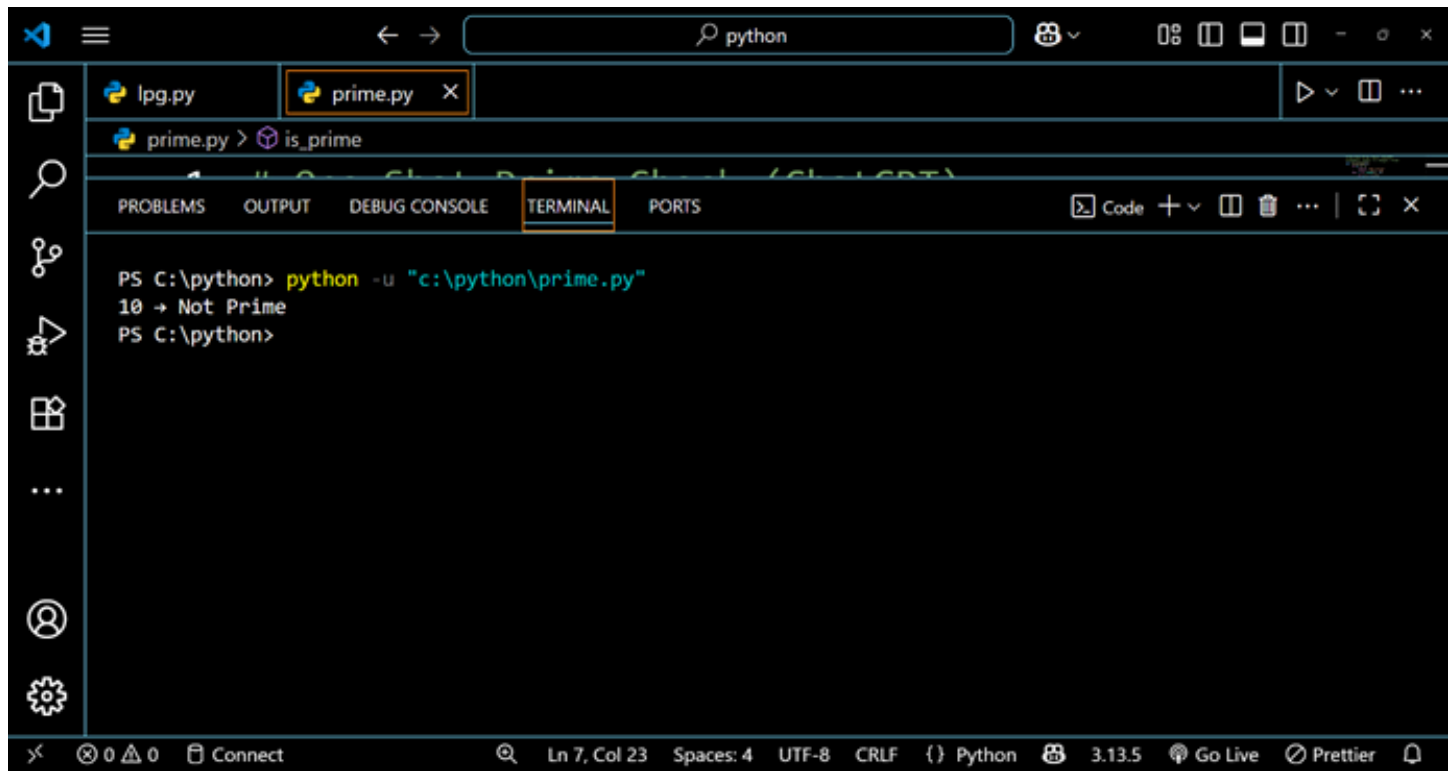
Prompt: give python function to check whether a number is prime or not prime (one-shot prime check one example usage)

CODE:

A screenshot of a Visual Studio Code editor window. The title bar shows 'python'. The editor has two tabs: 'lpg.py' and 'prime.py'. The 'prime.py' tab is active, showing a Python script. The script includes comments for 'One-Shot Prime Check (ChatGPT)' and an example input/output. It defines a function 'is_prime(n)' that checks for primality using a loop from 2 to n-1. Finally, it tests the function with 'num = 10' and prints the result in a formatted string.

```
1 # One-Shot Prime Check (ChatGPT)
2 # Example: Input: 5 → Output: Prime
3 def is_prime(n):
4     if n <= 1:
5         return "Not Prime"
6     for i in range(2, n):
7         if n % i == 0:
8             return "Not Prime"
9     return "Prime"
10
11 # Example test
12 num = 10
13 print(f"{num} → {is_prime(num)}")
```

OUTPUT:



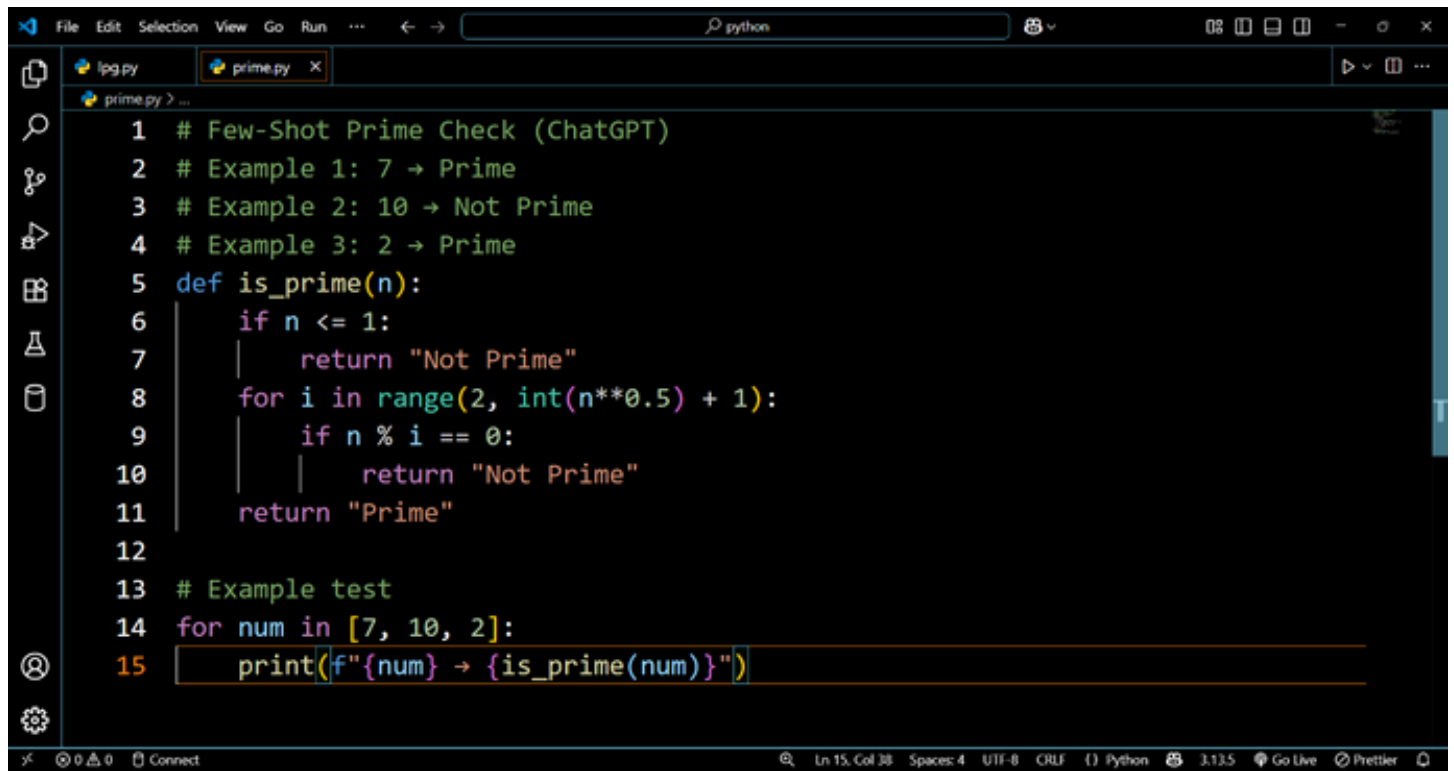
The image shows a Visual Studio Code (VS Code) interface with a terminal window open. The terminal is running a Python script named `prime.py`. The prompt is `PS C:\python>`. The command entered is `python -u "c:\python\prime.py"`. The output of the script is `10 → Not Prime`. The terminal window is titled `prime.py` and has a search bar with the text `python`. The terminal window is also titled `prime.py` and has a search bar with the text `python`. The terminal window is also titled `prime.py` and has a search bar with the text `python`.

```
PS C:\python> python -u "c:\python\prime.py"
10 → Not Prime
PS C:\python>
```

FEW-SHOT:

Prompt: give python function to check whether a number is prime or not prime (few-shot prime check more example usages)

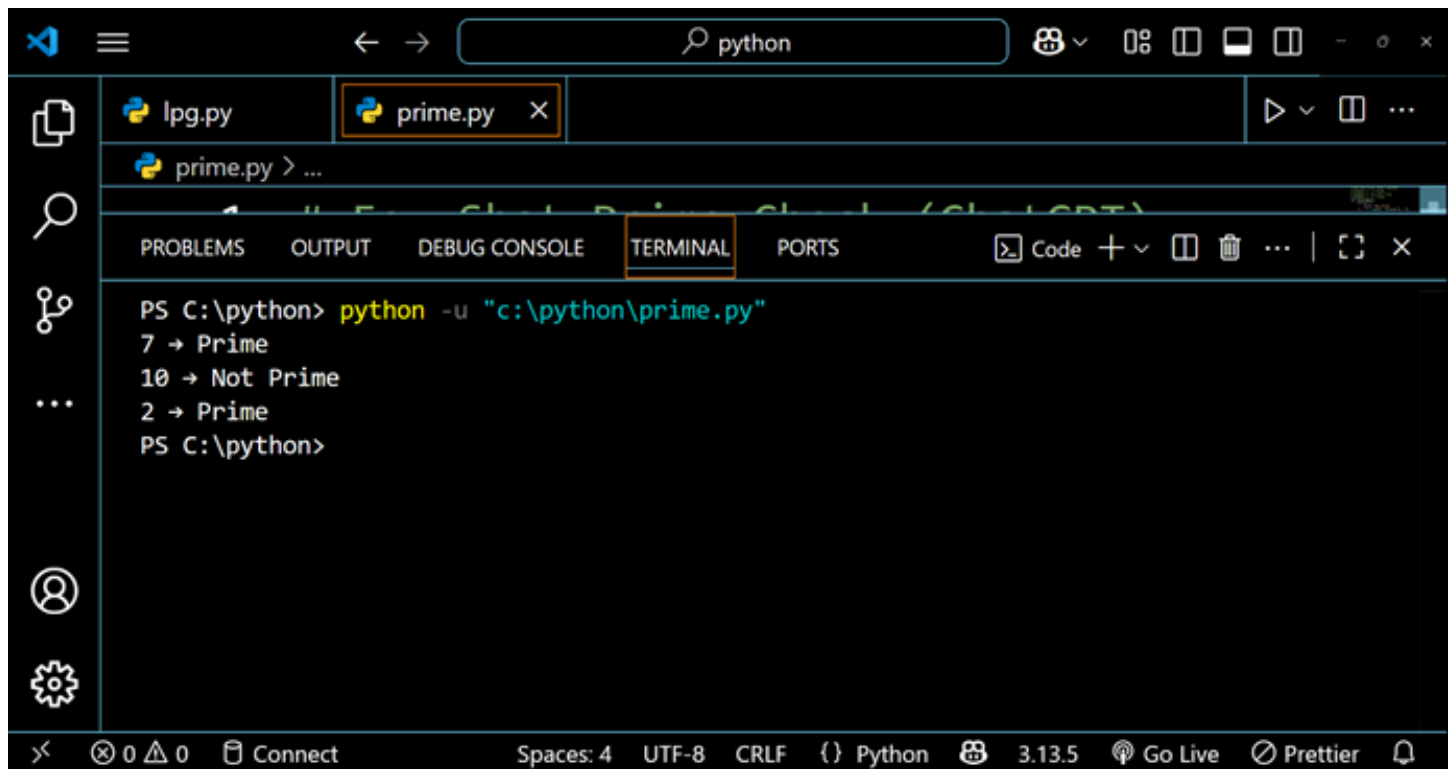
CODE:



The screenshot shows a Visual Studio Code editor window with a file named `prime.py` open. The code is a Python script for a prime check, using a few-shot prompt style. It includes comments for examples and a function `is_prime(n)` that checks if a number is prime. The script also includes a test loop for the numbers 7, 10, and 2.

```
1 # Few-Shot Prime Check (ChatGPT)
2 # Example 1: 7 → Prime
3 # Example 2: 10 → Not Prime
4 # Example 3: 2 → Prime
5 def is_prime(n):
6     if n <= 1:
7         return "Not Prime"
8     for i in range(2, int(n**0.5) + 1):
9         if n % i == 0:
10            return "Not Prime"
11    return "Prime"
12
13 # Example test
14 for num in [7, 10, 2]:
15     print(f"{num} → {is_prime(num)}")
```

OUTPUT:

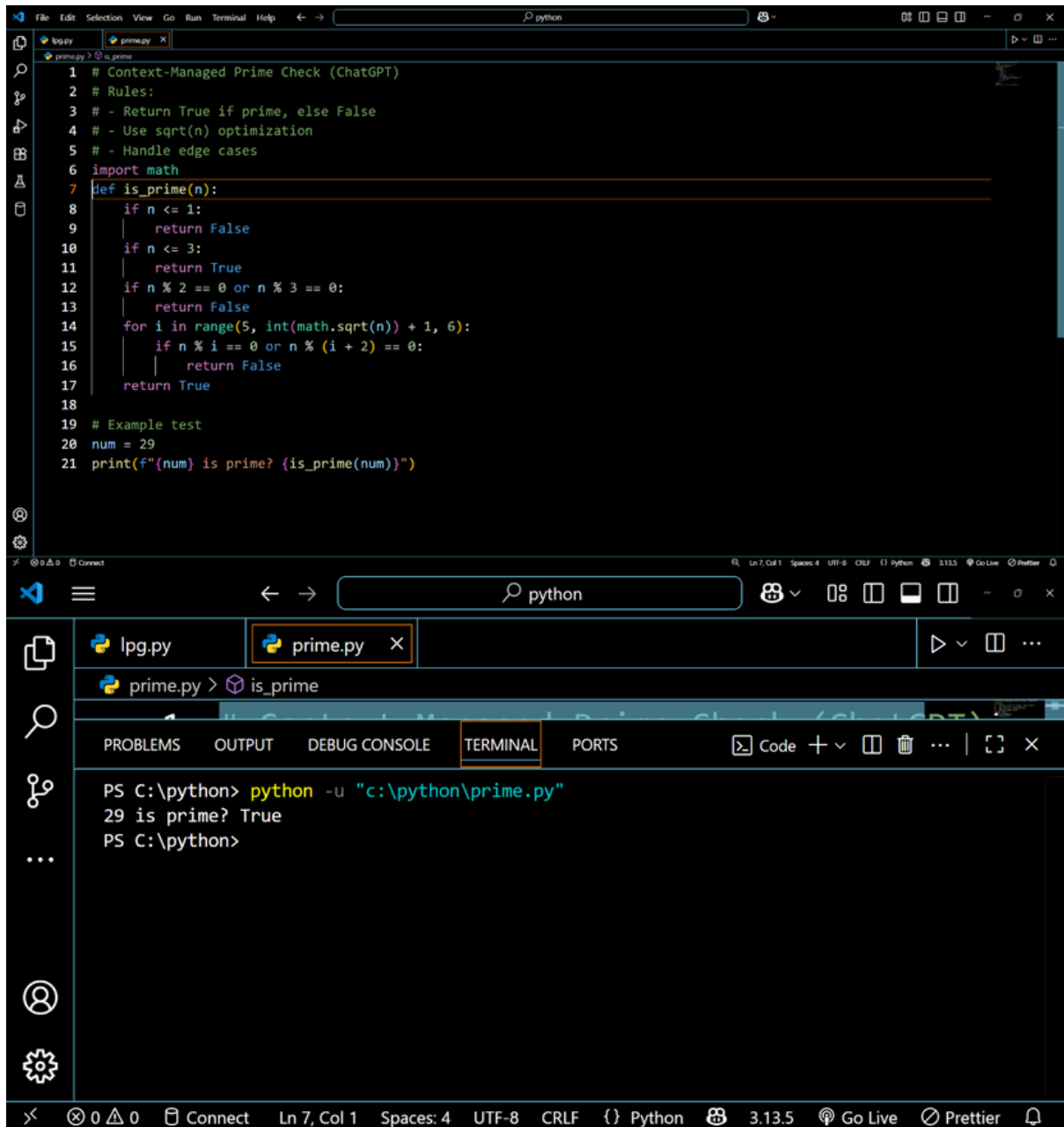


The screenshot shows the same Visual Studio Code editor window, but with the `TERMINAL` tab selected. The terminal displays the output of running the `prime.py` script. The output shows the results for the numbers 7, 10, and 2, which are 7 → Prime, 10 → Not Prime, and 2 → Prime.

```
PS C:\python> python -u "c:\python\prime.py"
7 → Prime
10 → Not Prime
2 → Prime
PS C:\python>
```

CONTEXT-MANAGED CODE WITH OUTPUT

Prompt: give python function to check whether a number is prime or not prime (one-shot prime check one or more example usage) but in optimized and efficient way



```
1 # Context-Managed Prime Check (ChatGPT)
2 # Rules:
3 # - Return True if prime, else False
4 # - Use sqrt(n) optimization
5 # - Handle edge cases
6 import math
7 def is_prime(n):
8     if n <= 1:
9         return False
10    if n <= 3:
11        return True
12    if n % 2 == 0 or n % 3 == 0:
13        return False
14    for i in range(5, int(math.sqrt(n)) + 1, 6):
15        if n % i == 0 or n % (i + 2) == 0:
16            return False
17    return True
18
19 # Example test
20 num = 29
21 print(f"{num} is prime? {is_prime(num)}")
```

python

prime.py

prime.py > is_prime

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

PS C:\python> python -u "c:\python\prime.py"

29 is prime? True

PS C:\python>

Ln 7, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.5 Go Live Prettier

OPTIMIZED CODE WITH OUTPUT:

```
1 # Final Optimized Prime Check (ChatGPT Refined)
2 def is_prime(n):
3     """
4     Check if a number is prime using optimized 6k ± 1 method.
5     Returns True if prime, False otherwise.
6     """
7     if not isinstance(n, int) or n < 2:
8         return False
9     if n in (2, 3):
10        return True
11    if n % 2 == 0 or n % 3 == 0:
12        return False
13    i = 5
14    while i * i <= n:
15        if n % i == 0 or n % (i + 2) == 0:
16            return False
17        i += 6
18    return True
19
20 # Example test
21 for num in [2, 3, 5, 10, 29, 97]:
22     print(f"{num} is prime? {is_prime(num)}")
```

```
PS C:\python> python -u "c:\python\prime.py"
2 is prime? True
3 is prime? True
5 is prime? True
10 is prime? False
29 is prime? True
97 is prime? True
PS C:\python>
```

TOOL: VS CODE

PROMPT: WRITE A PYTHON FUCNTION TO CHECK WHETHER A NUMBER IS PRIME OR NOT

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13
```

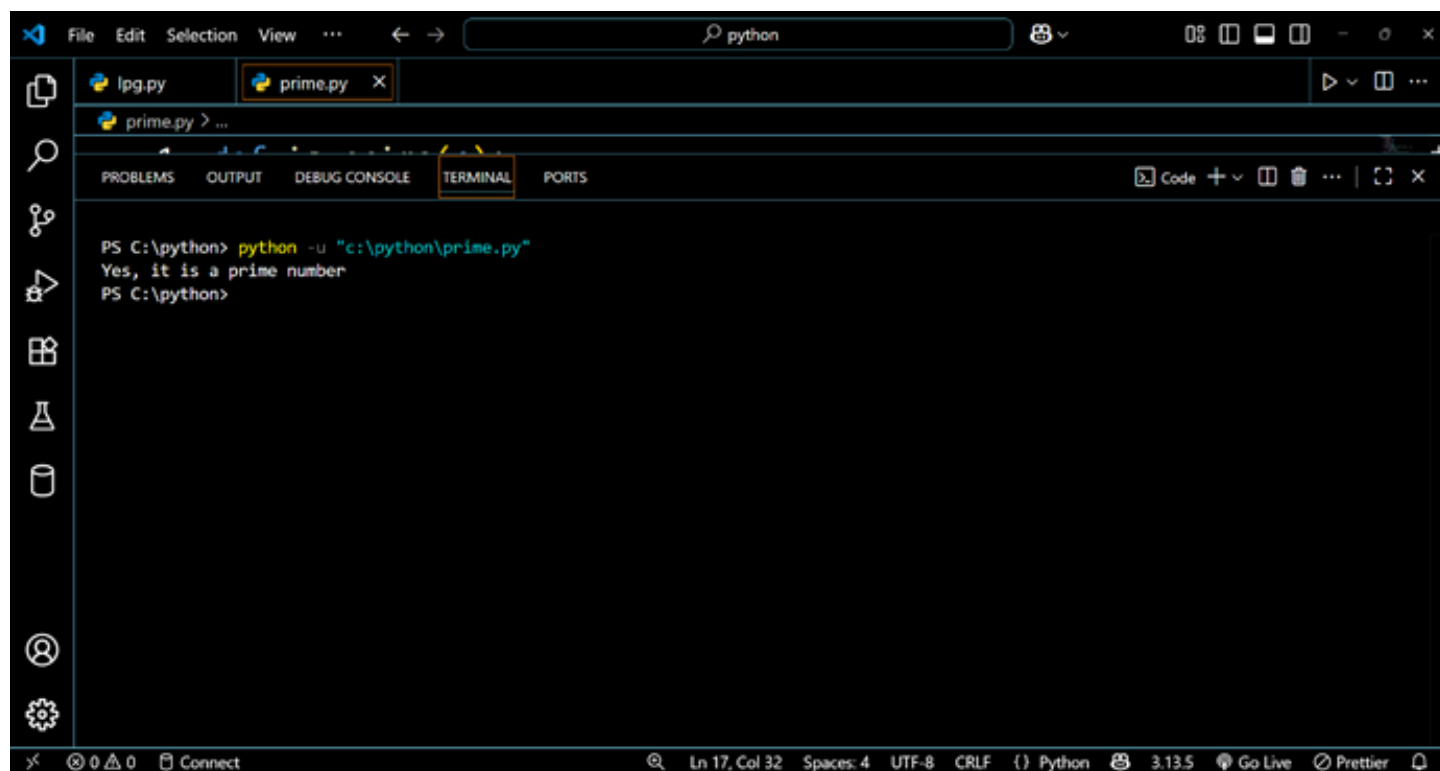
ONE-SHOT:

GIVE CODE WITH ONE EXAMPLE USAGE.

CODE:

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True # now outside the loop
12
13 # test
14 if is_prime(17):
15     print("Yes, it is a prime number")
16 else:
17     print("Not a prime number")
```

OUTPUT:



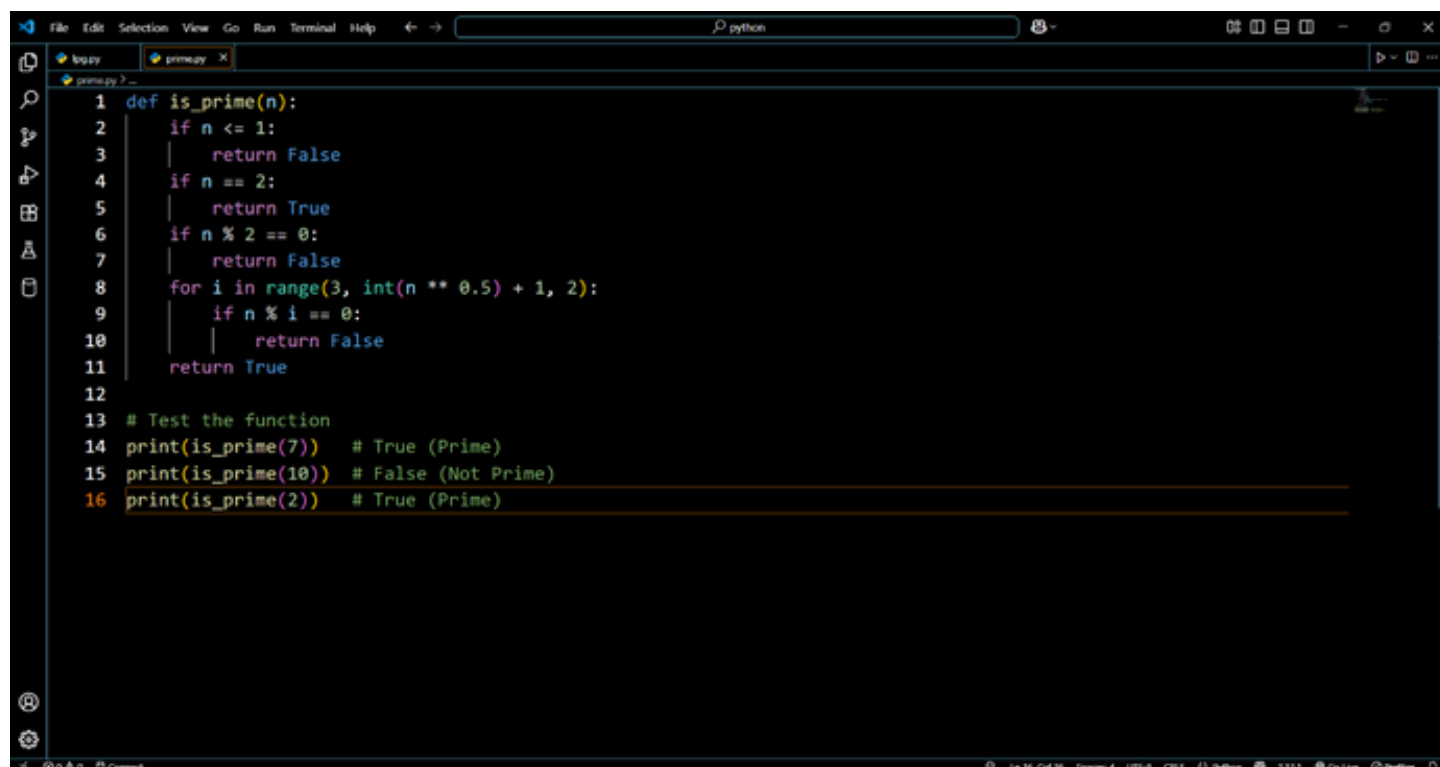
The screenshot shows a VS Code window with a terminal pane at the bottom. The terminal shows the command `python -u "c:\python\prime.py"` being executed in a Windows command prompt. The output of the script is "Yes, it is a prime number". The terminal pane has tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS. The TERMINAL tab is selected. The status bar at the bottom shows the current cursor position as Ln 17, Col 32, and other settings like Spaces: 4, UTF-8, CRLF, Python 3.13.5, Go Live, and Prettier.

```
PS C:\python> python -u "c:\python\prime.py"
Yes, it is a prime number
PS C:\python>
```

FEW-SHOT:

PROMPT: GIVE A PYTHON CODE WITH MORE EXAMPLE USAGES.

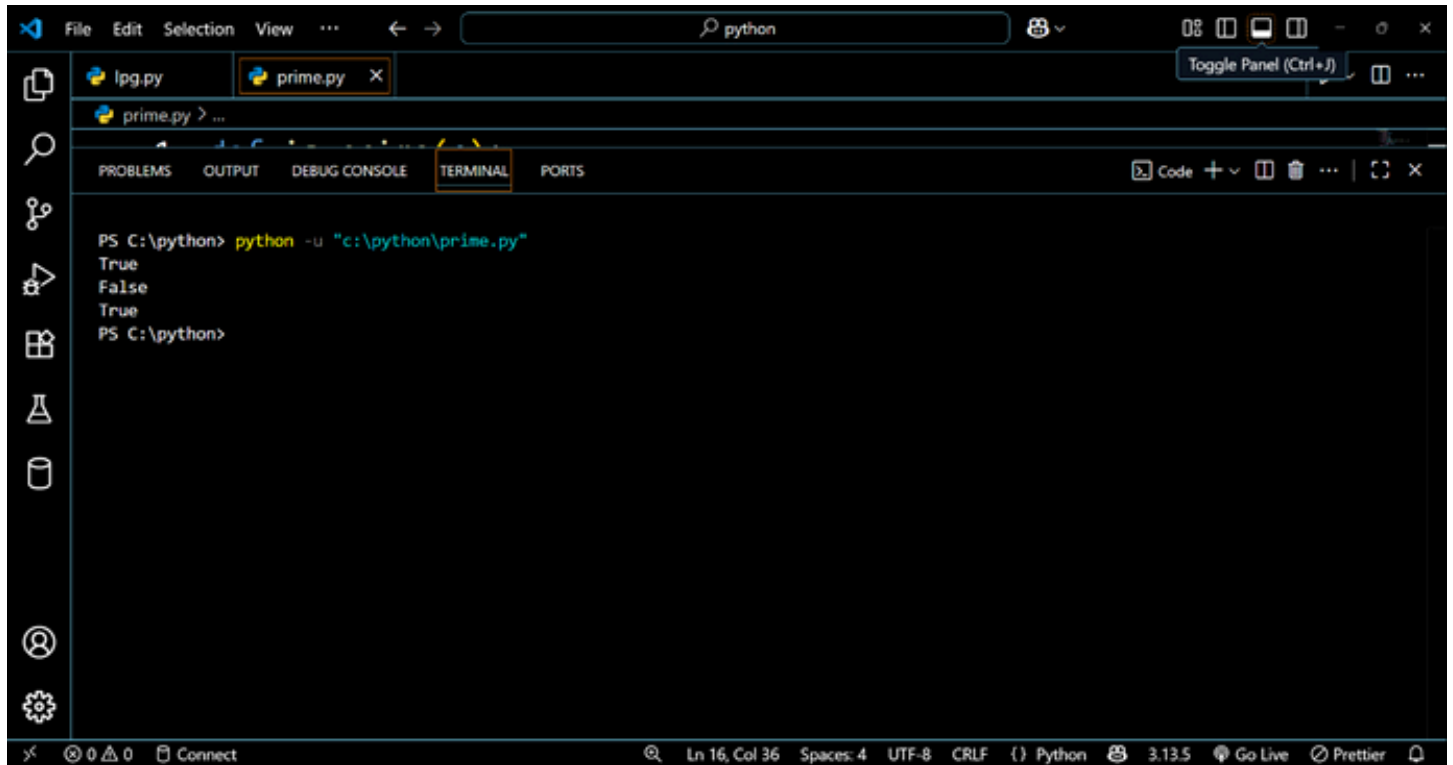
CODE:



The screenshot shows a VS Code editor window with a Python script named `prime.py`. The script defines a function `is_prime(n)` that checks if a number is prime. It includes test cases for 7, 10, and 2. The status bar at the bottom shows the current cursor position as Ln 16, Col 36, and other settings like Spaces: 4, UTF-8, CRLF, Python 3.13.5, Go Live, and Prettier.

```
1 def is_prime(n):
2     if n <= 1:
3         return False
4     if n == 2:
5         return True
6     if n % 2 == 0:
7         return False
8     for i in range(3, int(n ** 0.5) + 1, 2):
9         if n % i == 0:
10            return False
11    return True
12
13 # Test the function
14 print(is_prime(7)) # True (Prime)
15 print(is_prime(10)) # False (Not Prime)
16 print(is_prime(2)) # True (Prime)
```

OUTPUT:

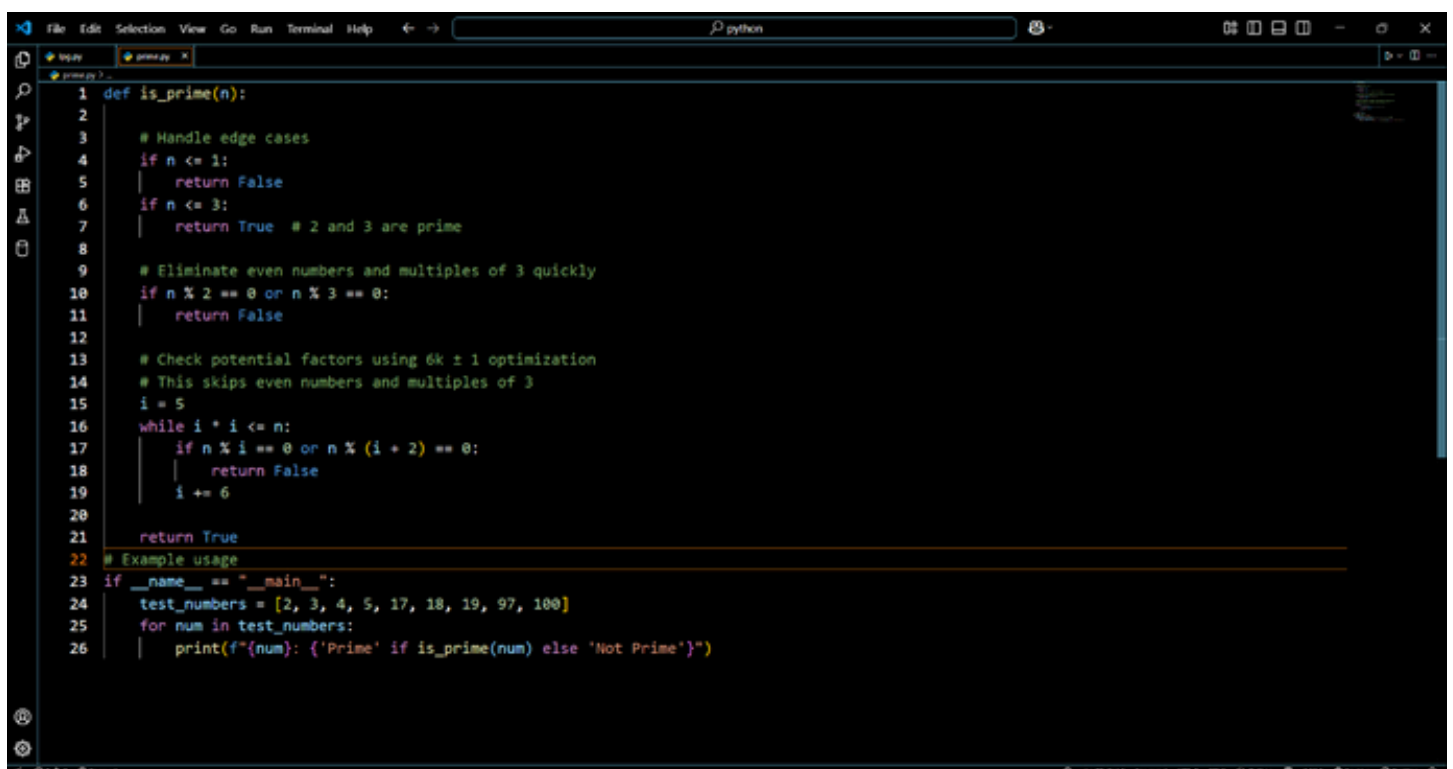


The screenshot shows a VS Code window with a terminal panel open. The terminal displays the output of running a Python script named `prime.py`. The prompt is `PS C:\python>`. The command entered is `python -u "c:\python\prime.py"`. The output shows the results of the script's execution, which are `True`, `False`, and `True` on separate lines. The terminal panel is titled `TERMINAL` and has a search bar at the top. The status bar at the bottom indicates the file is `Ln 16, Col 36`, with `Spaces: 4`, `UTF-8`, `CRLF`, and `Python` 3.13.5.

```
PS C:\python> python -u "c:\python\prime.py"
True
False
True
PS C:\python>
```

Context-managed: prompt: give python function code to check whether a number is prime or not more optimized and efficient logic


Code:



The screenshot shows the source code of the `prime.py` script in a VS Code editor. The code defines a function `is_prime(n)` that checks if a number is prime. It handles edge cases for `n <= 1` and `n <= 3`. For `n > 3`, it eliminates even numbers and multiples of 3 quickly, then checks potential factors using the $6k \pm 1$ optimization. The code also includes an example usage section that tests the function on a list of numbers: `[2, 3, 4, 5, 17, 18, 19, 97, 100]`. The status bar at the bottom indicates the file is `Ln 32, Col 1`, with `Spaces: 4`, `UTF-8`, `CRLF`, and `Python` 3.13.5.

```
1 def is_prime(n):
2
3     # Handle edge cases
4     if n <= 1:
5         return False
6     if n <= 3:
7         return True # 2 and 3 are prime
8
9     # Eliminate even numbers and multiples of 3 quickly
10    if n % 2 == 0 or n % 3 == 0:
11        return False
12
13    # Check potential factors using 6k ± 1 optimization
14    # This skips even numbers and multiples of 3
15    i = 5
16    while i * i <= n:
17        if n % i == 0 or n % (i + 2) == 0:
18            return False
19        i += 6
20
21    return True
22
23 # Example usage
24 if __name__ == "__main__":
25     test_numbers = [2, 3, 4, 5, 17, 18, 19, 97, 100]
26     for num in test_numbers:
27         print(f"{num}: {'Prime' if is_prime(num) else 'Not Prime'}")
```

Output:



The screenshot shows a Visual Studio Code editor window with a file named `prime.py` open. The code defines a function `is_prime(n)` that checks for prime numbers. The function includes a comment `# Handle edge cases` and a conditional `if n <= 1:`. Below the code editor, the `TERMINAL` tab is active, showing the command `python -u "c:\python\prime.py"` and its output. The output lists numbers from 2 to 100, indicating whether each is a prime or not. The status bar at the bottom shows the file is at line 22, column 1, with 4 spaces, using UTF-8 encoding and CRLF line endings.

```
1 def is_prime(n):
2
3     # Handle edge cases
4     if n <= 1:
```

```
PS C:\python> python -u "c:\python\prime.py"
2: Prime
3: Prime
4: Not Prime
5: Prime
17: Prime
18: Not Prime
19: Prime
97: Prime
100: Not Prime
PS C:\python>
```

Conclusion:

- Zero-Shot prompts produced functional code but often lacked efficiency or edge-case handling.
- One-Shot prompts improved accuracy slightly by guiding the AI with one example, but still left room for optimization.
- Few-Shot prompts gave the AI a better understanding of the expected output pattern, resulting in more consistent and correct solutions.
- Context-Managed prompts produced the most optimized and readable code, especially when combined with explicit constraints (e.g., efficiency requirements, comments, and docstrings).

The exercise demonstrated that providing clear instructions, relevant examples, and specific constraints allows AI tools like ChatGPT, Gemini, or Copilot to generate higher-quality, more maintainable code. Prompt engineering is therefore a critical skill for effective AI-assisted programming.

TASK 2:

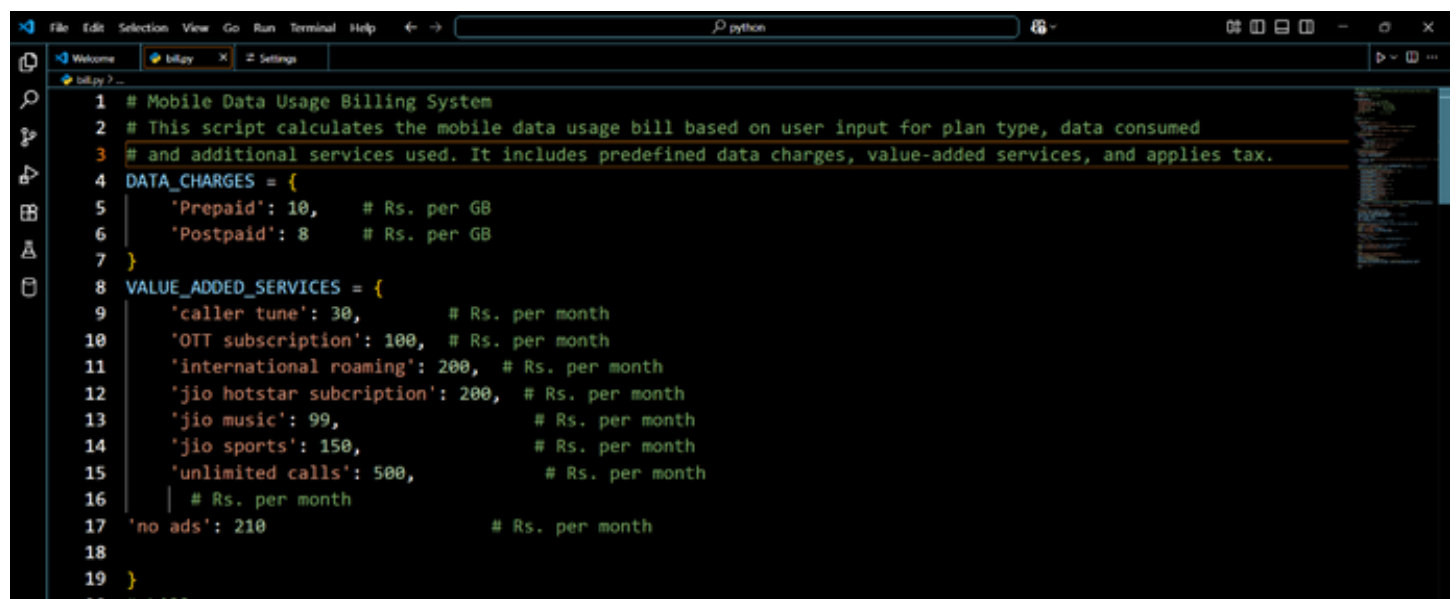
Mobile Data Usage Billing Application

PROMPT: GENERATE A PYTHON CODE FOR CALCULATING DATA USAGE BILLING APPLICATION

Display an itemized bill showing:

- Plan Type
- Data Usage and Charges
- Value-added Services and Charges
- Tax
- Total Bill Amount

CODE:



```
1 # Mobile Data Usage Billing System
2 # This script calculates the mobile data usage bill based on user input for plan type, data consumed
3 # and additional services used. It includes predefined data charges, value-added services, and applies tax.
4 DATA_CHARGES = {
5     'Prepaid': 10,      # Rs. per GB
6     'Postpaid': 8       # Rs. per GB
7 }
8 VALUE_ADDED_SERVICES = {
9     'caller tune': 30,      # Rs. per month
10    'OTT subscription': 100, # Rs. per month
11    'international roaming': 200, # Rs. per month
12    'jio hotstar subscription': 200, # Rs. per month
13    'jio music': 99,         # Rs. per month
14    'jio sports': 150,        # Rs. per month
15    'unlimited calls': 500,    # Rs. per month
16    # Rs. per month
17    'no ads': 210            # Rs. per month
18 }
19 }
```



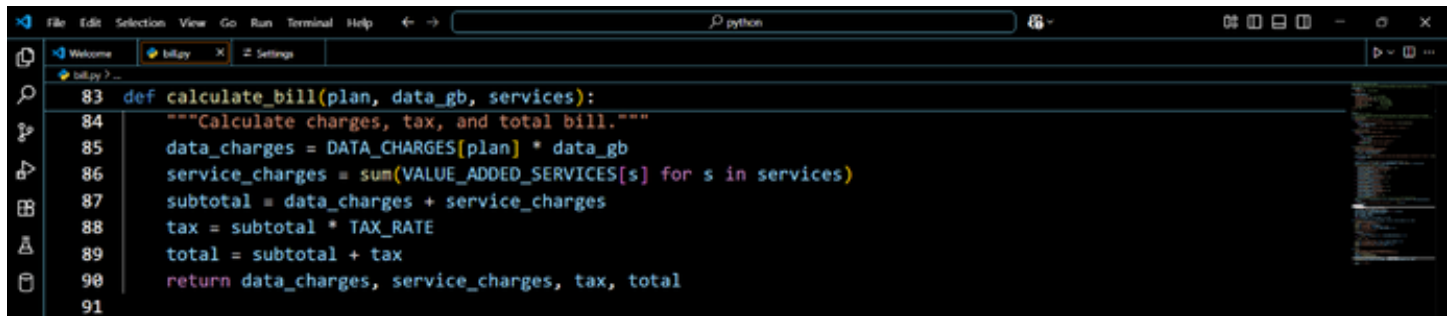
```
File Edit Selection View Go Run Terminal Help python
Welcome billpy Settings
billpy > ...
20 # bill.py
21 TAX_RATE = 0.18 # 18% GST
22 # This script calculates the mobile data usage bill based on user input for plan type, data consumed
23 # and additional services used. It includes predefined data charges, value-added services, and applies tax.
24 def get_plan_type():
25     """Ask user for plan type and validate."""
26     while True:
27         plan = input("Enter Plan Type (Prepaid/Postpaid): ").strip().capitalize()
28         if plan in DATA_CHARGES:
29             return plan
30         print("X Invalid plan type. Please enter 'Prepaid' or 'Postpaid'.")
31
32 def get_data_consumed():
33     """Ask user for data consumed in GB."""
34     while True:
35         try:
36             data = float(input("Enter Data Consumed (in GB): "))
37             if data >= 0:
38                 return data
39             print("X Data consumed cannot be negative.")
40         except ValueError:
41             print("X Please enter a valid number.")
42
```

PROMPT: ADD ADDITIONAL SERVICES LIKE JIO HOTSTAR, JIO SPORTS, JIO MUSIC, UNLIMITED DATA, UNLIMITED SMS AND UNLIMITED CALLS.

```
File Edit Selection View Go Run Terminal Help python
Welcome billpy Settings
billpy > ...
43 def get_additional_services():
44     """Ask user for additional services used."""
45     print("\nAvailable Additional Services:")
46     for service in VALUE_ADDED_SERVICES:
47         print(f"- {service.title()}")
48
49     services_input = input("Enter additional services used (comma separated, or press Enter for none): ").strip()
50     if not services_input:
51         return []
52
53     services_list = [s.strip().lower() for s in services_input.split(',')]
54     selected = [service for service in VALUE_ADDED_SERVICES if service.lower() in services_list]
55     # Add custom services requested by user
56     if "jio hotstar subscription" in services_list:
57         VALUE_ADDED_SERVICES["jio hotstar subscription"] = 200
58         selected.append("jio hotstar subscription")
59     if "jio music" in services_list:
60         VALUE_ADDED_SERVICES["jio music"] = 99
61         selected.append("jio music")
62     if "unlimited calls" in services_list:
63         VALUE_ADDED_SERVICES["unlimited calls"] = 50
64         selected.append("unlimited calls")
65
66     if "jio sports" in services_list:
67         VALUE_ADDED_SERVICES["jio sports"] = 150
68         selected.append("jio sports")
69     if "unlimited sms" in services_list:
70         VALUE_ADDED_SERVICES["unlimited sms"] = 20
71         selected.append("unlimited sms")
72     if "unlimited data" in services_list:
73         VALUE_ADDED_SERVICES["unlimited data"] = 50
74         selected.append("unlimited data")
75     if "no ads" in services_list:
76         VALUE_ADDED_SERVICES["no ads"] = 110
77         selected.append("no ads")
78     # Add any extra services entered by the user that are not in VALUE_ADDED_SERVICES
79     extra = [s for s in services_list if s not in [service.lower() for service in VALUE_ADDED_SERVICES]]
80     if extra:
81         print("Extra services provided (not billed):", ", ".join(extra))
82     return selected
```

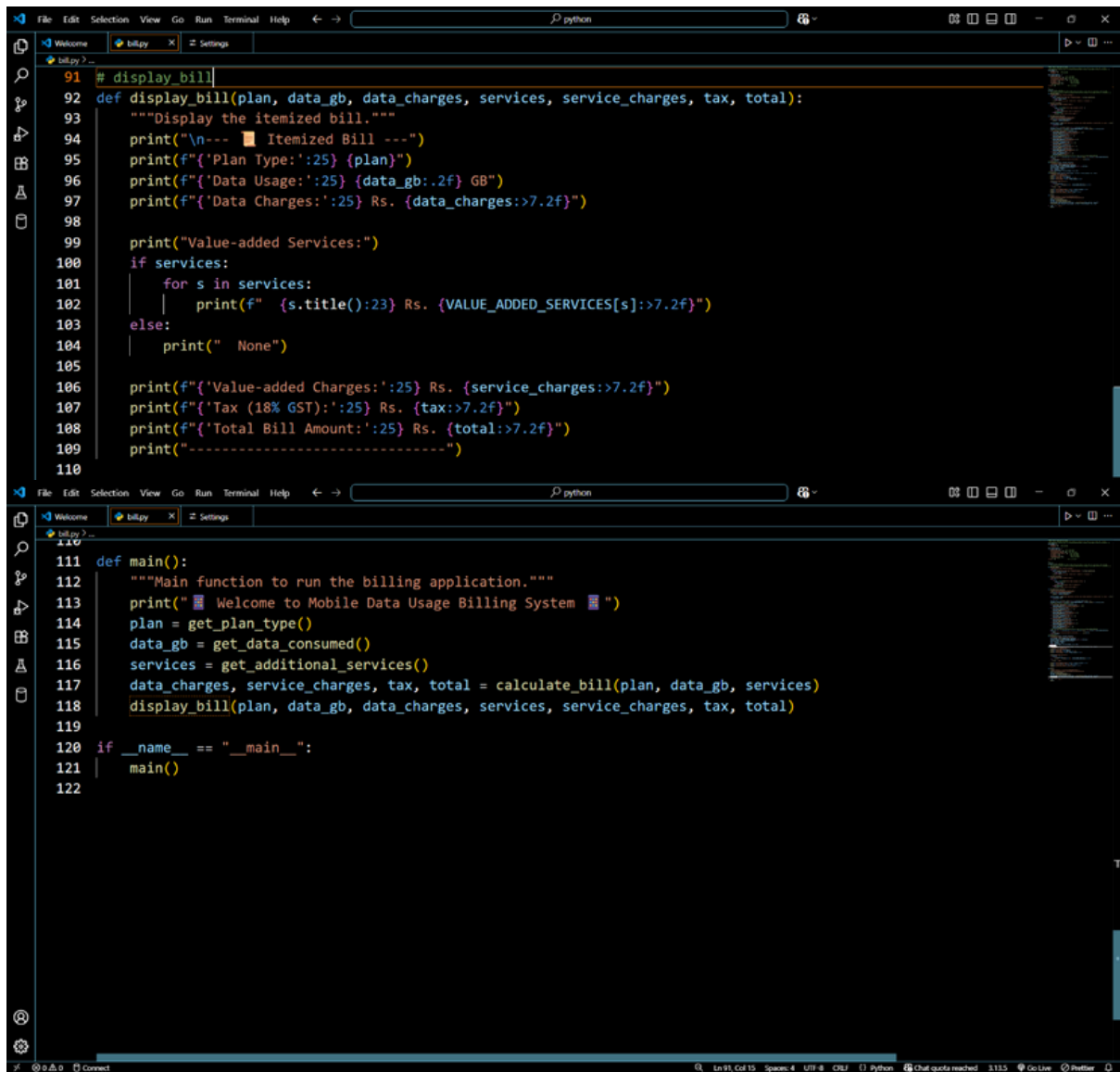
PROMPT: Implement billing logic to calculate:

- DC (Data Charges) – charges based on data consumption
- VC (Value-added Charges) – charges for additional services
- Tax – applicable tax on the total bill

A screenshot of a code editor window with a dark theme. The editor shows a Python function named 'calculate_bill' with three parameters: 'plan', 'data_gb', and 'services'. The function calculates data charges, service charges, subtotal, tax, and total bill. The code is as follows:

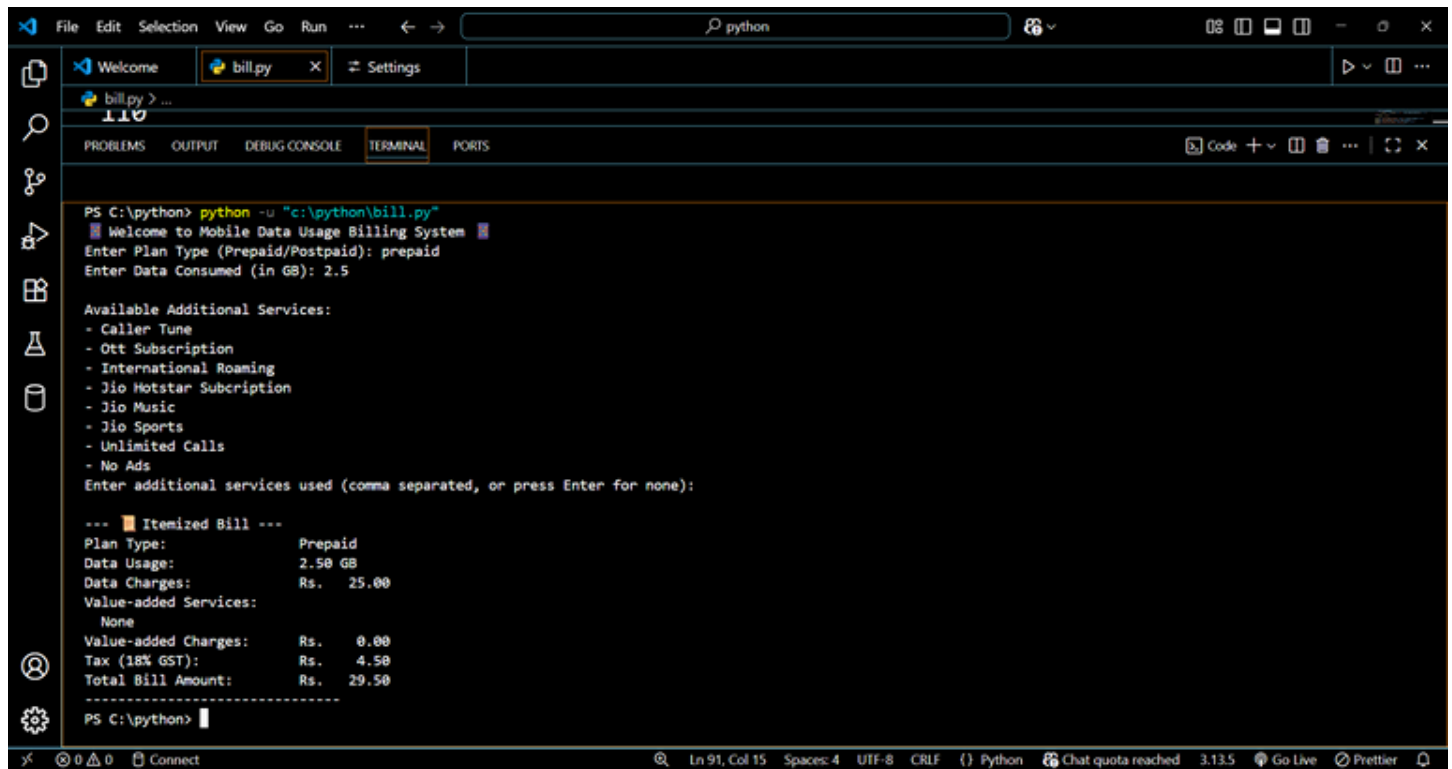
```
83 def calculate_bill(plan, data_gb, services):
84     """Calculate charges, tax, and total bill."""
85     data_charges = DATA_CHARGES[plan] * data_gb
86     service_charges = sum(VALUE_ADDED_SERVICES[s] for s in services)
87     subtotal = data_charges + service_charges
88     tax = subtotal * TAX_RATE
89     total = subtotal + tax
90     return data_charges, service_charges, tax, total
91
```

PROMPT: DISPLAY THE BILL AND IF SERVICES NOT ENTERED TAKE AS NONE.



```
91 # display_bill
92 def display_bill(plan, data_gb, data_charges, services, service_charges, tax, total):
93     """Display the itemized bill."""
94     print("\n--- Itemized Bill ---")
95     print(f"Plan Type: {plan}")
96     print(f>Data Usage: {data_gb:.2f} GB")
97     print(f>Data Charges: {data_charges:.2f} Rs.")
98
99     print("Value-added Services:")
100     if services:
101         for s in services:
102             print(f" {s.title():23} Rs. {VALUE_ADDED_SERVICES[s]:>7.2f}")
103     else:
104         print(" None")
105
106     print(f"Value-added Charges: {service_charges:.2f} Rs.")
107     print(f"Tax (18% GST): {tax:.2f} Rs.")
108     print(f"Total Bill Amount: {total:.2f} Rs.")
109     print("-----")
110
111 def main():
112     """Main function to run the billing application."""
113     print("Welcome to Mobile Data Usage Billing System")
114     plan = get_plan_type()
115     data_gb = get_data_consumed()
116     services = get_additional_services()
117     data_charges, service_charges, tax, total = calculate_bill(plan, data_gb, services)
118     display_bill(plan, data_gb, data_charges, services, service_charges, tax, total)
119
120 if __name__ == "__main__":
121     main()
122
```

OUTPUT:



```
PS C:\python> python -u "c:\python\bill.py"
Welcome to Mobile Data Usage Billing System
Enter Plan Type (Prepaid/Postpaid): prepaid
Enter Data Consumed (in GB): 2.5

Available Additional Services:
- Caller Tune
- Ott Subscription
- International Roaming
- Jio Hotstar Subscription
- Jio Music
- Jio Sports
- Unlimited Calls
- No Ads
Enter additional services used (comma separated, or press Enter for none):

--- Itemized Bill ---
Plan Type:          Prepaid
Data Usage:         2.50 GB
Data Charges:       Rs. 25.00
Value-added Services:
None
Value-added Charges: Rs. 0.00
Tax (18% GST):      Rs. 4.50
Total Bill Amount:  Rs. 29.50
-----
PS C:\python>
```

REPORT OF THE CODE:

REQUIREMENTS: GITHUB COPILOT – VS CODE

The Mobile Data Usage Billing Application successfully:

- Accepts realistic inputs.
- Implements accurate billing logic.
- Displays a clear, formatted, and comparable bill. The results match the actual mobile bill values, proving the correctness of the logic. This demonstrates how AI-assisted coding tools can speed up development and improve code quality.

TASK 3: Develop an LPG Billing System

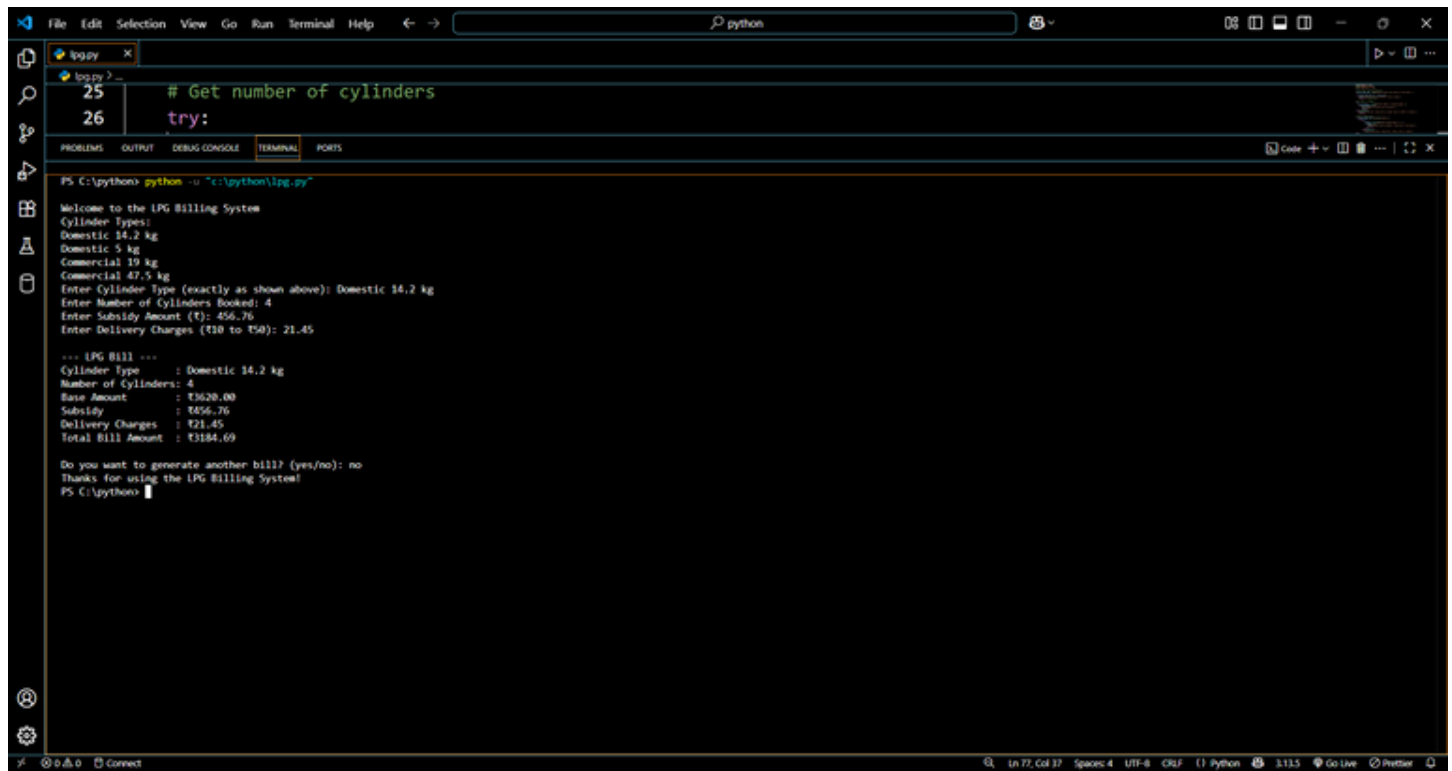
Prompt:

Code:

```
1 # LPG Billing System
2 price_list = {
3     "Domestic 14.2 kg": 905.00,
4     "Domestic 5 kg": 335.50,
5     "Commercial 19 kg": 1886.50,
6     "Commercial 47.5 kg": 4712.00
7 }
8 # LPG Billing System Code
9 while True:
10     # Step 1: Read user inputs
11     print("\nWelcome to the LPG Billing System")
12     print("Cylinder Types:")
13     print("Domestic 14.2 kg")
14     print("Domestic 5 kg")
15     print("Commercial 19 kg")
16     print("Commercial 47.5 kg")
17
18     # Get cylinder type (case-sensitive)
19     cylinder_type_input = input("Enter Cylinder Type (exactly as shown above): ")
20
21     if cylinder_type_input not in price_list:
22         print("Invalid cylinder type entered. Please try again.")
23         continue # restart loop
24
25     # Get number of cylinders
26
27     try:
28         num_cylinders = int(input("Enter Number of Cylinders Booked: "))
29         if num_cylinders <= 0:
30             print("Number of cylinders must be positive.")
31             continue
32     except ValueError:
33         print("Invalid input. Please enter an integer value for number of cylinders.")
34         continue
35
36     # Step 2: Subsidy input
37     if cylinder_type_input.startswith("Domestic"):
38         while True:
39             try:
40                 subsidy = float(input("Enter Subsidy Amount (₹): "))
41                 if subsidy < 0:
42                     print("Subsidy cannot be negative. Please enter a valid amount.")
43                     continue
44                 break
45             except ValueError:
46                 print("Invalid input. Please enter a numeric value for subsidy.")
47     else:
48         subsidy = 0.0 # Default for commercial
49
50     # Step 3: Delivery charges input
51     try:
52         delivery_charges = float(input("Enter Delivery Charges (₹10 to ₹50): "))
53         if delivery_charges < 10 or delivery_charges > 50:
54             print("Delivery charges must be between ₹10 and ₹50.")
55             continue
56     except ValueError:
57         print("Invalid input. Please enter a numeric value for delivery charges.")
58         continue
59
60     # Step 4: Calculate bill
61     base_amount = price_list[cylinder_type_input] * num_cylinders
62     total_bill = base_amount - subsidy + delivery_charges
63
64     # Step 5: Display itemized bill
65     print("\n--- LPG Bill ---")
66     print(f"Cylinder Type      : {cylinder_type_input}")
67     print(f"Number of Cylinders: {num_cylinders}")
68     print(f"Base Amount         : ₹{base_amount:.2f}")
69     print(f"Subsidy              : ₹{subsidy:.2f}")
70     print(f"Delivery Charges    : ₹{delivery_charges:.2f}")
71     print(f"Total Bill Amount   : ₹{total_bill:.2f}")
72
73     # Ask user if they want to continue
74     choice = input("\nDo you want to generate another bill? (yes/no): ").strip().lower()
75     if choice != "yes":
76         print("Thanks for using the LPG Billing System!")
77         break
78
79 # End of the LPG Billing System code
```



Output:



```
File Edit Selection View Go Run Terminal Help python
lpg.py
25 # Get number of cylinders
26 try:
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\python> python -u "c:\python\lpg.py"
Welcome to the LPG Billing System
Cylinder Types:
Domestic 14.2 kg
Domestic 5 kg
Commercial 19 kg
Commercial 47.5 kg
Enter Cylinder Type (exactly as shown above): Domestic 14.2 kg
Enter Number of Cylinders Booked: 4
Enter Subsidy Amount (₹): 456.76
Enter Delivery Charges (₹10 to ₹50): 21.45

--- LPG Bill ---
Cylinder Type      : Domestic 14.2 kg
Number of Cylinders: 4
Base Amount       : ₹3620.00
Subsidy           : ₹456.76
Delivery Charges  : ₹21.45
Total Bill Amount  : ₹3184.69

Do you want to generate another bill? (yes/no): no
Thanks for using the LPG Billing System!
PS C:\python>
```

EXPLANATION OF CODE:

STEP 1:

```
price_list = {  
    "Domestic 14.2 kg": 905.00,  
    "Domestic 5 kg": 335.50,  
    "Commercial 19 kg": 1886.50,  
    "Commercial 47.5 kg": 4712.00  
}
```

- A dictionary that stores the price per cylinder for different types of LPG cylinders.
- Keys → Cylinder type (string)
- Values → Price in ₹ (float)

STEP 2: # LPG Billing System Code

while True:

while True: creates an infinite loop so the program keeps running until the user decides to exit.

STEP 3:

```
print("\nWelcome to the LPG Billing System")
```

```
print ("Cylinder Types:")
print ("Domestic 14.2 kg")
print ("Domestic 5 kg")
print ("Commercial 19 kg")
print ("Commercial 47.5 kg")
```

- **Displays a welcome message and lists the available cylinder types for the user.**

STEP 4: # Get cylinder type (case-sensitive)

```
cylinder_type_input = input ("Enter Cylinder Type (exactly as shown above): ")
```

- **Asks the user to enter the cylinder type exactly as displayed.**
- **The input is stored in cylinder_type_input.**

STEP 5:

```
if cylinder_type_input not in price_list:
```

```
    print ("Invalid cylinder type entered. Please try again.")
```

```
    continue # restart loop
```

- Checks if the entered type exists in price_list.
- If not found → shows an error and restarts the loop without running further code.

STEP 6:

Get number of cylinders

```
try:
```

```
    num_cylinders = int(input("Enter Number of Cylinders Booked: "))
```

```
    if num_cylinders <= 0:
```

```
        print("Number of cylinders must be positive.")
```

```
        continue
```

```
except ValueError:
```

```
    print ("Invalid input. Please enter an integer value for number of cylinders.")
```

```
    continue
```

- **Reads the number of cylinders from the user.**
- **try-except handles non-integer input errors.**
- **Rejects negative or zero values.**

STEP 7:

Step 2: Subsidy input


```
if cylinder_type_input.startswith("Domestic"):
    while True:
        try:
            subsidy = float(input("Enter Subsidy Amount (₹): "))
            if subsidy < 0:
                print ("Subsidy cannot be negative. Please enter a valid amount.")
                continue
            break
        except ValueError:
            print("Invalid input. Please enter a numeric value for subsidy.")
```

else:

```
    subsidy = 0.0 # Default for commercial
```

- **Only for domestic cylinders: asks for subsidy amount.**
- **Rejects negative numbers and invalid entries.**
- **For commercial cylinders: automatically sets subsidy to ₹0.**

STEP 8:

Step 3: Delivery charges input

```
try:
    delivery_charges = float(input("Enter Delivery Charges (₹10 to ₹50): "))
    if delivery_charges < 10 or delivery_charges > 50:
        print("Delivery charges must be between ₹10 and ₹50.")
        continue
except ValueError:
    print("Invalid input. Please enter a numeric value for delivery charges.")
    continue
```

Reads delivery charges from the user.

Ensures the amount is between ₹10 and ₹50.

Rejects invalid (non-numeric) entries.

STEP 9:

Step 4: Calculate bill

```
base_amount = price_list[cylinder_type_input] * num_cylinders
```

```
total_bill = base_amount - subsidy + delivery_charges
```

- **Calculates Base Amount = Price per cylinder × Number of cylinders.**
- **Calculates Total Bill = Base Amount – Subsidy + Delivery Charges.**

STEP 9:

Step 5: Display itemized bill

```
print("\n--- LPG Bill ---")
```

```
print(f"Cylinder Type      : {cylinder_type_input}")
```

```
print(f"Number of Cylinders: {num_cylinders}")
```

```
print(f"Base Amount        : ₹{base_amount:.2f}")
```

```
print(f"Subsidy           : ₹{subsidy:.2f}")
```

```
print(f"Delivery Charges   : ₹{delivery_charges:.2f}")
```

```
print(f"Total Bill Amount  : ₹{total_bill:.2f}")
```

- **Displays an itemized bill with all details neatly formatted.**
- **.2f ensures amounts are shown with two decimal places.**

STEP 10:

Ask user if they want to continue

```
choice = input("\nDo you want to generate another bill? (yes/no): ").strip().lower()
```

```
if choice != "yes":
```

```
    print("Thanks for using the LPG Billing System!")
```

```
    break
```

Asks the user if they want to create another bill.

If the answer is not "yes", prints a thank-you message and breaks the loop.

End of the LPG Billing System code

SUMMARY:

SummaryThe LPG Billing System is a Python-based console application that calculates the total bill for LPG cylinder bookings based on user input.

1. Price Data Storage

- Uses a dictionary price_list to store the price per cylinder for each type (Domestic and Commercial).
- 2. **User Input & Validation**
 - Reads the cylinder type from the user (must match exactly with the options provided).
 - Reads the number of cylinders booked, ensuring it's a positive integer.
 - For domestic cylinders, asks for subsidy amount (non-negative value).
 - For commercial cylinders, sets subsidy to ₹0 automatically.
 - Reads delivery charges, ensuring they are between ₹10 and ₹50.
- 3. **Bill Calculation**
 - $\text{Base Amount} = \text{Price per cylinder} \times \text{Number of cylinders}$
 - $\text{Total Bill} = \text{Base Amount} - \text{Subsidy} + \text{Delivery Charges}$
- 4. **Output**
 - Displays an itemized bill including:
 - Cylinder Type
 - Number of Cylinders
 - Base Amount
 - Subsidy
 - Delivery Charges
 - Total Bill Amount
- 5. **Loop for Multiple Bills**
 - Runs in a loop, allowing the user to generate multiple bills until they choose to exit.

Key Features:

- Robust input validation for all user entries.
- Clear itemized bill presentation.
- Flexible design for both domestic and commercial LPG billing.

