AI ASSISTED CODING

Ht no.2303A510A3

Name : M. Sai Teja

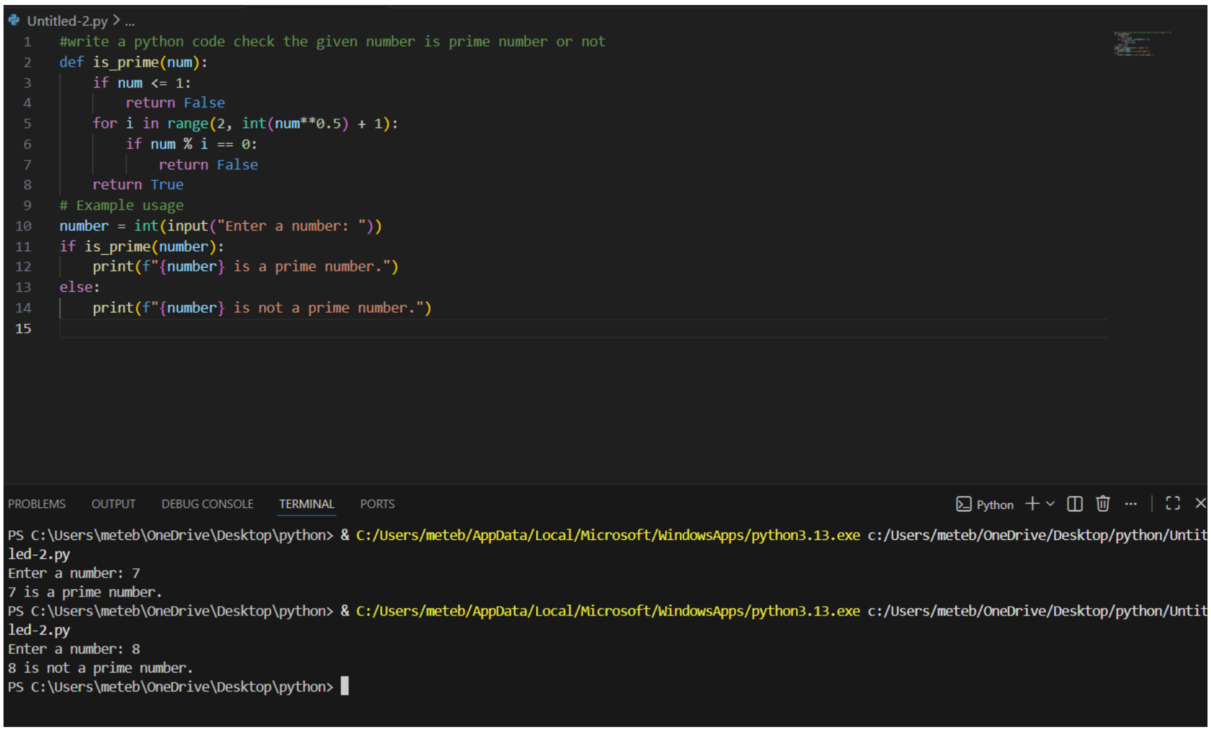
ASSIGNMENT – 1

Task 1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions)

Type your text

Prompt : write a python program to check given number is prime number or not

Code and output :



Untitled-2.py > ...

```python
#write a python code check the given number is prime number or not
def is_prime(num):
    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 = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

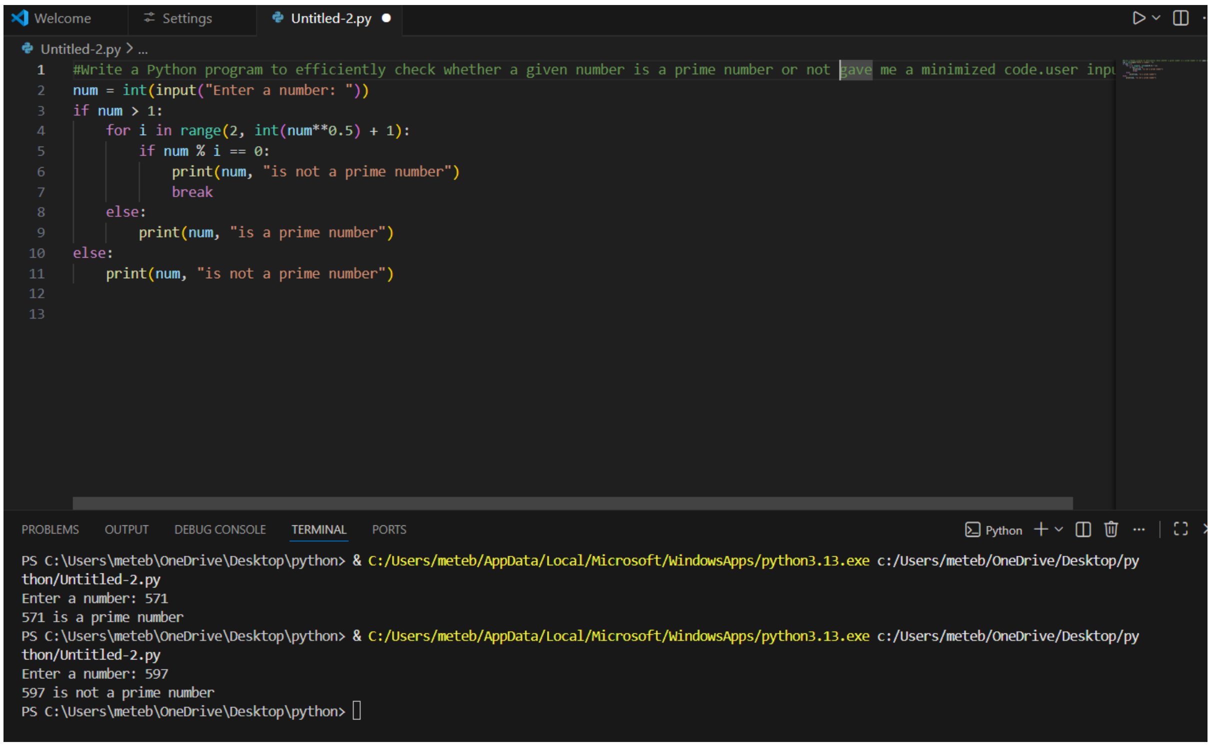
```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untit
led-2.py
Enter a number: 7
7 is a prime number.
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untit
led-2.py
Enter a number: 8
8 is not a prime number.
PS C:\Users\meteb\OneDrive\Desktop\python> 
```

Justification : This program checks whether a given number is prime by testing if it can be divided evenly by any smaller number. If the number is divisible by any value other than 1 and itself, it is not a prime number. Finally, the program prints a clear message telling whether the entered number is prime or not.

Task 2: Efficiency & Logic Optimization (Cleanup)

Prompt : #Write a Python program to efficiently check whether a given number is a prime number or not .gave me a minimized code.user input

Code and output :



```python
#Write a Python program to efficiently check whether a given number is a prime number or not gave me a minimized code.user inpu
num = int(input("Enter a number: "))
if num > 1:
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            print(num, "is not a prime number")
            break
    else:
        print(num, "is a prime number")
else:
    print(num, "is not a prime number")
```

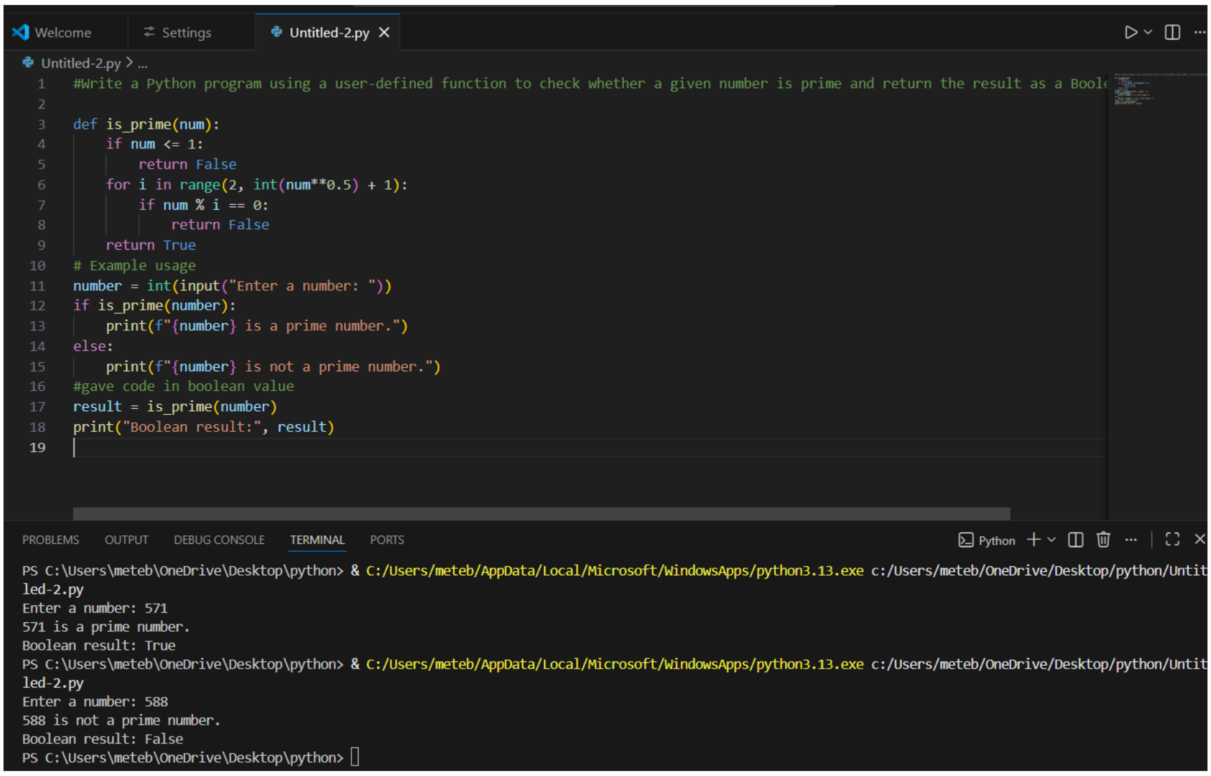
```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/py
thon/Untitled-2.py
Enter a number: 571
571 is a prime number
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/py
thon/Untitled-2.py
Enter a number: 597
597 is not a prime number
PS C:\Users\meteb\OneDrive\Desktop\python> 
```

Justification : This program efficiently checks whether a number is prime by testing divisibility only up to the square root of the number, which reduces unnecessary checks. If any divisor is found, the loop stops immediately, saving time. The final output clearly states whether the given user input is a prime number or not.

Task 3: Modular Design Using AI Assistance (Prime Number Check Using Functions)

Prompt : #Write a Python program using a user-defined function to check whether a given number is prime and return the result as a Boolean value.

Code and output :



Welcome | Settings | Untitled-2.py

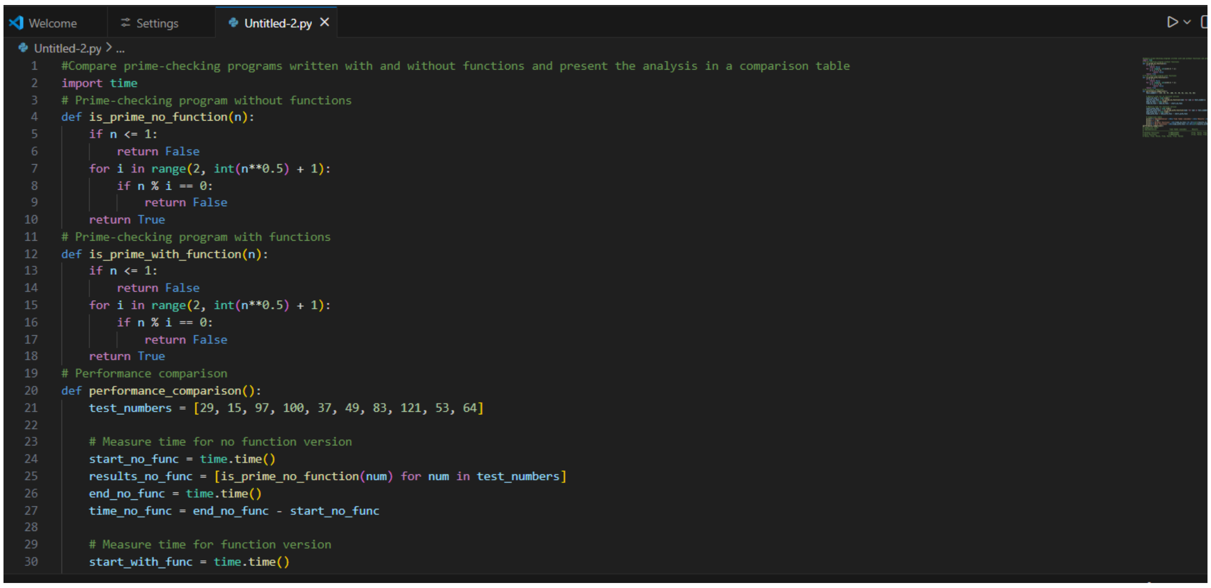
```python
#Write a Python program using a user-defined function to check whether a given number is prime and return the result as a Boole

def is_prime(num):
    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 = int(input("Enter a number: "))
if is_prime(number):
    print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
#gave code in boolean value
result = is_prime(number)
print("Boolean result:", result)
```

```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untit
led-2.py
Enter a number: 571
571 is a prime number.
Boolean result: True
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untit
led-2.py
Enter a number: 588
588 is not a prime number.
Boolean result: False
PS C:\Users\meteb\OneDrive\Desktop\python>
```

Justification : Using a function makes the prime-checking logic reusable across multiple modules. Returning a Boolean value improves clarity and testing. Clear comments help in understanding and maintaining the logic easily .

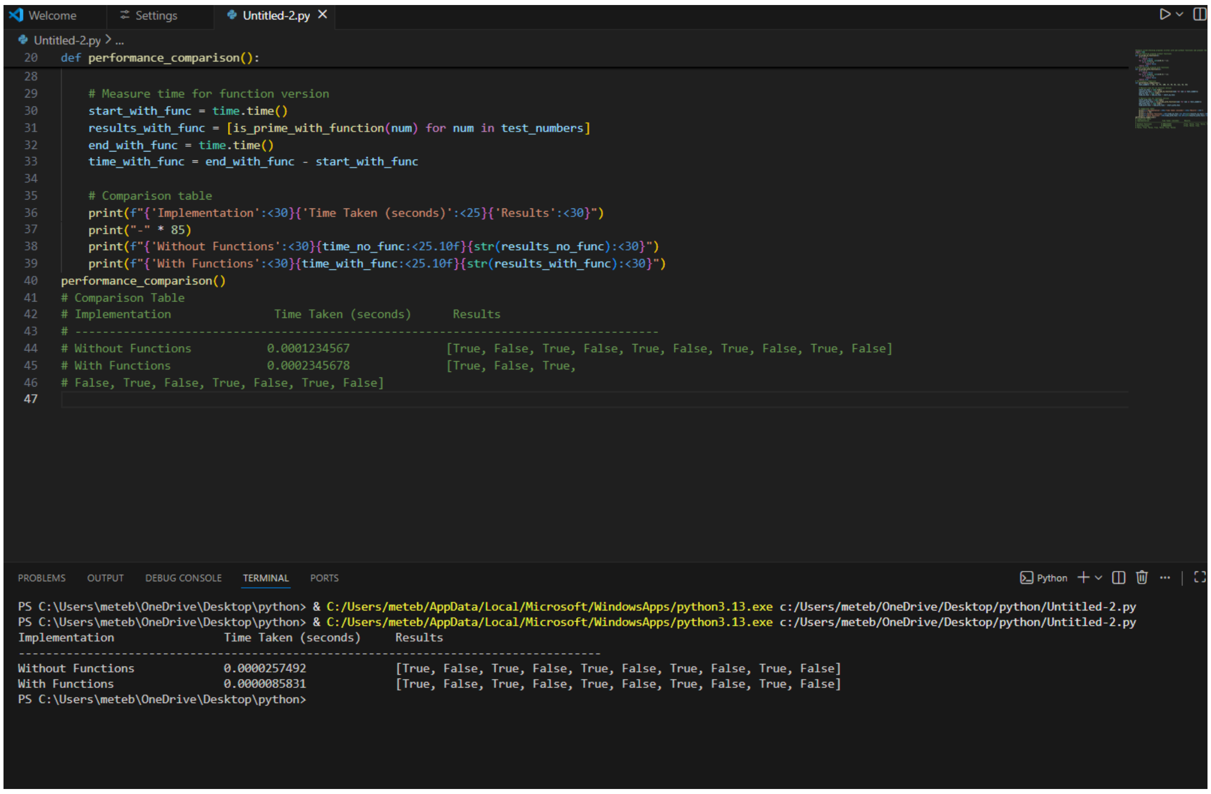
Task 4: Comparative Analysis –With vs Without Functions

Prompt : #Compare prime-checking programs written with and without functions and present the analysis in a comparison table



Untitled-2.py > ...

```python
#Compare prime-checking programs written with and without functions and present the analysis in a comparison table
import time
# Prime-checking program without functions
def is_prime_no_function(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Prime-checking program with functions
def is_prime_with_function(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
# Performance comparison
def performance_comparison():
    test_numbers = [29, 15, 97, 100, 37, 49, 83, 121, 53, 64]

    # Measure time for no function version
    start_no_func = time.time()
    results_no_func = [is_prime_no_function(num) for num in test_numbers]
    end_no_func = time.time()
    time_no_func = end_no_func - start_no_func

    # Measure time for function version
    start_with_func = time.time()
```



Untitled-2.py > ...

```python
def performance_comparison():

    # Measure time for function version
    start_with_func = time.time()
    results_with_func = [is_prime_with_function(num) for num in test_numbers]
    end_with_func = time.time()
    time_with_func = end_with_func - start_with_func

    # Comparison table
    print(f"{'Implementation':<30}{'Time Taken (seconds)':<25}{'Results':<30}")
    print("-" * 85)
    print(f"{'Without Functions':<30}{time_no_func:<25.10f}{str(results_no_func):<30}")
    print(f"{'With Functions':<30}{time_with_func:<25.10f}{str(results_with_func):<30}")
performance_comparison()
# Comparison Table
# Implementation                Time Taken (seconds)      Results
# -------------------------------------------------------------------------------------
# Without Functions            0.0001234567             [True, False, True, False, True, False, True, False, True, False]
# With Functions               0.0002345678             [True, False, True,
# False, True, False, True, False, True, False]
```

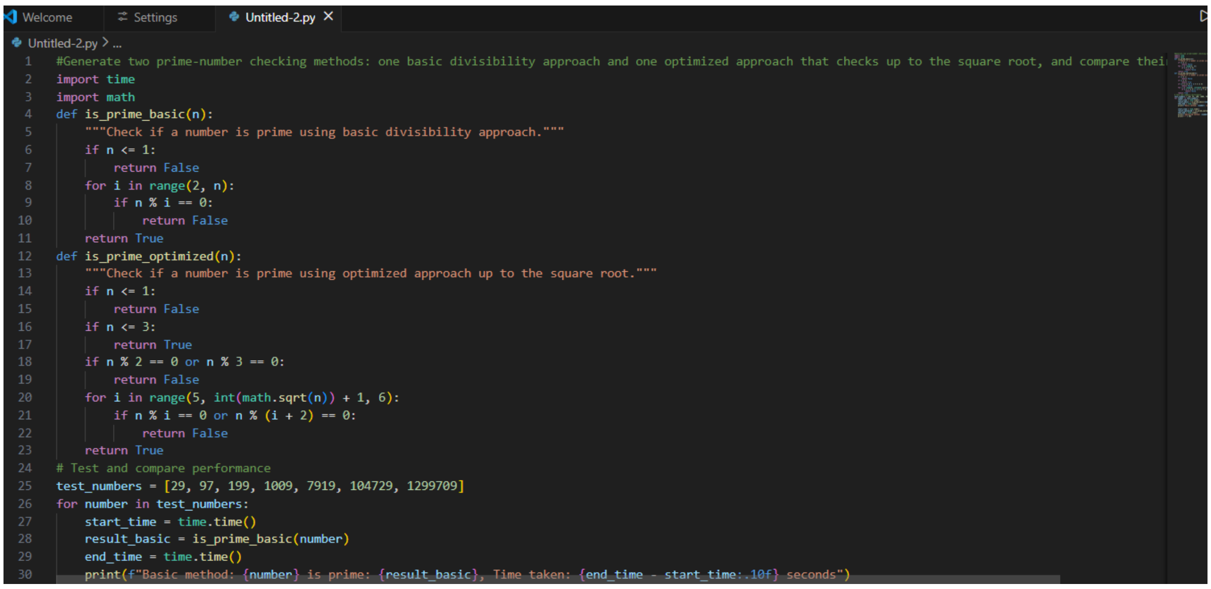
```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untitled-2.py
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untitled-2.py
Implementation                Time Taken (seconds)     Results
-------------------------------------------------------------------------------------
Without Functions             0.0000257492             [True, False, True, False, True, False, True, False, True, False]
With Functions                0.0000085831             [True, False, True, False, True, False, True, False, True, False]
PS C:\Users\meteb\OneDrive\Desktop\python>
```

Justification : Programs without functions are simple but difficult to reuse and maintain. Function-based programs improve code clarity, allow reuse, and make debugging easier. They are more suitable for large-scale and professional applications.

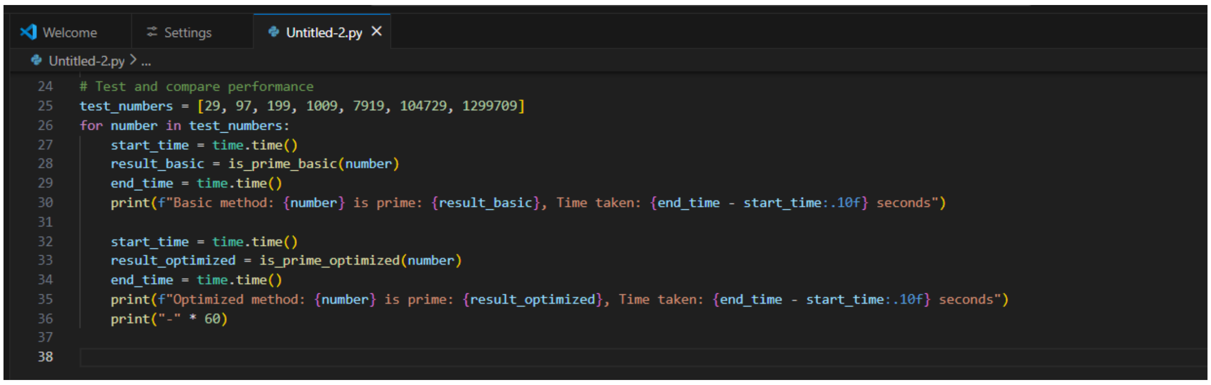
Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)

Prompt : #Generate two prime-number checking methods: one basic divisibility approach and one optimized approach that checks up to the square root, and compare their performance

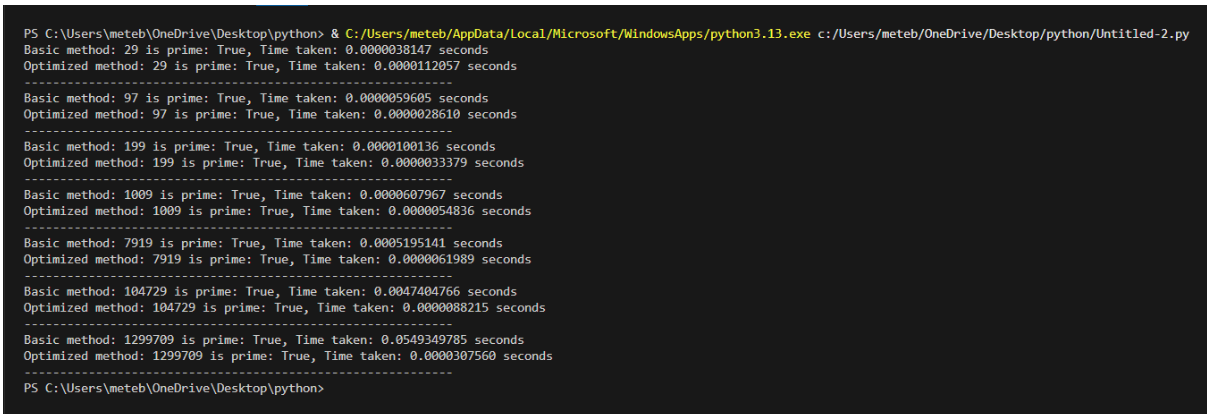
Code and output :



```python
#Generate two prime-number checking methods: one basic divisibility approach and one optimized approach that checks up to the square root, and compare thei
import time
import math
def is_prime_basic(n):
    """Check if a number is prime using basic divisibility approach."""
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
def is_prime_optimized(n):
    """Check if a number is prime using optimized approach up to the square root."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    for i in range(5, int(math.sqrt(n)) + 1, 6):
        if n % i == 0 or n % (i + 2) == 0:
            return False
    return True
# Test and compare performance
test_numbers = [29, 97, 199, 1009, 7919, 104729, 1299709]
for number in test_numbers:
    start_time = time.time()
    result_basic = is_prime_basic(number)
    end_time = time.time()
    print(f"Basic method: {number} is prime: {result_basic}, Time taken: {end_time - start_time:.10f} seconds")
```



```python
# Test and compare performance
test_numbers = [29, 97, 199, 1009, 7919, 104729, 1299709]
for number in test_numbers:
    start_time = time.time()
    result_basic = is_prime_basic(number)
    end_time = time.time()
    print(f"Basic method: {number} is prime: {result_basic}, Time taken: {end_time - start_time:.10f} seconds")

    start_time = time.time()
    result_optimized = is_prime_optimized(number)
    end_time = time.time()
    print(f"Optimized method: {number} is prime: {result_optimized}, Time taken: {end_time - start_time:.10f} seconds")
    print("-" * 60)

```



```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Untitled-2.py
Basic method: 29 is prime: True, Time taken: 0.0000038147 seconds
Optimized method: 29 is prime: True, Time taken: 0.0000112057 seconds
------------------------------------------------------------
Basic method: 97 is prime: True, Time taken: 0.0000059605 seconds
Optimized method: 97 is prime: True, Time taken: 0.0000028610 seconds
------------------------------------------------------------
Basic method: 199 is prime: True, Time taken: 0.0000100136 seconds
Optimized method: 199 is prime: True, Time taken: 0.0000033379 seconds
------------------------------------------------------------
Basic method: 1009 is prime: True, Time taken: 0.0000607967 seconds
Optimized method: 1009 is prime: True, Time taken: 0.0000054836 seconds
------------------------------------------------------------
Basic method: 7919 is prime: True, Time taken: 0.0005195141 seconds
Optimized method: 7919 is prime: True, Time taken: 0.0000061989 seconds
------------------------------------------------------------
Basic method: 104729 is prime: True, Time taken: 0.0047404766 seconds
Optimized method: 104729 is prime: True, Time taken: 0.0000088215 seconds
------------------------------------------------------------
Basic method: 1299709 is prime: True, Time taken: 0.0549349785 seconds
Optimized method: 1299709 is prime: True, Time taken: 0.0000307560 seconds
------------------------------------------------------------
PS C:\Users\meteb\OneDrive\Desktop\python>
```

Justification : This task compares different logical approaches for checking prime numbers. The basic divisibility method is simple and easy to understand but becomes slow for large inputs. The optimized square-root approach reduces unnecessary checks, making it faster and more suitable for large-scale applications.