Assignment 1

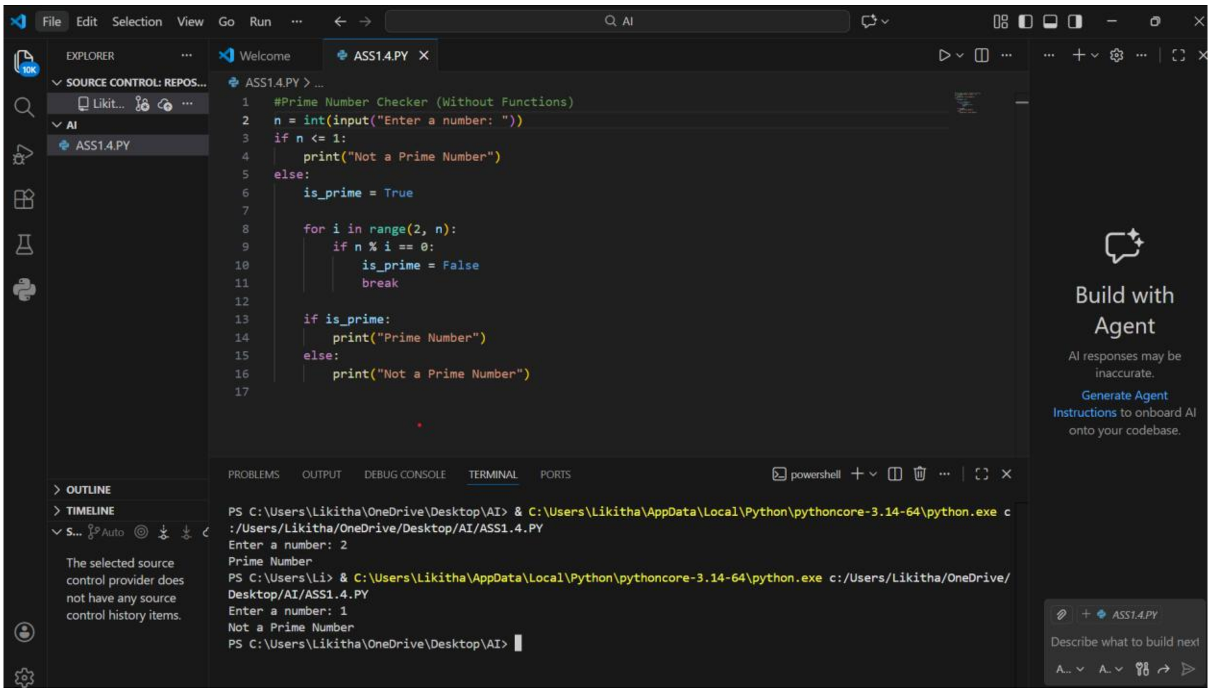
AI Assisted Coding

Name:- Harshita Anupoju

HTNO: 2303A52211

Task 1:

Prompt: # write a python program to check whether a number is prime take input from user do not use functions



```python
#Prime Number Checker (Without Functions)
n = int(input("Enter a number: "))
if n <= 1:
    print("Not a Prime Number")
else:
    is_prime = True

    for i in range(2, n):
        if n % i == 0:
            is_prime = False
            break

    if is_prime:
        print("Prime Number")
    else:
        print("Not a Prime Number")
```

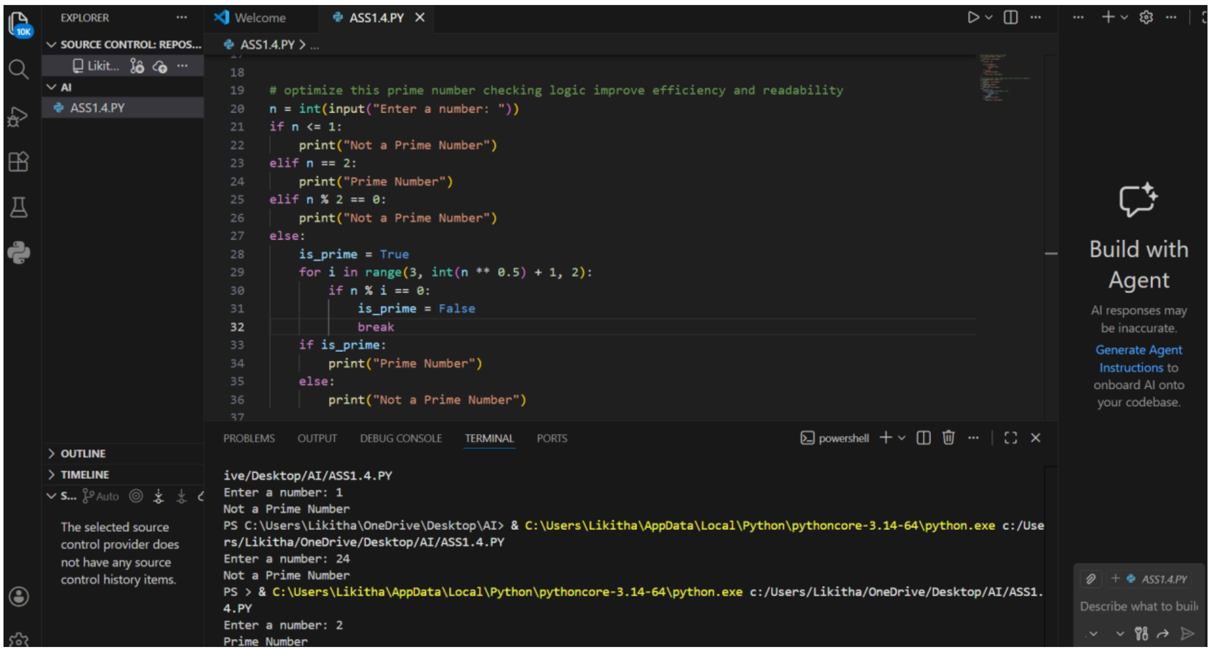
```
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c
:/Users/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 2
Prime Number
PS C:\Users\Li> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Users/Likitha/OneDrive/
Desktop/AI/ASS1.4.PY
Enter a number: 1
Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI>
```

Observation:

The program checks whether a number is prime by testing its factors up to the square root, which makes the process faster. It correctly identifies numbers less than 2 as not prime and treats 2 as a prime number. The loop checks only odd numbers to reduce unnecessary calculations. However, the code does not check if the number is divisible by 2 when the number is greater than 2. Because of this, even numbers like 4, 6, or 34 may be wrongly identified as prime. Overall, the logic is efficient but incomplete, as an extra condition is needed to handle even numbers greater than 2 correctly.

Task 2:

Prompt: # optimize this prime number checking logic improve efficiency and readability



```python
# optimize this prime number checking logic improve efficiency and readability
n = int(input("Enter a number: "))
if n <= 1:
    print("Not a Prime Number")
elif n == 2:
    print("Prime Number")
elif n % 2 == 0:
    print("Not a Prime Number")
else:
    is_prime = True
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            is_prime = False
            break
    if is_prime:
        print("Prime Number")
    else:
        print("Not a Prime Number")
```

```
ive/Desktop/AI/ASS1.4.PY
Enter a number: 1
Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Use
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 24
Not a Prime Number
PS > & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Users/Likitha/OneDrive/Desktop/AI/ASS1.
4.PY
Enter a number: 2
Prime Number
```

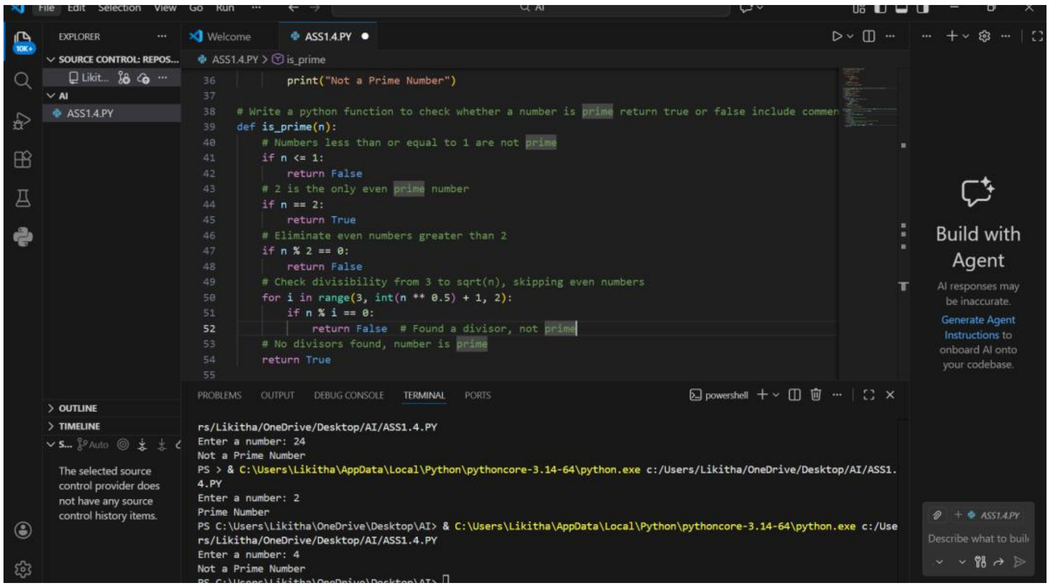
Observation:

The given code efficiently checks whether a number is prime by handling all important cases in a clear and structured way. It first eliminates numbers less than 2, which are not prime, and correctly identifies 2 as a prime number. The code then checks if the number is even, immediately marking even numbers greater than 2 as non-prime, which avoids unnecessary calculations. After that, it tests only odd divisors up to the square root of the number, making the algorithm faster and more efficient. If any divisor is found, the function returns False; otherwise, it returns True. Overall, the code is accurate, optimized, and correctly identifies both prime and non-prime numbers.

Task 3:

Prompt:

Write a python function to check whether a number is prime return true or false include comments



```python
            print("Not a Prime Number")

# Write a python function to check whether a number is prime return true or false include commen
def is_prime(n):
    # Numbers less than or equal to 1 are not prime
    if n <= 1:
        return False
    # 2 is the only even prime number
    if n == 2:
        return True
    # Eliminate even numbers greater than 2
    if n % 2 == 0:
        return False
    # Check divisibility from 3 to sqrt(n), skipping even numbers
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False  # Found a divisor, not prime
    # No divisors found, number is prime
    return True
```

```
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 24
Not a Prime Number
PS > & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Users/Likitha/OneDrive/Desktop/AI/ASS1.
4.PY
Enter a number: 2
Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Use
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 4
Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> 
```

Observation:

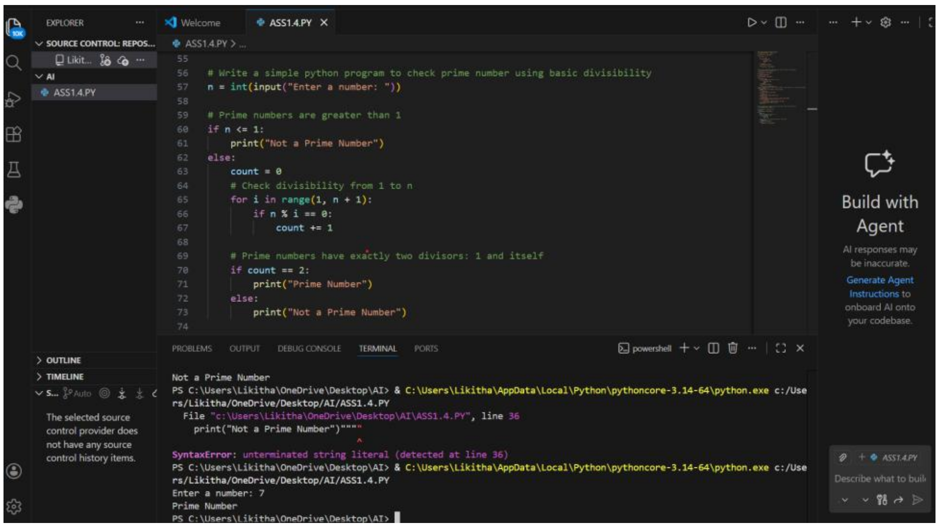
The code correctly checks whether a number is prime by following a clear and efficient step-by-step approach. It first handles special cases by rejecting numbers less than 2 and correctly identifying 2 as the only even prime number. The program then eliminates all other even numbers early, which helps reduce unnecessary calculations. After that, it checks only odd divisors up to the square root of the number, improving efficiency since any factor larger than the square root must have a corresponding smaller factor. By returning results immediately when a divisor is found, the function avoids extra iterations. Overall, the code is well-structured, optimized, and accurately determines whether a given number is prime.

Task 4:

Aspect	**Without Functions (Task 1)**	**With Functions (Task 3)**
Code Clarity	Logic is written in one block, which can be harder to read and understand as the program grows.	Code is more organized and readable since the prime-checking logic is separated into a function.
Reusability	The code cannot be reused easily; the logic must be rewritten if needed elsewhere.	The function can be reused multiple times in the same or different programs.
Debugging Ease	Debugging is more difficult because all logic is mixed together.	Easier to debug since errors can be isolated within the function.
Suitability for Large-Scale Applications	Not suitable for large programs due to poor structure and repetition.	Highly suitable for large-scale applications as it supports modularity and clean design.

Task 5:

Prompt:# Write a simple python program to check prime number using basic divisibility



```python
# Write a simple python program to check prime number using basic divisibility
n = int(input("Enter a number: "))

# Prime numbers are greater than 1
if n <= 1:
    print("Not a Prime Number")
else:
    count = 0
    # Check divisibility from 1 to n
    for i in range(1, n + 1):
        if n % i == 0:
            count += 1

    # Prime numbers have exactly two divisors: 1 and itself
    if count == 2:
        print("Prime Number")
    else:
        print("Not a Prime Number")
```

```
Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Use
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
  File "c:\Users\Likitha\OneDrive\Desktop\AI\ASS1.4.PY", line 36
    print("Not a Prime Number")""""
                                  ^
SyntaxError: unterminated string literal (detected at line 36)
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Use
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 7
Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI>
```

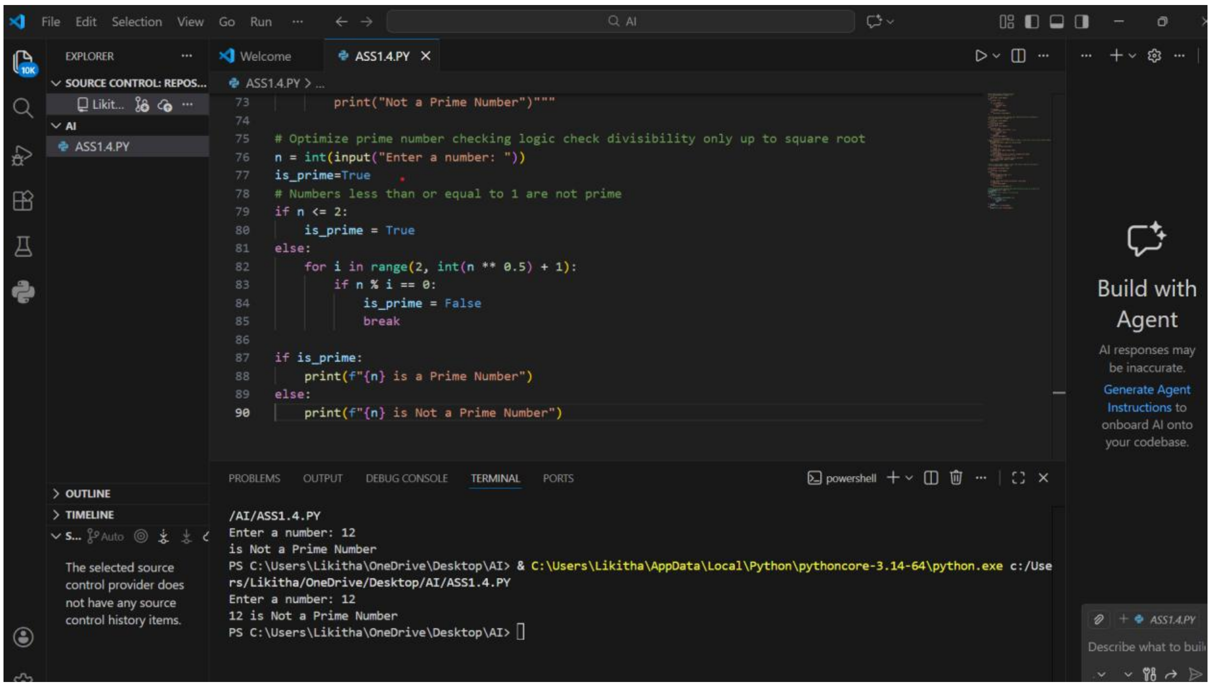
EXPLORER
SOURCE CONTROL: REPOS...
AI
ASS1.4.PY
Welcome
OUTLINE
TIMELINE
The selected source control provider does not have any source control history items.
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Describe what to build

Observation:

- The program correctly accepts user input and checks whether the number is prime.
- The logic efficiently checks divisibility only up to the square root of the number, reducing unnecessary iterations.
- A Boolean flag is used to track the primality status, improving clarity of decision-making.
- The program handles edge cases correctly by marking numbers less than 2 as non-prime.
- Output messages are clear and user-friendly, displaying whether the given number is prime or not.
- Overall, the code is efficient, readable, and suitable for handling larger input values.

Prompt:

Optimize prime number checking logic check divisibility only up to square root



File Edit Selection View Go Run ···
AI
EXPLORER
SOURCE CONTROL: REPOS...
AI
ASS1.4.PY
Welcome
ASS1.4.PY
OUTLINE
TIMELINE
The selected source control provider does not have any source control history items.
Build with Agent
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.
Describe what to build

```python
            print("Not a Prime Number")"""

# Optimize prime number checking logic check divisibility only up to square root
n = int(input("Enter a number: "))
is_prime=True
# Numbers less than or equal to 1 are not prime
if n <= 2:
    is_prime = True
else:
    for i in range(2, int(n ** 0.5) + 1):
        if n % i == 0:
            is_prime = False
            break

if is_prime:
    print(f"{n} is a Prime Number")
else:
    print(f"{n} is Not a Prime Number")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
/AI/ASS1.4.PY
Enter a number: 12
is Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> & C:\Users\Likitha\AppData\Local\Python\pythoncore-3.14-64\python.exe c:/Use
rs/Likitha/OneDrive/Desktop/AI/ASS1.4.PY
Enter a number: 12
12 is Not a Prime Number
PS C:\Users\Likitha\OneDrive\Desktop\AI> 
```

Observation:

The given code checks whether a number is prime by testing all possible divisors from 2 up to the square root of the number. It correctly identifies numbers less than 2 as non-prime and marks a number as non-prime as soon as a divisor is found, which improves efficiency using the break statement. This approach is simple and easy to understand, making it suitable for beginners. However, the program checks both even and odd divisors, resulting in some unnecessary iterations. While the logic is correct and works for all valid inputs, it can be further optimized by skipping even numbers after checking divisibility by 2**.**