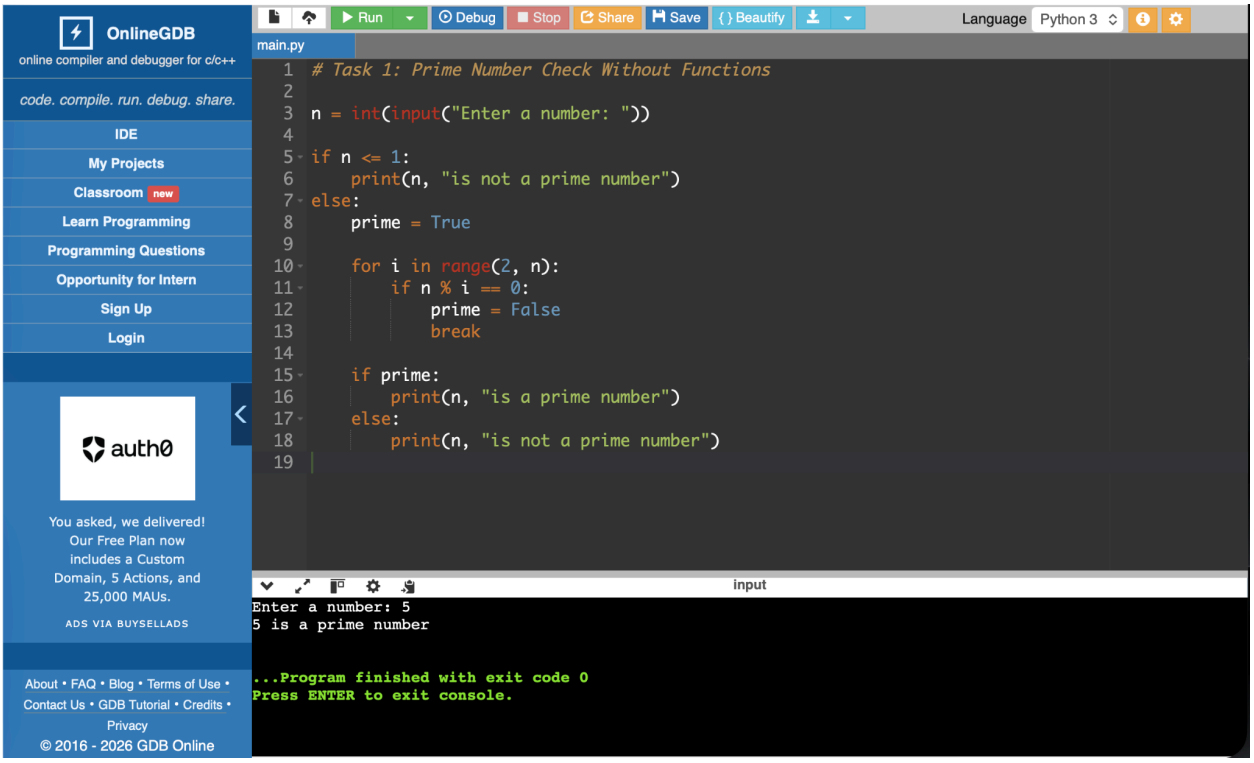
2303A521563
batch=10
Task 1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions)
❖ Scenario
➢ You are developing a basic validation script for a numerical learning application.
❖ Task Description
Use GitHub Copilot to generate a Python program that:
➢ Checks whether a given number is prime
➢ Accepts user input
➢ Implements logic directly in the main code
➢ Does not use any user-defined functions
❖ Expected Output
➢ Correct prime / non-prime result
➢ Screenshots showing Copilot-generated code suggestions
➢ Sample inputs and outputs

```
# Task 1: Prime Number Check Without Functions

n = int(input("Enter a number: "))

if n <= 1:
    print(n, "is not a prime number")
else:
    prime = True

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

    if prime:
        print(n, "is a prime number")
    else:
        print(n, "is not a prime number")
```



Explanation

Explanation

- User enters a number `n`
- If `n <= 1`, it is not prime
- Loop runs from `2` to `n-1`
- If divisible by any number → not prime

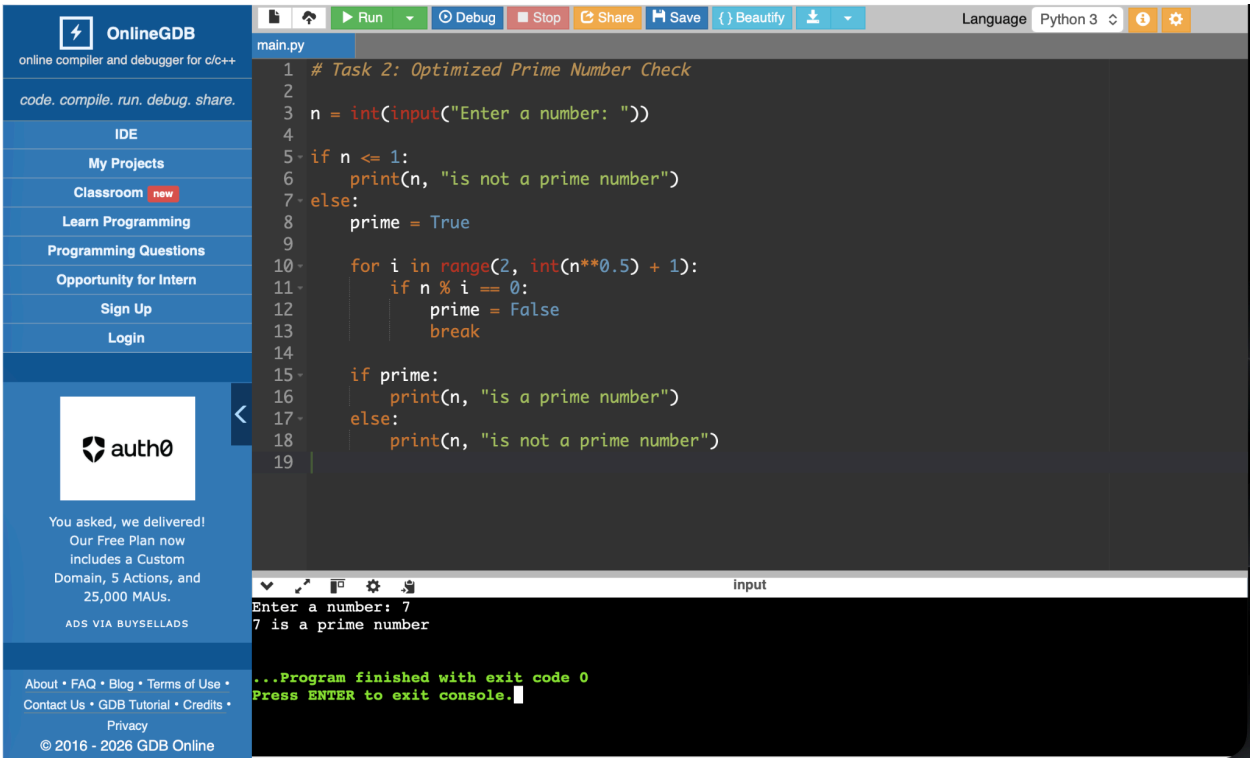
Time Complexity

- Worst case: **O(n)**
- Not efficient for large numbers

Task 2: Efficiency & Logic Optimization (Cleanup)
❖ Scenario
The script must handle larger input values efficiently.
❖ Task Description
Review the Copilot-generated code from Task 1 and improve it by:
➢ Reducing unnecessary iterations
➢ Optimizing the loop range (e.g., early termination)
➢ Improving readability
➢ Use Copilot prompts like:
▪ “Optimize prime number checking logic”

▪ “Improve efficiency of this code”
Hint:
Prompt Copilot with phrases like
“optimize this code”, “simplify logic”, or “make it more readable”
❖ Expected Output
➢ Original and optimized code versions
➢ Explanation of how the improvements reduce time complexity

```
# Task 2: Optimized Prime Number Check

n = int(input("Enter a number: "))

if n <= 1:
    print(n, "is not a prime number")
else:
    prime = True

    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            prime = False
            break

    if prime:
        print(n, "is a prime number")
    else:
        print(n, "is not a prime number")
```



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

❖ Scenario

The prime-checking logic will be reused across multiple modules.

❖ Task Description

Use GitHub Copilot to generate a function-based Python program that:

➢ Uses a user-defined function to check primality

➢ Returns a Boolean value

➢ Includes meaningful comments (AI-assisted)

❖ Expected Output

➢ Correctly working prime-checking function

➢ Screenshots documenting Copilot's function generation

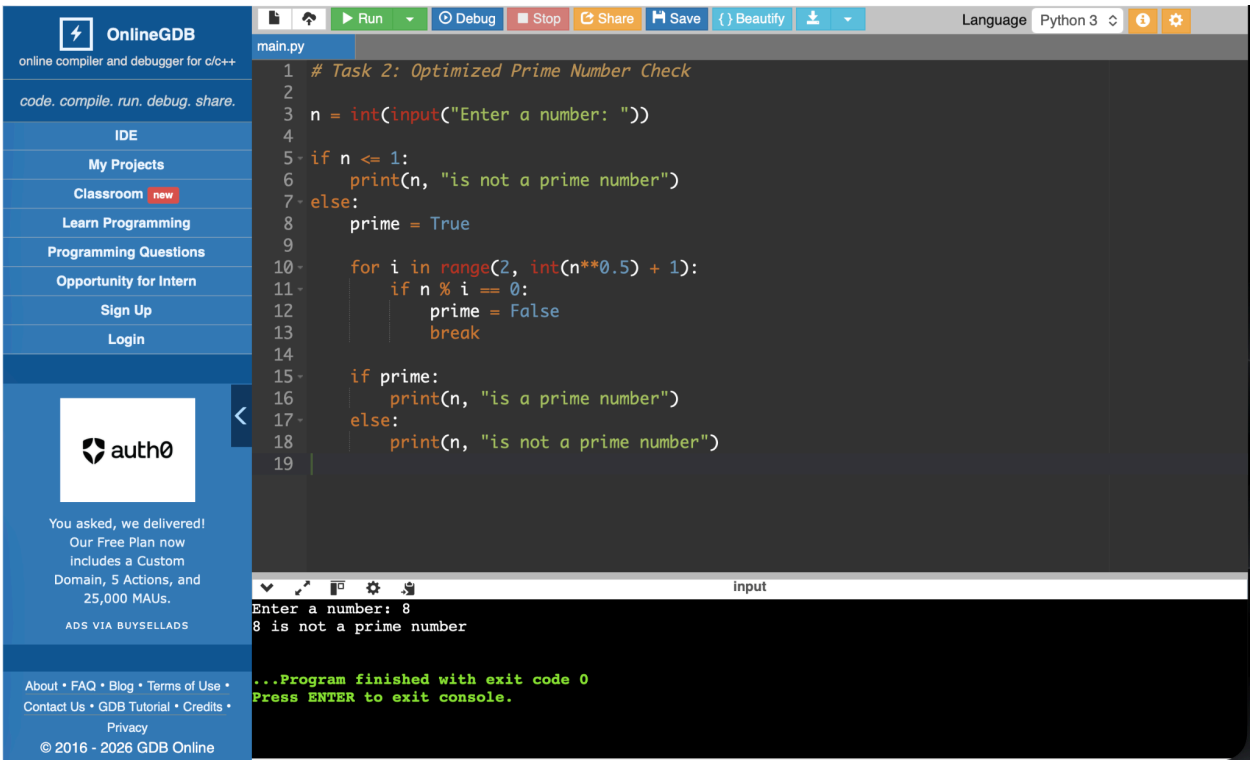
➢ Sample test cases and outputs

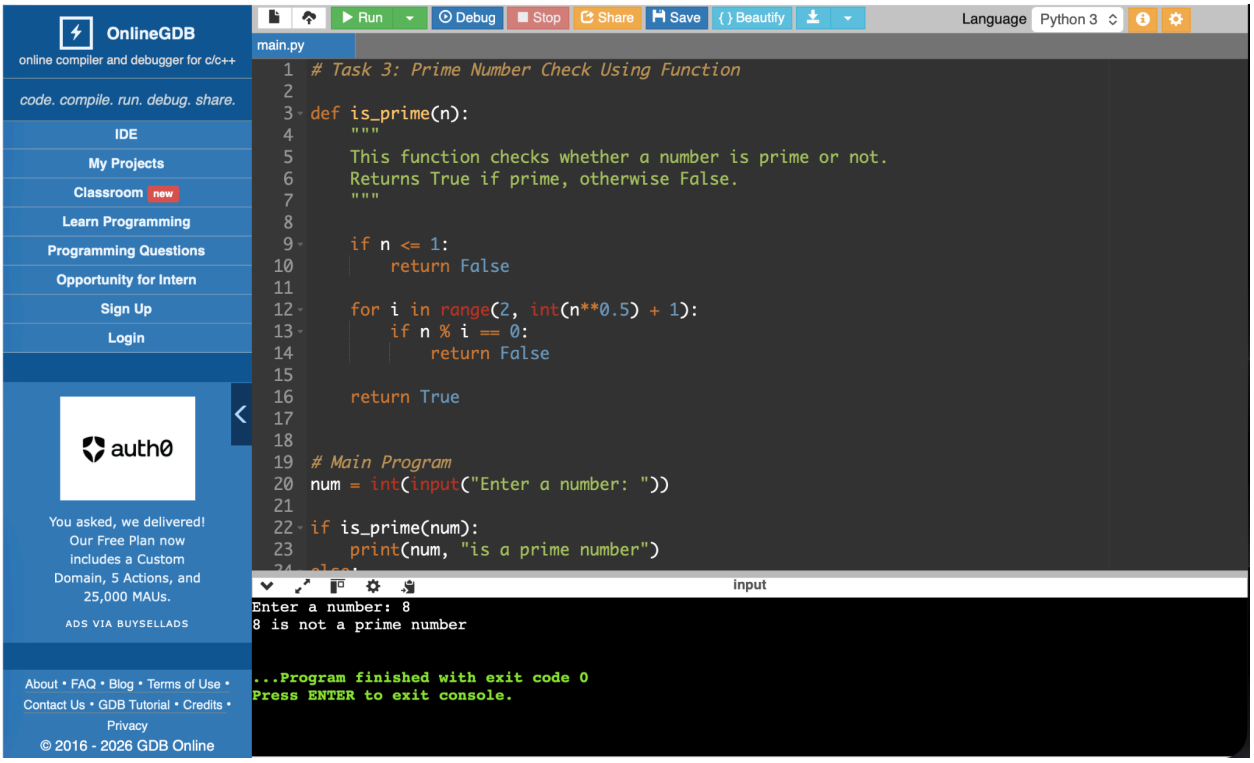
Task 2: Optimized Prime Number Check

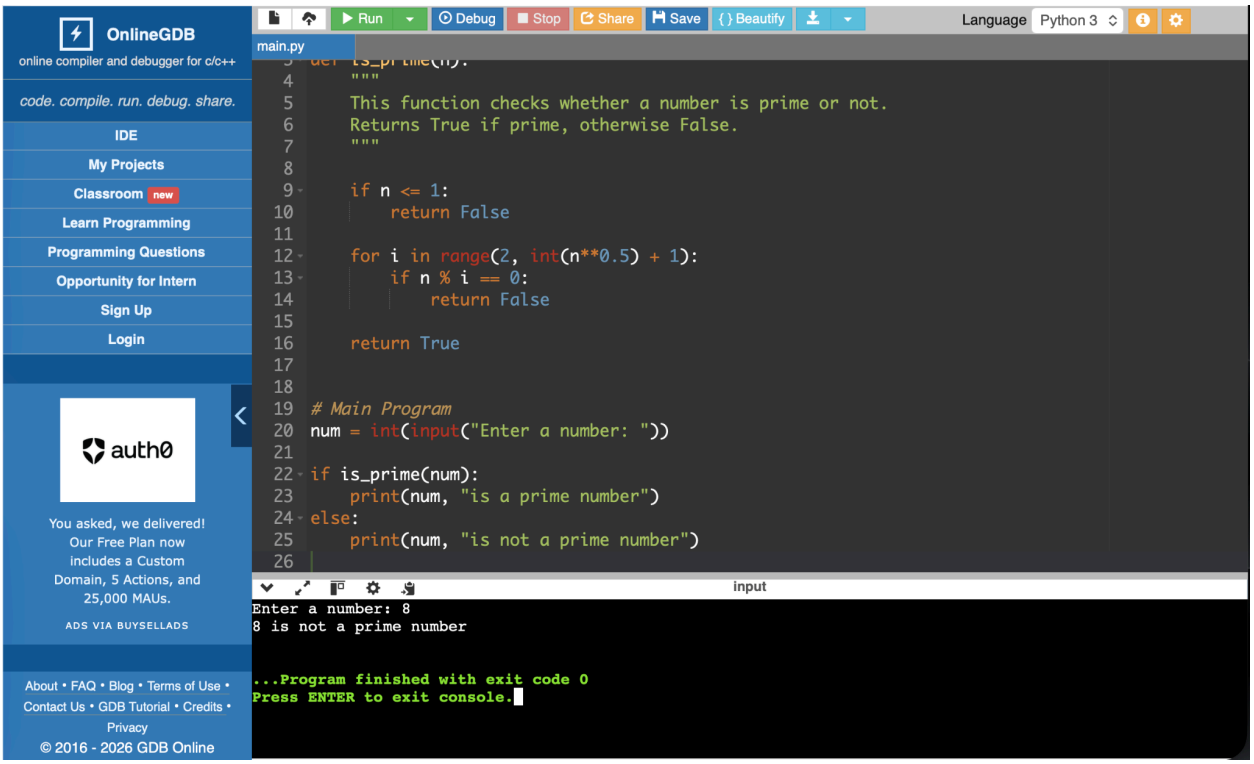
Task 3: Prime Number Check Using Function

```
def is_prime(n):
    """
    This function checks whether a number is prime or not.
    Returns True if prime, otherwise False.
    """

    if n <= 1:
```

```
    return False

for i in range(2, int(n**0.5) + 1):
    if n % i == 0:
        return False

return True
```







explanation

Optimization Done:

- Instead of checking till `n-1`
- We check only till √**n**

Because:

> If `n` has a factor greater than √n, it must also have a factor smaller than √n.

Time Complexity

- Before: **O(n)**
- After: **O(√n)** ✅ Faster for large inputs

Task 4: Comparative Analysis –With vs Without Functions

❖ Scenario

You are participating in a technical review discussion.

❖ Task Description

Compare the Copilot-generated programs:

➢ Without functions (Task 1)

➢ With functions (Task 3)

➢ Analyze them based on:

➢ Code clarity

➢ Reusability

➢ Debugging ease

➢ Suitability for large-scale applications

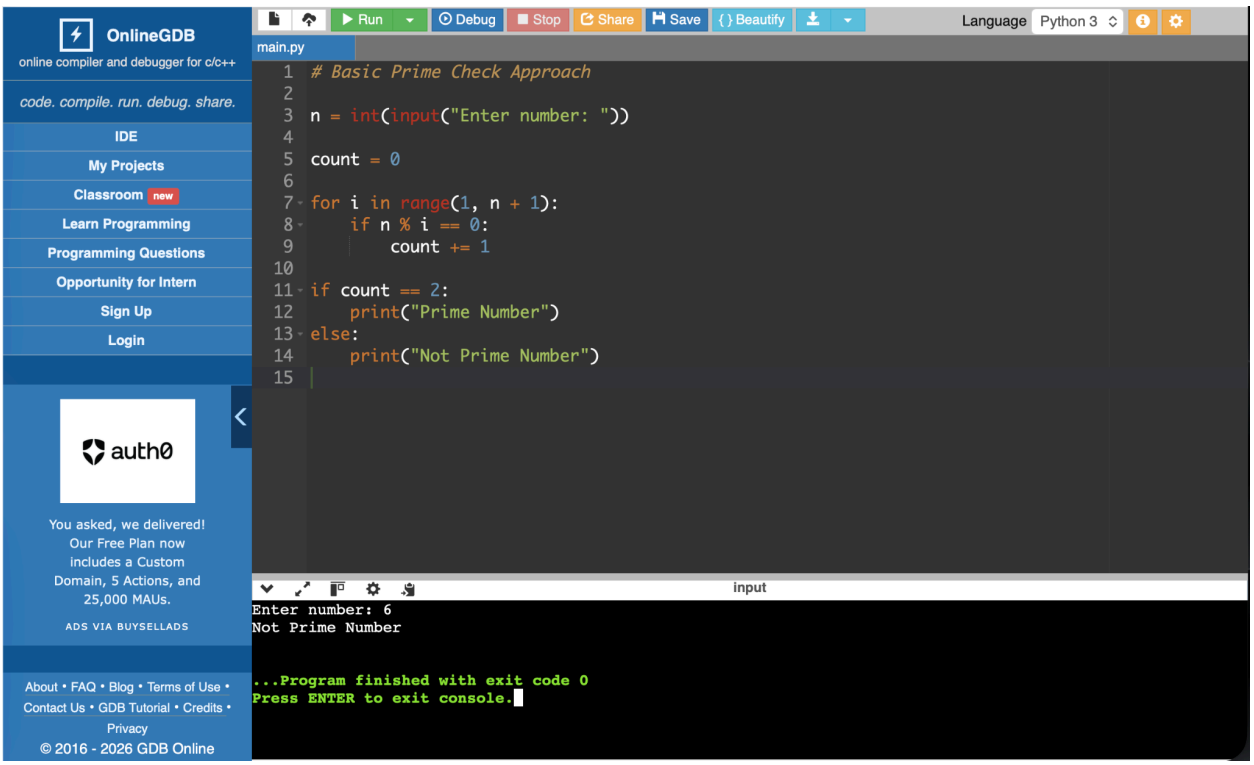
❖ Expected Output

Comparison table or short analytical report

Basic Prime Check Approach

n = int(input("Enter number: "))

```
count = 0


for i in range(1, n + 1):
    if n % i == 0:
        count += 1


if count == 2:
    print("Prime Number")
else:
    print("Not Prime Number")
```



Explanation

- Counts total divisors
- Prime numbers have exactly 2 divisors

Complexity

- **O(n)** very slow

Task 5: AI-Generated Iterative vs Recursive Fibonacci Approaches (Different

Algorithmic Approaches to Prime Checking)

❖ Scenario

Your mentor wants to evaluate how AI handles alternative logical

strategies.

❖ Task Description

Prompt GitHub Copilot to generate:

➢ A basic divisibility check approach

➢ An optimized approach (e.g., checking up to √n)

❖ Expected Output

➢ Two correct implementations

➢ Comparison discussing:

▪ Execution flow

▪ Time complexity

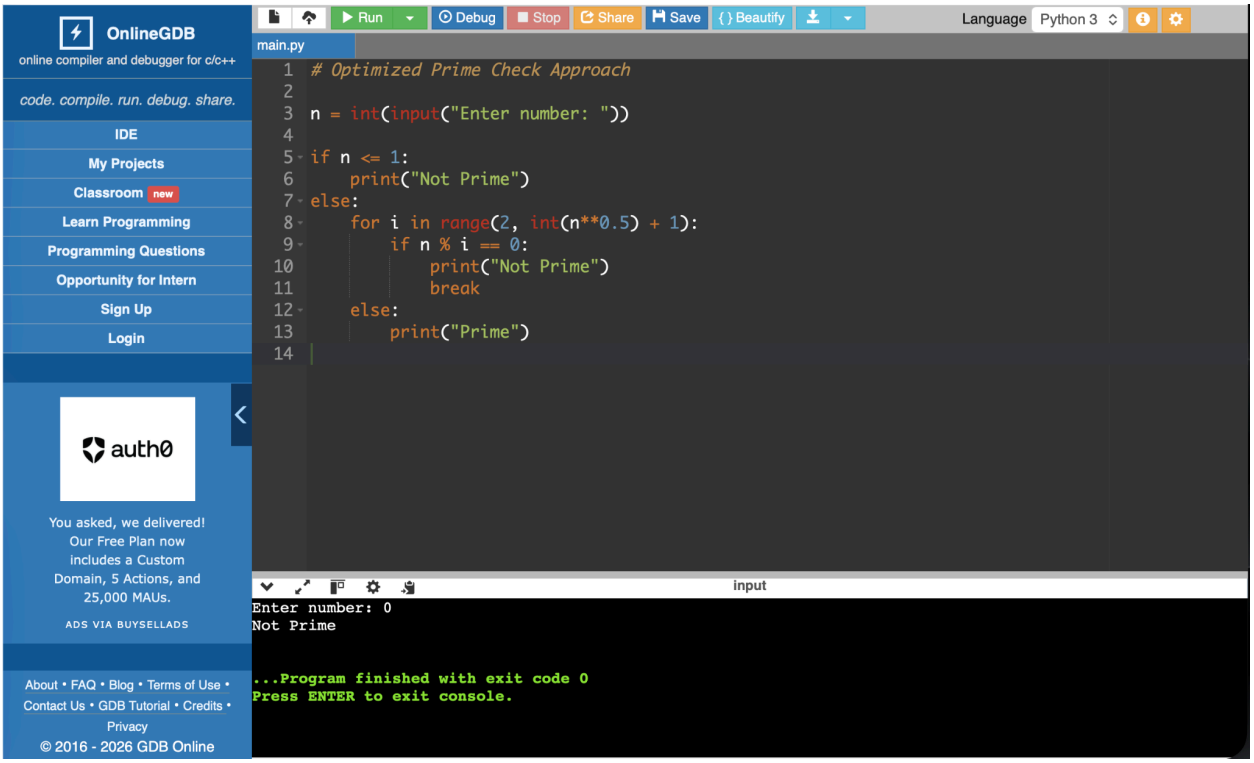
▪ Performance for large inputs

▪ When each approach is appropriate

Optimized Prime Check Approach

```
n = int(input("Enter number: "))


if n <= 1:
```

```
    print("Not Prime")
else:
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            print("Not Prime")
            break
    else:
        print("Prime")
```



Explanation

- Counts total divisors
- Prime numbers have exactly 2 divisors

Complexity

- **O(n)** very slow