AI ASSISTANT CODING

Assignment-1.4

Hall Ticket No: 2303A51873

Batch: 14

Name: R.Vineesha

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

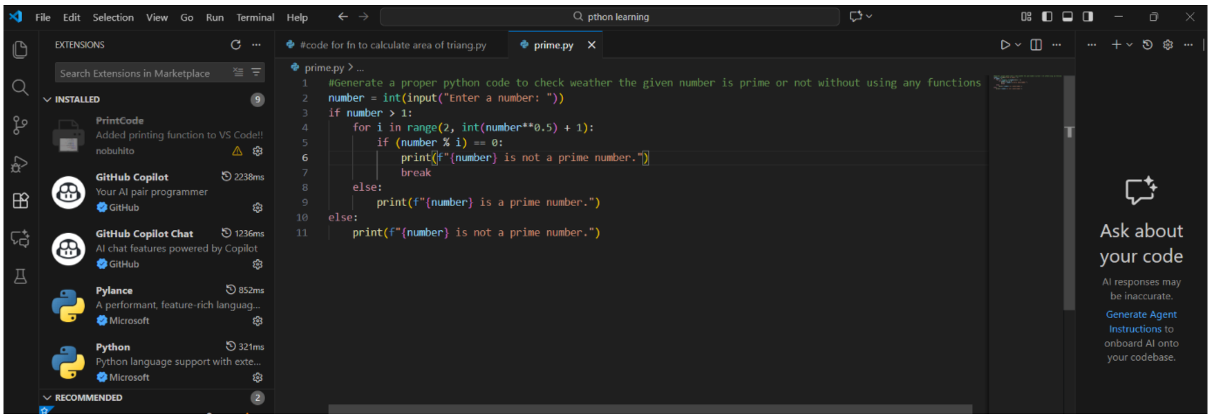
➢ Screenshots showing Copilot-generated code suggestions

➢ Sample inputs and outputs

Prompt

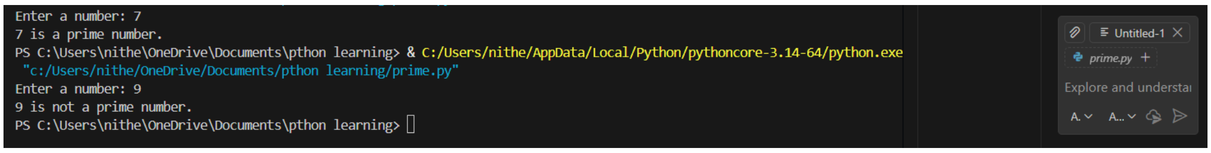
#Generate a proper python code to check weather the given number is prime or not without using any functions

Code



```python
#Generate a proper python code to check weather the given number is prime or not without using any functions
number = int(input("Enter a number: "))
if number > 1:
    for i in range(2, int(number**0.5) + 1):
        if (number % i) == 0:
            print(f"{number} is not a prime number.")
            break
    else:
        print(f"{number} is a prime number.")
else:
    print(f"{number} is not a prime number.")
```

Output:



```
Enter a number: 7
7 is a prime number.
PS C:\Users\nithe\OneDrive\Documents\pthon learning> & C:/Users/nithe/AppData/Local/Python/pythoncore-3.14-64/python.exe
 "c:/Users/nithe/OneDrive/Documents/pthon learning/prime.py"
Enter a number: 9
9 is not a prime number.
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

Justification:

This program checks whether a given number is prime using direct conditional logic without defining any functions. All computations are performed sequentially in a single block, making the logic easy to follow and suitable for beginners.

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

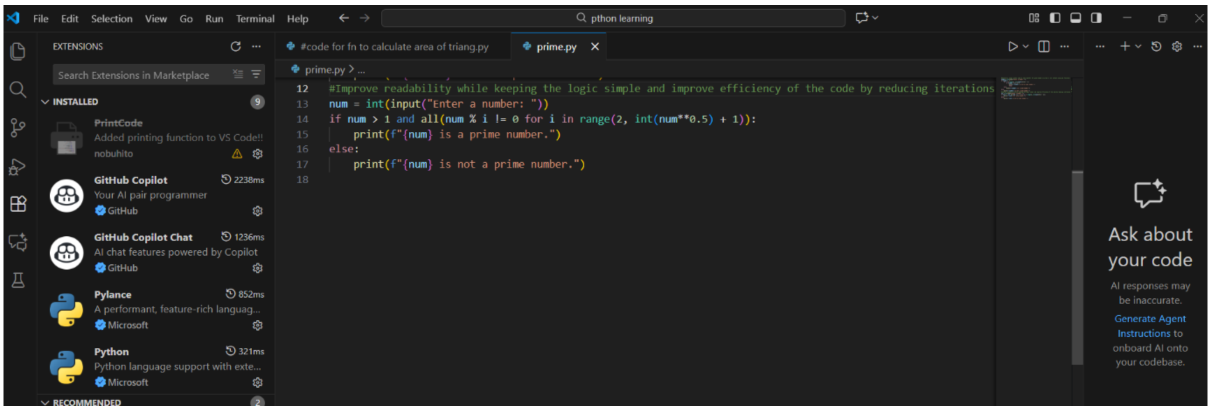
➢ Original and optimized code versions

➢ Explanation of how the improvements reduce time complexity

Prompt

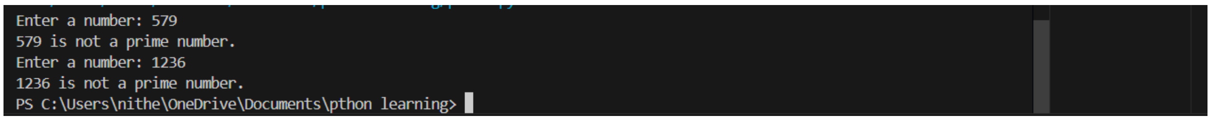
#Improve readability while keeping the logic simple and improve efficiency of the code by reducing iterations also minimize the code length

Code:


File
Edit
Selection
View
Go
Run
Terminal
Help
pthon learning
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
PrintCode
Added printing function to VS Code!!
GitHub Copilot
Your AI pair programmer
GitHub Copilot Chat
AI chat features powered by Copilot
Pylance
Python
RECOMMENDED
#code for fn to calculate area of triang.py
prime.py
Ask about your code
AI responses may be inaccurate.
Generate Agent Instructions to onboard AI onto your codebase.

```python
#Improve readability while keeping the logic simple and improve efficiency of the code by reducing iterations
num = int(input("Enter a number: "))
if num > 1 and all(num % i != 0 for i in range(2, int(num**0.5) + 1)):
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")
```

Output:



```
Enter a number: 579
579 is not a prime number.
Enter a number: 1236
1236 is not a prime number.
PS C:\Users\nithe\OneDrive\Documents\pthon learning>
```

Justification:

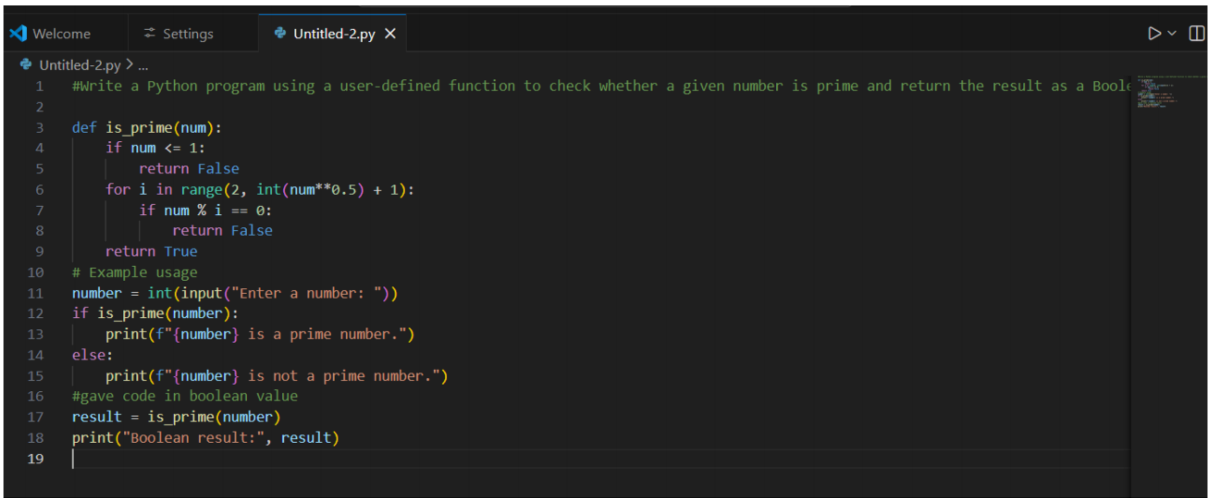
The optimized script improves performance by reducing unnecessary iterations and limiting the loop range, enabling faster execution for larger input values.
Early termination and simplified conditions lower the overall time complexity while maintaining correct prime number validation.

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

Prompt:

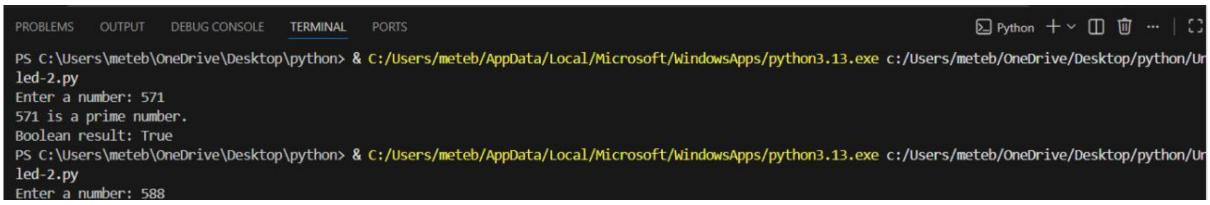
#The function must return a Boolean value (True if prime, False otherwise)



```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)
```

Output:



```
PS C:\Users\meteb\OneDrive\Desktop\python> & C:/Users/meteb/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/meteb/OneDrive/Desktop/python/Un
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/Un
led-2.py
Enter a number: 588
```

Justification:

Using a user-defined function makes the prime-checking logic reusable across multiple modules, improving code modularity and maintainability. Returning a Boolean value enables easy integration with conditional statements and other program components.

Task-4: Comparative Analysis –With vs Without Functions

Prompt:

# Compare both code with function without function Analyze and compare two Python programs for checking whether a number is prime

❖ 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

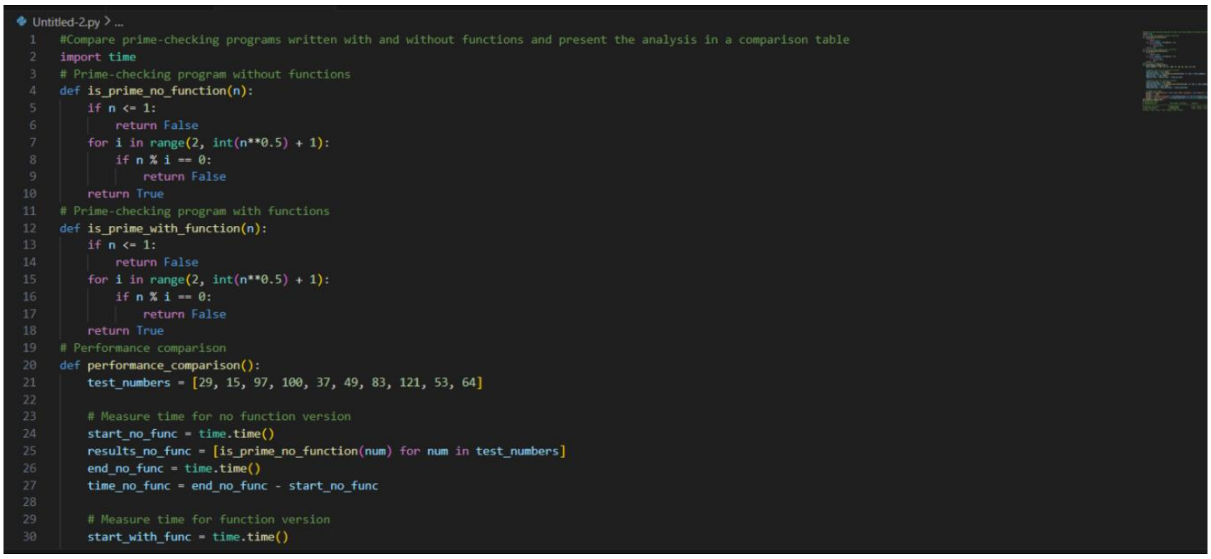
➢ Debugging ease

➢ Suitability for large-scale applications

❖ Expected Output

Comparison table or short analytical report

Code:

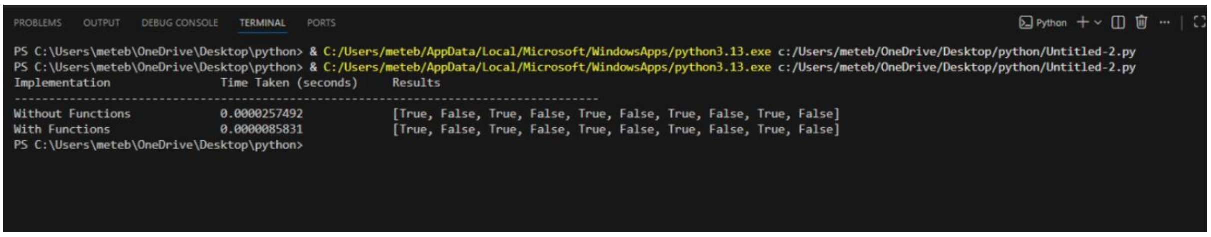


```python
#Compare prime-checking programs written with and without functions and present the analysis in a comparison table
import time
# Prime-checking program without functions
def is_prime_no_function(n):
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 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()
```

Output:



```
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 written with functions offer better code clarity by separating logic into well-defined blocks, making them easier to read and understand. Function-based designs improve reusability and debugging ease, as changes or fixes can be applied in one place without affecting the entire code.

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

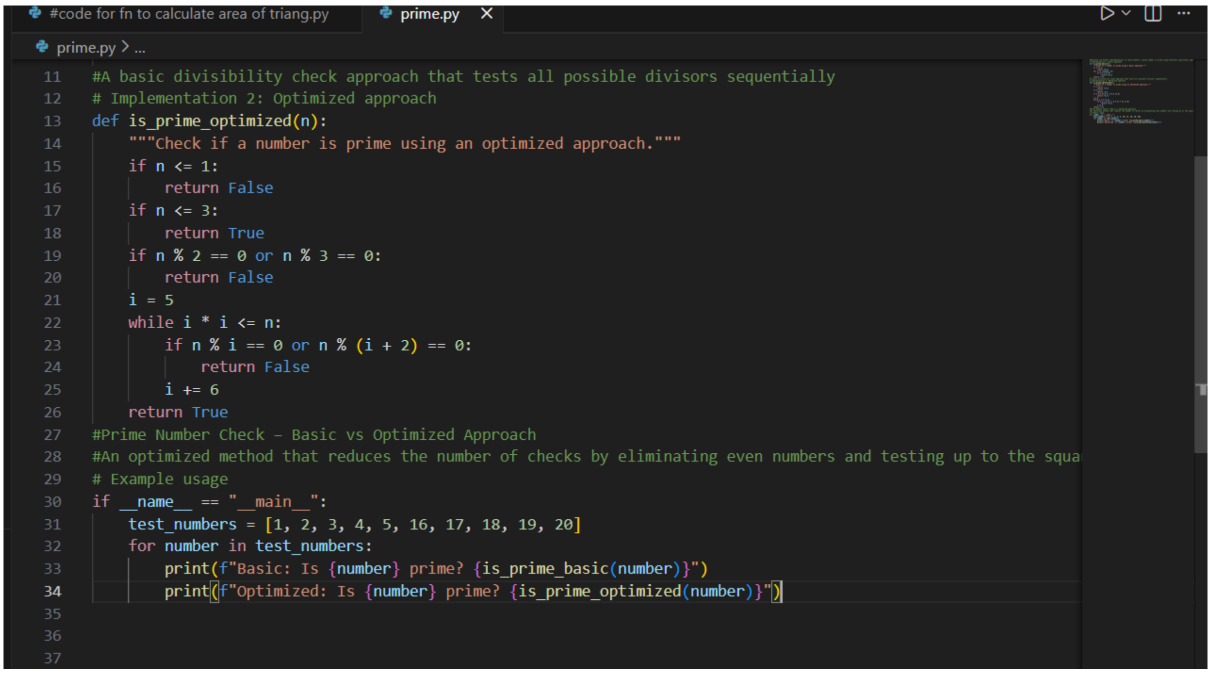
▪ Time complexity

▪ Performance for large inputs

▪ When each approach is appropriate

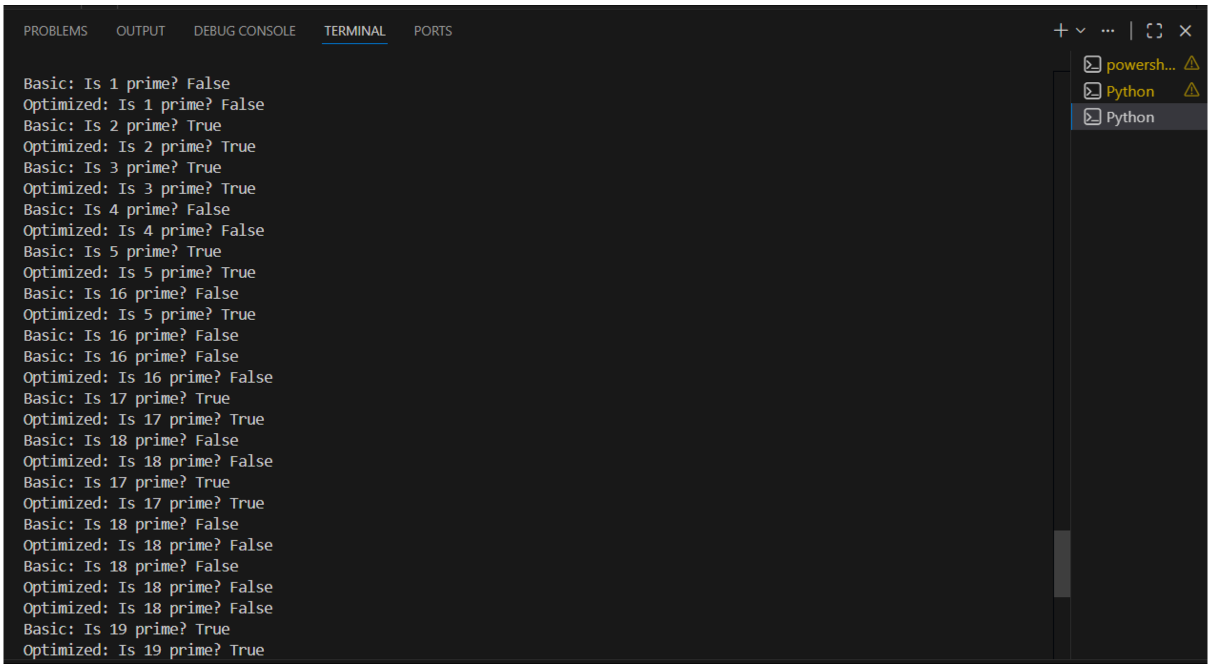
Prompt: Prime Number Check – Basic vs Optimized Approach

Code:



```python
#A basic divisibility check approach that tests all possible divisors sequentially
# Implementation 2: Optimized approach
def is_prime_optimized(n):
    """Check if a number is prime using an optimized approach."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
#Prime Number Check – Basic vs Optimized Approach
#An optimized method that reduces the number of checks by eliminating even numbers and testing up to the squa
# Example usage
if __name__ == "__main__":
    test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20]
    for number in test_numbers:
        print(f"Basic: Is {number} prime? {is_prime_basic(number)}")
        print(f"Optimized: Is {number} prime? {is_prime_optimized(number)}")
```

Output:



```
Basic: Is 1 prime? False
Optimized: Is 1 prime? False
Basic: Is 2 prime? True
Optimized: Is 2 prime? True
Basic: Is 3 prime? True
Optimized: Is 3 prime? True
Basic: Is 4 prime? False
Optimized: Is 4 prime? False
Basic: Is 5 prime? True
Optimized: Is 5 prime? True
Basic: Is 16 prime? False
Optimized: Is 5 prime? True
Basic: Is 16 prime? False
Basic: Is 16 prime? False
Optimized: Is 16 prime? False
Basic: Is 17 prime? True
Optimized: Is 17 prime? True
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 17 prime? True
Optimized: Is 17 prime? True
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 18 prime? False
Optimized: Is 18 prime? False
Optimized: Is 18 prime? False
Basic: Is 19 prime? True
Optimized: Is 19 prime? True
```

Justification: The basic approach checks divisibility up to N−1, resulting in unnecessary iterations and higher time complexity. The optimized approach checks only up to √N because any factor larger than √N must have a corresponding smaller factor.