NAME:T.RAMYASRI

H.NO:2303A52136

ASSIGNMENT-1.4

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

CODE:

```python
# Check if a number is prime. Read from input, print "prime" or "not prime". No functions.
number = int(input("Enter a number: "))
if number <= 1:
    print("not prime")
else:
    is_prime = True
    # naive check: try every number from 2 to n-1
    for i in range(2, number):
        if number % i == 0:
            is_prime = False
            break
    if is_prime:
        print("prime")
    else:
        print("not prime")
```

Output:

```
PROBLEMS 1    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

/coding/code.py
Enter a number: 5
rive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/pyt
Enter a number: 14
not prime
PS C:\Users\ramya\OneDrive\Desktop\coding>
```

TASK-2:

Efficiency & Logic Optimization (Cleanup) (Optimize the Task 1 code for larger inputs)

CODE:

```python
# Optimized prime check: reads integer from input, checks up to sqrt(n), prints "prime" or "not prime".

import math

n_str = input("Enter an integer: ")
try:
    n = int(n_str)
except ValueError:
    print("Invalid input. Please enter an integer.")
    exit(1)

if n <= 1:
    print("not prime")
elif n <= 3:
    # 2 and 3 are prime
    print("prime")
elif n % 2 == 0:
    print("not prime")
else:
    is_prime = True
    limit = math.isqrt(n)  # integer sqrt
    # check only odd divisors from 3 to sqrt(n)
    for i in range(3, limit + 1, 2):
        if n % i == 0:
            is_prime = False
            break
    print("prime" if is_prime else "not prime")
```

OUTPUT:

```
PS C:\Users\ramya\OneDrive\Desktop\coding> 71
71
PS C:\Users\ramya\OneDrive\Desktop\coding>        Open file in editor (ctrl + click)
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/python3.11
Enter an integer: 71
prime
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/python3.11
Enter an integer: 35
not prime
PS C:\Users\ramya\OneDrive\Desktop\coding>
```

Task 3 — Modular Design Using AI Assistance (Prime checking function)

code.py > ...

```python
# Prime checking using a reusable function. Returns True if n is prime.

import math

def is_prime(n: int) -> bool:
    """Return True if n is a prime number, else False.
    Uses trial division up to the integer square root and skips even numbers.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False
    limit = math.isqrt(n)
    for i in range(3, limit + 1, 2):
        if n % i == 0:
            return False
    return True

if __name__ == "__main__":
    n_str = input("Enter an integer: ")
    try:
        n = int(n_str)
    except ValueError:
        print("Invalid input. Please enter an integer.")
        exit(1)

    print("prime" if is_prime(n) else "not prime")

```

Output:

```
.py
Enter an integer: 5
prime
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/Windows/
Enter an integer: 13
prime
PS C:\Users\ramya\OneDrive\Desktop\coding>
```

TASK-4: **Comparative Analysis — With vs Without Functions**

ANALYSIS:

- The modular version (Task 3) is preferable for maintainability, reusability, and automated testing. The non-modular version (Task 1) is acceptable for a one-off quick script but scales poorly. Debugging and profiling are more convenient with a function. For production or library code, always prefer modularization and unit tests.

TASK-5: **Basic divisibility vs Optimized sqrt approach (two algorithmic strategies)**

A. Basic divisibility approach

```
code.py > ...
# Basic divisibility approach: check from 2 up to n-1

n = int(input("Enter an integer: "))
if n <= 1:
    print("not prime")
else:
    is_prime = True
    for i in range(2, n):
        if n % i == 0:
            is_prime = False
            break
    print("prime" if is_prime else "not prime")
```

OUTPUT:

```
Microsoft/WindowsApps/python3.11.exe c:/Users/ramya/OneDrive/Desktop/codi
g/code.py
Enter an integer: 13
prime
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local
Microsoft/WindowsApps/python3.11.exe c:/Users/ramya/OneDrive/Desktop/codi
g/code.py
Enter an integer: 23
prime
PS C:\Users\ramya\OneDrive\Desktop\coding>
```

B. Optimized sqrt approach

CODE:

```
e.py > ...
# Optimized: check up to sqrt(n), skip even divisors.

import math

def is_prime(n: int) -> bool:
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False
    limit = math.isqrt(n)
    for i in range(3, limit + 1, 2):
        if n % i == 0:
            return False
    return True

n = int(input("Enter an integer: "))
print("prime" if is_prime(n) else "not prime")
```

OUTPUT:

```
PROBLEMS 1   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS

PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/python3.1
Enter an integer: 100
not prime
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/python3.1
Enter an integer: 250
not prime
PS C:\Users\ramya\OneDrive\Desktop\coding> & C:/Users/ramya/AppData/Local/Microsoft/WindowsApps/python3.1
Enter an integer: 377
not prime
PS C:\Users\ramya\OneDrive\Desktop\coding>
```

JUSTIFICATION:

- Quick setup: VS Code + GitHub Copilot mirrors real developer workflow and enables prompt-based coding practice.
- Baseline (Task 1): Naive, non-modular code shows typical first-draft AI output—easy to understand and use as a comparison point.
- Optimization (Task 2): Teach students to reduce iterations (check to √n, skip evens) to move from O(n) → O(√n).

- Modularization (Task 3): Encapsulating logic in a function (is_prime) improves reusability, testability, and readability.
- Comparison (Task 4): Evaluating clarity, reusability, debugging, and scalability highlights trade-offs between quick scripts and production code.
- Algorithm variants (Task 5): Showing basic vs optimized approaches reinforces algorithm selection and when each is appropriate.
- Evidence & reproducibility: Screenshots + saved prompts prove the human+AI process and support grading.
- Prompt auditing: Recording prompts teaches how phrasing changes Copilot output—an important skill.
- Human oversight: Emphasizes reviewing AI suggestions for correctness, edge cases, and security.
- Testing & measurement: Unit tests and runtime comparisons provide empirical support for claims about correctness and performance.
- Pedagogy: The iterative generate→review→optimize→modularize cycle models real software engineering with AI as an assistant.
- Final recommendation: Use Copilot to accelerate scaffolding but always validate, optimize, and modularize before production use.