Assignment-1.4

K.Akshay
2303A51817

| SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE | | DEPARTMENT OF COMPUTER SCIENCE ENGINEERING | |
|---|---|---|---|
| **Program Name:** B. Tech | | **Assignment Type: Lab** | **Academic Year:**2025-2026 |
| **Course Coordinator Name** | Dr. Rishabh Mittal | | |
| **Instructor(s) Name** | Mr. S Naresh Kumar<br>Ms. B. Swathi<br>Dr. Sasanko Shekhar Gantayat<br>Mr. Md Sallauddin<br>Dr. Mathivanan<br>Mr. Y Srikanth<br>Ms. N Shilpa<br>Dr. Rishabh Mittal (Coordinator)<br>Dr. R. Prashant Kumar<br>Mr. Ankushavali MD<br>Mr. B Viswanath<br>Ms. Sujitha Reddy<br>Ms. A. Anitha<br>Ms. M.Madhuri<br>Ms. Katherashala Swetha<br>Ms. Velpula sumalatha<br>Mr. Bingi Raju | | |
| **CourseCode** | 23CS002PC304 | **Course Title** | AI Assisted Coding |
| **Year/Sem** | III/II | **Regulation** | R23 |
| **Date and Day of Assignment** | **Week1 – Thursday** | **Time(s)** | 23CSBTB01 To 23CSBTB52 |
| **Duration** | 2 Hours | **Applicable to Batches** | All batches |
| **Assignment Number:1.3**(Present assignment number)/**24**(Total number of assignments) | | | |

| Q.No. | Question | ***Expected Time to complete*** |
|---|---|---|
| 1 | Lab 1: Environment Setup – *GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow* | Week1 - Monday |

| | | |
|---|---|---|
| | **Lab Objectives:**<br>● To install and configure GitHub Copilot in Visual Studio Code.<br>● To explore AI-assisted code generation using GitHub Copilot.<br>● To analyze the accuracy and effectiveness of Copilot's code suggestions.<br>● To understand prompt-based programming using comments and code context<br><br>**Lab Outcomes (LOs):**<br>After completing this lab, students will be able to:<br>● Set up GitHub Copilot in VS Code successfully.<br>● Use inline comments and context to generate code with Copilot.<br>● Evaluate AI-generated code for correctness and readability.<br>● Compare code suggestions based on different prompts and programming styles. | |
| | Task 0<br>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step.<br>Expected Output<br>● Install and configure GitHub Copilot in VS Code. Take screenshots of each step. | |
| | Task 1: AI-Generated Logic Without Modularization (Prime Number Check Without Functions) | |

fibonacci.py | factorial.py | prime num.py

C: > Users > prade > OneDrive > Pradeep > prime num.py > ...

```python
# AI-Generated Prime Number Check Without Functions
# This script checks if a given number is prime without using any functions

# Get input from user
num = int(input("Enter a number to check if it's prime: "))

# Initialize flag
is_prime = True

# Check if number is less than or equal to 1
if num <= 1:
    is_prime = False
else:
    # Check for factors from 2 to sqrt(num)
    for i in range(2, int(num**0.5) + 1):
        if num % i == 0:
            is_prime = False
            break

# Output the result
if is_prime:
    print(f"{num} is a prime number.")
else:
    print(f"{num} is not a prime number.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

```
C:\Users\prade>C:/Users/prade/AppData/Local/Microsoft/WindowsApps/python3.12.exe "c:/Users/prade/OneDrive/Pradeep/prime num.py"
Enter a number to check if it's prime: 2
2 is a prime number.
```

Here we are checking whether the given number is prime or not.

❖ **Scenario**

- You are developing a **basic validation script** for a numerical learning application.

❖ **Task Description**

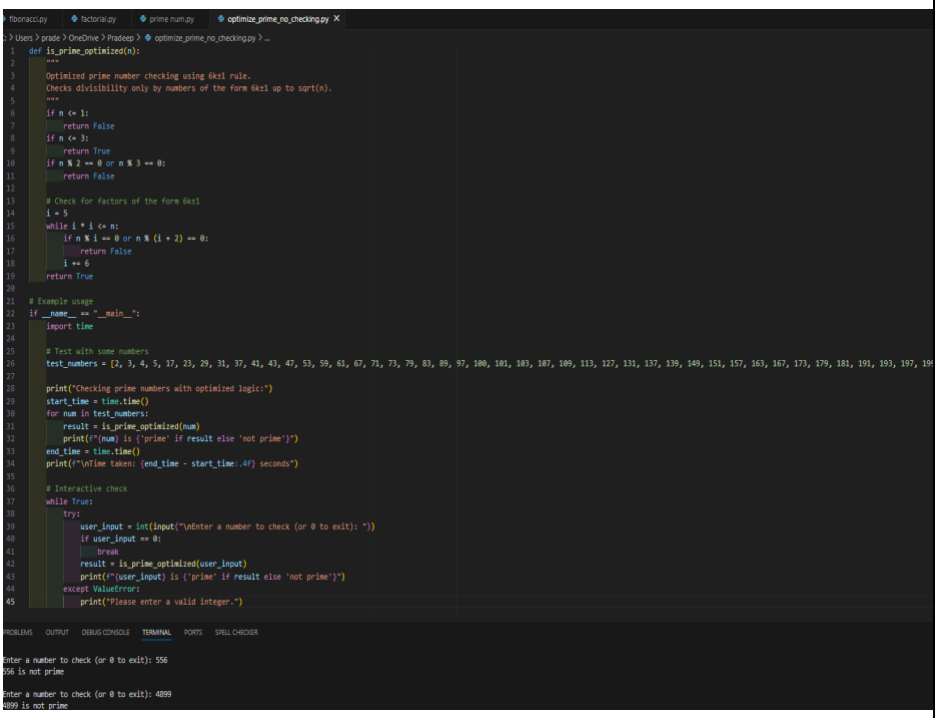
Use GitHub Copilot to generate a Python program that:

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

fibonacci.py | factorial.py | prime num.py | optimize_prime_no_checking.py | prime check in functions.py

C: > Users > prade > OneDrive > Pradeep > prime check in functions.py > ...

```python
# AI-assisted Python program for prime number checking using functions
# This program defines a function to check if a number is prime and returns a Boolean value

def is_prime(number):
    """
    Check if a given number is prime.

    Args:
    number (int): The number to check for primality.

    Returns:
    bool: True if the number is prime, False otherwise.
    """
    # Numbers less than or equal to 1 are not prime
    if number <= 1:
        return False

    # 2 and 3 are prime numbers
    if number <= 3:
        return True

    # Check if the number is divisible by 2 or 3
    if number % 2 == 0 or number % 3 == 0:
        return False

    # Check for factors from 5 to sqrt(number), skipping even numbers and multiples of 3
    i = 5
    while i * i <= number:
        if number % i == 0 or number % (i + 2) == 0:
            return False
        i += 6

    # If no factors found, the number is prime
    return True
```

```python
# Main function to test the is_prime function
def main():
    """
    Main function to demonstrate the prime checking functionality.
    """
    # Test cases
    test_numbers = [2, 3, 4, 5, 17, 23, 29, 31, 37, 41, 43, 47]

    print("Prime number checker (AI-assisted):")
    for num in test_numbers:
        result = is_prime(num)
        status = "is prime" if result else "is not prime"
        print(f"{num} {status}")

    # Interactive input
    while True:
        try:
            user_input = int(input("\nEnter a number to check (or 0 to exit): "))
            if user_input == 0:
                break
            result = is_prime(user_input)
            status = "is prime" if result else "is not prime"
            print(f"{user_input} {status}")
        except ValueError:
            print("Please enter a valid integer.")

# Run the main function if the script is executed directly
if __name__ == "__main__":
    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 1

```
Enter a number to check (or 0 to exit): 3
3 is prime

Enter a number to check (or 0 to exit): 1
1 is not prime

Enter a number to check (or 0 to exit):
```

**Here we find prime numbers using functions**

- **Scenario**

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

❖ **Task Description**

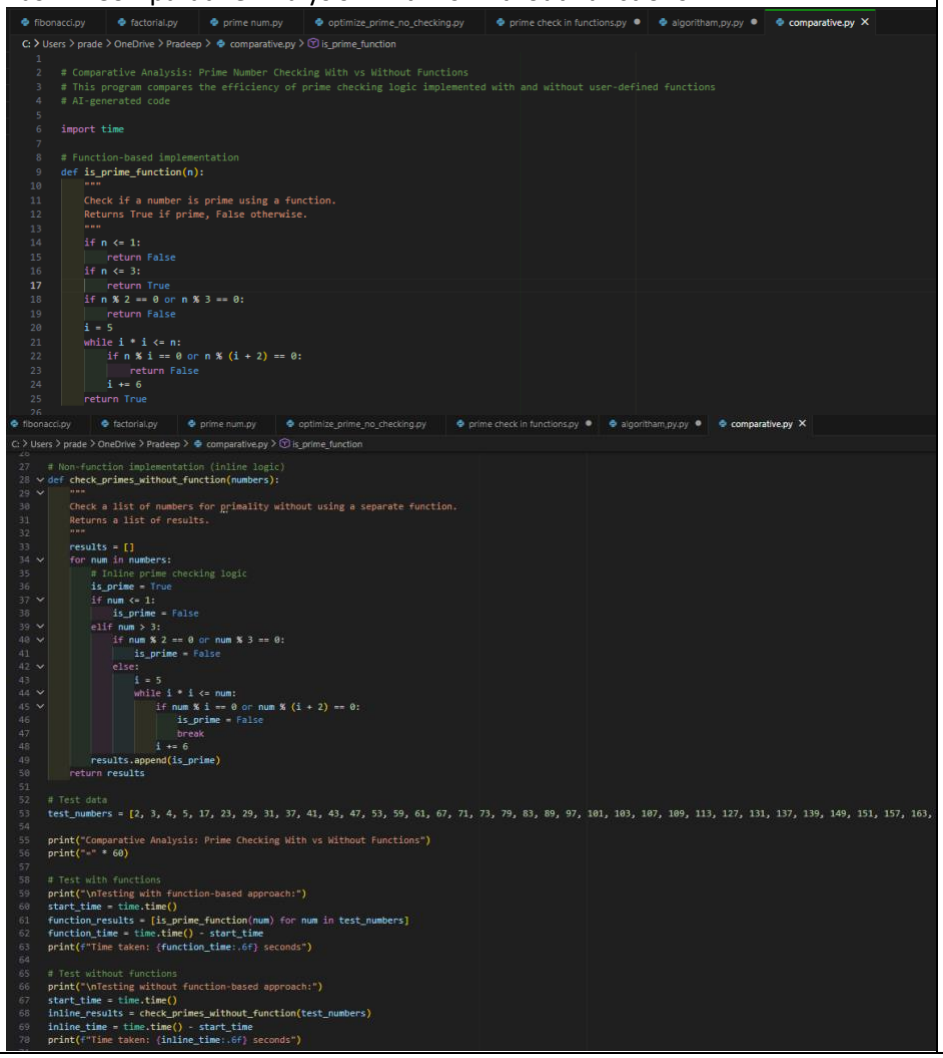
Use GitHub Copilot to generate a function-based Python program that:

- ➢ 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 4: Comparative Analysis –With vs Without Functions



```
# Comparative Analysis: Prime Number Checking With vs Without Functions
# This program compares the efficiency of prime checking logic implemented with and without user-defined functions
# AI-generated code

import time

# Function-based implementation
def is_prime_function(n):
    """
    Check if a number is prime using a function.
    Returns True if prime, False otherwise.
    """
    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
```

```
# Non-function implementation (inline logic)
def check_primes_without_function(numbers):
    """
    Check a list of numbers for primality without using a separate function.
    Returns a list of results.
    """
    results = []
    for num in numbers:
        # Inline prime checking logic
        is_prime = True
        if num <= 1:
            is_prime = False
        elif num > 3:
            if num % 2 == 0 or num % 3 == 0:
                is_prime = False
            else:
                i = 5
                while i * i <= num:
                    if num % i == 0 or num % (i + 2) == 0:
                        is_prime = False
                        break
                    i += 6
        results.append(is_prime)
    return results

# Test data
test_numbers = [2, 3, 4, 5, 17, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163,

print("Comparative Analysis: Prime Checking With vs Without Functions")
print("=" * 60)

# Test with functions
print("\nTesting with function-based approach:")
start_time = time.time()
function_results = [is_prime_function(num) for num in test_numbers]
function_time = time.time() - start_time
print(f"Time taken: {function_time:.6f} seconds")

# Test without functions
print("\nTesting without function-based approach:")
start_time = time.time()
inline_results = check_primes_without_function(test_numbers)
inline_time = time.time() - start_time
print(f"Time taken: {inline_time:.6f} seconds")
```

```python
# Verify results are the same
if function_results == inline_results:
    print("\n✓ Results are identical - both approaches produce the same output")
else:
    print("\n✗ Results differ - there might be an error in implementation")

# Performance comparison
print("Performance Comparison:")
print(f"Function-based time: {function_time:.6f} seconds")
print(f"Inline time: {inline_time:.6f} seconds")
if function_time < inline_time:
    print("Function-based approach is faster")
elif inline_time < function_time:
    print("Inline approach is faster")
else:
    print("Both approaches have similar performance")

# Display some results
print("Sample Results:")
for i in range(min(10, len(test_numbers))):
    num = test_numbers[i]
    print(f"{num}: {'Prime' if function_results[i] else 'Not Prime'}")

print("\nNote: Function-based code is more modular, reusable, and easier to maintain,")
print("while inline code might have slight performance advantages for simple cases.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 2

```
Enter a number to check (or 0 to exit): 5
5 is prime

Enter a number to check (or 0 to exit): 1
1 is not prime
```

❖ **Scenario**

You are participating in a technical review discussion.

❖ **Task Description**

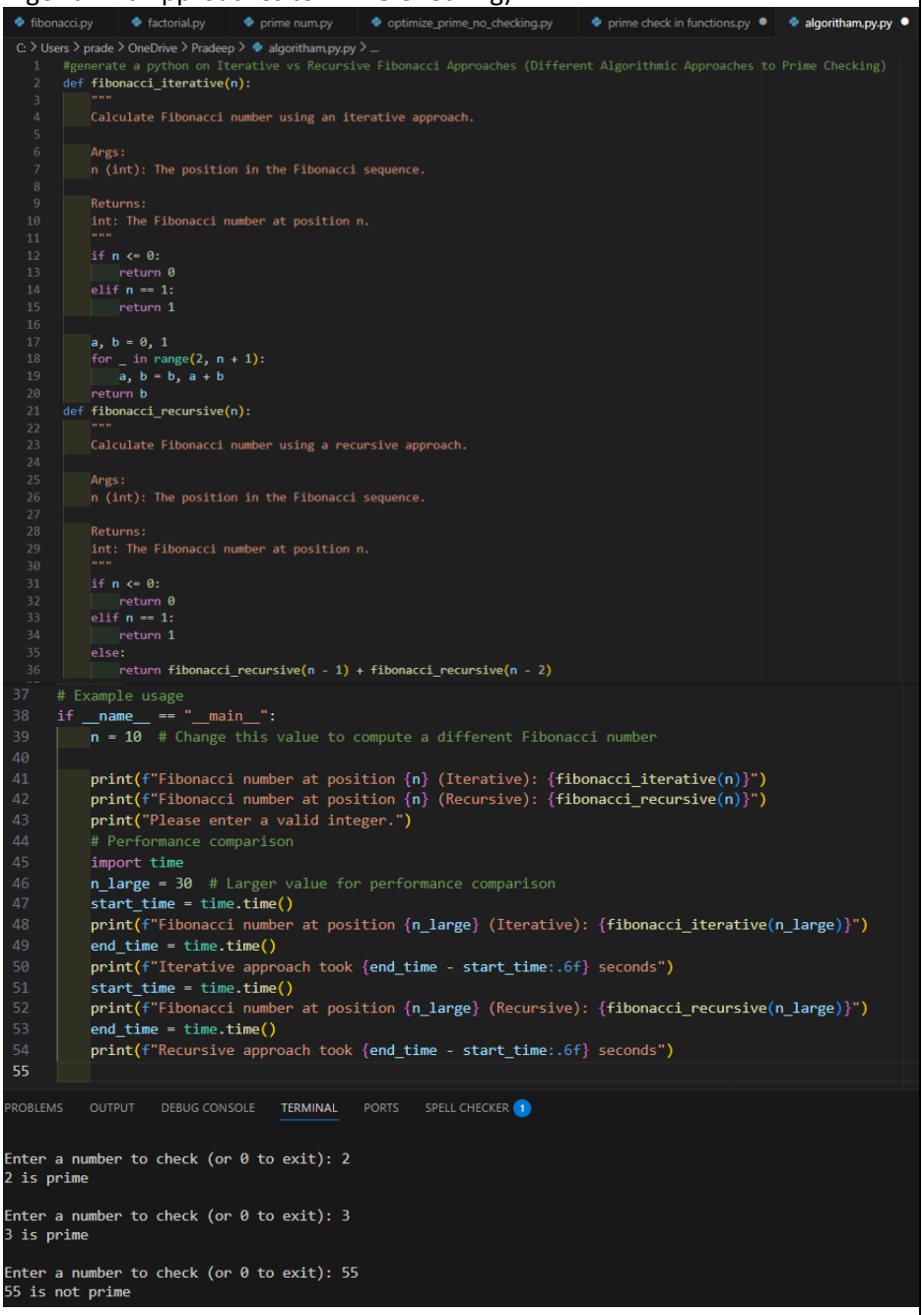
Compare the Copilot-generated programs:

- 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

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**<br>➢ Two correct implementations<br>➢ Comparison discussing:<br>▪ Execution flow<br>▪ Time complexity<br>▪ Performance for large inputs<br>▪ When each approach is appropriate<br><br>**Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots.** | |