

Assignment-1.4

S.Aishwarya
2303A51833

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		<div>Mr. S Naresh Kumar</div> <div>Ms. B. Swathi</div> <div>Dr. Sasanko Shekhar Gantayat</div> <div>Mr. Md Sallauddin</div> <div>Dr. Mathivanan</div> <div>Mr. Y Srikanth</div> <div>Ms. N Shilpa</div> <div>Dr. Rishabh Mittal (Coordinator)</div> <div>Dr. R. Prashant Kumar</div> <div>Mr. Ankushavali MD</div> <div>Mr. B Viswanath</div> <div>Ms. Sujitha Reddy</div> <div>Ms. A. Anitha</div> <div>Ms. M.Madhuri</div> <div>Ms. Katherashala Swetha</div> <div>Ms. Velpula sumalatha</div> <div>Mr. Bingi Raju</div>	
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 – <i>GitHub Copilot and VS Code Integration + Understanding AI-assisted Coding Workflow</i>	Week1 - Monday	

	<p>Lab Objectives:</p> <ul style="list-style-type: none"> ● To install and configure GitHub Copilot in Visual Studio Code. ● To explore AI-assisted code generation using GitHub Copilot. ● To analyze the accuracy and effectiveness of Copilot's code suggestions. ● To understand prompt-based programming using comments and code context <p>Lab Outcomes (LOs): After completing this lab, students will be able to:</p> <ul style="list-style-type: none"> ● Set up GitHub Copilot in VS Code successfully. ● Use inline comments and context to generate code with Copilot. ● Evaluate AI-generated code for correctness and readability. ● Compare code suggestions based on different prompts and programming styles. 	
	<p>Task 0</p> <ul style="list-style-type: none"> ● Install and configure GitHub Copilot in VS Code. Take screenshots of each step. <p>Expected Output</p> <ul style="list-style-type: none"> ● 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 X
C:\Users\prade > prade > OneDrive > Pradeep > prime num.py > ...
1  # AI-Generated Prime Number Check Without Functions
2  # This script checks if a given number is prime without using any functions
3
4  # Get input from user
5  num = int(input("Enter a number to check if it's prime: "))
6
7  # Initialize flag
8  is_prime = True
9
10 # Check if number is less than or equal to 1
11 if num <= 1:
12     is_prime = False
13 else:
14     # Check for factors from 2 to sqrt(num)
15     for i in range(2, int(num**0.5) + 1):
16         if num % i == 0:
17             is_prime = False
18             break
19
20 # Output the result
21 if is_prime:
22     print(f"{num} is a prime number.")
23 else:
24     print(f"{num} is not a prime number.")
25

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)

```
1 def is_prime_optimized(n):
2     """
3     Optimized prime number checking using 6k±1 rule.
4     Checks divisibility only by numbers of the form 6k±1 up to sqrt(n).
5     """
6     if n <= 1:
7         return False
8     if n <= 3:
9         return True
10    if n % 2 == 0 or n % 3 == 0:
11        return False
12
13    # Check for factors of the form 6k±1
14    i = 5
15    while i * i <= n:
16        if n % i == 0 or n % (i + 2) == 0:
17            return False
18        i += 6
19    return True
20
21 # Example usage
22 if __name__ == "__main__":
23     import time
24
25     # Test with some numbers
26     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, 167, 173, 179, 181, 191, 193, 197, 199]
27
28     print("Checking prime numbers with optimized logic:")
29     start_time = time.time()
30     for num in test_numbers:
31         result = is_prime_optimized(num)
32         print(f"{num} is {'prime' if result else 'not prime'}")
33     end_time = time.time()
34     print(f"Total time taken: {end_time - start_time:.4f} seconds")
35
36     # Interactive check
37     while True:
38         try:
39             user_input = int(input("Enter a number to check (or 0 to exit): "))
40             if user_input == 0:
41                 break
42             result = is_prime_optimized(user_input)
43             print(f"{user_input} is {'prime' if result else 'not prime'}")
44         except ValueError:
45             print("Please enter a valid integer.")
```

❖ **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\pradeep> OneDrive\Pradeep> prime check in functions.py
1
2 # AI-assisted Python program for prime number checking using functions
3 # This program defines a function to check if a number is prime and returns a Boolean value
4
5 def is_prime(number):
6     """
7     Check if a given number is prime.
8
9     Args:
10     number (int): The number to check for primality.
11
12     Returns:
13     bool: True if the number is prime, False otherwise.
14     """
15     # Numbers less than or equal to 1 are not prime
16     if number <= 1:
17         return False
18
19     # 2 and 3 are prime numbers
20     if number <= 3:
21         return True
22
23     # Check if the number is divisible by 2 or 3
24     if number % 2 == 0 or number % 3 == 0:
25         return False
26
27     # Check for factors from 5 to sqrt(number), skipping even numbers and multiples of 3
28     i = 5
29     while i * i <= number:
30         if number % i == 0 or number % (i + 2) == 0:
31             return False
32         i += 6
33
34     # If no factors found, the number is prime
35     return True
36
37 # Main function to test the is_prime function
38 def main():
39     """
40     Main function to demonstrate the prime checking functionality.
41
42     # Test cases
43     test_numbers = [2, 3, 4, 5, 17, 23, 29, 31, 37, 41, 43, 47]
44
45     print("Prime number checker (AI-assisted):")
46     for num in test_numbers:
47         result = is_prime(num)
48         status = "is prime" if result else "is not prime"
49         print(f"{num} {status}")
50
51     # Interactive input
52     while True:
53         try:
54             user_input = int(input("\nEnter a number to check (or 0 to exit): "))
55             if user_input == 0:
56                 break
57             result = is_prime(user_input)
58             status = "is prime" if result else "is not prime"
59             print(f"{user_input} {status}")
60         except ValueError:
61             print("Please enter a valid integer.")
62
63     # Run the main function if the script is executed directly
64     if __name__ == "__main__":
65         main()
66
67 PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 1
68
69 Enter a number to check (or 0 to exit): 3
70 3 is prime
71
72 Enter a number to check (or 0 to exit): 1
73 1 is not prime
74
75 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

```
fibonacci.py factorial.py prime_num.py optimize_prime_no_checking.py prime_check_in_functions.py algorithm.py.py comparative.py X
C:\Users\prade> OneDrive > Pradeep > comparative.py > is_prime_function
1
2 # Comparative Analysis: Prime Number Checking With vs Without Functions
3 # This program compares the efficiency of prime checking logic implemented with and without user-defined functions
4 # AI-generated code
5
6 import time
7
8 # Function-based implementation
9 def is_prime_function(n):
10     """
11     Check if a number is prime using a function.
12     Returns True if prime, False otherwise.
13     """
14     if n <= 1:
15         return False
16     if n <= 3:
17         return True
18     if n % 2 == 0 or n % 3 == 0:
19         return False
20     i = 5
21     while i * i <= n:
22         if n % i == 0 or n % (i + 2) == 0:
23             return False
24         i += 6
25     return True
26
27 # Non-function implementation (inline logic)
28 def check_primes_without_function(numbers):
29     """
30     Check a list of numbers for primality without using a separate function.
31     Returns a list of results.
32     """
33     results = []
34     for num in numbers:
35         # Inline prime checking logic
36         is_prime = True
37         if num <= 1:
38             is_prime = False
39         elif num > 3:
40             if num % 2 == 0 or num % 3 == 0:
41                 is_prime = False
42             else:
43                 i = 5
44                 while i * i <= num:
45                     if num % i == 0 or num % (i + 2) == 0:
46                         is_prime = False
47                     break
48                 i += 6
49             results.append(is_prime)
50     return results
51
52 # Test data
53 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]
54
55 print("Comparative Analysis: Prime Checking With vs Without Functions")
56 print("-" * 60)
57
58 # Test with functions
59 print("\nTesting with function-based approach:")
60 start_time = time.time()
61 function_results = [is_prime_function(num) for num in test_numbers]
62 function_time = time.time() - start_time
63 print(f"Time taken: (function_time:.6f) seconds")
64
65 # Test without functions
66 print("\nTesting without function-based approach:")
67 start_time = time.time()
68 inline_results = check_primes_without_function(test_numbers)
69 inline_time = time.time() - start_time
70 print(f"Time taken: (inline_time:.6f) seconds")
```

```

72 # Verify results are the same
73 if function_results == inline_results:
74     print("\n✓ Results are identical - both approaches produce the same output")
75 else:
76     print("\nX Results differ - there might be an error in implementation")
77
78 # Performance comparison
79 print("Performance Comparison:")
80 print(f"Function-based time: {function_time:.6f} seconds")
81 print(f"Inline time: {inline_time:.6f} seconds")
82 if function_time < inline_time:
83     print("Function-based approach is faster")
84 elif inline_time < function_time:
85     print("Inline approach is faster")
86 else:
87     print("Both approaches have similar performance")
88
89 # Display some results
90 print("Sample Results:")
91 for i in range(min(10, len(test_numbers))):
92     num = test_numbers[i]
93     print(f"{num}: {'Prime' if function_results[i] else 'Not Prime'}")
94
95 print("\nNote: Function-based code is more modular, reusable, and easier to maintain,")
96 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)

```
fibonacci.py  factorial.py  prime num.py  optimize_prime_no_checking.py  prime check in functions.py  algorithm.py.py
C:\Users\prade > OneDrive > Pradeep > algorithm.py.py > ...
1  #generate a python on Iterative vs Recursive Fibonacci Approaches (Different Algorithmic Approaches to Prime Checking)
2  def fibonacci_iterative(n):
3      """
4      Calculate Fibonacci number using an iterative approach.
5
6      Args:
7      n (int): The position in the Fibonacci sequence.
8
9      Returns:
10     int: The Fibonacci number at position n.
11     """
12     if n <= 0:
13         return 0
14     elif n == 1:
15         return 1
16
17     a, b = 0, 1
18     for _ in range(2, n + 1):
19         a, b = b, a + b
20     return b
21
22 def fibonacci_recursive(n):
23     """
24     Calculate Fibonacci number using a recursive approach.
25
26     Args:
27     n (int): The position in the Fibonacci sequence.
28
29     Returns:
30     int: The Fibonacci number at position n.
31     """
32     if n <= 0:
33         return 0
34     elif n == 1:
35         return 1
36     else:
37         return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
38
39 # Example usage
40 if __name__ == "__main__":
41     n = 10 # Change this value to compute a different Fibonacci number
42
43     print(f"Fibonacci number at position {n} (Iterative): {fibonacci_iterative(n)}")
44     print(f"Fibonacci number at position {n} (Recursive): {fibonacci_recursive(n)}")
45     print("Please enter a valid integer.")
46     # Performance comparison
47     import time
48     n_large = 30 # Larger value for performance comparison
49     start_time = time.time()
50     print(f"Fibonacci number at position {n_large} (Iterative): {fibonacci_iterative(n_large)}")
51     end_time = time.time()
52     print(f"Iterative approach took {end_time - start_time:.6f} seconds")
53     start_time = time.time()
54     print(f"Fibonacci number at position {n_large} (Recursive): {fibonacci_recursive(n_large)}")
55     end_time = time.time()
56     print(f"Recursive approach took {end_time - start_time:.6f} seconds")
57
58 PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  SPELL CHECKER 1
59
60 Enter a number to check (or 0 to exit): 2
61 2 is prime
62
63 Enter a number to check (or 0 to exit): 3
64 3 is prime
65
66 Enter a number to check (or 0 to exit): 55
67 55 is not prime
```

❖ 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 \sqrt{n})

	<p>❖ Expected Output</p> <ul style="list-style-type: none"> ➤ Two correct implementations ➤ Comparison discussing: <ul style="list-style-type: none"> ▪ Execution flow ▪ Time complexity ▪ Performance for large inputs ▪ When each approach is appropriate <p>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.</p>	
--	--	--