

AI ASSISTED CODING

Hall Ticket No: 2303A51850

Batch:13

Assignment-1.4

Task-1. AI-Generated Logic Without Modularization (Prime Number Check)

Without Functions)

Prompt

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

Code & Output

The screenshot shows a dark-themed instance of Visual Studio Code. The top bar includes standard menu items like File, Edit, Selection, View, Go, Run, etc., followed by a search bar containing "AI CODING". Below the menu is a toolbar with icons for file operations. The main workspace displays a Python file named "LAB1.py". The code itself is as follows:

```
1  #Use GitHub Copilot to generate a Python program that:
2  #> Does not use any user-defined functions
3  #> Checks whether a given number is prime
4  number = int(input("Enter a number: "))#> Accepts user input
5  is_prime = True
6  if number <= 1:
7      is_prime = False
8  for i in range(2, int(number**0.5) + 1):
9      if number % i == 0:#> Implements logic directly in the main code
10         is_prime = False
11         break
12 if is_prime:
13     print(f"{number} is a prime number.")
14 else:
15     print(f"{number} is not a prime number.)
```

Below the code editor is a status bar showing "Ln 15, Col 46" and other file details. The bottom of the screen features a "PROBLEMS" panel which lists several errors and warnings, notably an "IndentationError: unexpected indent" at line 11. To the right of the problems panel is a "TERMINAL" tab showing command-line output from a terminal window. The terminal output includes the command "python.exe" and the user's input "8", followed by the message "8 is not a prime number."

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)

Prompt

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

Code & Ouput :

```
LAB1.py
16 # Task 2: Efficiency & Logic Optimization (Cleanup)
17 #Review the Copilot-generated code from Task 1 and improve it by:
18 # Reducing unnecessary iterations
19 #> Optimizing the loop range (e.g., early termination)
20 #> Improving readability
21 #> Use Copilot prompts like:
22 # "Optimize prime number checking logic"
23 # "Improve efficiency of this code"
24 number = int(input("Enter a number: ")) # accepts user input
25
26 original_is_prime = True
27 if number <= 1:
28     original_is_prime = False
29 else:
30     for i in range(2, int(number**0.5) + 1):
31         if number % i == 0:
32             original_is_prime = False
33             break
34
35 # Task 2: Optimized implementation (no user-defined functions)
36 import time
37 start = time.perf_counter()
38 optimized_is_prime = True
39 if number <= 1:
40     optimized_is_prime = False
41 elif number <= 3: # 2 and 3 are prime
42     optimized_is_prime = True
43 elif number % 2 == 0: # eliminate even numbers early
44     optimized_is_prime = False
45 else:
46     limit = int(number**0.5) + 1
47     i = 3
48     while i <= limit: # check only odd divisors
49         if number % i == 0:
50             optimized_is_prime = False
51             break
52         i += 2
53
54 # Outputs comparing both versions
55 print(f"Original check: {number} is {'prime' if original_is_prime else 'not prime'}")
56 print(f"Optimized check: {number} is {'prime' if optimized_is_prime else 'not prime'}")

PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING> & "C:/Users/Spriha Gajula/anaconda3/envs/spriha1850/python.exe" "c:/Users/Spriha Gajula/Desktop/python.py\LAB1.py"
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING> & "C:/Users/Spriha Gajula/anaconda3/envs/spriha1850/python.exe" "c:/Users/Spriha Gajula/Desktop/python.py\LAB1.py"
Enter a number: 5
Original check: 5 is prime.
Optimized check: 5 is prime.
Optimized check time: 0.000004 seconds
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING>
```

Justification:

The optimized script improves performance by reducing unnecessary iterations and limiting the loop range, enabling faster execution for large

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)

Prompt:

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

Code & output :

The screenshot shows a code editor interface with a dark theme. The main area displays Python code for a prime number checker. The code includes docstrings and comments explaining the logic. The terminal tab at the bottom shows the code being run and a prime number being checked. The sidebar features a 'RECENT SESSIONS' list and an 'Ask about your code' section.

```
File Edit Selection View ... 🔍 Q: AI CODING
LAB1.py •
LAB1.py > i_prime
65  '''Task 3: Modular Design Using AI Assistance (Prime Number Check Using
66  Functions)
67  Scenario
68  The prime-checking logic will be reused across multiple modules.
69  Task Description
70  Use GitHub Copilot to generate a function-based Python program that:
71  ➤ Use a user-defined function to check primality
72  ➤ Returns a Boolean value
73  ➤ Includes meaningful comments (AI-assisted)
74  #= Accepts user input in the main code
75  #= Calls the function and displays the result ...
76 def is_prime(number):
77     """
78     Check if a number is prime.
79     """
80     Parameters:
81     number (int): The number to check for primality.
82     Returns:
83     bool: True if the number is prime, False otherwise.
84     """
85     if number <= 1:
86         return False
87     if number < 3:
88         return True # 2 and 3 are prime numbers
89     if number % 2 == 0:
90         return False # eliminate even numbers early
91     limit = int(number**0.5) + 1
92     for i in range(3, limit, 2): # check only odd divisors
93         if number % i == 0:
94             return False
95     return True
96     #give user input
97     number = int(input("Enter a number: "))
98     #call the function and display the result
99     if is_prime(number):
100         print(f"{number} is a prime number.")
101     else:
102         print(f"{number} is not a prime number.")
103
104
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING> & "C:/Users/Spriha Gajula/anaconda3/envs/sprisha1850/python.exe" "c:/Users/Spriha Gajula/OneDrive/Desktop/python.py\AI CODING\LAB1.py"
Enter a number: 5
5 is a prime number.
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING> & "C:/Users/Spriha Gajula/anaconda3/envs/sprisha1850/python.exe" "c:/Users/Spriha Gajula/OneDrive/Desktop/python.py\AI CODING\LAB1.py"
Enter a number: 4
4 is not a prime number.
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING>
Ln 97, Col 16  Spaces: 4  UTF-8  CRLF  {} Python  3.12.9 (sprisha1850)  #9 Go Live
```

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

Code & Output :

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows a tree view with files like `LAB1.py`, `LAB1.ipynb`, and `Task 4: Comparative Analysis With vs Without Functions`.
- Code Editor:** Displays a Python script for prime number checking. The code uses a for loop to check divisibility from 2 to the square root of the input number. It prints "True" if no divisor is found, and "False" otherwise.
- Terminal:** Shows two command-line sessions in a split terminal:
 - Session 1: `PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING & "C:/Users/Spriha Gajula/anaconda3/envs/sprisha1850/python.exe" "c:/Use rs/sriha Gajula/OneDrive/Desktop/python.py\AI CODING\LAB1.py"`. The user enters "5" and gets the output "True".
 - Session 2: `PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING & "C:/Users/spriha Gajula/anaconda3/envs/sprisha1850/python.exe" "c:/Use rs/sriha Gajula/OneDrive/Desktop/python.py\AI CODING\LAB1.py"`. The user enters "4" and gets the output "False".
- Bottom Status Bar:** Shows file paths, line numbers (Ln 118, Col 44), character counts (Spaces: 4), and file formats (UTF-8, CRLF).
- Right Sidebar:** Features an AI Coding interface with a search bar ("Q AI CODING"), a "CHAT" section with a message icon, and a "RECENT SESSIONS" list. The list includes:
 - prime number checking implementation ... (Completed, Local • 1 day ago)
 - Fibonacci function implementation in Py... (Completed, Local • 1 day ago)
 - Creating a basic HTML file (Completed, Local • 1 day ago)A "Show More" button is also present.
- Bottom Right Panel:** A "Ask about your code" panel with a message icon, stating "All responses may be inaccurate. Generate Agent Instructions to onboard AI onto your codebase."

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)

Prompt: Prime Number Check – Basic vs Optimized Approach

Code & output:

```
LAB1.py
145     # Time complexity
146     # O(n) for large inputs
147     # When each approach is appropriate
148     Note: Results should be submitted as a word document for all tasks in a
149     single document with prompts, comments & code explanation, and output
150     and if required.
151
152     # Basic Divisibility Check Approach
153     number = int(input("Enter a number: ")) # accepts user input
154     is_prime_basic = True
155     if number <= 1:
156         is_prime_basic = False
157         for i in range(2, number):
158             if number % i == 0:
159                 is_prime_basic = False
160                 break
161             if is_prime_basic:
162                 print("Basic Check: {number} is a prime number.")
163             else:
164                 print("Basic Check: {number} is not a prime number.")
165
166     # Optimized Approach (Checking up to √n)
167     is_prime_optimized = True
168     if number <= 1:
169         is_prime_optimized = False
170     elif number <= 3:
171         is_prime_optimized = True
172     else:
173         limit = int(number**0.5) + 1
174         for i in range(3, limit, 2):
175             if number % i == 0:
176                 is_prime_optimized = False
177                 break
178
179     if is_prime_optimized:
180         print("Optimized Check: {number} is a prime number.")
181     else:
182         print("Optimized Check: {number} is not a prime number.")
183
184     # Conclusion Discussion
```

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING> & "C:/Users/Spriha Gajula/anaconda3/envs/sprinha1850/python.exe" "c:/Users/Spriha Gajula/OneDrive/Desktop/python.py\LAB1.py"
G:\LAB1.py
Enter a number: 25
Basic Check: 25 is not a prime number.
Optimized Check: 25 is not a prime number.
PS C:\Users\Spriha Gajula\OneDrive\Desktop\python.py\AI CODING>
```

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 \sqrt{N} because any factor larger than \sqrt{N} must have a corresponding smaller factor.

