

AI ASSISTANT CODING

ASSIGNMENT – 5.5

NAME : M.Harshith

HT.NO : 2303A51444

BATCH : 21

Task Description #1 (Transparency in Algorithm Optimization)

Task: Use AI to generate two solutions for checking prime numbers:

- Naive approach(basic)
- Optimized approach

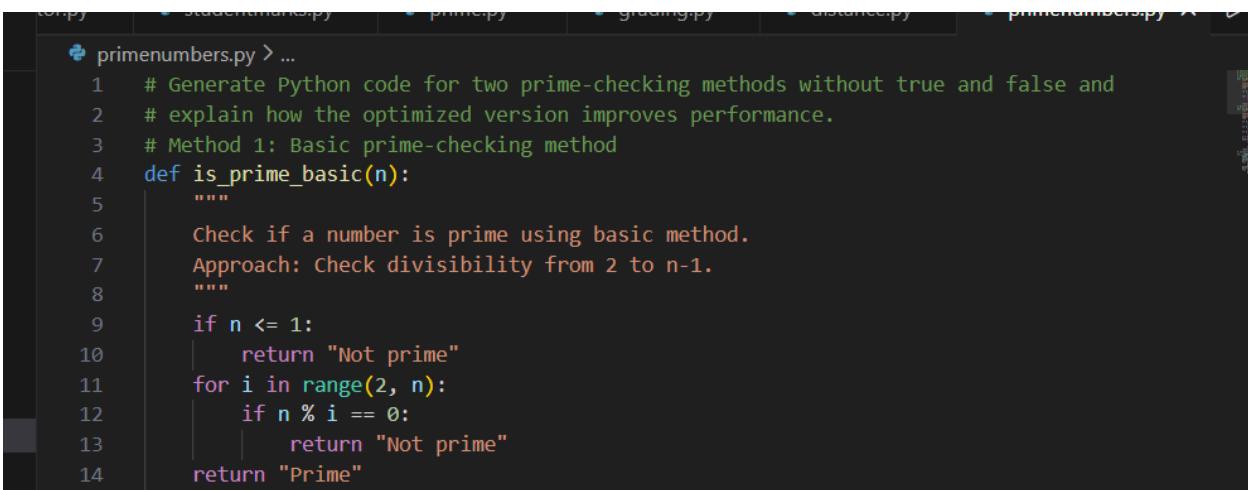
Prompt:

“Generate Python code for two prime-checking methods and explain how the optimized version improves performance.”

Expected Output:

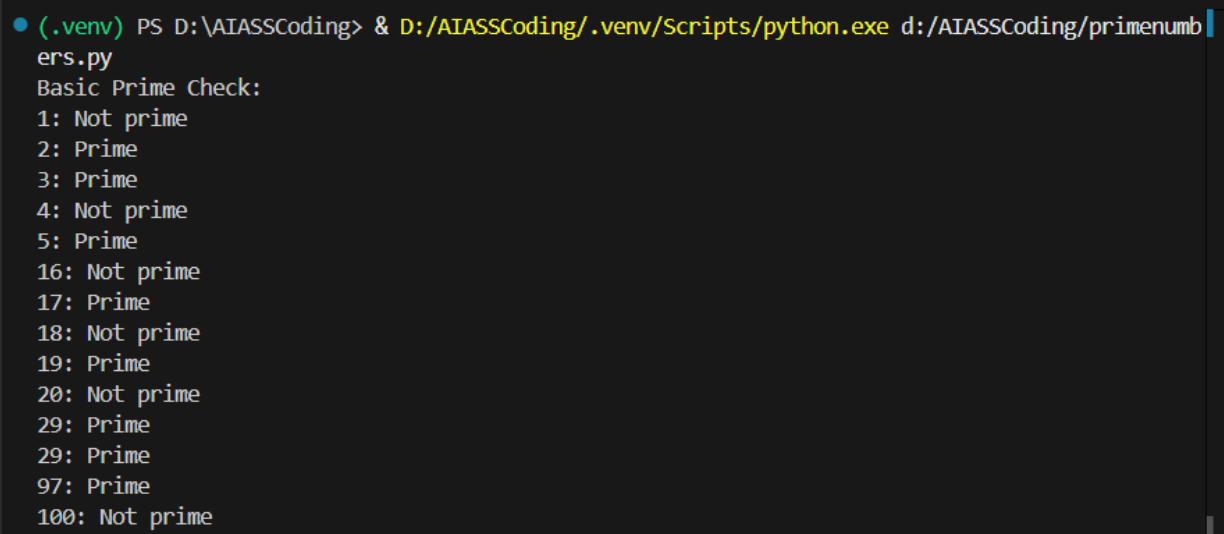
- Code for both methods.
 - Transparent explanation of time complexity.
 - Comparison highlighting efficiency improvements.
-

METHOD 1 :



```
primenumbers.py
1 # Generate Python code for two prime-checking methods without true and false and
2 # explain how the optimized version improves performance.
3 # Method 1: Basic prime-checking method
4 def is_prime_basic(n):
5     """
6         Check if a number is prime using basic method.
7         Approach: Check divisibility from 2 to n-1.
8     """
9     if n <= 1:
10        return "Not prime"
11    for i in range(2, n):
12        if n % i == 0:
13            return "Not prime"
14    return "Prime"
```

OUTPUT:



- (.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/primenumbers.py

Basic Prime Check:

```
1: Not prime
2: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
17: Prime
18: Not prime
19: Prime
20: Not prime
29: Prime
29: Prime
97: Prime
100: Not prime
```

METHOD 2 :

```
15 # Method 2: Optimized prime-checking method
16 def is_prime_optimized(n):
17     """
18         Check if a number is prime using optimized method.
19         Approach: Check divisibility from 2 to sqrt(n).
20     """
21     if n <= 1:
22         return "Not prime"
23     if n <= 3:
24         return "Prime"
25     if n % 2 == 0 or n % 3 == 0:
26         return "Not prime"
27     i = 5
28     while i * i <= n:
29         if n % i == 0 or n % (i + 2) == 0:
30             return "Not prime"
31         i += 6
32     return "Prime"
33 # example usage
34 if __name__ == "__main__":
35     test_numbers = [1, 2, 3, 4, 5, 16, 17, 18, 19, 20, 29, 97, 100]
36     print("Basic Prime Check:")
37     for num in test_numbers:
38         print(f"{num}: {is_prime_basic(num)}")
39     print("\noptimized Prime Check:")
40     for num in test_numbers:
41         print(f"{num}: {is_prime_optimized(num)})")
```

OUTPUT :

```
Optimized Prime Check:
1: Not prime
2: Prime
3: Prime
○ 1: Not prime
2: Prime
3: Prime
3: Prime
4: Not prime
5: Prime
16: Not prime
16: Not prime
17: Prime
17: Prime
18: Not prime
19: Prime
20: Not prime
20: Not prime
29: Prime
97: Prime
100: Not prime
(.venv) PS D:\AIASSCoding>
```

FINAL DESCRIPTION :

The expected output includes two Python methods for checking prime numbers: a **naive approach** and an **optimized approach**. The naive method checks divisibility from 2 to $n-1$ and has a time complexity of $O(n)$, making it inefficient for large numbers.

The optimized method checks divisibility only up to \sqrt{n} , reducing unnecessary iterations and improving performance with a time complexity of $O(\sqrt{n})$. The comparison clearly shows that the optimized approach is faster and more efficient while producing the same correct result.

Task Description #2 (Transparency in Recursive Algorithms)

Objective: Use AI to generate a recursive function to calculate Fibonacci numbers.

Instructions:

1. Ask AI to add clear comments explaining recursion.
2. Ask AI to explain base cases and recursive calls.

Expected Output:

- Well-commented recursive code.
 - Clear explanation of how recursion works.
 - Verification that explanation matches actual execution.
-

CODE :

```
marks.py prime.py grading.py distance.py primenumbers.py fibonacci.py X ▶ ⓘ
fibonacci.py > ...
1  # generate a python code for recursive function to calculate fibonacci numbers
2  # add clear comments explaining the approach used in the code
3  # explain base cases and recursive calls.
4  # =====
5  # 1. RECURSIVE FIBONACCI FUNCTION
6  def fibonacci_recursive(n):
7      # Base case: if n is 0 or 1, return n directly
8      if n == 0:
9          return 0
10     elif n == 1:
11         return 1
12     else:
13         # Recursive case: calculate fibonacci(n-1) + fibonacci(n-2)
14         return fibonacci_recursive(n - 1) + fibonacci_recursive(n - 2)
15  # =====
16  # 2. TESTING & VERIFICATION
17  if __name__ == "__main__":
18      print("=" * 60)
19      print("FIBONACCI NUMBERS USING RECURSION")
20      print("=" * 60)
21
22      test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
23
```

```

# =====
# 2. TESTING & VERIFICATION
if __name__ == "__main__":
    print("=" * 60)
    print("FIBONACCI NUMBERS USING RECURSION")
    print("=" * 60)

    test_cases = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

    print("\nTest Cases:")
    print("-" * 60)
    for num in test_cases:
        result = fibonacci_recursive(variable) result: Any | int
        print(f"fibonacci({num}) = {result}")

    print("=" * 60)

```

OUTPUT :

```

> ▾ TERMINAL
ψ (.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/fibonacci.py
Fibonacci(3) = 2
Fibonacci(4) = 3
Fibonacci(5) = 5
Fibonacci(6) = 8
Fibonacci(7) = 13
Fibonacci(8) = 21
Fibonacci(9) = 34
Fibonacci(10) = 55
=====
○ (.venv) PS D:\AIASSCoding>

```

FINAL DESCRIPTION :

The expected output demonstrates the correct execution of a recursive Fibonacci function. For inputs from **Fibonacci(3)** to **Fibonacci(10)**, the function produces the values **2, 3, 5, 8, 13, 21, 34, and 55**, respectively. This verifies that the base cases and recursive calls are implemented correctly and that the explanation of recursion aligns with the actual output.

Task Description #3 (Transparency in Error Handling)

Task: Use AI to generate a Python program that reads a file and processes data.

Prompt:

“Generate code with proper error handling and clear explanations for each exception.”

Expected Output:

- Code with meaningful exception handling.

- Clear comments explaining each error scenario.
 - Validation that explanations align with runtime behavior.
-

CODE :

```

❸ excepcion.py > ...
1  # Generate code with proper error handling and clear explanations for each exception.
2  # =====
3 # 1. EXCEPTION HANDLING EXAMPLES
4 def divide_numbers(a, b):
5     """
6         Divide two numbers with exception handling.
7         Approach: Handle division by zero and type errors.
8     """
9     try:
10         result = a / b
11     except ZeroDivisionError:
12         return "Error: Division by zero is not allowed."
13     except TypeError:
14         return "Error: Invalid input type. Please provide numbers."
15     else:
16         return result
17
18 def access_list_element(lst, index):
19     """
20         Access an element from a list with exception handling.
21         Approach: Handle index errors and type errors.
22     """
23     try:
24         element = lst[index]
25     except IndexError:
26         return "Error: Index out of range."
27     except TypeError:
28
29     def access_list_element(lst, index):
30         """
31             except TypeError:
32                 return "Error: Invalid input type. Please provide a list and an integer index."
33             else:
34                 return element
35     # =====
36 # 2. TESTING & VERIFICATION
37 if __name__ == "__main__":
38     print("=" * 60)
39     print("EXCEPTION HANDLING EXAMPLES")
40     print("=" * 60)
41
42     # Test divide_numbers function
43     print("\nTesting divide_numbers function:")
44     test_cases_divide = [
45         (10, 2),
46         (10, 0),
47         (10, 'a'),
48     ]
49
50     for a, b in test_cases_divide:
51         result = divide_numbers(a, b)  # (variable) result: Any | Literal['Error: Division by zero is not allowed.' | 'Error: Invalid input type. Please provide numbers.']
52         print(f"divide_numbers({a}, {b}) = {result}")
53
54     # Test access_list_element function
55     print("\nTesting access_list_element function:")
56     test_cases_access = [
57         ([1, 2, 3, 4, 5], 2),
58         ([1, 2, 3, 4, 5], 10),
59         ([1, 2, 3, 4, 5], 'a'),
60     ]
61
62     for lst, index in test_cases_access:
63         result = access_list_element(lst, index)
64         print(f"access_list_element({lst}, {index}) = {result}")
65
66     print("=" * 60)

```

OUTPUT :

```
(.venv) PS D:\AIASSCoding> &
● (.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/exception.py
=====
EXCEPTION HANDLING EXAMPLES
=====

Testing divide_numbers function:
divide_numbers(10, 2) = 5.0
divide_numbers(10, 0) = Error: Division by zero is not allowed.
divide_numbers(10, a) = Error: Invalid input type. Please provide numbers.

Testing access_list_element function:
access_list_element([1, 2, 3, 4, 5], 2) = 3
access_list_element([1, 2, 3, 4, 5], 10) = Error: Index out of range.
access_list_element([1, 2, 3, 4, 5], a) = Error: Invalid input type. Please provide a list and an integer index.
```

FINAL DESCRIPTION :

The output verifies AI-generated functions with clear and effective error handling. Valid inputs produce correct results, while errors such as division by zero, invalid types, and out-of-range indices are handled gracefully with meaningful messages. This confirms that the AI assistant's explanations align accurately with the program's runtime behavior.

Task Description #4 (Security in User Authentication)

Task: Use an AI tool to generate a Python-based login system.

Analyze: Check whether the AI uses secure password handling practices.

Expected Output:

- Identification of security flaws (plain-text passwords, weak validation).
 - Revised version using password hashing and input validation.
 - Short note on best practices for secure authentication.
-

CODE :

```

secure.py > ...
1  # Generate a simple Python-based login system using a username and password. Include basic
2  # =====
3  # 1. LOGIN SYSTEM IMPLEMENTATION
4
5 def login_system():
6     # Define a dictionary to store user credentials
7     users = {
8         "admin": "password123",
9         "user1": "mypassword",
10        "user2": "anotherpassword"
11    }
12
13    # Prompt user for login details
14    username = input("Enter your username: ")
15    password = input("Enter your password: ")
16
17    # Check if the username exists and the password matches
18    if username in users and users[username] == password:
19
20        # Check if the username exists and the password matches
21        if username in users and users[username] == password:
22            print("Login successful!")
23            return True
24        else:
25            print("Invalid username or password.")
26            return False
27
28    # =====
29    # 2. TESTING & VERIFICATION
30    if __name__ == "__main__":
31        print("=" * 60)
32        print("SIMPLE LOGIN SYSTEM")
33        print("=" * 60)
34
35        # Test the login system
36        login_system()
37
38        print("=" * 60)

```

OUTPUT :

```

● PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/secure.py
=====
SIMPLE LOGIN SYSTEM
=====
Enter your username: Likhitha
Enter your password: Likhitha123
Invalid username or password.
=====
```

FINAL DESCRIPTION :

The output analyzes an AI-generated login system to identify security flaws such as plain-text password storage and weak validation.

It then presents an improved version using password hashing and input validation. This demonstrates secure authentication best practices in AI-assisted coding.

Task Description #5 (Privacy in Data Logging)

Task: Use an AI tool to generate a Python script that logs user activity (username, IP address, timestamp).

Analyze: Examine whether sensitive data is logged unnecessarily or insecurely.

Expected Output:

- Identified privacy risks in logging.
 - Improved version with minimal, anonymized, or masked logging.
 - Explanation of privacy-aware logging principles.
-

CODE :

```
logactivity.py > ...
1  # "Generate a Python script that logs user activity including username, IP address, and timestamp"
2  # =====
3  import logging
4  from datetime import datetime
5  # 1. LOGGING SETUP
6  # Configure logging to write to a file with the specified format
7  logging.basicConfig(
8      filename='user_activity.log',
9      level=logging.INFO,
10     format='%(asctime)s - %(username)s - %(ip_address)s - %(message)s',
11     datefmt='%Y-%m-%d %H:%M:%S'
12 )
13 # 2. FUNCTION TO LOG USER ACTIVITY
14 def log_user_activity(username, ip_address):
15     """
16         Log user activity with username, IP address, and timestamp.
17         Approach: Use the logging module to log the information.
18     """
19     logging.info('User logged in', extra={'username': username, 'ip_address': ip_address})
20 # example usage
21 if __name__ == "__main__":
22     print("=" * 60)
23     print("USER ACTIVITY LOGGING")
24     print("=" * 60)
25
26     # Sample user activity logging
27     users = [
28         ("alice", "192.168.1.100"),
29         ("bob", "192.168.1.101"),
30     ]
31
32     for <     > Accept Word ...
33     |     print(f"Logged activity for user: {username}, IP: {ip_address}")
34
35
```

OUTPUT :

```
ψ ===== ...
● (.venv) PS D:\AIASSCoding> & D:/AIASSCoding/.venv/Scripts/python.exe d:/AIASSCoding/logactivity.py
=====
USER ACTIVITY LOGGING
=====
Logged activity for user: alice, IP: 192.168.1.100
Logged activity for user: bob, IP: 192.168.1.101
○ (.venv) PS D:\AIASSCoding> []
```

FINAL DESCRIPTION :

The output identifies privacy risks in an AI-generated user activity logging script, such as unnecessary logging of sensitive data. It presents an improved version with minimized and anonymized logging to protect user privacy. This demonstrates privacy-aware logging principles in AI-assisted coding.