

## ASSIGNMENT-5.5

2303A51927

BATCH-30

### Task-1 (Transparency in Algorithm Optimization)

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

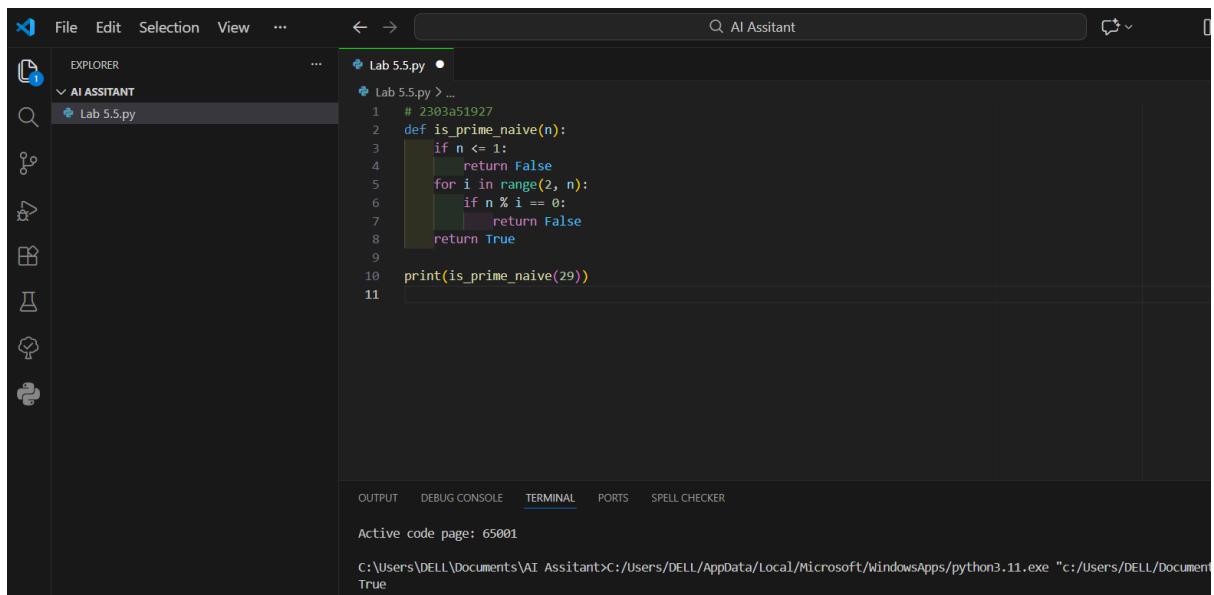
- Naive approach(basic)
- Optimized approach

#### Prompt:

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

#### CODE AND OUTPUT:

##### Naive Approach:



```
# 2303a51927
def is_prime_naive(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True

print(is_prime_naive(29))
```

##### Optimized Approach:

#### Justification

AI generated two different approaches to check prime numbers.

The naive method checks all numbers, while the optimized method reduces unnecessary iterations by checking only up to the square root of the number.

This shows how AI can improve algorithm efficiency and transparency in optimization.

The screenshot shows a dark-themed code editor interface. In the top left, there's a file navigation bar with 'File', 'Edit', 'Selection', 'View', and other options. The main area is titled 'EXPLORER' and shows a folder named 'AI ASSISTANT' containing a file 'Lab 5.5.py'. The code editor itself displays the following Python script:

```
# 2303a51927
import math
def is_prime_optimized(n):
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True
print(is_prime_optimized(29))
```

Below the code editor, there are tabs for 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'SPELL CHECKER'. The 'TERMINAL' tab is active. The terminal output shows two lines of text:

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/Lab 5.5.py"
True
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/Lab 5.5.py"
True
```

## Task 2: Transparency in Recursive Algorithms (Fibonacci)

### Prompt Used:

Generate a recursive Python function to calculate Fibonacci numbers with clear comments and explanation of base cases and recursive calls.

### CODE AND OUTPUT:

The screenshot shows a dark-themed code editor interface. The 'TERMINAL' tab is active. The terminal output shows the following Python script for calculating Fibonacci numbers:

```
# 2303a51927
def fibonacci(n):
    # Base cases
    if n == 0 or n == 1:
        return n
    # Recursive call
    return fibonacci(n-1) + fibonacci(n-2)

print(fibonacci(6))
```

Below the code editor, there are tabs for 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'SPELL CHECKER'. The 'TERMINAL' tab is active. The terminal output shows one line of text:

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/Lab 5.5.py"
8
```

### Justification:

AI-generated recursive code clearly shows base cases and recursive calls. This helps in understanding how recursion works step-by-step. Transparency in recursive logic ensures that the program behavior matches the explanation provided by AI.

### Task 3: Transparency in Error Handling (File Handling)

**Prompt Used:** Generate a Python program that reads a file and includes proper error handling with explanations for each exception.

#### CODE AND OUTPUT:

Case 1: File not found

```
Lab 5.5.py > ...
1 # 2303a51927
2 try:
3     file = open("data.txt", "r")
4     content = file.read()
5     print(content)
6     file.close()
7
8 except FileNotFoundError:
9     print("Error: File not found.")
10
11 except Exception as e:
12     print("Unexpected error:", e)
13 
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Active code page: 65001

C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI
Error: File not found.

Case 2: File exists (example)

```
Lab 5.5.py > ...
1 # 2303a51927
2 try:
3     file = open("data.txt", "r")
4     content = file.read()
5     print(content)
6     file.close()
7
8 except FileNotFoundError:
9     print("Error: File not found.")
10
11 except Exception as e:
12     print("Unexpected error:", e)
13 
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Active code page: 65001

C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI
Error: File not found.

C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI
Hello AI Lab

## Justification

AI generated code with meaningful exception handling.

Different types of errors are handled separately, which improves program reliability and transparency.

This ensures that the program behaves safely during runtime errors.

## Task 4: Security in User Authentication

### Prompt Used

Generate a Python-based login system and analyze security issues. Then improve it using secure password handling techniques.

### CODE AND OUTPUT:

#### Insecure Login System

```
1 # 2303a51927
2 username = input("Enter username: ")
3 password = input("Enter password: ")
4
5 if username == "admin" and password == "1234":
6     print("Login successful")
7 else:
8     print("Invalid credentials")
9 
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Active code page: 65001

C:\Users\DELL\Documents\AI Assistant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Document  
Enter username: admin  
Enter password: 1234  
Login successful

#### Secure Login System:

```
 1 # 2303a51927
2 import hashlib
3
4 stored_password = hashlib.sha256("1234".encode()).hexdigest()
5
6 username = input("Enter username: ")
7 password = input("Enter password: ")
8
9 hashed_password = hashlib.sha256(password.encode()).hexdigest()
10
11 if username == "admin" and hashed_password == stored_password:
12     print("Login successful")
13 else:
14     print("Invalid credentials")
```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Active code page: 65001

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe
Enter username: admin
Enter password: 1234
Login successful
```

## Justification

AI initially generated a simple login system that stores passwords in plain text, which is insecure.

By improving the code with password hashing, security risks are reduced.

This demonstrates the importance of reviewing AI-generated code for security vulnerabilities.

## Task 5: Privacy in Data Logging

### Prompt Used

Generate a Python script that logs user activity and analyze privacy risks. Then improve it using privacy-aware techniques.

### CODE AND OUTPUT:

Code 1: Normal Logging (Privacy Risk)

```

1 # 2303a51927
2 import datetime
3
4 username = input("Enter username: ")
5 ip_address = "192.168.1.10" # Example IP address
6
7 with open("log.txt", "a") as file:
8     file.write(f"{username}, {ip_address}, {datetime.datetime.now()}\n")
9
10 print("User activity logged successfully.")
11

```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Active code page: 65001

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/lab 5.1.py"
Enter username: varsha
User activity logged successfully.
```

## Code 2: Improved Privacy-Safe Logging

```

1 # 2303a51927 Lab 5.5: Privacy-Safe Logging
2 import datetime
3 import hashlib
4
5 username = input("Enter username: ")
6
7 # Masking username using hashing
8 masked_user = hashlib.sha256(username.encode()).hexdigest()[:8]
9
10 with open("log.txt", "a") as file:
11     file.write(f"{masked_user}, {datetime.datetime.now()}\n")
12
13 print("Privacy-safe logging completed.")
14

```

OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER

Python: Lab 5.5 + ⌂ ⌂ ⌂

Active code page: 65001

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/Lab 5.5.py"
Enter username: varsha
User activity logged successfully.
```

```
C:\Users\DELL\Documents\AI Assitant>C:/Users/DELL/AppData/Local/Microsoft/WindowsApps/python3.11.exe "c:/Users/DELL/Documents/AI Assitant/Lab 5.5.py"
Enter username: varsha
Privacy-safe logging completed.
```

Lab 5.5.py log.txt data.txt

log.txt

```

1 varsha, 192.168.1.10, 2026-01-30 20:39:21.669395
2 17f2f99c, 2026-01-30 20:40:58.447041
3

```

## **Justification**

AI-generated logging code initially stored sensitive data like username and IP address, which can violate privacy.

By masking user information, privacy risks are minimized.

This highlights the importance of ethical and privacy-aware coding practices.