

TASK1:

Prompt: Create a simple login system using Python to check whether the credentials are hardcoded in the code or not, passwords are stored or compared in plain text, and insecure logic is used for authentication. Give biased code.

CODE:

```
#create a simple login system using python to check whether the credentials are hardcoded in the code or not,passwords are stored or compared in plain text, and insecure logic is used for authentication. Give biased code.
# Sample hardcoded credentials
hardcoded_username = "admin"
hardcoded_password = "password123"
# Function to simulate login
def login(username, password):
    if username == hardcoded_username and password == hardcoded_password:
        return "Login successful!"
    else:
        return "Login failed! Invalid credentials."
# Test the login function with hardcoded credentials
print(login("admin", "password123")) # Expected: Login successful!
print(login("user", "wrongpassword")) # Expected: Login failed! Invalid credentials.
```

OUTPUT:

```
PS D:\AI assistance coding> cd 'd:\AI assistance coding'; & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '59573' '--' 'D:\AI assistance coding\assignment5.3.py'
Login successful!
Login failed! Invalid credentials.
PS D:\AI assistance coding>
```

JUSTIFICATION: This login system is biased toward simplicity over security. It assumes trusted users and ignores privacy and data protection. Using hardcoded and plain-text passwords can easily lead to unauthorized access and data breaches.

Unbiased prompt: Generate a python code for login system for privacy and data security scenario. Keep a choice that user selects whether the user wants to register or login. Encrypt the data and password. Check whether the credentials are hardcoded. If the user registered then store the passwords and details and check whether the passwords are stored or compared in plain text. Improve the overall security. Generate a efficient and readable code.

CODE:

```
import base64
import hashlib
import getpass
import hmac
import json
import os
import secrets
import sys

USER_FILE = "user_data.json"
ITERATIONS = 200_000

def generate_salt(length: int = 16) -> bytes:
    return secrets.token_bytes(length)

def hash_password(password: str, salt: bytes, iterations: int = ITERATIONS) -> str:
    dk = hashlib.pbkdf2_hmac("sha256", password.encode(), salt, iterations)
    return base64.b64encode(dk).decode()

def verify_password(stored_hash_b64: str, password: str, salt_b64: str, iterations: int) -> bool:
    salt = base64.b64decode(salt_b64)
    derived = hash_password(password, salt, iterations)
    return hmac.compare_digest(derived, stored_hash_b64)
```

OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
Confirm password:  
Registration successful!  
PS D:\AI assistance coding> █
```

JUSTIFICATION:

The unbiased login system avoids hardcoded credentials and does not store passwords in plain text. It uses hashing to protect passwords and hides input while typing. Authentication is based on secure verification instead of direct string comparison. This improves privacy, prevents data leaks, and makes the login process safe and fair.

TASK2:

BAISED PROMPT: Generate a simple Python loan approval system. Use different applicant names and genders in examples. Analyze the code to check if decisions depend on gender or names instead of financial factors. Identify any biased logic, discuss fairness issues, and rewrite the code to remove bias.baised version

CODE:

```
#Generate a simple Python loan approval system.  
#Use different applicant names and genders in examples.  
#Analyze the code to check if decisions depend on gender or names instead of financial factors.  
#Identify any biased logic, discuss fairness issues, and rewrite the code to remove bias.  
#Finally, suggest methods to ensure fair and unbiased decisions.  
def approve_loan(name, gender, income, credit_score):  
    # Biased logic (wrong approach)  
    if gender.lower() == "male":  
        bonus = 20  
    else:  
        bonus = 0  
  
    final_score = credit_score + bonus  
  
    if income > 30000 and final_score > 650:  
        return "Loan Approved"  
    else:  
        return "Loan Rejected"  
  
# Test cases  
applicants = [  
    ("Rahul", "male", 40000, 640),  
    ("Anita", "female", 40000, 640),  
    ("John", "male", 28000, 700),  
    ♀ ("Priya", "female", 28000, 700),  
]  
  
for a in applicants:  
    print(a[0], approve_loan(*a))
```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\AI assistance coding> D:\AI assistance coding , a C:\Users\Saranya\appdata\local\temp\user7000\unwindapps\pythons\11.exe C:\Users\Saranya\vscode\ext
nsions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52543' '--' 'D:\AI assistance coding\assignments.3.py'
Rahul Loan Approved
Anita Loan Rejected
John Loan Rejected
Priya Loan Rejected
PS D:\AI assistance coding> 
```

JUSTIFICATION: The first program demonstrates how bias can be unintentionally introduced into AI-generated decision systems. In this code, gender is used as a factor in loan approval by adding a bonus score for male applicants. Even when applicants have the same income and credit score, males receive an unfair advantage. This shows how using irrelevant personal attributes such as gender can lead to discriminatory outcomes. The purpose of this code is to simulate biased logic so that its impact can be observed and analyzed.

UNBIASED PROMPT: Generate a simple Python loan approval system. Use different applicant names and genders in examples. Analyze the code to check if decisions depend on gender or names instead of financial factors. Identify any biased logic, discuss fairness issues, and rewrite the code to remove bias.unbiased version

CODE:

```
#Generate a simple Python loan approval system.
#Use different applicant names and genders in examples.
#Analyze the code to check if decisions depend on gender or names instead of financial factors.
#Identify any biased logic, discuss fairness issues, and rewrite the code to remove bias.
#unbiased version

# Fair Loan Approval System

def approve_loan(income, credit_score):
    # Fair logic (no gender/name used)
    if income >= 30000 and credit_score >= 650:
        return "Loan Approved"
    else:
        return "Loan Rejected"

applicants = [
    ("Rahul", "male", 40000, 640),
    ("Anita", "female", 40000, 640),
    ("John", "male", 40000, 640),
    ("Priya", "female", 40000, 640),
]
print("Fair Loan System Output:\n")

for name, gender, income, credit_score in applicants:
    print(name, "-", approve_loan(income, credit_score))
```

OUTPUT:

```
nsions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60301' '--' 'D:\AI assistance coding\assignment5.3.py'
Fair Loan System Output:

Rahul - Loan Rejected
Anita - Loan Rejected
John - Loan Rejected
Priya - Loan Rejected
PS D:\AI assistance coding> 
```

JUSTIFICATION: The second program removes all sensitive and irrelevant attributes such as name and gender from the decision logic. Loan approval is based only on legitimate financial factors like income and credit score. This ensures that all applicants with the same financial profile receive the same decision. The goal of this code is to demonstrate how bias can be reduced by designing systems that rely only on relevant data, leading to fair, transparent, and ethical AI decisions.

TASK3:

Baised prompt: Generate binary search code quickly without much explanation. Keep it short and skip comments. Assume users already know recursion and Python.

CODE:

```
176 #Generate binary search code quickly without much explanation. Keep it short and skip comments. Assume users already know recursion and Python.
177 def binary_search(arr, target, low, high):
178     if low > high:
179         return -1
180     mid = (low + high) // 2
181     if arr[mid] == target:
182         return mid
183     elif arr[mid] < target:
184         return binary_search(arr, target, mid + 1, high)
185     else:
186         return binary_search(arr, target, low, mid - 1)
187 # Example usage
188 arr = [1, 3, 5, 7, 9, 11, 13, 15]
189 target = 7
190 result = binary_search(arr, target, 0, len(arr) - 1)
191 print(f"Element found at index: {result}" if result != -1 else "Element not found")
192 
```

OUTPUT:

```
PS D:\AI assistance coding> & 'c:\Users\anja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\anja\.vscode\extensions\ms-python.debugpy-2025.18.0
● -win32-x64\bundled\libs\debugpy\launcher' '64119' '--' 'D:\AI assistance coding\assignment5.3.py'
Element found at index: 3
○ PS D:\AI assistance coding>
```

JUSTIFICATION:

It focuses only on working output, skips comments and explanation, and assumes the user already understands recursion. This makes the code less transparent, harder for beginners, and difficult to verify.

UNBAISED PROMPT:

Generate a Python program that implements binary search using recursion on a sorted list. Include clear inline comments, beginner-friendly variable names, a step-by-step explanation of the base case and recursive case, and a small example for demonstration.

CODE:

```

#Generate a Python program that implements binary search using recursion on a sorted list. Include clear inline comments, beginner-friendly variable names, and appropriate whitespace.
def binary_search(sorted_list, target_value, low_index, high_index):
    # Base case: If the low index exceeds the high index, the target is not in the list
    if low_index > high_index:
        return -1 # Target not found

    # Calculate the middle index
    middle_index = (low_index + high_index) // 2

    # Check if the middle element is the target
    if sorted_list[middle_index] == target_value:
        return middle_index # Target found at middle index
    # If the target is greater than the middle element, search in the right half
    elif sorted_list[middle_index] < target_value:
        return binary_search(sorted_list, target_value, middle_index + 1, high_index)
    # If the target is less than the middle element, search in the left half
    else:
        return binary_search(sorted_list, target_value, low_index, middle_index - 1)

# Example usage
if __name__ == "__main__":
    example_list = [2, 4, 6, 8, 10, 12, 14, 16]
    target = 10
    result_index = binary_search(example_list, target, 0, len(example_list) - 1)

    if result_index != -1:
        print(f"Element found at index: {result_index}")
    else:
        print("Element not found")

```

OUTPUT:

```

PS D:\AI assistance coding> d;; cd 'd:\AI assistance coding'; & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '64411' '--' 'D:\AI assistance coding\assignment5.3.py'
Element found at index: 4
PS D:\AI assistance coding>

```

JUSTIFICATION: It includes clear comments, explains the base case and recursive case, and uses simple logic and names. This makes the code transparent, beginner-friendly, and easy for humans to understand and trust.

TASK4:

BAISED PROMPT: Generate a Python job applicant scoring system. Use skills, experience, education, and also consider the applicant's name and gender to adjust the final score.

CODE:

```

#Generate a Python job applicant scoring system. Use skills, experience, education, and also consider the applicant's name and gender to adjust the score.
#baised version
def score_applicant(name, gender, skills, experience, education):
    score = 0
    score += skills * 5
    score += experience * 3
    score += education * 2
    if gender.lower() == "male":
        score += 10
    else:
        score += 5
    if name.lower() in ["john", "michael", "robert"]:
        score += 5
    return score

# Test cases
applicants = [
    ("John", "male", 8, 5, 4),
    ("Anita", "female", 8, 5, 4),
    ("Michael", "male", 7, 6, 5),
    ("Priya", "female", 7, 6, 5),
]
for a in applicants:
    print(a[0], "Score:", score_applicant(*a))

```

OUTPUT:

```
PS D:\AI assistance coding> d;; cd 'd:\AI assistance coding'; & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '50406' '--' 'D:\AI assistance coding\assignment5.3.py'
John Score: 78
Anita Score: 68
Michael Score: 78
Priya Score: 68
PS D:\AI assistance coding>
```

JUSTIFICATION:

The biased code uses gender, name, or other unrelated personal features while calculating the applicant's score. These factors have no connection to job performance, so they can unfairly increase or decrease scores. This makes the system subjective, unethical, and discriminatory, and it can lead to wrong hiring decisions.

UNBAISED PROMPT: Generate a Python job applicant scoring system using only job-relevant features such as skills, years of experience, and education level. Do not use gender, name, age, or any personal attributes. Include fair and transparent scoring logic with comments.

CODE:

```
#Generate a Python job applicant scoring system using only job-relevant features such as skills, years of experience, and education level. Do not use gender, name, age, or any personal attributes.
# Fair Job Applicant Scoring System
def score_applicant.skills, experience, education):
    score = 0
    score += skills * 5 # Each skill point adds 5 to the score
    score += experience * 3 # Each year of experience adds 3 to the score
    score += education * 2 # Each level of education adds 2 to the score
    return score
# Test cases
applicants = [
    (8, 5, 4),
    (8, 5, 4),
    (7, 6, 5),
    (7, 6, 5),
]
print("\nFair Job Applicant Scoring System Output:")
for idx, a in enumerate(applicants, start=1):
    print(f"Applicant {idx} Score:", score_applicant(*a))
```

OUTPUT:

```
PS D:\AI assistance coding> d;; cd 'd:\AI assistance coding'; & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '60022' '--' 'D:\AI assistance coding\assignment5.3.py'
Fair Job Applicant Scoring System Output:
Applicant 1 Score: 63
Applicant 2 Score: 63
Applicant 3 Score: 63
Applicant 4 Score: 63
PS D:\AI assistance coding>
```

JUSTIFICATION:

The unbiased code calculates scores using only job-relevant features like skills, experience, and education. It ignores personal attributes such as gender or name. This keeps the logic objective, fair, and transparent, ensuring that all applicants are evaluated equally based on merit.

TASK5:

Baised prompt: Generate a Python program to process employee details using variables like male, female, husband_name, wife_name and apply different rules based on gender.

CODE:

```
#Generate a Python program to process employee details using variables like male, female, husband_name, wife_name and apply different rules based on
def process_employee_details(name , gender, spouse_name):
    if gender.lower() == "male":
        print(f"Employee Name: {name}")
        print(f"Gender: Male ")
        print(f"Husband Name: N/A")
        print(f"Wife Name: {spouse_name}")
    elif gender.lower() == "female":
        print(f"Employee Name: {name}")
        print(f"Gender: Female")
        print(f"Husband Name: {spouse_name}")
        print(f"Wife Name: N/A")
    else:
        print("Invalid gender specified.")
# Test cases
employees = [
    ("Rahul", "male", "Anita"),
    ("Anita", "female", "Rahul"),
]
for emp in employees:
    print("\nEmployee Details:")
    process_employee_details(*emp)
```

OUTPUT:



```
Employee Details:
Employee Name: Rahul
Gender: Male
Husband Name: N/A
Wife Name: Anita

Employee Details:
Employee Name: Anita
Gender: Female
Husband Name: Rahul
```

Ln 284, Col 1 Spaces: 4 UTF-8 CRLF [] Python 3.11.9 (Microsoft Store)

JUSTIFICATION:

The biased code:

- Uses gender-specific variables like male, female, husband_name, wife_name.
- Makes assumptions based on identity, such as different logic for men and women.
- Includes non-inclusive naming that excludes non-binary or unspecified users.
- Applies conditions unrelated to job performance.

This makes the system discriminatory, unfair, and unethical, and it violates inclusive software design principles.

UNBIASED PROMPT:

Generate a Python program to process employee details using gender-neutral variable names and logic. Avoid gender-based conditions unless absolutely required and ensure inclusive, respectful coding practices.

CODE:

```
process_employee_details(*emp)
#Generate a Python program to process employee details using gender-neutral variable names and logic. Avoid gender-based conditions unless absolutely
def process_employee_details(name, spouse_name):
    print(f"Employee Name: {name}")
    print(f"Spouse Name: {spouse_name}")
# Test cases
employees = [
    ("Rahul", "Anita"),
    ("Anita", "Rahul"),
]
for emp in employees:
    print("\nEmployee Details:")
    process_employee_details(*emp)
```

OUTPUT:

```
PS D:\AI assistance coding> d:; cd 'd:\AI assistance coding'; & 'c:\Users\sanja\AppData\Local\Microsoft\WindowsApps\python3.11.exe' 'c:\Users\sanja\vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '52477' '--' 'D:\AI assistance coding\assignment5.3.py'

Employee Details:
Employee Name: Rahul
Spouse Name: Anita

Employee Details:
Employee Name: Anita
Spouse Name: Rahul
PS D:\AI assistance coding>
```

JUSTIFICATION:

The unbiased code:

- Uses gender-neutral variables like employee_name, role, experience, department.
- Avoids gender-based conditions unless necessary.
- Treats all users equally with the same logic.
- Improves fairness, accessibility, and inclusiveness.

This ensures the program is ethical, transparent, and respectful of all identities.