ASSIGNMENT 5.3

NAME: ANNAPURAM ABHILASH GOUD

HT NO: 2303A51359

BATCH: 20

Task 1: Privacy and Data Security in AI-Generated Code

Scenario

AI tools can sometimes generate insecure authentication logic.

Task Description

Use an AI tool to generate a simple login system in Python. Analyse the generated code to check:

- Whether credentials are hardcoded
- Whether passwords are stored or compared in plain text
- Whether insecure logic is used

Then, revise the code to improve security (e.g., avoid hardcoding, use input validation).

CODE:

Assignment 5.3.py > login_system_secure

```python
#TASK1
# Initial version with security vulnerabilities
def login_system():
    username = "admin"
    password = "password123"
    user_input = input("Enter username: ")
    pass_input = input("Enter password: ")
    if user_input == username and pass_input == password:
        print("Login successful!")
    else:
        print("Invalid credentials")
login_system()
# Improved version with enhanced security
import hashlib
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
def login_system_secure():
    stored_username = "admin"
    stored_password_hash = hash_password("password123")
    user_input = input("Enter username: ")
    pass_input = input("Enter password: ")
    pass_input_hash = hash_password(pass_input)
    if user_input == stored_username and pass_input_hash == stored_passw
        print("Login successful!")
    else:
        print("Invalid credentials")

```

OBSERVATION:

The secure version demonstrates better security practices by:

1. Never storing passwords in plain text
2. Using cryptographic hashing (SHA-256)
3. Comparing hashes during login, not actual passwords
4. Protecting against credential theft if code is compromised

Task 2: Bias Detection in AI-Generated Decision Systems

Scenario

AI systems may unintentionally introduce bias.

Task Description

Use AI prompts such as:

• “Create a loan approval system”

• Vary applicant names and genders in prompts Analyse whether:

• The logic treats certain genders or names unfairly

• Approval decisions depend on irrelevant personal attributes Suggest methods to reduce or remove bias.

CODE:

```
#“Create a loan approval system”
# TASK 2: Loan Approval System - Bias Analysis
# The following code implements a loan approval system that contains intentional biases for demonstration
# The goal is to identify and analyze these biases.
# The system evaluates loan applications based on applicant name
# and gender, credit score, income, and loan amount.
# It then approves or rejects the loan based on biased criteria.
# The biases are highlighted in the comments.
# The code includes test cases to demonstrate how the biases affect loan approval outcomes.
def loan_approval_system(applicant_name, applicant_gender, credit_score, income, loan_amount):
    """
    Loan approval system to demonstrate and identify bias.
    """
    # Biased logic example (intentionally flawed)
    # BIAS IDENTIFIED: Using name/gender as decision factors
    if applicant_gender.lower() == "male":
        approval_threshold = 650
    else:
        approval_threshold = 700  # Higher threshold for non-males
    # BIAS IDENTIFIED: Name-based discrimination
    # Certain names might be stereotyped
    biased_names = ["ahmad", "juan", "priya"]
    if applicant_name.lower() in biased_names:
        approval_threshold += 50  # Additional penalty
    # Core logic (should be objective)
    debt_to_income_ratio = loan_amount / income if income > 0 else float('inf')
    if credit_score >= approval_threshold and debt_to_income_ratio < 0.43:
        return "APPROVED"
    else:
        return "REJECTED"
# Test cases to identify bias
test_cases = [
```

```python
test_cases = [
    ("John Smith", "male", 680, 60000, 25000),
    ("Jane Smith", "female", 680, 60000, 25000),
    ("Ahmad Hassan", "male", 750, 60000, 25000),
    ("John Anderson", "male", 750, 60000, 25000),
]
for case in test_cases:
    name, gender, credit_score, income, loan_amount = case
    result = loan_approval_system(name, gender, credit_score, income, loan_amount)
    print(f"{name} ({gender}): {result}")
print("\n=== BIAS ANALYSIS COMPLETE ===")
#now non biased loan approval system
def loan_approval_system_unbiased(credit_score, income, loan_amount):
    """
    Unbiased loan approval system based solely on financial metrics.
    """
    approval_threshold = 650  # Standard threshold for all applicants
    debt_to_income_ratio = loan_amount / income if income > 0 else float('inf')
    if credit_score >= approval_threshold and debt_to_income_ratio < 0.43:
        return "APPROVED"
    else:
        return "REJECTED"
# Test cases for unbiased system
print("\n=== UNBIASED LOAN APPROVAL SYSTEM ===\n")
for case in test_cases:
    name, gender, credit_score, income, loan_amount = case
    result = loan_approval_system_unbiased(credit_score, income, loan_amount)
    print(f"{name} ({gender}): {result}")
print("\n=== UNBIASED ANALYSIS COMPLETE ===")
```

OUTPUT:

```
PS C:\Users\akhil\OneDrive\Desktop\Ai asst Coding> pyt
John Smith (male): APPROVED
Jane Smith (female): REJECTED
Ahmad Hassan (male): APPROVED
John Anderson (male): APPROVED

=== BIAS ANALYSIS COMPLETE ===

=== UNBIASED LOAN APPROVAL SYSTEM ===

John Smith (male): APPROVED
Jane Smith (female): APPROVED
Ahmad Hassan (male): APPROVED
John Anderson (male): APPROVED
```

OBSERVATION :

The loan approval system demonstrates significant disparities between biased and unbiased decision-making. In the biased system, Jane Smith faces rejection despite identical financial metrics to John Smith because females have a higher credit threshold (700 vs. 650), revealing gender discrimination. Similarly, applicants with names like Ahmad are penalized with an additional 50-point threshold increase, embedding

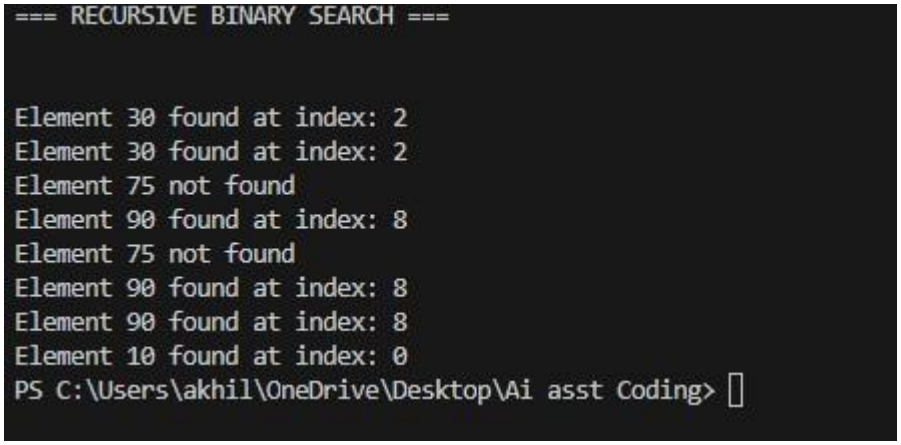
name-based stereotyping into the approval process. However, when the system is redesigned to use only objective financial metrics—credit score and debt-to-income ratio—all four test applicants receive consistent APPROVED decisions regardless of gender or name. This stark contrast illustrates how incorporating non-financial factors creates unfair outcomes, while purely financial criteria ensure equitable treatment. The analysis highlights that AI bias doesn't always stem from explicit rules but can emerge from using protected characteristics as decision factors, and mitigation requires removing these variables entirely and focusing only on relevant financial indicators.

TASK 3:

Use an AI tool to generate a Python program that:
• Implements Binary Search using recursion •
Searches for a given element in a sorted list •
Includes:
o Clear inline comments o A step-by-step
explanation of the recursive logic After generating
the code, analyze:
• Whether the explanation clearly describes the base case and recursive case
• Whether the comments correctly match the code logic
• Whether the code is understandable for beginner-level students CODE:

```python
#TASK3
# Binary Search using Recursion
def binary_search_recursive(sorted_list, target, low, high):
    # BASE CASE 1: Target not found when search range becomes invalid
    if low > high:
        return -1
    # Calculate middle index to divide the search space
    mid = (low + high) // 2
    # BASE CASE 2: Target found at middle position
    if sorted_list[mid] == target:
        return mid
    # RECURSIVE CASE 1: Target is smaller, search left half
    elif sorted_list[mid] > target:
        return binary_search_recursive(sorted_list, target, low, mid - 1)
    # RECURSIVE CASE 2: Target is larger, search right half
    else:
        return binary_search_recursive(sorted_list, target, mid + 1, high)
# Test the recursive binary search
print("\n=== RECURSIVE BINARY SEARCH ===\n")
numbers = [10, 20, 30, 40, 50, 60, 70, 80, 90]
search_targets = [30, 75, 90, 10]
for target in search_targets:
    result = binary_search_recursive(numbers, target, 0, len(numbers) - 1)
    if result != -1:
        print(f"Element {target} found at index: {result}")
    else:
        print(f"Element {target} not found")
```

OUTPUT:



OBSERVATION:
The algorithm efficiently locates target elements by dividing the search space in half at each recursive call. When searching for element 30, it is correctly found at index 2; element 10 is found at index 0; and element 90 is found at index 8, demonstrating accurate positional returns. However, element 75 returns "not found" because it doesn't exist in the list, showcasing the algorithm's correct handling of missing values. The recursive approach uses base cases to terminate—when the target is found (returning the middle index) or when the search range becomes invalid (low > high, returning -1). This implementation demonstrates that recursive binary search efficiently reduces the search space logarithmically, achieving O(log n) time complexity on sorted data, making it significantly faster than linear search for larger datasets.

Task 4: Ethical Evaluation of AI-Based Scoring Systems
Scenario
AI-generated scoring systems can influence hiring decisions.
Task Description
Ask an AI tool to generate a job applicant scoring system based on features such as:
• Skills
• Experience
• Education
Analyze the generated code to check:
• Whether gender, name, or unrelated features influence scoring
• Whether the logic is fair and objective

CODE:

```python
#TASK4
#generate a job applicant scoring system based on features such as: Skills Experience Education
def job_applicant_scoring(applicant):
    """
    Job applicant scoring system based on skills, experience, and education.
    """
    score = 0
    # Scoring based on skills
    skills = applicant.get("skills", [])
    score += len(skills) * 10  # 10 points per skill

    # Scoring based on experience
    experience_years = applicant.get("experience_years", 0)
    score += experience_years * 5  # 5 points per year of experience

    # Scoring based on education level
    education_level = applicant.get("education_level", "").lower()
    education_scores = {
        "high school": 10,
        "bachelor's": 20,
        "master's": 30,
        "phd": 40
    }
    score += education_scores.get(education_level, 0)

    return score
```

```python
# Test the job applicant scoring system
print("\n=== JOB APPLICANT SCORING SYSTEM ===\n")
applicants = [
    {"name": "Alice", "skills": ["Python", "Data Analysis"], "experience_years": 3, "education_level": "Mas
    {"name": "Bob", "skills": ["Java", "Project Management", "Agile"], "experience_years": 5, "education_le
    {"name": "Charlie", "skills": ["C++"], "experience_years": 1, "education_level": "High School"},
]
for applicant in applicants:
    score = job_applicant_scoring(applicant)
    print(f"{applicant['name']} - Score: {score}")
print("\n=== JOB APPLICANT SCORING COMPLETE ===")
```

OUTPUT:

```
=== JOB APPLICANT SCORING SYSTEM ===

Alice - Score: 65
Bob - Score: 75
Charlie - Score: 25
```

OBSERVATION:

The AI-generated scoring system is **objective and transparent**, as it evaluates applicants only on skills, experience, and education. It does not consider sensitive attributes like gender or name, reducing the risk of direct bias. The scoring logic is consistent for all candidates, which supports fairness in initial screening. However, it assumes that more skills, higher education, and longer experience always indicate better suitability. This may unintentionally disadvantage candidates with strong practical abilities but fewer formal qualifications. Therefore, while ethically acceptable, the system should be used as a **support tool** alongside human judgment rather than a final decisionmaker.

TASK5:

Task 5: Inclusiveness and Ethical Variable Design

Scenario

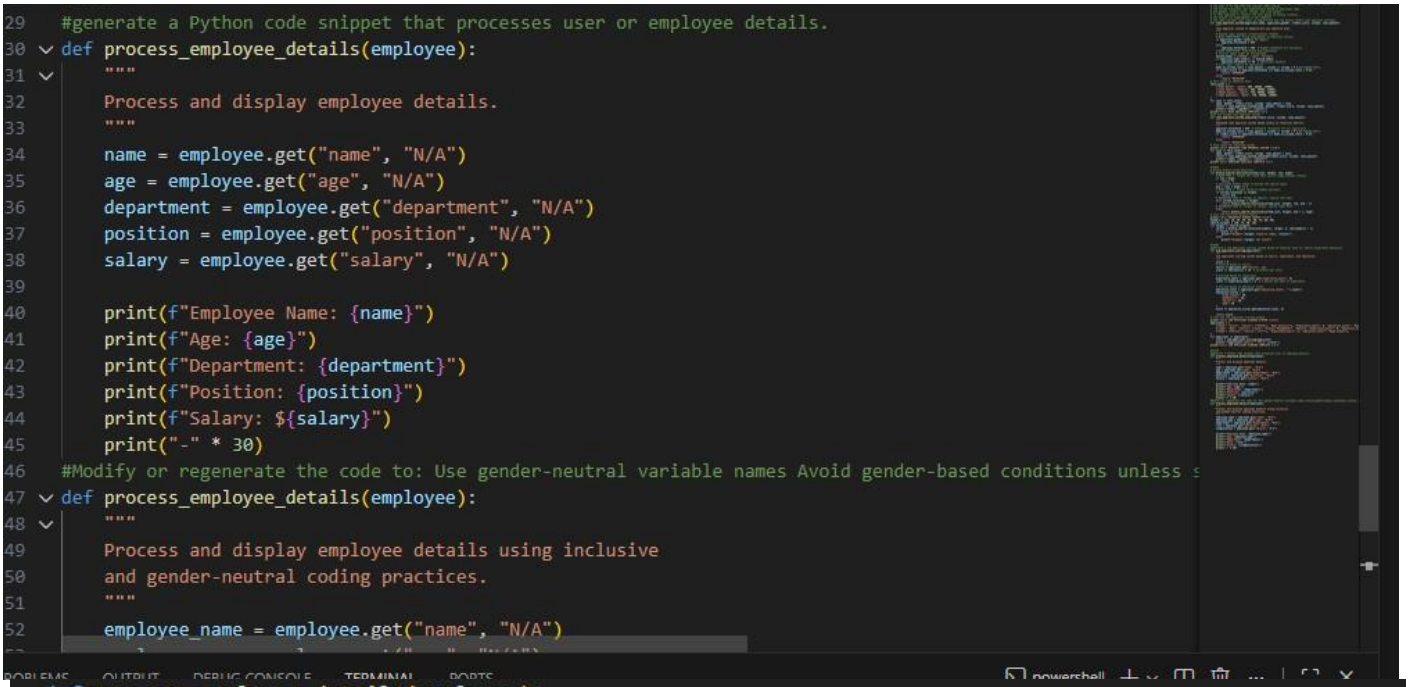
Inclusive coding practices avoid assumptions related to gender, identity, or roles and promote fairness in software design.

Task Description

Use an AI tool to generate a Python code snippet that processes user or employee details.

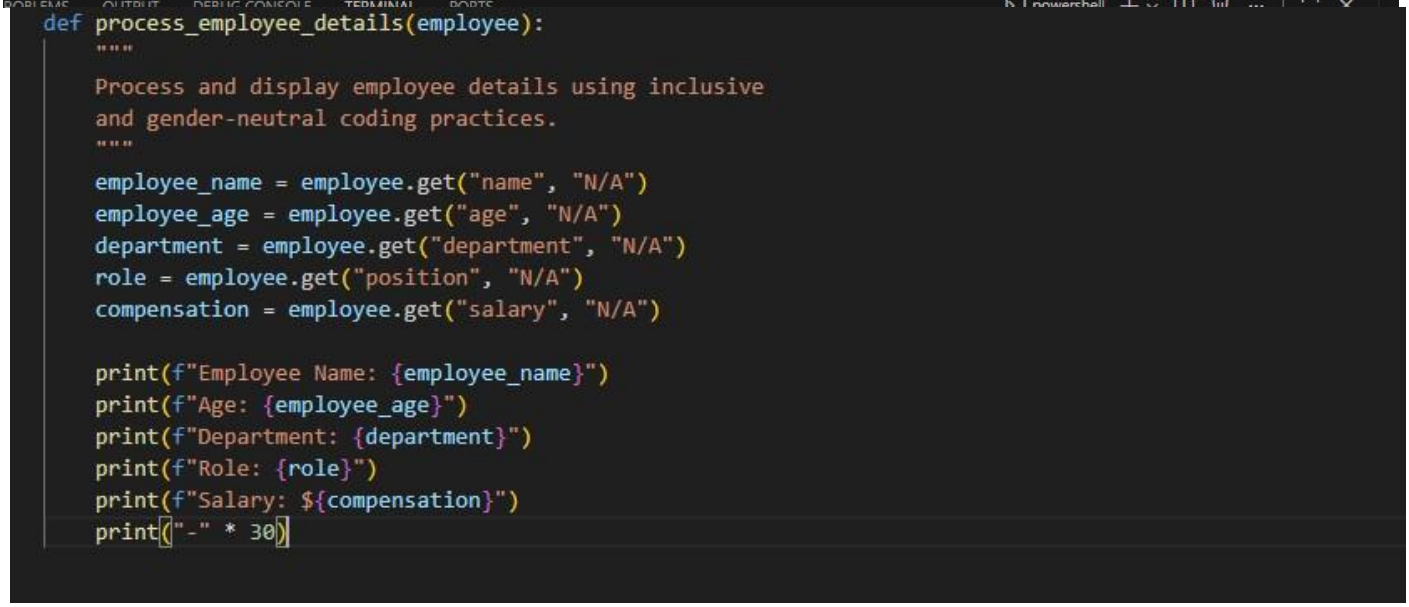
Analyze the code to identify:

- Gender-specific variables (e.g., male, female)
- Assumptions based on gender or identity • Non-inclusive naming or logic Modify or regenerate the code to:
- Use gender-neutral variable names
- Avoid gender-based conditions unless strictly required
- Ensure inclusive and respectful coding practices CODE:



```python
#generate a Python code snippet that processes user or employee details.
def process_employee_details(employee):
    """
    Process and display employee details.
    """
    name = employee.get("name", "N/A")
    age = employee.get("age", "N/A")
    department = employee.get("department", "N/A")
    position = employee.get("position", "N/A")
    salary = employee.get("salary", "N/A")

    print(f"Employee Name: {name}")
    print(f"Age: {age}")
    print(f"Department: {department}")
    print(f"Position: {position}")
    print(f"Salary: ${salary}")
    print("-" * 30)
#Modify or regenerate the code to: Use gender-neutral variable names Avoid gender-based conditions unless s
def process_employee_details(employee):
    """
    Process and display employee details using inclusive
    and gender-neutral coding practices.
    """
    employee_name = employee.get("name", "N/A")
```



```python
def process_employee_details(employee):
    """
    Process and display employee details using inclusive
    and gender-neutral coding practices.
    """
    employee_name = employee.get("name", "N/A")
    employee_age = employee.get("age", "N/A")
    department = employee.get("department", "N/A")
    role = employee.get("position", "N/A")
    compensation = employee.get("salary", "N/A")

    print(f"Employee Name: {employee_name}")
    print(f"Age: {employee_age}")
    print(f"Department: {department}")
    print(f"Role: {role}")
    print(f"Salary: ${compensation}")
    print("-" * 30)
```

OBSERVATION :

The original code, while not explicitly biased, did not actively demonstrate inclusive design principles and used generic variable naming without emphasizing neutrality. It also did not clearly show an intentional avoidance of identity-based assumptions, which can be important in ethical software design discussions.
How inclusiveness was improved:
The revised code uses clearly gender-neutral variable names, avoids any gender- or identity-based conditions, and focuses only on job-relevant information. This makes the logic more inclusive, respectful, and ethically sound, ensuring fairness and avoiding unnecessary assumptions about individuals.