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. Analyze 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). Expected Output • AI-generated login code • Identification of security risks • Revised secure version of the code • Brief explanation of improvements

```python
import hashlib
import os
import sys

# Temporary database: {username: (salt, hashed_password)}
user_database = {}

def hash_password(password, salt=None):
    """Encodes the password using SHA-256 and a random salt."""
    if salt is None:
        salt = os.urandom(16)  # Generate a random 16-byte salt

    # Combine password and salt, then hash
    combined = salt + password.encode('utf-8')
    hashed = hashlib.sha256(combined).hexdigest()
    return salt, hashed

def register():
    print("\n--- Registration ---")
    username = input("Enter a new username: ").strip()

    if username in user_database:
        print("Error: Username already taken.")
        return

    password = input("Enter a password: ").strip()
    if len(password) < 6:
        print("Error: Password too short.")
        return

    # Create secure hash
    salt, hashed_pass = hash_password(password)
    user_database[username] = (salt, hashed_pass)
    print(f"Success! Account for '{username}' created.")

def login():
    print("\n--- Login ---")
    username = input("Username: ").strip()
    password = input("Password: ").strip()

    if username in user_database:
        stored_salt, stored_hash = user_database[username]
        # Hash the input password using the SAME salt
        _, input_hash = hash_password(password, stored_salt)

        if input_hash == stored_hash:
            print(">>> Access Granted!")
            return

    print(">>> Access Denied!")

def main():
    while True:
        print("\n--- MAIN MENU ---")
        print("1. Register")
        print("2. Login")
        print("3. Exit")

        choice = input("Select (1-3): ")

        if choice == '1':
            register()
        elif choice == '2':
            login()
        elif choice == '3':
            print("Goodbye!")
            break
        else:
            print("Invalid choice.")

if __name__ == "__main__":
    main()
```

```
--- MAIN MENU ---
1. Register
2. Login
3. Exit
Select (1-3): 1

--- Registration ---
Enter a new username: bellamkonda bhanu prasad
Enter a password: 2005
```

```
Error: Password too short.

--- MAIN MENU ---
1. Register
2. Login
3. Exit
Select (1-3): 3
Goodbye!
```

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 Analyze whether: • The logic treats certain genders or names unfairly • Approval decisions depend on irrelevant personal attributes Suggest methods to reduce or remove bias. Expected Output • Python code generated by AI • Identification of biased logic (if any) • Discussion on fairness issue

```python
def check_loan_eligibility(name, gender, age, income, debt, credit_score):
    """
    Naive AI-generated logic for loan approval.
    """
    # Logic: High credit score and high income are primary.
    # Flaw: The 'risk_multiplier' incorporates gender and name length.
    risk_multiplier = 1.0

    # Biased Logic Example: AI might 'perceive' certain genders as higher risk
    # based on flawed historical training data.
    if gender.lower() == "female":
        risk_multiplier += 0.1  # UNFAIR BIAS

    # Name-based bias: Longer names or specific prefixes sometimes
    # trigger 'complexity' penalties in poorly designed models.
    if len(name) > 15:
        risk_multiplier += 0.05

    final_score = (credit_score * 0.6) + ((income / 1000) * 0.4) - (debt / 500)

    if (final_score / risk_multiplier) > 400:
        return "Approved"
    else:
        return "Rejected"

# Testing the system
print(f"Result for John: {check_loan_eligibility('John Doe', 'Male', 35, 60000, 2000, 720)}'
print(f"Result for Jane: {check_loan_eligibility('Jane Doe', 'Female', 35, 60000, 2000, 720
```

```
Result for John: Approved
Result for Jane: Approved
```

Transparency and Explainability in AI-Generated Code (Recursive Binary Search) Scenario AI-generated code should be transparent, well-documented, and easy for humans to understand and verify. Task Description 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 Expected Output • Python program for recursive binary search • AI-generated comments and explanation • Student's assessment on clarity, correctness, and transparency

```python
def recursive_binary_search(data_list, target, low, high):
    """
    Performs a binary search on a sorted list using recursion.
    """

    # --- BASE CASE 1: Target not found ---
    # If the 'low' index passes 'high', the search space is exhausted.
    if low > high:
        return -1

    # Calculate the middle index
    # (low + high) // 2 finds the floor of the average
    mid = (low + high) // 2

    # --- BASE CASE 2: Target found ---
    # If the middle element is our target, return the index.
    if data_list[mid] == target:
        return mid

    # --- RECURSIVE CASE 1: Search the Left Half ---
    # If the target is smaller than the middle element,
    # it must be in the lower half of the list.
    elif target < data_list[mid]:
        return recursive_binary_search(data_list, target, low, mid - 1)

    # --- RECURSIVE CASE 2: Search the Right Half ---
    # If the target is larger than the middle element,
    # it must be in the upper half of the list.
    else:
        return recursive_binary_search(data_list, target, mid + 1, high)

# --- Testing the code ---
```

```
my_list = [10, 22, 35, 47, 50, 63, 75, 88, 99]
target_val = 63

# Note: Initial low is 0, initial high is the last index of the list
result = recursive_binary_search(my_list, target_val, 0, len(my_list) - 1)

if result != -1:
    print(f"Element found at index: {result}")
else:
    print("Element not present in the list.")
```

```
Element found at index: 5
```

• Whether gender, name, or unrelated features influence scoring • Whether the logic is fair and objective Expected Output • Python scoring system code • Identification of potential bias (if any) • Ethical analysis of the scoring logic 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 Expected Output • Python scoring system code • Identification of potential bias (if any) • Ethical analysis of the scoring logic

```
def calculate_applicant_score(name, gender, education, years_experience, skills_list):
    """
    Scoring logic for job applicants.
    """
    score = 0

    # 1. Education Scoring
    education_weights = {
        "PhD": 50,
        "Masters": 40,
        "Bachelors": 30,
        "None": 0
    }
    score += education_weights.get(education, 0)

    # 2. Experience Scoring
    # Adds 5 points per year, but caps it at 10 years to avoid 'over-seniority'
    score += min(years_experience, 10) * 5

    # 3. Skills Scoring
    # Each matching keyword adds 10 points
    target_skills = ["Python", "Machine Learning", "Cloud Computing", "Leadership"]
    for skill in skills_list:
        if skill in target_skills:
            score += 10

    # 4. Hidden Pattern Bias (Simulated AI 'Heuristic')
    # AI often learns that shorter gaps or specific names correlate with 'stability'
    if gender.lower() == "male":
        # Flawed logic: AI might have seen more 'Male' tenure in historical data
        score += 5

    return score

# Testing the applicants
candidate_a = calculate_applicant_score("Robert Smith", "Male", "Masters", 8, ["Python", "Le
candidate_b = calculate_applicant_score("Maria Garcia", "Female", "Masters", 8, ["Python", '

print(f"Robert's Score: {candidate_a}")
print(f"Maria's Score: {candidate_b}")
```

```
Robert's Score: 105
Maria's Score: 100
```

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 Expected Output • Original AI-generated code snippet • Revised inclusive and gender-neutral code • Brief explanation of: o What was non-inclusive o How inclusiveness was improved

```
class TeamMember:
    def __init__(self, full_name, preferred_pronouns=None, honorific=None):
        self.full_name = full_name
        self.preferred_pronouns = preferred_pronouns
        self.honorific = honorific

    def get_formal_name(self):
        if self.honorific:
            return f"{self.honorific} {self.full_name}"
        return self.full_name

def send_invitation(member):
    # Uses gender-neutral 'guest' instead of 'wife/husband'
    print(f"Dear {member.get_formal_name()}, you and your guest are cordially invited.")
```

```python
# Example usage:
member1 = TeamMember("Alex Johnson", preferred_pronouns="they/them")
member2 = TeamMember("Dr. Evelyn Reed", honorific="Dr.")

send_invitation(member1)
send_invitation(member2)
```

```
Dear Alex Johnson, you and your guest are cordially invited.
Dear Dr. Dr. Evelyn Reed, you and your guest are cordially invited.
```