

**Name: K.Likitha**

**Hall Ticket: 2303A52144**

**Batch: 41**

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

**Prompt:**

Generate a simple Python program where a user first registers using input() and the data is stored. Then allow login using that data and include common security issues like plain-text password storage.

**Code:**

```
"""
REGISTRATION & LOGIN SYSTEM WITH SECURITY VULNERABILITIES
⚠ Educational demonstration of POOR security practices
    NEVER use this approach in production!
"""

import getpass
import json
from datetime import datetime

# ❌ ISSUE 1: User data stored in memory as plain-text dictionary
# In real systems, this would be a plain-text file or unencrypted database
USER_DATABASE = {}

# ❌ ISSUE 2: No encryption, no hashing, just storing credentials as-is
PASSWORD_STORAGE_METHOD = "PLAIN TEXT" # Should be bcrypt, argon2, etc.

# ❌ ISSUE 3: Tracking failed attempts but with very high limit
failed_login_attempts = {}
MAX_LOGIN_ATTEMPTS = 999

def display_welcome():
    """Display welcome message"""
    print("\n" + "="*70)
    print("INSECURE REGISTRATION & LOGIN SYSTEM - EDUCATIONAL PURPOSE ONLY")
    print("="*70)
    print(f"Password Storage Method: {PASSWORD_STORAGE_METHOD}")
    print("="*70 + "\n")

def register_user():
    """
    User registration with security vulnerabilities.

    ❌ VULNERABILITIES:
    1. Plain-text password storage
    2. No password strength validation
    3. No input sanitization
    4. Data stored in memory (not persistent)
    5. No email verification
    6. No CAPTCHA protection
    """
    pass
```

```
7. Passwords visible in logs/memory dumps
"""

print("\n" + "-"*70)
print("USER REGISTRATION")
print("-"*70 + "\n")

# Get username
while True:
    username = input("Enter username: ").strip()

    # ❌ ISSUE 4: Minimal validation
    if len(username) < 3:
        print("❌ Username must be at least 3 characters")
        continue

    # ❌ ISSUE 5: No check for SQL injection, special characters, etc.
    if username in USER_DATABASE:
        print(f"❌ Username '{username}' already exists")
        continue

    break

# Get email (no validation)
email = input("Enter email: ").strip()

# ❌ ISSUE 6: No email format validation
# ❌ ISSUE 7: Email stored in plain text

# Get password - multiple times to confirm
while True:
    password = getpass.getpass("Enter password: ")
    confirm_password = getpass.getpass("Confirm password: ")

    # ❌ ISSUE 8: Very weak password requirements
    if len(password) < 4:
        print("❌ Password must be at least 4 characters")
        continue

    # ❌ ISSUE 9: No complexity requirements
    # Should require: uppercase, lowercase, numbers, special chars

    if password != confirm_password:
        print("❌ Passwords don't match")
        continue

    break

# ❌ ISSUE 10: CRITICAL - Store password in PLAIN TEXT
# This is extremely dangerous!
user_data = {
    "username": username,
    "email": email,
    "password": password, # STORED AS PLAIN TEXT!
```

```

        "registered_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "failed_login_attempts": 0,
        "account_locked": False
    }

USER_DATABASE[username] = user_data

print(f"\n\x25 Registration successful!")
print(f"  Username: {username}")
print(f"  Email: {email}")
print(f"  Password stored as: {PASSWORD_STORAGE_METHOD}")
print(f"  Registered at: {user_data['registered_at']}")

return True

def login_user():
    """
    User login with vulnerabilities.

    ✗ VULNERABILITIES:
    1. Compares passwords in plain text
    2. No timing attack protection
    3. No rate limiting/account lockout
    4. Reveals if username exists
    5. No multi-factor authentication
    6. No session token generation
    """

    print("\n" + "-"*70)
    print("USER LOGIN")
    print("-"*70 + "\n")

    # Get username
    username = input("Enter username: ").strip()

    # ✗ ISSUE 11: Leaks whether username exists
    if username not in USER_DATABASE:
        print(f"✗ Login failed: Username or password incorrect")
        print("  (Actually, we revealed the username doesn't exist!)")
        return False

    user_data = USER_DATABASE[username]

    # ✗ ISSUE 12: Check if account is locked
    if user_data["account_locked"]:
        print(f"✗ Account locked due to too many failed attempts")
        return False

    # Get password from user
    password = getpass.getpass("Enter password: ")

    # ✗ ISSUE 13: CRITICAL - Plain text password comparison
    # This should use bcrypt.checkpw() or similar!

```

```

if user_data["password"] != password:
    user_data["failed_login_attempts"] += 1
    print(f"❌ Login failed: Username or password incorrect")
    print(f"    Failed attempts: {user_data['failed_login_attempts']}/3")

    # ❌ ISSUE 14: Account lockout threshold too high
    if user_data["failed_login_attempts"] >= 3:
        user_data["account_locked"] = True
        print(f"⚠️ Account locked for security (this is actually good!)")

return False

# ❌ ISSUE 15: No session token generation
# Just acknowledges successful login
print(f"\n✅ Login successful!")
print(f"    Welcome, {username}!")
print(f"    Email: {user_data['email']}")
print(f"    Account created: {user_data['registered_at']}")

# Reset failed attempts on successful login
user_data["failed_login_attempts"] = 0

return True


def view_all_users():
    """
    Shows all registered users with their passwords in plain text.

    ❌ ISSUE 16: Ability to dump all user credentials
    This demonstrates how easy it is to access all passwords
    """

    if not USER_DATABASE:
        print("\n❌ No users registered yet\n")
        return

    print("\n" + "-"*70)
    print("ALL REGISTERED USERS (PLAIN TEXT CREDENTIALS EXPOSED!)")
    print("-"*70)

    print(f"\n{'Username':<15} {'Email':<25} {'Password':<20} {'Registered':<20}")
    print("-"*70)

    for username, data in USER_DATABASE.items():
        print(f"{username:<15} {data['email']:<25} {data['password']:<20}")
        print(f" {data['registered_at']:<20}")

    print("\n⚠️ All passwords are visible in plain text!")
    print("⚠️ This would be catastrophic if this database was breached!\n")


def export_user_database():
    """

```

```
Export entire user database to JSON (plain text).
```

```
X ISSUE 17: Easy export of all credentials in plain text
"""

if not USER_DATABASE:
    print("\nX No users to export\n")
    return

print("\n" + "-"*70)
print("USER DATABASE EXPORT (JSON FORMAT)")
print("-"*70 + "\n")

json_output = json.dumps(USER_DATABASE, indent=2)
print(json_output)

print("\n⚠ This shows how easily the entire database could be leaked")
print("⚠ Passwords are stored in plain text in the JSON!\n")
```

```
def show_vulnerabilities():
    """Display all vulnerabilities in this system"""

    vulnerabilities = """
```

#### SECURITY VULNERABILITIES IN THIS SYSTEM

1. X PLAIN-TEXT PASSWORD STORAGE
  - └ Passwords stored as-is without any encryption or hashing
  - └ If database is breached, all passwords are immediately compromised
  - └ Attackers can use passwords on other sites (password reuse)
2. X NO PASSWORD HASHING
  - └ Should use: bcrypt, argon2, PBKDF2, or scrypt
  - └ Makes passwords useless even if database is compromised
3. X WEAK PASSWORD REQUIREMENTS
  - └ Only checks minimum length (4 characters)
  - └ No complexity requirements (uppercase, lowercase, numbers, symbols)
  - └ Vulnerable to dictionary attacks and brute force
4. X USERNAME ENUMERATION
  - └ Can determine which usernames are registered
  - └ Should say "Username or password incorrect" (generic)
  - └ Helps attackers narrow down valid usernames
5. X NO INPUT VALIDATION/SANITIZATION
  - └ Could be vulnerable to SQL injection
  - └ No checks for malicious input patterns
  - └ No length limits on inputs
6. X NO EMAIL VERIFICATION
  - └ Anyone can register with any email address

- └ No confirmation required
  - └ Leads to account takeover via email spoofing
7. ✗ NO RATE LIMITING
- └ Unlimited login attempts allowed
  - └ Vulnerable to brute force attacks
  - └ Could try thousands of passwords per second
8. ✗ POOR ACCOUNT LOCKOUT
- └ Only locks after 3 attempts (good!)
  - └ But lockout has no time limit in this code
  - └ Doesn't notify user or send security alerts
9. ✗ IN-MEMORY STORAGE (NOT PERSISTENT)
- └ Data lost when program exits
  - └ In real systems, this would be a plain-text file or database
  - └ Would need encryption at rest
10. ✗ NO MULTI-FACTOR AUTHENTICATION (MFA)
- └ Single factor authentication is insufficient
  - └ Vulnerable if password is compromised
  - └ Should use TOTP, SMS, or hardware keys
11. ✗ NO HTTPS/TLS ENCRYPTION
- └ If this were a web app, data sent in plain text
  - └ Passwords could be intercepted over the network
  - └ Man-in-the-middle attacks possible
12. ✗ NO SESSION TOKENS
- └ Just confirms successful login
  - └ No token generation for continued access
  - └ Should use JWT or secure session IDs
13. ✗ NO LOGGING OR MONITORING
- └ No record of login attempts
  - └ Can't detect suspicious activity
  - └ No audit trail for compliance
14. ✗ DATA EXPOSED TO ANYONE WITH CODE ACCESS
- └ View all users function shows all passwords
  - └ Export function dumps entire database
  - └ No access controls

### SECURE ALTERNATIVES

#### PASSWORD HASHING:

```
import bcrypt
hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
# Store hashed, verify with: bcrypt.checkpw(password.encode(), hashed)
```

#### USE ENVIRONMENT VARIABLES:

```
import os
```

```

from dotenv import load_dotenv
SECRET_KEY = os.getenv("SECRET_KEY") # Never hardcode!

SECURE STORAGE:
- Use a proper database (PostgreSQL, MongoDB)
- Enable encryption at rest
- Use AWS Secrets Manager, HashiCorp Vault

STRONG PASSWORD REQUIREMENTS:
- Minimum 12 characters
- Require: uppercase, lowercase, numbers, special characters
- Check against common password lists
- Use libraries like: pyuservalidate, owasp-password-strength

RATE LIMITING:
- Use Flask-Limiter or similar
- Max 5 attempts per minute
- Exponential backoff after failures
- Lock account for 15-30 minutes

MULTI-FACTOR AUTHENTICATION:
- TOTP: pyotp library
- SMS: Twilio
- Hardware keys: FIDO2
- Backup codes

SESSION MANAGEMENT:
- Use JWT tokens (PyJWT library)
- Set expiration time (15-60 minutes)
- Refresh tokens for long sessions
- Use HttpOnly + Secure cookies

SECURE FRAMEWORKS:
- Flask: Flask-Login, Flask-SQLAlchemy
- Django: Built-in auth system
- FastAPI: FastAPI Security
- ASP.NET Core: Identity system
"""

print(vulnerabilities)

def main():
    """Main program loop"""
    display_welcome()

    # ❌ ISSUE: No enforcement of registration before login
    # In a real system, this should be enforced

    while True:
        # Check if any users exist
        if not USER_DATABASE:
            print("\n" + "="*70)
            print("⚠️ NO USERS REGISTERED YET - PLEASE REGISTER FIRST")

```

```
print("=*70")
print("\nYou must create an account before you can login.\n")

print("Available Actions:")
print("-" * 70)
print("1. 📄 REGISTER NEW ACCOUNT")
print("    └ Create a new user account with username and password")
print()
print("2. ⓘ SHOW SECURITY VULNERABILITIES")
print("    └ Learn about the security issues in this system")
print()
print("3. 💾 EXIT")
print("    └ Close the application")
print("-" * 70)

choice = input("\nEnter your choice (1-3): ").strip()

if choice == "1":
    register_user()
elif choice == "2":
    show_vulnerabilities()
elif choice == "3":
    print("\n👋 Goodbye!")
    break
else:
    print("\n✖ Invalid choice. Please enter 1-3.")

else:
    # Users exist - show full menu
    print("\n" + "*70")
    print("📄 MAIN MENU - Welcome to Registration & Login System")
    print("*70")
    print(f"\n    Currently registered users: {len(USER_DATABASE)}")
    print()

    print("Authentication Options:")
    print("-" * 70)
    print("1. 📄 REGISTER NEW ACCOUNT")
    print("    └ Create another user account with username and password")
    print()
    print("2. 🔑 LOGIN TO ACCOUNT")
    print("    └ Login with existing credentials")
    print()

    print("\nSystem Administration (Educational):")
    print("-" * 70)
    print("3. 👤 VIEW ALL USERS (INSECURE!)")
    print("    └ Display all registered users and their plain-text passwords")
    print("    └ ⚠ This shows why plain-text storage is dangerous!")
    print()
    print("4. 🗂 EXPORT USER DATABASE (INSECURE!)")
    print("    └ Export all user data in JSON format")
    print("    └ ⚠ Shows how easily credentials can be leaked!")
    print()
```

```
print("Information:")
print("-" * 70)
print("5. 🗝 SHOW SECURITY VULNERABILITIES")
print("    └ Learn about the 17+ security issues in this system")
print("    └ See secure alternatives and best practices")
print()
print("6. 💾 EXIT")
print("    └ Close the application")
print("-" * 70)

choice = input("\nEnter your choice (1-6): ").strip()

if choice == "1":
    register_user()

elif choice == "2":
    login_user()

elif choice == "3":
    view_all_users()

elif choice == "4":
    export_user_database()

elif choice == "5":
    show_vulnerabilities()

elif choice == "6":
    print("\n👋 Goodbye!")
    break

else:
    print("\n❌ Invalid choice. Please enter 1-6.")

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

**Output:**

=====

⚠ NO USERS REGISTERED YET - PLEASE REGISTER FIRST

=====

You must create an account before you can login.

Available Actions:

- 
1. REGISTER NEW ACCOUNT
    - └ Create a new user account with username and password
  2. SHOW SECURITY VULNERABILITIES
    - └ Learn about the security issues in this system
  3. EXIT
    - └ Close the application
- 

Enter your choice (1-3): 1

-----  
USER REGISTRATION  
-----

Enter username: likitha  
Enter email: likithaa@gmail.com  
Enter password:  
Confirm password:

Registration successful!  
Username: likitha  
Email: likithaa@gmail.com  
Password stored as: PLAIN TEXT  
Registered at: 2026-01-28 11:39:17

=====

MAIN MENU - Welcome to Registration & Login System

=====

currently registered users: 1

Authentication Options:

- 
1. REGISTER NEW ACCOUNT
    - └ Create another user account with username and password
  2. LOGIN TO ACCOUNT
    - └ Login with existing credentials

System Administration (Educational):

- 
3. VIEW ALL USERS (INSECURE!)
    - └ Display all registered users and their plain-text passwords
    - └ ⚠ This shows why plain-text storage is dangerous!
  4. EXPORT USER DATABASE (INSECURE!)
    - └ Export all user data in JSON format
    - └ ⚠ Shows how easily credentials can be leaked!

Information:

- 
5. SHOW SECURITY VULNERABILITIES
    - └ Learn about the 17+ security issues in this system
    - └ See secure alternatives and best practices
  5. SHOW SECURITY VULNERABILITIES
    - └ Learn about the 17+ security issues in this system
    - └ See secure alternatives and best practices

- 
- 5. ● SHOW SECURITY VULNERABILITIES
    - └ Learn about the 17+ security issues in this system
    - └ See secure alternatives and best practices
  
  - 6. ■ EXIT
    - └ Close the application
- 

Enter your choice (1-6):

**Prompt:**

Rewrite the same program to make it more secure.

Avoid hardcoding, validate user input, and improve how passwords are handled.

**Revised Code:**

```
"""
REGISTRATION & LOGIN SYSTEM WITH SECURITY VULNERABILITIES
⚠ Educational demonstration of POOR security practices
    NEVER use this approach in production!
"""

import getpass
import json
from datetime import datetime

# ❌ ISSUE 1: User data stored in memory as plain-text dictionary
# In real systems, this would be a plain-text file or unencrypted database
USER_DATABASE = {}

# ❌ ISSUE 2: No encryption, no hashing, just storing credentials as-is
PASSWORD_STORAGE_METHOD = "PLAIN TEXT" # Should be bcrypt, argon2, etc.

# ❌ ISSUE 3: Tracking failed attempts but with very high limit
failed_login_attempts = {}
MAX_LOGIN_ATTEMPTS = 999

def display_welcome():
    """Display welcome message"""
    print("\n" + "="*70)
    print("INSECURE REGISTRATION & LOGIN SYSTEM - EDUCATIONAL PURPOSE ONLY")
    print("="*70)
    print(f"Password Storage Method: {PASSWORD_STORAGE_METHOD}")
    print("="*70 + "\n")

def register_user():
    """
    User registration with security vulnerabilities.

    ❌ VULNERABILITIES:
    1. Plain-text password storage
    2. No password strength validation
    3. No input sanitization
    """
    pass
```

```
4. Data stored in memory (not persistent)
5. No email verification
6. No CAPTCHA protection
7. Passwords visible in logs/memory dumps
"""

print("\n" + "-"*70)
print("USER REGISTRATION")
print("-"*70 + "\n")

# Get username
while True:
    username = input("Enter username: ").strip()

    # ❌ ISSUE 4: Minimal validation
    if len(username) < 3:
        print("❌ Username must be at least 3 characters")
        continue

    # ❌ ISSUE 5: No check for SQL injection, special characters, etc.
    if username in USER_DATABASE:
        print(f"❌ Username '{username}' already exists")
        continue

    break

# Get email (no validation)
email = input("Enter email: ").strip()

# ❌ ISSUE 6: No email format validation
# ❌ ISSUE 7: Email stored in plain text

# Get password - multiple times to confirm
while True:
    password = getpass.getpass("Enter password: ")
    confirm_password = getpass.getpass("Confirm password: ")

    # ❌ ISSUE 8: Very weak password requirements
    if len(password) < 4:
        print("❌ Password must be at least 4 characters")
        continue

    # ❌ ISSUE 9: No complexity requirements
    # Should require: uppercase, lowercase, numbers, special chars

    if password != confirm_password:
        print("❌ Passwords don't match")
        continue

    break

# ❌ ISSUE 10: CRITICAL - Store password in PLAIN TEXT
# This is extremely dangerous!
user_data = {
```

```

        "username": username,
        "email": email,
        "password": password, # STORED AS PLAIN TEXT!
        "registered_at": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "failed_login_attempts": 0,
        "account_locked": False
    }

USER_DATABASE[username] = user_data

print(f"\n✅ Registration successful!")
print(f"  Username: {username}")
print(f"  Email: {email}")
print(f"  Password stored as: {PASSWORD_STORAGE_METHOD}")
print(f"  Registered at: {user_data['registered_at']}")

return True

def login_user():
    """
    User login with vulnerabilities.

    ✗ VULNERABILITIES:
    1. Compares passwords in plain text
    2. No timing attack protection
    3. No rate limiting/account lockout
    4. Reveals if username exists
    5. No multi-factor authentication
    6. No session token generation
    """

    print("\n" + "-"*70)
    print("USER LOGIN")
    print("-"*70 + "\n")

    # Get username
    username = input("Enter username: ").strip()

    # ✗ ISSUE 11: Leaks whether username exists
    if username not in USER_DATABASE:
        print(f"✗ Login failed: Username or password incorrect")
        print("  (Actually, we revealed the username doesn't exist!)")
        return False

    user_data = USER_DATABASE[username]

    # ✗ ISSUE 12: Check if account is locked
    if user_data["account_locked"]:
        print(f"✗ Account locked due to too many failed attempts")
        return False

    # Get password from user
    password = getpass.getpass("Enter password: ")

```

```

# ❌ ISSUE 13: CRITICAL - Plain text password comparison
# This should use bcrypt.checkpw() or similar!
if user_data["password"] != password:
    user_data["failed_login_attempts"] += 1
    print(f"❌ Login failed: Username or password incorrect")
    print(f"    Failed attempts: {user_data['failed_login_attempts']}")

# ❌ ISSUE 14: Account lockout threshold too high
if user_data["failed_login_attempts"] >= 3:
    user_data["account_locked"] = True
    print(f"⚠️ Account locked for security (this is actually good!)")

return False

# ❌ ISSUE 15: No session token generation
# Just acknowledges successful login
print(f"\n✅ Login successful!")
print(f"    Welcome, {username}!")
print(f"    Email: {user_data['email']}")
print(f"    Account created: {user_data['registered_at']}")

# Reset failed attempts on successful login
user_data["failed_login_attempts"] = 0

return True


def view_all_users():
    """
    Shows all registered users with their passwords in plain text.

    ❌ ISSUE 16: Ability to dump all user credentials
    This demonstrates how easy it is to access all passwords
    """

    if not USER_DATABASE:
        print("\n❌ No users registered yet\n")
        return

    print("\n" + "-"*70)
    print("ALL REGISTERED USERS (PLAIN TEXT CREDENTIALS EXPOSED!)")
    print("-"*70)

    print(f"\n{'Username':<15} {'Email':<25} {'Password':<20} {'Registered':<20}")
    print("-"*70)

    for username, data in USER_DATABASE.items():
        print(f"{username:<15} {data['email']:<25} {data['password']:<20}")
        print(f" {data['registered_at']:<20}")

    print("\n⚠️ All passwords are visible in plain text!")
    print("⚠️ This would be catastrophic if this database was breached!\n")

```

```

def export_user_database():
    """
        Export entire user database to JSON (plain text).

    X ISSUE 17: Easy export of all credentials in plain text
    """

    if not USER_DATABASE:
        print("\nX No users to export\n")
        return

    print("\n" + "-"*70)
    print("USER DATABASE EXPORT (JSON FORMAT)")
    print("-"*70 + "\n")

    json_output = json.dumps(USER_DATABASE, indent=2)
    print(json_output)

    print("\n⚠ This shows how easily the entire database could be leaked")
    print("⚠ Passwords are stored in plain text in the JSON!\n")

```

```

def show_vulnerabilities():
    """Display all vulnerabilities in this system"""

    vulnerabilities = """

```

### SECURITY VULNERABILITIES IN THIS SYSTEM

1. X PLAIN-TEXT PASSWORD STORAGE
  - └ Passwords stored as-is without any encryption or hashing
  - └ If database is breached, all passwords are immediately compromised
  - └ Attackers can use passwords on other sites (password reuse)
2. X NO PASSWORD HASHING
  - └ Should use: bcrypt, argon2, PBKDF2, or scrypt
  - └ Makes passwords useless even if database is compromised
3. X WEAK PASSWORD REQUIREMENTS
  - └ Only checks minimum length (4 characters)
  - └ No complexity requirements (uppercase, lowercase, numbers, symbols)
  - └ Vulnerable to dictionary attacks and brute force
4. X USERNAME ENUMERATION
  - └ Can determine which usernames are registered
  - └ Should say "Username or password incorrect" (generic)
  - └ Helps attackers narrow down valid usernames
5. X NO INPUT VALIDATION/SANITIZATION
  - └ Could be vulnerable to SQL injection
  - └ No checks for malicious input patterns
  - └ No length limits on inputs

6. **✗ NO EMAIL VERIFICATION**
  - └ Anyone can register with any email address
  - └ No confirmation required
  - └ Leads to account takeover via email spoofing
7. **✗ NO RATE LIMITING**
  - └ Unlimited login attempts allowed
  - └ Vulnerable to brute force attacks
  - └ Could try thousands of passwords per second
8. **✗ POOR ACCOUNT LOCKOUT**
  - └ Only locks after 3 attempts (good!)
  - └ But lockout has no time limit in this code
  - └ Doesn't notify user or send security alerts
9. **✗ IN-MEMORY STORAGE (NOT PERSISTENT)**
  - └ Data lost when program exits
  - └ In real systems, this would be a plain-text file or database
  - └ Would need encryption at rest
10. **✗ NO MULTI-FACTOR AUTHENTICATION (MFA)**
  - └ Single factor authentication is insufficient
  - └ Vulnerable if password is compromised
  - └ Should use TOTP, SMS, or hardware keys
11. **✗ NO HTTPS/TLS ENCRYPTION**
  - └ If this were a web app, data sent in plain text
  - └ Passwords could be intercepted over the network
  - └ Man-in-the-middle attacks possible
12. **✗ NO SESSION TOKENS**
  - └ Just confirms successful login
  - └ No token generation for continued access
  - └ Should use JWT or secure session IDs
13. **✗ NO LOGGING OR MONITORING**
  - └ No record of login attempts
  - └ Can't detect suspicious activity
  - └ No audit trail for compliance
14. **✗ DATA EXPOSED TO ANYONE WITH CODE ACCESS**
  - └ View all users function shows all passwords
  - └ Export function dumps entire database
  - └ No access controls

<input checked="" type="checkbox"/> SECURE ALTERNATIVES
---

PASSWORD HASHING:

```
import bcrypt
hashed = bcrypt.hashpw(password.encode(), bcrypt.gensalt())
# Store hashed, verify with: bcrypt.checkpw(password.encode(), hashed)
```

```

USE ENVIRONMENT VARIABLES:
import os
from dotenv import load_dotenv
SECRET_KEY = os.getenv("SECRET_KEY") # Never hardcode!

SECURE STORAGE:
- Use a proper database (PostgreSQL, MongoDB)
- Enable encryption at rest
- Use AWS Secrets Manager, HashiCorp Vault

STRONG PASSWORD REQUIREMENTS:
- Minimum 12 characters
- Require: uppercase, lowercase, numbers, special characters
- Check against common password lists
- Use libraries like: pyuservalidate, owasp-password-strength

RATE LIMITING:
- Use Flask-Limiter or similar
- Max 5 attempts per minute
- Exponential backoff after failures
- Lock account for 15-30 minutes

MULTI-FACTOR AUTHENTICATION:
- TOTP: pyotp library
- SMS: Twilio
- Hardware keys: FIDO2
- Backup codes

SESSION MANAGEMENT:
- Use JWT tokens (PyJWT library)
- Set expiration time (15-60 minutes)
- Refresh tokens for long sessions
- Use HttpOnly + Secure cookies

SECURE FRAMEWORKS:
- Flask: Flask-Login, Flask-SQLAlchemy
- Django: Built-in auth system
- FastAPI: FastAPI Security
- ASP.NET Core: Identity system
"""

print(vulnerabilities)

def main():
    """Main program loop"""
    display_welcome()

    # ❌ ISSUE: No enforcement of registration before login
    # In a real system, this should be enforced

    while True:
        # Check if any users exist
        if not USER_DATABASE:

```

```

print("\n" + "*70)
print("⚠ NO USERS REGISTERED YET - PLEASE REGISTER FIRST")
print("*70)
print("\nYou must create an account before you can login.\n")

print("Available Actions:")
print("-" * 70)
print("1. 📝 REGISTER NEW ACCOUNT")
print("    └ Create a new user account with username and password")
print()
print("2. 🗃 SHOW SECURITY VULNERABILITIES")
print("    └ Learn about the security issues in this system")
print()
print("3. 🚪 EXIT")
print("    └ Close the application")
print("-" * 70)

choice = input("\nEnter your choice (1-3): ").strip()

if choice == "1":
    register_user()
elif choice == "2":
    show_vulnerabilities()
elif choice == "3":
    print("\n👋 Goodbye!")
    break
else:
    print("\n❌ Invalid choice. Please enter 1-3.")

else:
    # Users exist - show full menu
    print("\n" + "*70)
    print("📝 MAIN MENU - Welcome to Registration & Login System")
    print("*70)
    print(f"\n    Currently registered users: {len(USER_DATABASE)}")
    print()

    print("Authentication Options:")
    print("-" * 70)
    print("1. 📝 REGISTER NEW ACCOUNT")
    print("    └ Create another user account with username and password")
    print()
    print("2. 🔑 LOGIN TO ACCOUNT")
    print("    └ Login with existing credentials")
    print()

    print("\nSystem Administration (Educational):")
    print("-" * 70)
    print("3. 👤 VIEW ALL USERS (INSECURE!]")
    print("    └ Display all registered users and their plain-text passwords")
    print("    └ ⚠ This shows why plain-text storage is dangerous!")
    print()
    print("4. 📁 EXPORT USER DATABASE (INSECURE!]")
    print("    └ Export all user data in JSON format")

```

```

print("    └ ↗ Shows how easily credentials can be leaked!")
print()

print("Information:")
print("-" * 70)
print("5. 🗑 SHOW SECURITY VULNERABILITIES")
print("    └ Learn about the 17+ security issues in this system")
print("    └ See secure alternatives and best practices")
print()
print("6. ⚙ EXIT")
print("    └ Close the application")
print("-" * 70)

choice = input("\nEnter your choice (1-6): ").strip()

if choice == "1":
    register_user()

elif choice == "2":
    login_user()

elif choice == "3":
    view_all_users()

elif choice == "4":
    export_user_database()

elif choice == "5":
    show_vulnerabilities()

elif choice == "6":
    print("\n👋 Goodbye!")
    break

else:
    print("\n❌ Invalid choice. Please enter 1-6.")

if __name__ == "__main__":
    main()

```

**Output:**

```

224 def show_vulnerabilities():
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> python secure_login_system.py

You must create an account before you can login.

Available Actions:
1. 📁 REGISTER NEW ACCOUNT
   └ Create a new user account with strong password validation
2. 🌐 VIEW SECURITY FEATURES
   └ Learn about the security features in this system
3. 🚪 EXIT
   └ Close the application

Enter your choice (1-3): 1

-----
📁 USER REGISTRATION - SECURE
-----

Enter username: likitha
 Username is valid
Enter email: liki@gmail.com
 Email is valid

⚠️ Password Requirements:
• Minimum 12 characters
• At least one UPPERCASE letter (A-Z)
• At least one lowercase letter (a-z)
• At least one number (0-9)
• At least one special character (!@#$%^&*()_+-=[{}]{})etc.)

Enter password: []

```

#### Explanation:

The first program demonstrates an insecure login system where user registration data is stored in plain text. Such code is unsafe because credentials can be easily exposed or misused.

The revised version improves security by avoiding hardcoded values, validating input, and handling passwords more safely.

#### Task 2: Bias Detection in AI-Generated Decision Systems

##### Prompt:

Create a loan approval system and test it with applicants of different names and genders.  
Check for biased decisions and suggest ways to remove bias.

##### Code:

```

"""
INTERACTIVE BIASED LOAN APPROVAL SYSTEM
⚠️ This demonstrates INTENTIONAL BIAS in decision-making algorithms
    Shows how gender, name, and other factors can unfairly affect loan decisions
"""

from dataclasses import dataclass
from typing import Tuple
from datetime import datetime

# ❌ BIAS EXAMPLE 1: Different interest rates based on gender
GENDER_BIAS_RATES = {

```

```

        "male": 0.045,      # 4.5% - Lower rate for males
        "female": 0.065,    # 6.5% - Higher rate for females (BIASED!)
        "other": 0.070     # 7.0% - Highest rate for others
    }

# ❌ BIAS EXAMPLE 2: Different approval thresholds based on gender
GENDER_BIAS_THRESHOLDS = {
    "male": 0.60,       # 60% threshold for males (easier to approve)
    "female": 0.75,     # 75% threshold for females (harder to approve) - BIASED!
    "other": 0.80       # 80% threshold for others (hardest to approve)
}

# ❌ BIAS EXAMPLE 3: Name-based bias (assuming certain names are higher risk)
NAME_BIAS_SCORES = {
    # "Western" names get lower risk scores
    "john": 0.15, "michael": 0.15, "james": 0.15, "robert": 0.15, "david": 0.15,
    "william": 0.15, "mary": 0.20, "jennifer": 0.20, "sarah": 0.20, "emily": 0.20,
    # "Non-Western" names get higher risk scores (RACIST BIAS!)
    "ahmed": 0.45, "raj": 0.40, "fatima": 0.50, "priya": 0.45,
}

# ❌ BIAS EXAMPLE 4: Different minimum income requirements by gender
GENDER_INCOME_BIAS = {
    "male": 25000,      # $25k minimum for males
    "female": 35000,    # $35k minimum for females (BIASED!)
    "other": 45000      # $45k minimum for others
}

@dataclass
class LoanApplicant:
    """Loan applicant information"""
    name: str
    gender: str
    age: int
    income: float
    credit_score: int
    loan_amount: float
    employment_years: int
    debt_to_income_ratio: float

class BiasedLoanApprovalSystem:
    """
    ❌ BIASED LOAN APPROVAL SYSTEM
    Demonstrates intentional bias based on gender and name
    """

    def __init__(self):
        self.decisions = []

    def _get_name_bias_score(self, name: str) -> float:
        """
        ❌ BIAS: Assign risk score based on name
        """

```

```

Assumes certain names are higher risk (RACIST!)
"""

name_lower = name.lower()
default_bias = 0.35
bias_score = NAME_BIAS_SCORES.get(name_lower, default_bias)
return bias_score

def _get_gender_bias(self, gender: str) -> Tuple[float, float]:
"""
    ✗ BIAS: Get approval threshold and interest rate based on gender
    Makes different requirements for different genders (DISCRIMINATORY!)
"""

    threshold = GENDER_BIAS_THRESHOLDS.get(gender.lower(), 0.80)
    interest_rate = GENDER_BIAS_RATES.get(gender.lower(), 0.070)
    return threshold, interest_rate

def _check_income_requirement(self, applicant: LoanApplicant) -> Tuple[bool, str]:
"""
    ✗ BIAS: Different income requirements by gender
"""

    min_income = GENDER_INCOME_BIAS.get(applicant.gender.lower(), 45000)

    if applicant.income < min_income:
        return False, f"Income ${applicant.income:,.0f} below minimum
${min_income:,.0f} for {applicant.gender}s"

    return True, f"Income requirement met: ${applicant.income:,.0f}"

def _calculate_approval_score(self, applicant: LoanApplicant) -> float:
"""
    ✗ BIASED: Calculate approval score with intentional biases
"""

    # Income score (normalized to 0-100)
    income_score = min((applicant.income / 100000) * 100, 100)

    # Credit score (already normalized 300-850 to 0-100)
    credit_score_normalized = (applicant.credit_score - 300) / 550 * 100
    credit_score_normalized = min(max(credit_score_normalized, 0), 100)

    # Employment score (max 40 years)
    employment_score = min((applicant.employment_years / 40) * 100, 100)

    # ✗ NAME BIAS SCORE (discriminatory!)
    name_bias_score = self._get_name_bias_score(applicant.name) * 100

    # ✗ GENDER BIAS (discriminatory!)
    gender_multiplier = 1.0
    if applicant.gender.lower() == "male":
        gender_multiplier = 1.15 # +15% boost for males
    elif applicant.gender.lower() == "female":
        gender_multiplier = 0.85 # -15% penalty for females

    # Weighted calculation with biases included

```

```

approval_score = (
    (income_score * 0.40) +
    (credit_score_normalized * 0.30) +
    (employment_score * 0.15) +
    ((100 - name_bias_score) * 0.05) +
    (50 * 0.10)
) * gender_multiplier

# Normalize to 0-1
return min(max(approval_score / 100, 0), 1)

def evaluate_application(self, applicant: LoanApplicant) -> dict:
    """
    Evaluate loan application with BIASED logic.
    """

    print("\n" + "*85)
    print(f"LOAN APPLICATION REVIEW - {applicant.name.upper()}")
    print("*85)

    # Display applicant info
    print("\n📋 APPLICANT INFORMATION:")
    print("-"*85)
    print(f" Name: {applicant.name}")
    print(f" Gender: {applicant.gender}")
    print(f" Age: {applicant.age} years")
    print(f" Annual Income: ${applicant.income:.2f}")
    print(f" Credit Score: {applicant.credit_score}")
    print(f" Loan Amount: ${applicant.loan_amount:.2f}")
    print(f" Employment Years: {applicant.employment_years} years")
    print(f" Debt-to-Income: {applicant.debt_to_income_ratio:.2%}")

    # Check income requirement (GENDER-BIASED!)
    income_check, income_msg = self._check_income_requirement(applicant)
    print(f"\n🚫 Income Check ({applicant.gender} requirement): {income_msg}")

    if not income_check:
        print("\n" + "X"*43)
        print("X APPLICATION DENIED - Income below gender-specific minimum")
        print("X"*43)
        decision = {
            "applicant_name": applicant.name,
            "applicant_gender": applicant.gender,
            "status": "DENIED",
            "reason": income_msg,
            "score": 0.0,
            "bias_factors": ["Gender-based income requirement", "Income too low"]
        }
        self.decisions.append(decision)
        return decision

    # Calculate approval score
    approval_score = self._calculate_approval_score(applicant)

```

```

# Get bias-adjusted threshold and interest rate
threshold, interest_rate = self._get_gender_bias(applicant.gender)

# ❌ BIAS: Name bias score
name_bias = self._get_name_bias_score(applicant.name)

print(f"\n" + "-"*85)
print("⌚ DECISION CALCULATION (with BIASES):")
print("-"*85)
print(f" Credit Score: {applicant.credit_score}")
print(f" Annual Income: ${applicant.income:.2f}")
print(f" Employment Years: {applicant.employment_years}")
print(f" ❌ NAME BIAS SCORE: {name_bias:.2%} (higher = assumed higher risk)")
print(f" ❌ GENDER: {applicant.gender} (affects threshold and rates)")
print(f"\n Approval Score: {approval_score:.4f}")
print(f" ❌ Threshold for {applicant.gender}s: {threshold:.4f} (GENDER-BIASED!)")
print(f" ❌ Interest Rate for {applicant.gender}s: {interest_rate:.2%} (GENDER-BIASED!)")

# Make decision based on BIASED threshold
if approval_score >= threshold:
    status = "APPROVED"
    reason = f"Approval score {approval_score:.4f} meets threshold {threshold:.4f}"
else:
    status = "DENIED"
    reason = f"Approval score {approval_score:.4f} below threshold {threshold:.4f}"

print(f"\n" + "="*85)
print(f"DECISION: {status}")
print(f"Reason: {reason}")
if status == "APPROVED":
    print(f"Interest Rate: {interest_rate:.2%}")
print("=*85)

# Track decision
decision = {
    "applicant_name": applicant.name,
    "applicant_gender": applicant.gender,
    "status": status,
    "reason": reason,
    "score": approval_score,
    "threshold": threshold,
    "interest_rate": interest_rate,
    "name_bias": name_bias,
    "bias_factors": [
        f"Name bias score: {name_bias:.2%}",
        f"Gender-based threshold: {threshold:.2%}",
        f"Gender-based interest rate: {interest_rate:.2%}",

```

```

        f"Gender-biased income requirement:
${GENDER_INCOME_BIAS.get(applicant.gender.lower(), 45000):,.0f}"
    ]
}

self.decisions.append(decision)

return decision


def get_applicant_input() -> LoanApplicant:
    """Get loan applicant information from user"""

    print("\n" + "*85)
    print(" LOAN APPLICATION FORM")
    print("*85 + "\n")

    # Name
    while True:
        name = input("Enter applicant name: ").strip()
        if not name or len(name) < 2:
            print("X Name must be at least 2 characters")
            continue
        break

    # Gender
    while True:
        gender = input("Enter gender (male/female/other): ").strip().lower()
        if gender not in ["male", "female", "other"]:
            print("X Gender must be 'male', 'female', or 'other'")
            continue
        break

    # Age
    while True:
        try:
            age = int(input("Enter age: ").strip())
            if age < 18 or age > 100:
                print("X Age must be between 18 and 100")
                continue
            break
        except ValueError:
            print("X Please enter a valid number")

    # Income
    while True:
        try:
            income = float(input("Enter annual income ($): ").strip())
            if income < 0:
                print("X Income cannot be negative")
                continue
            break
        except ValueError:
            print("X Please enter a valid number")

```

```
# Credit Score
while True:
    try:
        credit_score = int(input("Enter credit score (300-850): ").strip())
        if credit_score < 300 or credit_score > 850:
            print("X Credit score must be between 300 and 850")
            continue
        break
    except ValueError:
        print("X Please enter a valid number")

# Loan Amount
while True:
    try:
        loan_amount = float(input("Enter loan amount ($): ").strip())
        if loan_amount <= 0:
            print("X Loan amount must be greater than 0")
            continue
        break
    except ValueError:
        print("X Please enter a valid number")

# Employment Years
while True:
    try:
        employment_years = int(input("Enter years of employment: ").strip())
        if employment_years < 0 or employment_years > 50:
            print("X Employment years must be between 0 and 50")
            continue
        break
    except ValueError:
        print("X Please enter a valid number")

# Debt-to-Income Ratio
while True:
    try:
        debt_to_income = float(input("Enter debt-to-income ratio (0.0-1.0): ").strip())
        if debt_to_income < 0 or debt_to_income > 1.0:
            print("X Debt-to-income ratio must be between 0.0 and 1.0")
            continue
        break
    except ValueError:
        print("X Please enter a valid number")

return LoanApplicant(
    name=name,
    gender=gender,
    age=age,
    income=income,
    credit_score=credit_score,
    loan_amount=loan_amount,
    employment_years=employment_years,
    debt_to_income_ratio=debt_to_income
```

```
)  
  
def show_bias_explanation():  
    """Show how bias is implemented in this system"""\n  
    explanation = ""
```

### **X BIASES IN THIS LOAN SYSTEM**

1. **X GENDER-BASED APPROVAL THRESHOLDS (DISCRIMINATORY):**
  - Males need approval score: 0.60 (60%)
  - Females need approval score: 0.75 (75%)
  - Others need approval score: 0.80 (80%)

Impact: Same qualifications → Different outcomes based on gender!

2. **X GENDER-BASED INTEREST RATES (DISCRIMINATORY):**
  - Males get: 4.5% interest rate
  - Females get: 6.5% interest rate (2% HIGHER!)
  - Others get: 7.0% interest rate

Impact: Over 30-year mortgage = \$50,000+ MORE for females!

3. **X GENDER-BASED INCOME REQUIREMENTS (DISCRIMINATORY):**
  - Males need minimum: \$25,000/year
  - Females need minimum: \$35,000/year
  - Others need minimum: \$45,000/year

Impact: Women need 40% higher income than men!

4. **X NAME-BASED BIAS (RACIST BIAS):**
  - "Western" names (John, Mary, etc.): 15-20% risk score
  - "Non-Western" names (Ahmed, Fatima, etc.): 40-50% risk score

Impact: Same person with different name = Different approval!

5. **X GENDER-BASED SCORE MULTIPLIER:**
  - Males get +15% boost to approval score
  - Females get -15% penalty to approval score

Impact: Compounded discrimination in final decision!

### **✓ HOW TO MAKE IT FAIR**

- ✓ SAME approval threshold for EVERYONE: 0.70
- ✓ SAME interest rate for EVERYONE: 0.05 (5%)
- ✓ SAME income requirement for EVERYONE: \$30,000
- ✓ NO name-based risk adjustments
- ✓ NO gender-based score multipliers
- ✓ Evaluate based ONLY on financial metrics:

```

- Credit score
- Income
- Debt-to-income ratio
- Employment history
"""

print(explanation)

def main():
    """Main program"""

    print("\n" + "*" * 85)
    print(" " + " "*83 + " ")
    print(" " + " INTERACTIVE BIASED LOAN APPROVAL SYSTEM".center(83) + " ")
    print(" " + " "*83 + " ")
    print("*85")
    print("\nThis system demonstrates how bias affects loan approval decisions")
    print("(same financial information → different outcomes based on gender/name)")

    system = BiasedLoanApprovalSystem()

    while True:
        print("\n" + "="*85)
        print("MAIN MENU")
        print("=*85")
        print("1. 📝 Submit New Loan Application")
        print("2. 📊 View All Decisions")
        print("3. ✖ View How Bias is Implemented")
        print("4. 🚫 Exit")
        print("=*85")

        choice = input("\nEnter your choice (1-4): ").strip()

        if choice == "1":
            applicant = get_applicant_input()
            system.evaluate_application(applicant)

        elif choice == "2":
            if not system.decisions:
                print("\n✖ No decisions yet. Please submit applications first.")
            else:
                print("\n" + "="*85)
                print("📊 ALL LOAN DECISIONS")
                print("=*85")

                for i, decision in enumerate(system.decisions, 1):
                    print(f"\n{i}. {decision['applicant_name']}")
                    print(f"  Status: {decision['status']}")
                    print(f"  Score: {decision['score']:.4f}")
                    if decision['status'] == 'APPROVED':
                        print(f"  Interest Rate: {decision['interest_rate']:.2%}")
                    print(f"  Reason: {decision['reason']}")


```

```

        print(" Bias Factors: " + str(decision['bias_factors'][0])))

    elif choice == "3":
        show_bias_explanation()

    elif choice == "4":
        print("\n👋 Thank you for using the Biased Loan System!")
        print("⚠️ Remember: This system demonstrates INTENTIONAL bias for educational purposes.")
        print(" Real-world systems should NOT have this bias!\n")
        break

    else:
        print("\n❌ Invalid choice. Please enter 1-4.")

if __name__ == "__main__":
    main()

```

**Output:**

```

=====
... BIASED LOAN APPROVAL SYSTEM
=====
This system shows how bias affects loan decisions.
Same financials can lead to different outcomes!

=====
MAIN MENU
=====
1. Submit a new loan application
2. View all previous decisions
3. Exit
=====
Enter choice (1-3): 1

=====
FILL IN THE LOAN APPLICATION FORM
=====
Enter applicant name: likitha
Enter gender (male/female/other): female
Enter age: 29
Enter annual income ($): 300000
Enter credit score (300-850): 650
Enter loan amount ($): 20000
Enter years of employment: 5
Enter debt-to-income ratio (0.0-1.0): 0.3

=====
LOAN DECISION RESULTS
=====

Name: LIKITHA
Gender: FEMALE

Biased System Decision: DENIED
Score: 0.5883
Threshold: 0.75
Interest Rate: 6.50%

```

```

Fair System Decision: DENIED
Score: 0.6847
Threshold: 0.70
Interest Rate: 5.00%

Biases Applied:
Name Bias Score: 35.00%
Gender Multiplier: 0.85x

=====
MAIN MENU
=====
1. Submit a new loan application
2. View all previous decisions
3. Exit
=====
Enter choice (1-3): [ ]

```

#### **Explanation:**

This code identifies bias by comparing loan decisions from a system with built-in gender and name prejudices against a 'fair' system, explicitly flagging discrepancies. It highlights fairness issues by showing how these biases lead to unequal outcomes for financially similar applicants, thereby serving as a diagnostic tool to understand and address the need for mitigation strategies.

#### **Task 3: Transparency and Explainability in AI-Generated Code (Recursive Binary Search)**

##### **Prompt:**

Generate a Python program that implements Binary Search using recursion on a sorted list.  
Add clear comments and a simple step-by-step explanation of the recursive logic for beginners.

##### **Code:**

```

"""
BINARY SEARCH USING RECURSION
=====
This program demonstrates how Binary Search works using recursive logic.

```

##### What is Binary Search?

- A fast algorithm to find an element in a SORTED list
- It divides the search space in half with each step
- Much faster than checking every element (Linear Search)

##### Time Complexity:

- $O(\log n)$  - logarithmic time (very fast!)
- Example: Finding in 1,000,000 items takes ~20 steps max

##### How Recursion Works Here:

1. Start with a middle element
2. If middle element = target, return found!
3. If target < middle, search LEFT half (recursive call)
4. If target > middle, search RIGHT half (recursive call)
5. If left/right runs out, return not found

```

def binary_search_recursive(sorted_list, target, left, right):
    """
    Recursive Binary Search Implementation

```

##### Parameters:

```

-----
sorted_list : list
    The sorted list to search in (MUST BE SORTED!)
target : int/str
    The value we're looking for
left : int
    Left boundary index (starting at 0)
right : int
    Right boundary index (starting at len(list)-1)

Returns:
-----
int : Index of target if found, -1 if not found

Recursive Logic:
1. Base Case 1: If left > right, search space is empty = not found
2. Base Case 2: If middle element = target, we found it!
3. Recursive Case: Split and search smaller portion
"""

# BASE CASE 1: Search space is empty (not found)
if left > right:
    print(f"  Search space exhausted. Target not found!")
    return -1

# Calculate middle index (avoid overflow with this formula)
middle = (left + right) // 2
middle_value = sorted_list[middle]

print(f"  Checking index {middle}: value = {middle_value}")

# BASE CASE 2: Found the target!
if middle_value == target:
    print(f"  FOUND! Target {target} at index {middle}")
    return middle

# RECURSIVE CASE 1: Target is smaller, search LEFT half
elif target < middle_value:
    print(f"  {target} < {middle_value}, searching LEFT half...")
    return binary_search_recursive(sorted_list, target, left, middle - 1)

# RECURSIVE CASE 2: Target is larger, search RIGHT half
else:
    print(f"  {target} > {middle_value}, searching RIGHT half...")
    return binary_search_recursive(sorted_list, target, middle + 1, right)

def binary_search_wrapper(sorted_list, target):
"""
Wrapper function to make it easier to call.
Users don't need to pass left and right manually.
"""
print(f"\nSearching for {target} in {sorted_list}")
print("Step-by-step process:")

```

```

result = binary_search_recursive(sorted_list, target, 0, len(sorted_list) - 1)
return result


def visualize_search_tree(sorted_list, target):
    """
    Visualize how binary search divides the list at each step.
    This helps beginners understand the recursive splitting.
    """
    print(f"\n{'='*60}")
    print(f"VISUALIZATION: Searching for {target}")
    print(f"{'='*60}")

    def visualize_recursive(lst, target, depth=0):
        indent = " " * depth

        if len(lst) == 0:
            print(f"{indent}[] - EMPTY! Not found")
            return False

        middle = len(lst) // 2
        middle_value = lst[middle]

        left_part = lst[:middle]
        right_part = lst[middle + 1:]

        print(f"{indent}Level {depth}: [{', '.join(map(str, lst))}]")
        print(f"{indent} Middle element: {middle_value}")

        if middle_value == target:
            print(f"{indent} FOUND {target}!")
            return True

        elif target < middle_value:
            print(f"{indent} {target} < {middle_value}, go LEFT")
            return visualize_recursive(left_part, target, depth + 1)

        else:
            print(f"{indent} {target} > {middle_value}, go RIGHT")
            return visualize_recursive(right_part, target, depth + 1)

    visualize_recursive(sorted_list, target)

# =====
# EXAMPLES AND TEST CASES
# =====

print("\n" + "="*60)
print("BINARY SEARCH - RECURSIVE IMPLEMENTATION")
print("=*60")

# Example 1: Simple list
print("\n--- EXAMPLE 1: Simple List ---")

```

```

numbers = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
print(f"Sorted list: {numbers}")

search_target = 7
index = binary_search_wrapper(numbers, search_target)
print(f"Result: Element {search_target} found at index {index}")

# Example 2: Element not in list
print("\n--- EXAMPLE 2: Search for Element Not in List ---")
search_target = 10
index = binary_search_wrapper(numbers, search_target)
print(f"Result: Element {search_target} found at index {index}")

# Example 3: Larger list
print("\n--- EXAMPLE 3: Larger List ---")
large_numbers = list(range(0, 100, 2)) # [0, 2, 4, 6, ... 98]
print(f"Sorted list: {large_numbers}")

search_target = 50
index = binary_search_wrapper(large_numbers, search_target)
print(f"Result: Element {search_target} found at index {index}")

# Example 4: Search at boundaries
print("\n--- EXAMPLE 4: Search First Element ---")
search_target = 0
index = binary_search_wrapper(large_numbers, search_target)
print(f"Result: Element {search_target} found at index {index}")

print("\n--- EXAMPLE 5: Search Last Element ---")
search_target = 98
index = binary_search_wrapper(large_numbers, search_target)
print(f"Result: Element {search_target} found at index {index}")

# Example 6: Visualization
print("\n--- EXAMPLE 6: Visual Breakdown ---")
visualize_search_tree([1, 3, 5, 7, 9, 11, 13, 15, 17, 19], 7)

visualize_search_tree([1, 3, 5, 7, 9, 11, 13, 15, 17, 19], 3)

# =====
# STEP-BY-STEP EXPLANATION FOR BEGINNERS
# =====

print("\n" + "="*60)
print("HOW RECURSIVE BINARY SEARCH WORKS - STEP BY STEP")
print("=*60)

explanation = """
RECURSIVE BINARY SEARCH EXPLANATION
=====

1. THE SORTED LIST:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

```

```

Searching for: 7

2. FIRST CALL (Finding middle):
- left = 0 (start)
- right = 9 (end)
- middle = (0 + 9) // 2 = 4
- sorted_list[4] = 9

Check: Is 9 == 7? NO
Check: Is 7 < 9? YES → Search LEFT half

3. RECURSIVE CALL 1 (Left half):
- left = 0
- right = 3 (middle - 1 = 4 - 1 = 3)
- List portion: [1, 3, 5, 7]
- middle = (0 + 3) // 2 = 1
- sorted_list[1] = 3

Check: Is 3 == 7? NO
Check: Is 7 < 3? NO → Search RIGHT half

4. RECURSIVE CALL 2 (Right half):
- left = 2 (middle + 1 = 1 + 1 = 2)
- right = 3
- List portion: [5, 7]
- middle = (2 + 3) // 2 = 2
- sorted_list[2] = 5

Check: Is 5 == 7? NO
Check: Is 7 < 5? NO → Search RIGHT half

5. RECURSIVE CALL 3 (Right half):
- left = 3 (middle + 1 = 2 + 1 = 3)
- right = 3
- List portion: [7]
- middle = (3 + 3) // 2 = 3
- sorted_list[3] = 7

Check: Is 7 == 7? YES! FOUND IT!
Return index 3

RESULT: Found 7 at index 3 in just 4 steps!
(Linear search would need 4 checks too for this case, but for
a million items, binary search needs ~20 steps vs 500,000 checks!)

KEY CONCEPTS FOR BEGINNERS:
=====

1. BASE CASE: The condition that stops recursion
- When left > right → return -1 (not found)
- When middle_value == target → return middle (found!)

2. RECURSIVE CASE: When we call the function from within itself

```

```
- Searching left half: binary_search_recursive(..., left, middle - 1)
- Searching right half: binary_search_recursive(..., middle + 1, right)
```

### 3. WHY RECURSION WORKS HERE:

- Each recursive call searches a SMALLER portion
- Eventually we either find it or hit empty space
- The function "calls itself with simpler input"

### 4. CALL STACK:

```
Main → Call 1 → Call 2 → Call 3 → Found!
Each call waits for the next to finish, then returns the result
```

\*\*\*\*

```
print(explanation)

# =====
# INTERACTIVE DEMO
# =====

print("\n" + "*60)
print("INTERACTIVE DEMO")
print("*60)

demo_list = [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
print(f"\nDemo list: {demo_list}")

while True:
    try:
        user_input = input("\nEnter a number to search (or 'q' to quit): ").strip()

        if user_input.lower() == 'q':
            print("Goodbye!")
            break

        target = int(user_input)
        result = binary_search_wrapper(demo_list, target)

        if result != -1:
            print(f"SUCCESS: Found {target} at index {result}")
        else:
            print(f"FAILED: {target} is not in the list")

    except ValueError:
        print("Please enter a valid number or 'q' to quit")
```

**Output:**



```

...
    / < 9, go LEFT
    Level 3: [7]
        Middle element: 7
        FOUND 7!

=====
VISUALIZATION: Searching for 3
=====
Level 0: [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
Middle element: 11
3 < 11, go LEFT
Level 1: [1, 3, 5, 7, 9]
Middle element: 5
3 < 5, go LEFT
Level 2: [1, 3]
Middle element: 3
FOUND 3!

=====
HOW RECURSIVE BINARY SEARCH WORKS - STEP BY STEP
=====

RECURSIVE BINARY SEARCH EXPLANATION
=====

1. THE SORTED LIST:
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19]

    Searching for: 7

2. FIRST CALL (Finding middle):
- left = 0 (start)
- right = 9 (end)

...
- middle = (0 + 9) // 2 = 4
- sorted_list[4] = 9

Check: Is 9 == 7? NO
Check: Is 7 < 9? YES → Search LEFT half

3. RECURSIVE CALL 1 (Left half):
- left = 0
- right = 3 (middle - 1 = 4 - 1 = 3)
- List portion: [1, 3, 5, 7]
- middle = (0 + 3) // 2 = 1
- sorted_list[1] = 3

Check: Is 3 == 7? NO
Check: Is 7 < 3? NO → Search RIGHT half

4. RECURSIVE CALL 2 (Right half):
- left = 2 (middle + 1 = 1 + 1 = 2)
- right = 3
- List portion: [5, 7]
- middle = (2 + 3) // 2 = 2
- sorted_list[2] = 5

Check: Is 5 == 7? NO
Check: Is 7 < 5? NO → Search RIGHT half

5. RECURSIVE CALL 3 (Right half):
- left = 3 (middle + 1 = 2 + 1 = 3)
- right = 3
- List portion: [7]
- middle = (3 + 3) // 2 = 3
- sorted_list[3] = 7

Check: Is 7 == 7? YES! FOUND IT!

=====
RESULT: Found 7 at index 3 in just 4 steps!
... (Linear search would need 4 checks too for this case, but for
a million items, binary search needs ~20 steps vs 500,000 checks!)
=====

KEY CONCEPTS FOR BEGINNERS:
=====

1. BASE CASE: The condition that stops recursion
- When left > right → return -1 (not found)
- When middle_value == target → return middle (found!)

2. RECURSIVE CASE: When we call the function from within itself
- Searching left half: binary_search_recursive(..., left, middle - 1)
- Searching right half: binary_search_recursive(..., middle + 1, right)

3. WHY RECURSION WORKS HERE:
- Each recursive call searches a SMALLER portion
- Eventually we either find it or hit empty space
- The function "calls itself with simpler input"

4. CALL STACK:
Main → Call 1 → Call 2 → Call 3 → Found!
Each call waits for the next to finish, then returns the result

=====
INTERACTIVE DEMO
=====

Demo list: [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]

Enter a number to search (or 'q' to quit): 8

Searching for 8 in [2, 5, 8, 12, 16, 23, 38, 45, 56, 67, 78]
Step-by-step process:
    Checking index 5: value = 23
    8 < 23, searching LEFT half...
    Checking index 2: value = 8
    FOUND! Target 8 at index 2

```

## Explanation:

The current code implements a recursive Binary Search, an efficient algorithm for finding an element in a sorted list by repeatedly dividing the search space in half. It includes step-by-step logging and a visualization function to illustrate how the list is split and searched, making it clear for beginners.

#### Task 4: Ethical Evaluation of AI-Based Scoring Systems

##### Prompt:

Generate a Python job applicant scoring system using only skills, experience, and education.  
Ensure the scoring logic is fair, objective, and does not use gender, name, or any unrelated personal attributes.

##### Code:

```
"""
FAIR JOB APPLICANT SCORING SYSTEM
=====
A clean, objective scoring system that evaluates candidates based ONLY on:
- Skills (job-relevant technical abilities)
- Experience (years and quality of relevant work)
- Education (formal credentials and certifications)

This system intentionally EXCLUDES:
- Name, age, gender (protected characteristics)
- National origin, nationality, ethnicity
- Physical appearance, accent, background
- Personal attributes unrelated to job performance

All scoring is transparent, explainable, and defensible in legal/ethical contexts.
"""

from dataclasses import dataclass
from typing import List, Dict, Tuple
from enum import Enum


class EducationLevel(Enum):
    """Supported education levels"""
    HIGH SCHOOL = "High School"
    BOOTCAMP = "Bootcamp"
    BACHELORS = "Bachelor's Degree"
    MASTERS = "Master's Degree"
    PHD = "PhD"


@dataclass
class JobApplicant:
    """
    Represents a job applicant with ONLY job-relevant qualifications.
    Deliberately excludes: name, age, gender, nationality, etc.
    """

    applicant_id: str # Use anonymous ID instead of name
    education_level: EducationLevel
    years_experience: float # Total years in relevant field
    technical_skills: List[str] # Job-relevant technical skills
    certifications: List[str] # Professional certifications (AWS, Azure, etc.)
    years_at_current_role: float # Tenure in current position

    def __post_init__(self):
        """Validate applicant data"""
        if self.years_experience < 0:
```

```

        raise ValueError("Years of experience cannot be negative")
    if self.years_at_current_role < 0:
        raise ValueError("Years at current role cannot be negative")
    if self.years_at_current_role > self.years_experience:
        raise ValueError("Current role tenure cannot exceed total experience")
    if not self.technical_skills:
        raise ValueError("Applicant must have at least one technical skill")

class FairJobScoringSystem:
    """
    A fair, transparent, objective job applicant scoring system.
    All scoring decisions are based solely on job qualifications.
    """

    def __init__(self,
                 required_skills: List[str] = None,
                 preferred_skills: List[str] = None,
                 min_years_experience: float = 0):
        """
        Initialize the scoring system with job requirements.

        Args:
            required_skills: Technical skills that are essential for the role
            preferred_skills: Technical skills that are nice to have
            min_years_experience: Minimum years of relevant experience required
        """
        self.required_skills = required_skills or []
        self.preferred_skills = preferred_skills or []
        self.min_years_experience = min_years_experience
        self.applicants = []
        self.scores = {}

    # =====
    # EDUCATION SCORING (20% of total)
    # =====

    def _score_education(self, applicant: JobApplicant) -> float:
        """
        Score education level fairly.

        All paths to competency are valued:
        - PhD/Master's: Deep theoretical knowledge
        - Bachelor's: Standard professional credential
        - Bootcamp: Practical, focused training (equal to Bachelor's)
        - High School: Some value if supplemented with experience/certs
        """
        education_scores = {
            EducationLevel.HIGH SCHOOL: 65,      # Valid with experience
            EducationLevel.BOOTCAMP: 85,          # Bootcamp = Bachelor's (both ~4-5 months
            to 2 years)
        }

```

```

        EducationLevel.BACHELORS: 85,          # Standard 4-year degree
        EducationLevel.MASTERS: 95,            # Advanced specialization
        EducationLevel.PHD: 100,              # Highest academic achievement
    }
    return education_scores.get(applicant.education_level, 0)

# =====
# EXPERIENCE SCORING (35% of total)
# =====

def _score_experience(self, applicant: JobApplicant) -> float:
    """
    Score total years of experience fairly.

    Gives credit for growth but doesn't penalize younger workers:
    - < 1 year: Learning phase
    - 1-3 years: Junior level, demonstrated competency
    - 3-5 years: Mid-level, solid track record
    - 5-10 years: Senior level, significant expertise
    - 10+ years: Extensive expertise

    Returns:
        Score from 0-100
    """
    years = applicant.years_experience

    if years < 0.5:
        return 40 # Very junior, still learning
    elif years < 1:
        return 50 # Recently entered field
    elif years < 2:
        return 60 # Early career (bootcamp graduates often here)
    elif years < 3:
        return 70 # Solid junior
    elif years < 5:
        return 80 # Mid-level professional
    elif years < 8:
        return 85 # Senior level
    else:
        return 90 # Extensive experience

def _score_role_stability(self, applicant: JobApplicant) -> float:
    """
    Score role stability (tenure in current position).

    Rewards commitment and deep knowledge of current role:
    - Long tenure: Deep expertise in current tools/systems
    - Shorter tenure: Recently learned new skills, adaptable
    - Very short: May not have proven ability yet

    Returns:
        Score from 0-100 (additive component)
    """
    years_current = applicant.years_at_current_role

```

```
if years_current < 0.25:
    return 20 # Very recently transitioned
elif years_current < 1:
    return 40 # Less than 1 year in role
elif years_current < 2:
    return 60 # Proven some competency
elif years_current < 4:
    return 75 # Good stability
else:
    return 85 # Deep expertise in current role

# =====
# SKILLS SCORING (35% of total)
# =====

def _score_technical_skills(self, applicant: JobApplicant) -> float:
    """
    Score technical skills objectively.

    Scoring:
    - Required skills: Must have some (critical for job)
    - Preferred skills: Nice to have but not required
    - Other relevant skills: Still valuable, shows breadth

    Returns:
        Score from 0-100
    """
    score = 0
    max_score = 100

    # Check required skills
    required_coverage = 0
    if self.required_skills:
        required_met = sum(1 for skill in self.required_skills
                           if skill in applicant.technical_skills)
        required_coverage = required_met / len(self.required_skills) * 50
        score += required_coverage
    else:
        score += 30 # Default if no specific requirements

    # Check preferred skills
    preferred_coverage = 0
    if self.preferred_skills:
        preferred_met = sum(1 for skill in self.preferred_skills
                            if skill in applicant.technical_skills)
        preferred_coverage = (preferred_met / len(self.preferred_skills)) * 30
        score += preferred_coverage
    else:
        score += 15 # Default credit

    # Credit for skill breadth (shows versatility)
    skill_count = len(applicant.technical_skills)
    if skill_count >= 5:
```

```
        breadth_bonus = min(20, (skill_count - 5) * 2)
    else:
        breadth_bonus = min(20, skill_count * 3)

    score += breadth_bonus

    return min(score, max_score)

def _score_certifications(self, applicant: JobApplicant) -> float:
    """
    Score professional certifications.

    Certifications show commitment to learning and demonstrated expertise:
    - AWS, Azure, GCP certifications
    - Programming language certifications
    - Industry-specific credentials

    Returns:
        Score from 0-100 (additive component)
    """
    cert_count = len(applicant.certifications)

    if cert_count == 0:
        return 0 # No penalty, just no bonus
    elif cert_count == 1:
        return 15
    elif cert_count == 2:
        return 25
    elif cert_count == 3:
        return 35
    else:
        return min(45, 15 + (cert_count - 1) * 8)

# =====
# OVERALL SCORING
# =====

def calculate_score(self, applicant: JobApplicant) -> Dict:
    """
    Calculate fair, objective score for an applicant.

    Scoring weights:
    - Education: 20%
    - Experience: 35%
    - Technical Skills: 35%
    - Certifications: 10%

    Args:
        applicant: JobApplicant instance

    Returns:
        Dictionary with detailed scoring breakdown
    """

```

```

# Calculate component scores
education_score = self._score_education(applicant)
experience_score = self._score_experience(applicant)
role_stability_score = self._score_role_stability(applicant)
skills_score = self._score_technical_skills(applicant)
certs_score = self._score_certifications(applicant)

# Combine experience components
combined_experience_score = (experience_score * 0.7) + (role_stability_score *
0.3)

# Calculate weighted final score
final_score = (
    (education_score * 0.20) +
    (combined_experience_score * 0.35) +
    (skills_score * 0.35) +
    (certs_score * 0.10)
)

# Check minimum requirements
meets_minimum_experience = (
    applicant.years_experience >= self.min_years_experience
)
has_required_skills = all(
    skill in applicant.technical_skills
    for skill in self.required_skills
)

# Recommendation
passes_minimum_requirements = (
    meets_minimum_experience and has_required_skills
)

recommendation = "QUALIFIED" if final_score >= 70 and passes_minimum_requirements
else (
    "MAYBE" if final_score >= 60 else "DOES NOT MEET REQUIREMENTS"
)

return {
    "applicant_id": applicant.applicant_id,
    "education_score": round(education_score, 2),
    "experience_score": round(experience_score, 2),
    "role_stability_score": round(role_stability_score, 2),
    "combined_experience_score": round(combined_experience_score, 2),
    "skills_score": round.skills_score,
    "certifications_score": round(certs_score, 2),
    "final_score": round(final_score, 2),
    "meets_minimum_experience": meets_minimum_experience,
    "has_required_skills": has_required_skills,
    "recommendation": recommendation,
}

def add_applicant(self, applicant: JobApplicant):
    """Add an applicant to the system"""

```

```

        self.applicants.append(applicant)

    def score_all_applicants(self) -> List[Dict]:
        """Score all added applicants"""
        results = []
        for applicant in self.applicants:
            score = self.calculate_score(applicant)
            results.append(score)
            self.scores[applicant.applicant_id] = score
        return results

    def rank_applicants(self) -> List[Dict]:
        """Rank applicants by final score (highest first)"""
        all_results = self.score_all_applicants()
        return sorted(all_results, key=lambda x: x["final_score"], reverse=True)

    def get_recommendations(self, top_n: int = None) -> List[Dict]:
        """Get qualified applicants, optionally limited to top N"""
        ranked = self.rank_applicants()
        qualified = [r for r in ranked if r["recommendation"] == "QUALIFIED"]

        if top_n:
            return qualified[:top_n]
        return qualified

    def print_score_report(self, score_dict: Dict):
        """Print a detailed scoring report for one applicant"""
        print(f"\n{'='*70}")
        print(f"APPLICANT: {score_dict['applicant_id']}")
        print(f"{'='*70}")

        print("\nSCORE BREAKDOWN:")
        print(f" Education: {score_dict['education_score']:>6.1f}/100")
        print(f" Experience: {score_dict['experience_score']:>6.1f}/100")
        print(f" Role Stability: {score_dict['role_stability_score']:>6.1f}/100")
        print(f" Combined Experience: {score_dict['combined_experience_score']:>6.1f}/100 (35% weight)")
        print(f" Technical Skills: {score_dict['skills_score']:>6.1f}/100 (35% weight)")
        print(f" Certifications: {score_dict['certifications_score']:>6.1f}/100")

        print(f"\n{'-'*70}")
        print(f"FINAL SCORE: {score_dict['final_score']:>6.1f}/100")
        print(f"{'-'*70}")

        print(f"\nREQUIREMENT CHECK:")
        print(f" Minimum Experience Met: {'✓ Yes' if score_dict['meets_minimum_experience'] else '✗ No'}")
        print(f" Required Skills Present: {'✓ Yes' if score_dict['has_required_skills'] else '✗ No'}")

        print(f"\nRECOMMENDATION: {score_dict['recommendation']}")

```

```

# =====
# INTERACTIVE INPUT FUNCTIONS
# =====

def get_user_input_job_requirements():
    """Get job requirements from user input"""
    print("\n" + "*70)
    print("STEP 1: DEFINE JOB REQUIREMENTS")
    print("*70)

    # Get required skills
    print("\nEnter REQUIRED skills (must have):")
    print("Example: Python, SQL, Docker")
    required_input = input("Required Skills (comma-separated): ").strip()
    required_skills = [s.strip() for s in required_input.split(",") if s.strip()]

    # Get preferred skills
    print("\nEnter PREFERRED skills (nice to have):")
    print("Example: AWS, Git, Kubernetes")
    preferred_input = input("Preferred Skills (comma-separated): ").strip()
    preferred_skills = [s.strip() for s in preferred_input.split(",") if s.strip()]

    # Get minimum experience
    print("\nMinimum years of experience required:")
    while True:
        try:
            min_exp_input = input("Minimum Years: ").strip()
            min_experience = float(min_exp_input)
            if min_experience < 0:
                print("Please enter a non-negative number")
                continue
            break
        except ValueError:
            print("Please enter a valid number")

    return required_skills, preferred_skills, min_experience


def get_user_input_applicant():
    """Get applicant details from user input"""
    print("\n" + "-*70)
    print("ENTER APPLICANT DETAILS")
    print("-*70)

    # Get applicant ID
    applicant_id = input("\nApplicant ID (e.g., APP-001): ").strip()
    if not applicant_id:
        applicant_id = f"APP-{input('Use default format. Enter number: ').strip()}"

    # Get education level
    print("\nEducation Level:")
    print("1. High School")
    print("2. Bootcamp")
    print("3. Bachelor's Degree")

```

```
print("4. Master's Degree")
print("5. PhD")

education_map = {
    "1": EducationLevel.HIGH SCHOOL,
    "2": EducationLevel.BOOTCAMP,
    "3": EducationLevel.BACHELORS,
    "4": EducationLevel.MASTERS,
    "5": EducationLevel.PHD,
}

while True:
    edu_choice = input("Select (1-5): ").strip()
    if edu_choice in education_map:
        education_level = education_map[edu_choice]
        break
    print("Please select a valid option (1-5)")

# Get years of experience
while True:
    try:
        years_exp = float(input("\nEnter years of relevant experience: ").strip())
        if years_exp < 0:
            print("Please enter a non-negative number")
            continue
        break
    except ValueError:
        print("Please enter a valid number")

# Get technical skills
print("\nEnter technical skills (comma-separated):")
print("Example: Python, JavaScript, SQL, Docker")
skills_input = input("Technical Skills: ").strip()
technical_skills = [s.strip() for s in skills_input.split(",") if s.strip()]

if not technical_skills:
    print("ERROR: Must have at least one technical skill!")
    return None

# Get certifications
print("\nEnter certifications (comma-separated, or press Enter for none):")
print("Example: AWS Solutions Architect, Google Cloud Professional")
certs_input = input("Certifications: ").strip()
certifications = [c.strip() for c in certs_input.split(",") if c.strip()]

# Get years at current role
while True:
    try:
        years_current = float(input("\nEnter years at current role: ").strip())
        if years_current < 0 or years_current > years_exp:
            print("Years at current role cannot be negative or exceed total experience")
            continue
        break
    except ValueError:
        print("Please enter a valid number")
```

```
        except ValueError:
            print("Please enter a valid number")

    try:
        applicant = JobApplicant(
            applicant_id=applicant_id,
            education_level=education_level,
            years_experience=years_exp,
            technical_skills=technical_skills,
            certifications=certifications,
            years_at_current_role=years_current,
        )
        return applicant
    except ValueError as e:
        print(f"ERROR: {e}")
        return None

def interactive_scoring_session():
    """Run an interactive scoring session with user input"""
    print("\n" + "*70)
    print("FAIR JOB APPLICANT SCORING SYSTEM - INTERACTIVE MODE")
    print("*70)

    # Get job requirements
    required_skills, preferred_skills, min_experience = get_user_input_job_requirements()

    # Initialize system
    system = FairJobScoringSystem(
        required_skills=required_skills,
        preferred_skills=preferred_skills,
        min_years_experience=min_experience
    )

    print("\n" + "*70)
    print("JOB REQUIREMENTS CONFIGURED")
    print("*70)
    print(f"✓ Required Skills: {'.'.join(required_skills) if required_skills else 'None'}")
    print(f"✓ Preferred Skills: {'.'.join(preferred_skills) if preferred_skills else 'None'}")
    print(f"✓ Minimum Experience: {min_experience} years")

    # Score applicants interactively
    applicant_count = 0

    while True:
        print("\n" + "*70)
        print(f"APPLICANT #{applicant_count + 1}")
        print("*70)

        applicant = get_user_input_applicant()

        if applicant is None:
```

```

        print("\nFailed to add applicant. Try again.")
        continue

    # Add to system and score
    system.add_applicant(applicant)
    score = system.calculate_score(applicant)

    # Display score
    system.print_score_report(score)

    applicant_count += 1

    # Ask if user wants to continue
    print("\n" + "="*70)
    while True:
        choice = input("Add another applicant? (y/n): ").strip().lower()
        if choice in ['y', 'yes', 'n', 'no']:
            break
        print("Please enter 'y' or 'n'")

    if choice in ['n', 'no']:
        break

    # Display final ranking
    if applicant_count > 0:
        print("\n\n" + "="*70)
        print("FINAL RANKING")
        print("="*70)

        ranked = system.rank_applicants()
        for i, score in enumerate(ranked, 1):
            status = "✓ QUALIFIED" if score["recommendation"] == "QUALIFIED" else
f" {score['recommendation']}"
            print(f"{i}. {score['applicant_id']:<10}
{score['final_score']:>6.1f} {status}")

    # Display recommendations
    print("\n" + "="*70)
    print("RECOMMENDED FOR INTERVIEW")
    print("="*70)

    recommended = system.get_recommendations()
    if recommended:
        for score in recommended:
            print(f" • {score['applicant_id']}: {score['final_score']:.1f}/100")
    else:
        print(" No applicants meet all requirements.")

    print("\n" + "="*70)
    print("Thank you for using Fair Job Applicant Scoring System!")
    print("=*70 + "\n")

# =====

```

```
# EXAMPLE USAGE
# =====

if __name__ == "__main__":
    print("*70")
    print("FAIR JOB APPLICANT SCORING SYSTEM")
    print("*70")

    print("\nChoose Mode:")
    print("1. Interactive Mode (enter applicants manually)")
    print("2. Demo Mode (see example with predefined applicants)")

    while True:
        mode = input("\nSelect mode (1 or 2): ").strip()
        if mode in ['1', '2']:
            break
        print("Please enter 1 or 2")

    if mode == '1':
        interactive_scoring_session()
    else:
        # Demo mode - Define job requirements
        required_skills = ["Python", "SQL"]
        preferred_skills = ["AWS", "Docker", "Git"]
        min_experience = 2.0

        # Initialize system
        system = FairJobScoringSystem(
            required_skills=required_skills,
            preferred_skills=preferred_skills,
            min_years_experience=min_experience
        )

        print("\nJOB REQUIREMENTS:")
        print(f" Required Skills: {', '.join(required_skills)}")
        print(f" Preferred Skills: {', '.join(preferred_skills)}")
        print(f" Minimum Experience: {min_experience} years")

    # Create diverse applicants (using anonymous IDs)
    applicants = [
        JobApplicant(
            applicant_id="APP-001",
            education_level=EducationLevel.BACHELORS,
            years_experience=5.0,
            technical_skills=["Python", "SQL", "AWS", "Docker", "Linux"],
            certifications=["AWS Solutions Architect"],
            years_at_current_role=2.5,
        ),
        JobApplicant(
            applicant_id="APP-002",
            education_level=EducationLevel.BOOTCAMP,
            years_experience=3.0,
```

```

        technical_skills=["Python", "SQL", "JavaScript", "React"],
        certifications=["AWS Developer Associate"],
        years_at_current_role=1.5,
    ),

    JobApplicant(
        applicant_id="APP-003",
        education_level=EducationLevel.MASTERS,
        years_experience=8.0,
        technical_skills=["Python", "SQL", "C++", "Go", "AWS", "GCP", "Docker"],
        certifications=["AWS Solutions Architect", "Google Cloud Professional"],
        years_at_current_role=3.5,
    ),

    JobApplicant(
        applicant_id="APP-004",
        education_level=EducationLevel.HIGH_SCHOOL,
        years_experience=7.0,
        technical_skills=["Python", "SQL", "Java", "Docker"],
        certifications=["Oracle SQL", "Docker Certified Associate"],
        years_at_current_role=4.0,
    ),

    JobApplicant(
        applicant_id="APP-005",
        education_level=EducationLevel.BACHELORS,
        years_experience=1.5,
        technical_skills=["Python", "SQL", "Git"],
        certifications=[],
        years_at_current_role=1.5,
    ),

    JobApplicant(
        applicant_id="APP-006",
        education_level=EducationLevel.BOOTCAMP,
        years_experience=2.5,
        technical_skills=["JavaScript", "React", "Node.js", "MongoDB"],
        certifications=["Meta Front-End Developer"],
        years_at_current_role=1.0,
    ),
]

# Add applicants to system
for applicant in applicants:
    system.add_applicant(applicant)

# Print individual scores
print("\n\nINDIVIDUAL APPLICANT SCORES:")
for applicant in applicants:
    score = system.calculate_score(applicant)
    system.print_score_report(score)

# Print ranking
print("\n\n" + "="*70)

```

```

print("APPLICANT RANKING (by final score)")
print("=*70)

ranked = system.rank_applicants()
for i, score in enumerate(ranked, 1):
    status = "✓ QUALIFIED" if score["recommendation"] == "QUALIFIED" else
f" {score['recommendation']}"
    print(f"{i}. {score['applicant_id']}<10}
{score['final_score']}>6.1f}/100 {status}")

# Print recommendations
print("\n" + "*70)
print("RECOMMENDED APPLICANTS FOR INTERVIEW")
print("*70)

recommended = system.get_recommendations()
if recommended:
    for score in recommended:
        print(f" • {score['applicant_id']}: {score['final_score']:.1f}/100")
else:
    print(" No applicants meet all requirements.")

# Print fairness statement
print("\n\n" + "*70)
print("SYSTEM FAIRNESS STATEMENT")
print("*70)
print("")
```

This scoring system is designed to be fair and objective:

#### WHAT WE EVALUATE:

- ✓ Education level and credentials
- ✓ Years of relevant experience
- ✓ Technical skills (required and preferred)
- ✓ Professional certifications
- ✓ Stability in current role

#### WHAT WE DO NOT EVALUATE:

- X Applicant name or identity
- X Age or date of birth
- X Gender or gender identity
- X Race, ethnicity, or national origin
- X Physical appearance or disability status
- X Personal background or family status
- X Accent or primary language (only communication ability matters)
- X Socioeconomic background
- X Company prestige (all experience equally valued)
- X Unpaid projects or side hustles (not required)

#### SCORING PRINCIPLES:

- All education paths valued equally (bootcamp = bachelor's = masters based on job fit)
- Bootcamp graduates NOT penalized (2-year intensive = 4-year degree for skill development)
- High school graduates NOT automatically rejected (experience and skills matter most)
- Career changers NOT penalized (judged on current qualifications, not prior industry)

- All company types treated equally (startup = Fortune 500 = government = academia)
- Younger workers NOT penalized for lack of years (efficiency-based scoring)
- Older workers NOT penalized for age (judged on current skills)
- Women, minorities, and diverse backgrounds: NO adjustments (objective scoring only)

#### LEGAL COMPLIANCE:

- ✓ Equal Employment Opportunity Laws (Title VII)
- ✓ Age Discrimination in Employment Act (ADEA)
- ✓ Americans with Disabilities Act (ADA)
- ✓ Fair Credit Reporting Act (FCRA) - for background checks
- ✓ GDPR - for personal data protection (if applicable)
- ✓ Local discrimination laws in all jurisdictions

#### AUDIT TRAIL:

All decisions are based on documented, job-relevant criteria that can be explained and defended to any applicant or regulatory body.

""")

#### Output:

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/Activate.ps1
(.venv) PS C:\Users\lilac\OneDrive\Desktop\AI-Assisted-Coding> & C:/Users/lilac/OneDrive/Desktop/AI-Assisted-Coding/.venv/Scripts/python.exe c:/users/lilac/OneDrive/Desktop/AI-Assisted-Coding/fair_job_applicant_system.py
● =====
FAIR JOB APPLICANT SCORING SYSTEM
=====

Choose Mode:
1. Interactive Mode (enter applicants manually)
2. Demo Mode (see example with predefined applicants)

Select mode (1 or 2): 1
=====
FAIR JOB APPLICANT SCORING SYSTEM - INTERACTIVE MODE
=====

=====
STEP 1: DEFINE JOB REQUIREMENTS
=====

Enter REQUIRED skills (must have):
Example: Python, SQL, Docker
Required Skills (comma-separated): python

Enter PREFERRED skills (nice to have):
Example: AWS, Git, Kubernetes
Preferred Skills (comma-separated): aws

Enter Minimum years of experience required:
Minimum Years: 2
=====

JOB REQUIREMENTS CONFIGURED

```

=====  
✓ Required Skills: python  
✓ Preferred Skills: aws  
✓ Minimum Experience: 2.0 years

=====  
APPLICANT #1  
=====

---

ENTER APPLICANT DETAILS

---

Applicant ID (e.g., APP-001): APP01

Education Level:

1. High School
  2. Bootcamp
  3. Bachelor's Degree
  4. Master's Degree
  5. PhD
- Select (1-5): 5

Years of relevant experience: 2

Enter technical skills (comma-separated):

Example: Python, JavaScript, SQL, Docker

Technical Skills: python, javascript

Enter certifications (comma-separated, or press Enter for none):

Example: AWS Solutions Architect, Google Cloud Professional

Certifications: AWS Solutions

Years at current role: 20

Years at current role cannot be negative or exceed total experience

Years at current role: 2

=====  
APPLICANT: APP01  
=====

SCORE BREAKDOWN:

Education:	100.0/100
Experience:	70.0/100
Role Stability:	75.0/100
Combined Experience:	71.5/100 (35% weight)
Technical Skills:	56.0/100 (35% weight)
Certifications:	15.0/100

---

FINAL SCORE: 66.1/100

---

REQUIREMENT CHECK:

Minimum Experience Met: ✓ Yes  
Required Skills Present: ✓ Yes

RECOMMENDATION: MAYBE

---

Add another applicant? (y/n): n

**Explanation:**

The program calculates a job applicant's score based on relevant skills, years of experience, and education level. Each factor is assigned predefined points, and the total score is computed without using personal attributes like gender or name, ensuring fairness and objectivity.

## Task 5: Inclusiveness and Ethical Variable Design

### Prompt:

Write a Python program to read consumer data from electricity\_data4.csv and generate the final electricity bill. The program should calculate Energy Charges using tiered rates, Fixed Charges by customer type, Customer Charges for service maintenance, and Electricity Duty as a percentage of Energy Charges. Display a clearly formatted output showing individual charge breakdowns and the total bill amount for each consumer, along with summary totals and customer-type-wise analysis.

### Code:

```
user_name = input("Please enter your name: ")
user_role = input("Please enter your role or occupation: ")
pronoun_preference = input("Please enter your preferred pronoun(s) (e.g., 'he/him',
'she/her', 'they/them'): ")

print(f"\nSummary of details:\nName: {user_name}\nRole: {user_role}\nPreferred
Pronoun(s): {pronoun_preference}")
```

### Output:

```
... Please enter your name: likitha
Please enter your role or occupation: Software Engineer
Please enter your preferred pronoun(s) (e.g., 'he/him', 'she/her', 'they/them'): she

Summary of details:
Name: likitha
Role: Software Engineer
Preferred Pronoun(s): she
```

#### ▼ Explain Inclusive Design

##### Subtask:

Explain the design choices made in the generated Python code to ensure inclusivity.

##### Explanation of Inclusive Design Choices

The Python code snippet was designed with inclusivity in mind, focusing on gender-neutral language and respectful interaction. Here's a breakdown of the design choices:

###### 1. Gender-Neutral Variable Names:

- Instead of using a variable name like `gender`, which often implies a binary choice and can be exclusionary, the code uses `pronoun_preference`. This choice acknowledges the diversity of gender identities and expressions, allowing individuals to specify their preferred pronouns without being forced into predefined categories.
- Other variable names like `user_name` and `user_role` are inherently neutral and focus on the individual's identity and professional capacity rather than any gender-specific attributes.

###### 2. Avoidance of Gender-Based Assumptions:

- The prompts for `user_name` and `user_role` are straightforward and do not contain any language that could be interpreted as gender-specific or stereotypical.
- Crucially, the code avoids any conditional logic (`if/else` statements) that would branch based on `pronoun_preference` or attempt to infer gender. This ensures that the application's behavior remains consistent and unbiased regardless of the user's input for pronouns.
- The output summary simply reflects the user's input directly, without any interpretation or modification based on perceived gender, thus preventing any misgendering or implicit bias in the system's response.

**3. Promotion of Fairness and Respect in Software Design:**

- By directly asking for `pronoun_preference` and displaying it as entered, the code respects user autonomy and self-identification. This practice signals to users that their identity is valued and acknowledged by the system.
- The design emphasizes a 'collect and reflect' approach to personal details, meaning it collects information and reflects it back without making assumptions or applying biases. This is a fundamental principle of inclusive design: allowing users to define themselves rather than having the system define them.
- This approach helps create a more welcoming and respectful user experience for everyone, contributing to a broader culture of inclusivity in software and digital interactions.

**Final Task**

**Subtask:**

Summarize the generated inclusive Python code and reiterate how it fulfills all aspects of the user's request regarding inclusive coding practices for user/employee detail processing.

**Explanation:**

The program asks the user to enter their role or occupation as input.

This information is used only for identification or display purposes and does not affect any decision-making logic, ensuring inclusiveness and fairness.