SR UNIVERSITY

# AI Assistant Coding
## Assignment-6.4

**Name of Student : V. Abhinav Batch : 41**
**Enrollment No. : 2303A52486**

## Task 1: Student Performance Evaluation System

**Create the skeleton of a Python class named Student with the attributes:**

- **name**
- **roll_number**
- **marks**

**Prompt:**

*Write a python code with class and attributes name, roll_number and marks. That displaying student details and checks whether the student's marks are above the class average and returns an appropriate message. And takes inputs from the user and asks the user to know which Student details or Student is above or below average.*

**Code:**

```
class Student:
    def __init__(self, name, roll_number, marks):
        self.name = name
        self.roll_number = roll_number
        self.marks = marks

    def display_details(self):
        print(f"Name        : {self.name}")
        print(f"Roll No.    : {self.roll_number}")
        print(f"Marks       : {self.marks}")

    def above_average_message(self, class_average):
        if self.marks > class_average:
            return f"{self.name} (Roll No. {self.roll_number}) is ABOVE the class average."
        elif self.marks < class_average:
            return f"{self.name} (Roll No. {self.roll_number}) is BELOW the class average."
        else:
            return f"{self.name} (Roll No. {self.roll_number}) is EXACTLY at the class average."


def main():
    students = []

    # Take number of students
    n = int(input("Enter the number of students: "))

    # Take input for each student
    for i in range(n):
        print(f"\nEnter details for student {i + 1}:")
        name = input("  Enter name: ")
        roll_number = input("  Enter roll number: ")
        marks = float(input("  Enter marks: "))

        students.append(Student(name, roll_number, marks))

    # Compute class average
    total_marks = sum(s.marks for s in students)
    class_average = total_marks / n if n > 0 else 0
    print(f"\nClass average marks: {class_average:.2f}")
```

```python
def main():
    # Compute class average
    total_marks = sum(s.marks for s in students)
    class_average = total_marks / n if n > 0 else 0
    print(f"\nClass average marks: {class_average:.2f}")

    while True:
        print("\nWhat would you like to do?")
        print("1. See a student's details and above/below average status")
        print("2. See all students who are above average")
        print("3. See all students who are below average")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            roll = input("Enter the roll number of the student: ")
            found = False
            for s in students:
                if s.roll_number == roll:
                    found = True
                    print("\nStudent details:")
                    s.display_details()
                    print(s.above_average_message(class_average))
                    break
            if not found:
                print("No student found with that roll number.")

        elif choice == "2":
            print("\nStudents ABOVE class average:")
            any_above = False
            for s in students:
                if s.marks > class_average:
                    any_above = True
                    print(f"- {s.name} (Roll No. {s.roll_number}, Marks: {s.marks})")
            if not any_above:
                print("No student is above the class average.")
```

```python
            if not any_above:
                print("No student is above the class average.")

        elif choice == "3":
            print("\nStudents BELOW class average:")
            any_below = False
            for s in students:
                if s.marks < class_average:
                    any_below = True
                    print(f"- {s.name} (Roll No. {s.roll_number}, Marks: {s.marks})")
            if not any_below:
                print("No student is below the class average.")

        elif choice == "4":
            print("Exiting program.")
            break
        else:
            print("Invalid choice. Please enter 1, 2, 3, or 4.")


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

**Output:**

```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/A
.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 6.py"
Enter the number of students: 3

Enter details for student 1:
  Enter name: Abhi
  Enter roll number: 101
  Enter marks: 85

Enter details for student 2:
  Enter name: Boby
  Enter roll number: 102a
  Enter marks: 74

Enter details for student 3:
  Enter name: Shashi
  Enter roll number: a103
  Enter marks: 90.5

Class average marks: 83.17

What would you like to do?
1. See a student's details and above/below average status
2. See all students who are above average
3. See all students who are below average
4. Exit
Enter your choice (1-4): 1
Enter the roll number of the student: 101

Student details:
Name        : Abhi
Roll No.    : 101
Marks       : 85.0
Abhi (Roll No. 101) is ABOVE the class average.
```

**Explanation:**

This task implements an object-oriented Python program using a `Student` class with attributes such as name, roll number, and marks. The code collects multiple student records from user input, computes the class average, and compares each student's marks against it. Menu-driven options allow users to view individual student details or list students above or below the average. The output confirms correct calculation of the class average and accurate classification of students based on their performance.

**Task 2: Data Processing in a Monitoring System**
Write the initial part of a for loop to iterate over a list of integers representing sensor readings.

Add a comment prompt instructing GitHub Copilot to:

- Identify even numbers
- Calculate their square
- Print the result in a readable format

Allow Copilot to complete the remaining loop logic.

**Prompt:**

*Write a python code which has a for loop to iterate over a list of integers representing sensor readings, also Identifies even numbers, Calculates their square and Print the result in a readable format. And takes inputs from the user and asks the user to know which to display.*

**Code:**

Assign 6.py > ...

```python
# Step 1: Take sensor readings from the user
n = int(input("Enter the number of sensor readings: "))

sensor_readings = []
for i in range(n):
    value = int(input(f"Enter reading {i + 1}: "))
    sensor_readings.append(value)

# Step 2: Main loop for user choices
while True:
    print("\nWhat would you like to display?")
    print("1. All sensor readings")
    print("2. Only even sensor readings")
    print("3. Even sensor readings and their squares")
    print("4. Exit")

    choice = input("Enter your choice (1-4): ")

    if choice == "1":
        print("\nAll sensor readings:")
        for reading in sensor_readings:
            print(reading, end=" ")
        print()

    elif choice == "2":
        print("\nEven sensor readings:")
        found = False
        for reading in sensor_readings:
            if reading % 2 == 0:
                print(reading, end=" ")
                found = True
        if not found:
            print("No even readings found.", end="")
        print()

    elif choice == "3":
        print("\nEven sensor readings and their squares:")
        found = False
        for reading in sensor_readings:
            if reading % 2 == 0:
                square = reading ** 2
                print(f"Reading: {reading} -> Square: {square}")
                found = True
        if not found:
            print("No even readings found.")

    elif choice == "4":
        print("Exiting program.")
        break

    else:
        print("Invalid choice. Please enter 1, 2, 3, or 4.")
```

Assign 5.py > ...

```python
def get_float(prompt, min_value=None, max_value=None):
    while True:
        try:
            val = float(input(prompt).strip())
            if min_value is not None and val < min_value:
                print(f"Value must be >= {min_value}")
                continue
            if max_value is not None and val > max_value:
                print(f"Value must be <= {max_value}")
                continue
            return val
        except ValueError:
            print("Enter a valid number.")


def get_int(prompt, min_value=None, max_value=None):
    while True:
        try:
            val = int(input(prompt).strip())
            if min_value is not None and val < min_value:
                print(f"Value must be >= {min_value}")
                continue
            if max_value is not None and val > max_value:
                print(f"Value must be <= {max_value}")
                continue
            return val
        except ValueError:
            print("Enter a valid integer.")


def get_yes_no(prompt):
    while True:
        v = input(prompt + " (y/n): ").strip().lower()
        if v in ("y", "yes"):
            return True
        if v in ("n", "no"):
            return False
        print("Please type y or n.")
```

Assign 5.py > ...

```python
def loan_decision(income_monthly, expenses_monthly, credit_score, loan_amount, loan_years,
    reasons = []
    if 550 <= credit_score < 650:
        reasons.append("Borderline credit score (550–649).")
    if 0.45 < dti <= 0.60:
        reasons.append("High DTI (45%–60%).")
    if loan_years > 5:
        reasons.append("Long tenure (> 5 years).")
    if total_repay_est > annual_income * 1.0:
        reasons.append("Estimated total repayment is high vs annual income.")

    if reasons:
        return "MANUAL_REVIEW", reasons

    # Approve
    return "APPROVE", ["Meets basic eligibility rules."]


def main():
    print("Loan Approval System")
    print("--------------------")

    income = get_float("Monthly income: ", min_value=1)
    expenses = get_float("Monthly expenses: ", min_value=0, max_value=income)
    credit = get_int("Credit score (300-850): ", min_value=300, max_value=850)
    amount = get_float("Loan amount requested: ", min_value=1)
    years = get_int("Loan tenure (years): ", min_value=1, max_value=30)
    employed = get_yes_no("Employed")

    decision, reasons = loan_decision(income, expenses, credit, amount, years, employed)

    dti = expenses / income
    print("\nResult")
    print("------")
    print(f"Decision: {decision}")
    print(f"DTI (expenses/income): {dti:.2%}")
    print("Reasons:")
    for r in reasons:
        print(f"- {r}")
```

**Output:**

```
PS C:\Users\Abhi\Documents\AI Assistant Coding> &
.exe "c:/Users/Abhi/Documents/AI Assistant Coding
Enter the number of sensor readings: 5
Enter reading 1: 10
Enter reading 2: 7
Enter reading 3: 3
Enter reading 4: 4
Enter reading 5: 9

What would you like to display?
1. All sensor readings
2. Only even sensor readings
3. Even sensor readings and their squares
4. Exit
Enter your choice (1-4): 1

All sensor readings:
10 7 3 4 9

What would you like to display?
1. All sensor readings
2. Only even sensor readings
3. Even sensor readings and their squares
4. Exit
Enter your choice (1-4): 3

Even sensor readings and their squares:
Reading: 10 -> Square: 100
Reading: 4 -> Square: 16
```

This task processes a list of sensor readings using a `for` loop to demonstrate conditional logic and iteration. The program identifies even sensor readings, calculates their squares, and prints the results in a readable format. A menu-driven interface allows users to choose whether to display all readings, only even readings, or even readings with their squares. The output verifies that even numbers are correctly detected and processed, reflecting proper loop and condition handling.

**Task 3: Banking Transaction Simulation**

**Create the structure of a Python class named BankAccount with attributes:**

- **account_holder**
- **balance**

**Use GitHub Copilot to complete methods for:**

- **Depositing money**
- **Withdrawing money**
- **Preventing withdrawals when the balance is insufficient**

**Prompt:**

*Create a python class named BankAccount with attributes account_holder, balance and asks the user to for Depositing money and Withdrawing money also Preventing withdrawals when the balance is insufficient.*

**Code:**

Assign 6.py > ...

```
148  # # Task-3
149  class BankAccount:
150      def __init__(self, account_holder, balance=0.0):
151          self.account_holder = account_holder
152          self.balance = balance
153
154      def deposit(self, amount):
155          if amount <= 0:
156              print("Deposit amount must be positive.")
157          else:
158              self.balance += amount
159              print(f"Deposited {amount:.2f}. New balance: {self.balance:.2f}")
160
161      def withdraw(self, amount):
162          if amount <= 0:
163              print("Withdrawal amount must be positive.")
164          elif amount > self.balance:
165              print("Insufficient balance. Withdrawal not allowed.")
166          else:
167              self.balance -= amount
168              print(f"Withdrew {amount:.2f}. New balance: {self.balance:.2f}")
169
170      def display_details(self):
171          print(f"Account Holder: {self.account_holder}")
172          print(f"Current Balance: {self.balance:.2f}")
173
174
175  def main():
176      name = input("Enter account holder name: ")
177      opening_balance = float(input("Enter opening balance: "))
178
179      account = BankAccount(name, opening_balance)
180
181      while True:
182          print("\nWhat would you like to do?")
183          print("1. Deposit money")
184          print("2. Withdraw money")
185          print("3. Show account details")
186          print("4. Exit")
```

```python
def main():
    name = input("Enter account holder name: ")
    opening_balance = float(input("Enter opening balance: "))

    account = BankAccount(name, opening_balance)

    while True:
        print("\nWhat would you like to do?")
        print("1. Deposit money")
        print("2. Withdraw money")
        print("3. Show account details")
        print("4. Exit")

        choice = input("Enter your choice (1-4): ")

        if choice == "1":
            amount = float(input("Enter amount to deposit: "))
            account.deposit(amount)

        elif choice == "2":
            amount = float(input("Enter amount to withdraw: "))
            account.withdraw(amount)

        elif choice == "3":
            account.display_details()

        elif choice == "4":
            print("Exiting. Thank you!")
            break

        else:
            print("Invalid choice. Please enter 1, 2, 3, or 4.")


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

**Output:**

```
PS C:\Users\Abhi\Documents\AI Assistant Codi
.exe "c:/Users/Abhi/Documents/AI Assistant C
Enter account holder name: Abhi
Enter opening balance: 80000

What would you like to do?
1. Deposit money
2. Withdraw money
3. Show account details
4. Exit
Enter your choice (1-4): 1
Enter amount to deposit: 20000
Deposited 20000.00. New balance: 100000.00

What would you like to do?
1. Deposit money
2. Withdraw money
3. Show account details
4. Exit
Enter your choice (1-4): 3
Account Holder: Abhi
Current Balance: 100000.00
```

**Explanation:**

This task simulates basic banking operations using a `BankAccount` class with attributes for account holder and balance. The code includes methods for depositing money, withdrawing money, and preventing withdrawals when the balance is insufficient. User interaction is handled through a loop-based menu that allows repeated transactions. The output demonstrates successful deposits, secure withdrawal checks, and accurate balance updates, ensuring realistic transaction behavior.

**Task 4: Ethical Evaluation of AI-Based Scoring Systems Scenario**

**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**

**Prompt:**
Generate a job applicant scoring system in python based on features such as:
• Skills
• Experience
• Education etc.

**Code:**

Assign 5.py > ...

```python
# Task-4: Job Applicant Scoring System

from dataclasses import dataclass
from typing import Any, List
from enum import Enum


class EducationLevel(Enum):
    HIGH_SCHOOL = 1
    ASSOCIATE = 2
    BACHELOR = 3
    MASTER = 4
    PHD = 5


@dataclass
class Applicant:
    name: str
    skills: List[str]
    experience_years: float
    education_level: EducationLevel
    certifications: List[str]
    previous_roles: List[str]
    gpa: float = 0.0


class JobApplicantScoringSystem:
    """
    Scoring system for job applicants based on:
    - Skills match
    - Experience
    - Education
    - Certifications
    - Previous roles
    """

    def __init__(self, job_requirements: dict[str, Any]):
        """
        Initialize with job requirements
```

Assign 5.py > ...

```python
class JobApplicantScoringSystem:
    def score_skills(self, applicant: Applicant) -> float:
        required = set(self.job_requirements.get('required_skills', []))
        preferred = set(self.job_requirements.get('preferred_skills', []))
        applicant_skills = set(s.lower() for s in applicant.skills)

        if not required:
            return 100.0

        # Required skills: must-haves
        required_match = len(required & applicant_skills) / len(required)

        # Preferred skills: bonus
        if preferred:
            preferred_match = len(preferred & applicant_skills) / len(preferred)
        else:
            preferred_match = 0

        score = (required_match + preferred_match) * 100
        return min(100.0, score)

    def score_experience(self, applicant: Applicant) -> float:
        """Score based on years of experience"""
        min_experience = self.job_requirements.get('min_experience', 0)
        max_expected = self.job_requirements.get('max_experience', 15)

        if applicant.experience_years < min_experience:
            return 0.0

        if applicant.experience_years >= max_expected:
            return 100.0

        # Linear scale between min and max
        score = ((applicant.experience_years - min_experience) /
                 (max_expected - min_experience)) * 100
        return min(100.0, score)

    def score_education(self, applicant: Applicant) -> float:
        """Score based on education level"""
```

```python
class JobApplicantScoringSystem:
    def get_recommendation(self, applicant: Applicant) -> str:
        elif overall >= 70:
            return "ACCEPT - Good candidate"
        elif overall >= 55:
            return "MAYBE - Consider for interview"
        elif overall >= 40:
            return "WEAK - Not recommended"
        else:
            return "REJECT - Does not meet requirements"

    def print_detailed_report(self, applicant: Applicant) -> None:
        """Print detailed scoring report"""
        scores = self.calculate_overall_score(applicant)

        print("\n" + "="*60)
        print(f"APPLICANT SCORING REPORT: {applicant.name}")
        print("="*60)

        print(f"\nBackground:")
        print(f"  Skills: {', '.join(applicant.skills)}")
        print(f"  Experience: {applicant.experience_years} years")
        print(f"  Education: {applicant.education_level.name}")
        print(f"  Certifications: {', '.join(applicant.certifications) if applic
        print(f"  GPA: {applicant.gpa}")
        print(f"  Previous Roles: {', '.join(applicant.previous_roles)}")

        print(f"\nScores:")
        print(f"  Skills Match: {scores['skills']:.1f}/100")
        print(f"  Experience: {scores['experience']:.1f}/100")
        print(f"  Education: {scores['education']:.1f}/100")
        print(f"  Certifications: {scores['certifications']:.1f}/100")
        print(f"  GPA: {scores['gpa']:.1f}/100")

        print(f"\n{'Overall Score:':<25} {scores['overall']:.1f}/100")
        print(f"{'Recommendation:':<25} {self.get_recommendation(applicant)}")
        print("="*60 + "\n")
```

```python
# ========== INTERACTIVE DEMO ==========

def get_education_level() -> EducationLevel:
    """Get education level from user input"""
    print("\nEducation Levels:")
    for level in EducationLevel:
        print(f"  {level.value}) {level.name}")

    while True:
        try:
            choice = int(input("Select education level (1-5): "))
            return EducationLevel(choice)
        except (ValueError, KeyError):
            print("Invalid choice. Enter a number 1-5.")


def input_list(prompt: str) -> List[str]:
    """Get comma-separated input from user"""
    response = input(prompt).strip()
    if not response:
        return []
    return [item.strip() for item in response.split(',')]


def demo():
    """Interactive demo of the scoring system"""
    print("\n" + "="*60)
    print("JOB APPLICANT SCORING SYSTEM")
    print("="*60)

    # Define job requirements
    job_requirements = {
        'required_skills': ['python', 'sql', 'git'],
        'preferred_skills': ['aws', 'docker', 'kubernetes'],
        'min_experience': 2,
        'max_experience': 15,
        'min_education': EducationLevel.BACHELOR,
        'preferred_certifications': ['aws certified', 'docker certified'],
```

1. No bias-prone data - The scoring system only uses:
    - Skills (job-relevant)
    - Experience (measurable years)
    - Education level (verifiable credential)
    - Certifications (verifiable credential)
    - GPA (measurable)
    -
2. Name is NOT used in scoring - The name field is only used for reporting/display. It has zero impact on the score calculation.

3. No demographic data - There's no gender, age, race, ethnicity, or any protected characteristics in the scoring logic.
4. Transparent, deterministic logic - Each scoring method is objective and reproducible:

- Skills: percentage match against requirements
- Experience: linear scale between min/max
- Education: level-based scoring
- Certifications: percentage match
- GPA: linear scale

```
580  def run_examples():
607              experience_years=2,
608              education_level=EducationLevel.BACHELOR,
609              certifications=[],
610              gpa=3.2,
611              previous_roles=["Junior Developer"]
612          ),
613          Applicant(
614              name="Carol Davis",
615              skills=["Python", "SQL", "Git"],
616              experience_years=8,
617              education_level=EducationLevel.MASTER,
618              certifications=["AWS Certified"],
619              gpa=3.6,
620              previous_roles=["Senior Developer", "Tech Lead", "Architect"]
621          ),
622      ]
623
624      print("\n" + "="*60)
625      print("JOB APPLICANT SCORING SYSTEM - EXAMPLES")
626      print("="*60)
627
628      for applicant in applicants:
629          scorer.print_detailed_report(applicant)
630
631
632  if __name__ == "__main__":
633      print("\nChoose mode:")
634      print("1) Interactive mode (enter applicant details)")
635      print("2) View example applicants")
636
637      choice = input("> ").strip()
638
639      if choice == "1":
640          demo()
641      elif choice == "2":
642          run_examples()
643      else:
644          print("Invalid choice.")
```

**Output:**

```
--- Enter Applicant Information ---
Applicant name: Abhinav
Skills (comma-separated): Python, Java, AIML, Git
Years of experience: 2

Education Levels:
  1) HIGH_SCHOOL
  2) ASSOCIATE
  3) BACHELOR
  4) MASTER
  5) PHD
Select education level (1-5): 3
Certifications (comma-separated): AIML Specialization-IIITH, AWS Certified
GPA (0.0-4.0): 3.6
Previous roles (comma-separated): Computer Vision Intern

============================================================
APPLICANT SCORING REPORT: Abhinav
============================================================

Background:
  Skills: Python, Java, AIML, Git
  Experience: 2.0 years
  Education: BACHELOR
  Certifications: AIML Specialization-IIITH, AWS Certified
  GPA: 3.6
  Previous Roles: Computer Vision Intern

Scores:
  Skills Match: 66.7/100
  Experience: 0.0/100
  Education: 20.0/100
  Certifications: 50.0/100
  GPA: 88.9/100

Overall Score:            40.4/100
Recommendation:           WEAK - Not recommended
============================================================
```

**Explanation:**

This task evaluates the fairness of an AI-generated job applicant scoring system. The scoring logic is based solely on job-relevant features such as skills, experience, education level, certifications, and GPA. No demographic or sensitive attributes like gender, age, or name are used in decision-making, ensuring unbiased evaluation. The output presents a transparent score breakdown and recommendation, confirming ethical and objective system behavior.

SR UNIVERSITY

**Task 5: Inclusiveness and Ethical Variable Design Scenario**

**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**

**Prompt:**

*Generate a Python code that processes user or employee details, take user input.*

**Code:**

Assign 5.py > ...

```
649     from dataclasses import dataclass, field
650     from typing import List, Optional
651     from datetime import datetime
652     import json
653
654
655     @dataclass
656     class Employee:
657         """Employee data model"""
658         employee_id: str
659         first_name: str
660         last_name: str
661         email: str
662         department: str
663         position: str
664         salary: float
665         hire_date: str
666         phone: str
667         address: str
668         skills: List[str] = field(default_factory=list)
669         projects: List[str] = field(default_factory=list)
670         manager: Optional[str] = None
671
672         @property
673         def full_name(self) -> str:
674             return f"{self.first_name} {self.last_name}"
675
676         def to_dict(self) -> dict:
677             """Convert employee to dictionary"""
678             return {
679                 'employee_id': self.employee_id,
680                 'first_name': self.first_name,
681                 'last_name': self.last_name,
682                 'email': self.email,
683                 'department': self.department,
684                 'position': self.position,
685                 'salary': self.salary,
686                 'hire_date': self.hire_date,
687                 'phone': self.phone,
```

Assign 5.py > ...

```python
class EmployeeManagementSystem:

    def __init__(self):
        self.employees: dict[str, Employee] = {}

    def add_employee(self, employee: Employee) -> tuple[bool, str]:
        """Add new employee to system"""
        if employee.employee_id in self.employees:
            return False, f"Employee ID {employee.employee_id} already exists."

        self.employees[employee.employee_id] = employee
        return True, f"Employee {employee.full_name} added successfully."

    def get_employee(self, employee_id: str) -> Optional[Employee]:
        """Get employee by ID"""
        return self.employees.get(employee_id)

    def update_employee(self, employee_id: str, **kwargs) -> tuple[bool, str]:
        """Update employee details"""
        if employee_id not in self.employees:
            return False, f"Employee ID {employee_id} not found."

        employee = self.employees[employee_id]
        for key, value in kwargs.items():
            if hasattr(employee, key):
                setattr(employee, key, value)

        return True, f"Employee {employee.full_name} updated successfully."

    def delete_employee(self, employee_id: str) -> tuple[bool, str]:
        """Delete employee from system"""
        if employee_id not in self.employees:
            return False, f"Employee ID {employee_id} not found."

        employee = self.employees.pop(employee_id)
        return True, f"Employee {employee.full_name} deleted

    def search_by_department(self, department: str) -> List[Employee]:
        """Search employees by department"""
```

(class) Employee

Employee data model

```python
            summary[emp.department] = summary.get(emp.department, 0) + 1
        return summary

    def save_to_file(self, filename: str) -> None:
        """Save employee data to JSON file"""
        data = {emp_id: emp.to_dict() for emp_id, emp in self.employees.items()}
        with open(filename, 'w', encoding='utf-8') as f:
            json.dump(data, f, indent=2)

    def load_from_file(self, filename: str) -> tuple[bool, str]:
        """Load employee data from JSON file"""
        try:
            with open(filename, 'r', encoding='utf-8') as f:
                data = json.load(f)

            self.employees = {emp_id: Employee.from_dict(emp_data)
                              for emp_id, emp_data in data.items()}
            return True, f"Loaded {len(self.employees)} employees from {filename}"
        except FileNotFoundError:
            return False, f"File {filename} not found."
        except Exception as e:
            return False, f"Error loading file: {str(e)}"


# ========== INPUT HELPERS ==========

def get_valid_input(prompt: str, validator=None, error_msg: str = "Invalid input."):
    """Get validated input from user"""
    while True:
        value = input(prompt).strip()
        if not value:
            print("Input cannot be empty.")
            continue
        if validator is None or validator(value):
            return value
        print(error_msg)
```

```python
def get_float_input(prompt: str, min_value: float = None) -> float:
    """Get float input with validation"""
    while True:
        try:
            value = float(input(prompt).strip())
            if min_value is not None and value < min_value:
                print(f"Value must be >= {min_value}")
                continue
            return value
        except ValueError:
            print("Enter a valid number.")


def get_list_input(prompt: str) -> List[str]:
    """Get comma-separated list input"""
    value = input(prompt).strip()
    if not value:
        return []
    return [item.strip() for item in value.split(',')]


def validate_email(email: str) -> bool:
    """Basic email validation"""
    return '@' in email and '.' in email.split('@')[1]


def validate_phone(phone: str) -> bool:
    """Basic phone validation"""
    digits = ''.join(c for c in phone if c.isdigit())
    return len(digits) >= 10


# ========== USER INTERFACE ==========

def display_employee(employee: Employee) -> None:
    """Display employee details in formatted way"""
    print("\n" + "="*70)
    print(f"EMPLOYEE DETAILS: {employee.full_name}")
    print("-"*70)
```

Assign 5.py > ...

```
1054    def reports_menu(system: EmployeeManagementSystem) -> None:
1077            print("DEPARTMENT SUMMARY")
1078            print("="*50)
1079            for dept, count in sorted(summary.items()):
1080                print(f"  {dept}: {count} employee(s)")
1081            print("="*50 + "\n")
1082
1083        else:
1084            print("Invalid choice.")
1085
1086
1087    def main_menu():
1088        """Main program menu"""
1089        system = EmployeeManagementSystem()
1090
1091        print("\n" + "="*70)
1092        print("EMPLOYEE MANAGEMENT SYSTEM")
1093        print("="*70)
1094
1095        while True:
1096            print("\nMain Menu:")
1097            print("1)  Add Employee")
1098            print("2)  View Employee")
1099            print("3)  Update Employee")
1100            print("4)  Delete Employee")
1101            print("5)  List All Employees")
1102            print("6)  Search Employees")
1103            print("7)  Generate Reports")
1104            print("8)  Save to File")
1105            print("9)  Load from File")
1106            print("10) Exit")
1107
1108            choice = input("\nSelect option: ").strip()
1109
1110            if choice == "1":
1111                add_employee_menu(system)
1112            elif choice == "2":
1113                view_employee_menu(system)
```

```
1087    def main_menu():
1100            print("4)  Delete Employee")
1101            print("5)  List All Employees")
1102            print("6)  Search Employees")
1103            print("7)  Generate Reports")
1104            print("8)  Save to File")
1105            print("9)  Load from File")
1106            print("10) Exit")
1107
1108            choice = input("\nSelect option: ").strip()
1109
1110            if choice == "1":
1111                add_employee_menu(system)
1112            elif choice == "2":
1113                view_employee_menu(system)
1114            elif choice == "3":
1115                update_employee_menu(system)
1116            elif choice == "4":
1117                delete_employee_menu(system)
1118            elif choice == "5":
1119                list_all_employees_menu(system)
1120            elif choice == "6":
1121                search_menu(system)
1122            elif choice == "7":
1123                reports_menu(system)
1124            elif choice == "8":
1125                filename = input("Enter filename: ").strip()
1126                system.save_to_file(filename)
1127                print(f"✓ Data saved to {filename}")
1128            elif choice == "9":
1129                filename = input("Enter filename: ").strip()
1130                success, message = system.load_from_file(filename)
1131                print(f"{'✓' if success else 'X'} {message}")
1132            elif choice == "10":
1133                print("Goodbye!")
1134                break
1135            else:
1136                print("Invalid option. Please try again.")
1137
```

**Output:**

```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Loca
on.exe "c:/Users/Abhi/Documents/AI Assistant Coding/Assign 5.py"

======================================================================
EMPLOYEE MANAGEMENT SYSTEM
======================================================================

Main Menu:
1)  Add Employee
2)  View Employee
3)  Update Employee
4)  Delete Employee
5)  List All Employees
6)  Search Employees
7)  Generate Reports
8)  Save to File
9)  Load from File
10) Exit

Select option: 1

--- Enter Employee Details ---
Employee ID: EMP001
First Name: Sarah
Last Name: Sharma
Email: sarah.sharma@someone.com
Phone: 989384933
Invalid phone number (need at least 10 digits).
Phone: 9893849331
Department: Engineering
Position: Senior Software Engineer
Salary: $40000
Hire Date (YYYY-MM-DD): 2023-05-15
Address: 123 Main St, Seattle, WA 98101
Manager Name (optional):
Skills (comma-separated): Python, JavaScript, Docker, AWS
Projects (comma-separated): Cloud Migration, API Redesign, Mobile App

✓ Employee Sarah Sharma added successfully.
```

**Explanation:**

This task analyzes and improves AI-generated employee-processing code to ensure inclusiveness and ethical design. Gender-specific variables and assumptions are removed and replaced with gender-neutral naming conventions. The revised code focuses only on relevant employee attributes and includes proper input validation for robustness. The output shows successful employee data handling while adhering to inclusive and respectful coding practices.