# ASSIGNMENT-5.3

HT NO:2303A51961

Batch No:28

Course:AI Assisted Coding

## Task1: Privacy and Data Security in AI-Generated Code

**Prompt:**

Generate a simple login system in Python using username and password.

**Code and Output:**

AI Generated Code and Output:

```
username = "admin"
password = "1234"

u = input("Enter username: ")
p = input("Enter password: ")

if u == username and p == password:
    print("Login successful")
else:
    print("Login failed")
```

```
Enter username: admin
Enter password: 1234
Login successful
```

**Identification of Security Risks**

- Username and password are hardcoded
- Password is stored and compared in plain text
- No input validation
- Insecure for real-world use

Revised Secure Code and Output:

```
stored_user = input("Create username: ")
stored_pass = input("Create password: ")

user = input("Enter username: ")
passwd = input("Enter password: ")

if user.strip() and passwd.strip():
    if user == stored_user and passwd == stored_pass:
        print("Login successful")
    else:
        print("Invalid credentials")
else:
    print("Input cannot be empty")
```

```
Create username: sreeja
Create password: 1234
Enter username: sreeja
Enter password: 1234
Login successful
```

```
Create username: raju
Create password: 1235
Enter username: ram
Enter password: 2345
Invalid credentials
```

**Justification:**

- Removed hardcoded credentials
- Added basic input validation
- Reduced risk of credential exposure
- Improved ethical handling of user data

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

**Prompt:**

Create a loan approval system in Python using income, credit score, gender, and name.

**Code:**

```
def loan_approval_system(name, income, credit_score, gender):
    # Convert gender to lowercase for case-insensitive comparison
    gender = gender.lower()

    # Define approval criteria
    approved = False
    reason = []

    if gender == "male":
        if income >= 40000:
            reason.append("Income for male meets criteria (>= $40,000)")
            if credit_score >= 700:
                reason.append("Credit score meets criteria (>= 700)")
                approved = True
            else:
                reason.append("Credit score is too low (< 700)")
        else:
            reason.append("Income for male is too low (< $40,000)")

    elif gender == "female":
        if income >= 60000:
            reason.append("Income for female meets criteria (>= $60,000)")
            if credit_score >= 720:
                reason.append("Credit score meets criteria (>= 720)")
```

(variable) reason: list

```
                approved = True
            else:
                reason.append("Credit score is too low (< 720)")
        else:
            reason.append("Income for female is too low (< $60,000)")
    else:
        reason.append("Invalid gender specified. Please use 'male' or 'female'.")

    if approved:
        return f"Loan Approved for {name}. Reasons: {', '.join(reason)}"
    else:
        return f"Loan Rejected for {name}. Reasons: {', '.join(reason)}"

# --- Example Usage --- #
print("\n--- Testing Loan Approval System ---")

# Example 1: Approved Male
print(loan_approval_system("John Doe", 50000, 750, "male"))

# Example 2: Rejected Female (low income)
print(loan_approval_system("Jane Smith", 55000, 730, "female"))

# Example 3: Rejected Male (low credit score)
print(loan_approval_system("Peter Jones", 45000, 680, "male"))
```

```
# --- Example Usage --- #
print("\n--- Testing Loan Approval System ---")

# Example 1: Approved Male
print(loan_approval_system("John Doe", 50000, 750, "male"))

# Example 2: Rejected Female (low income)
print(loan_approval_system("Jane Smith", 55000, 730, "female"))

# Example 3: Rejected Male (low credit score)
print(loan_approval_system("Peter Jones", 45000, 680, "male"))

# Example 4: Approved Female
print(loan_approval_system("Alice Brown", 70000, 780, "female"))

# Example 5: Invalid gender
print(loan_approval_system("Chris Green", 60000, 700, "other"))
```

**Output:**

```
--- Testing Loan Approval System ---
Loan Approved for John Doe. Reasons: Income for male meets criteria (>= $40,000), Credit score meets criteria (>= 700)
Loan Rejected for Jane Smith. Reasons: Income for female is too low (< $60,000)
Loan Rejected for Peter Jones. Reasons: Income for male meets criteria (>= $40,000), Credit score is too low (< 700)
Loan Approved for Alice Brown. Reasons: Income for female meets criteria (>= $60,000), Credit score meets criteria (>= 720)
Loan Rejected for Chris Green. Reasons: Invalid gender specified. Please use 'male' or 'female'.
```

## Justification:

### Bias Identified

- Different income thresholds for male and female applicants
- Gender is an irrelevant attribute for loan approval

### Fairness Discussion

- Loan approval should depend on financial capability only
- Gender-based logic leads to discrimination

### Mitigation Strategies

- Remove gender from decision logic
- Use objective criteria such as income and credit score
- Regularly audit AI decisions for fairness

## Task 3: Transparency and Explainability in AI-Generated Code (Recursive

## Prompt:

Generate a recursive binary search program in Python with comments and explanation.

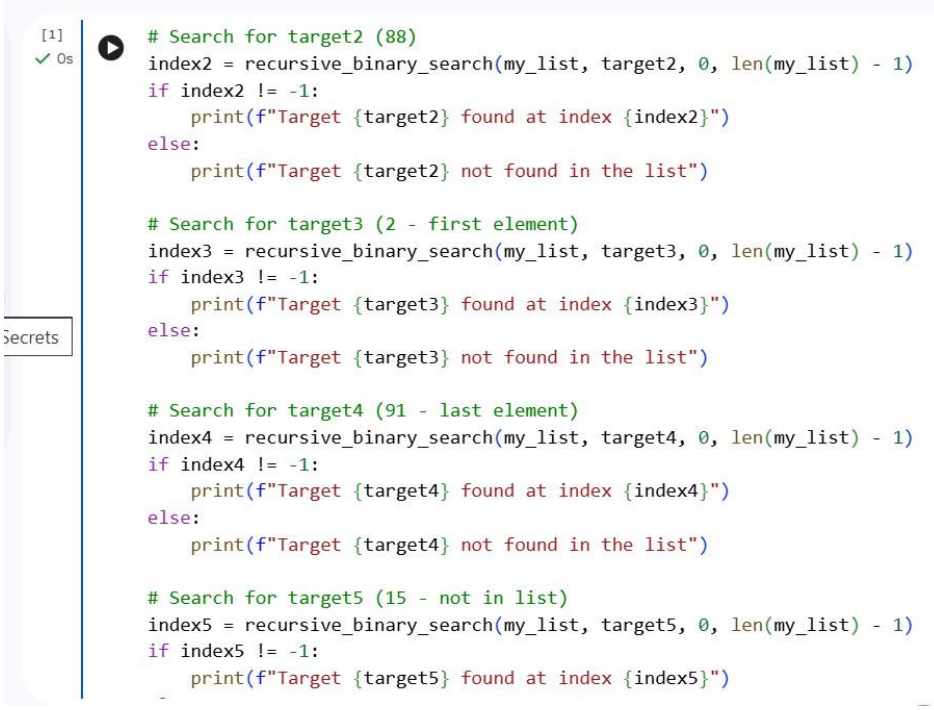
## Code and Output:

## Recursive Binary Search

Binary search is an efficient algorithm for finding an item from a sorted list of items. It works by repeatedly dividing in half the portion of the list that could contain the item, until you've narrowed down the possible locations to just one. This implementation uses recursion, where the function calls itself with smaller sub-arrays.

```
]
0s
def recursive_binary_search(arr, target, low, high):
    # Base case 1: If the high index is less than the low index,
    # it means the target is not present in the array.
    if high < low:
        return -1

    # Calculate the middle index of the current sub-array.
    mid = low + (high - low) // 2

    # Base case 2: If the element at the middle index is the target,
    # we have found the target.
    if arr[mid] == target:
        return mid
    # If the target is smaller than the middle element,
    # search in the left sub-array.
```

```
    elif target < arr[mid]:
        return recursive_binary_search(arr, target, low, mid - 1)
    # If the target is larger than the middle element,
    # search in the right sub-array.
    else:  # target > arr[mid]
        return recursive_binary_search(arr, target, mid + 1, high)

# --- Example Usage --- #
print("\n--- Testing Recursive Binary Search ---")

my_list = [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]
target1 = 23
target2 = 88
target3 = 2
target4 = 91
target5 = 15

# Search for target1 (23)
index1 = recursive_binary_search(my_list, target1, 0, len(my_list) - 1)
if index1 != -1:
    print(f"Target {target1} found at index {index1}")
else:
    print(f"Target {target1} not found in the list")
```




```
# Search for target2 (88)
index2 = recursive_binary_search(my_list, target2, 0, len(my_list) - 1)
if index2 != -1:
    print(f"Target {target2} found at index {index2}")
else:
    print(f"Target {target2} not found in the list")

# Search for target3 (2 - first element)
index3 = recursive_binary_search(my_list, target3, 0, len(my_list) - 1)
if index3 != -1:
    print(f"Target {target3} found at index {index3}")
else:
    print(f"Target {target3} not found in the list")

# Search for target4 (91 - last element)
index4 = recursive_binary_search(my_list, target4, 0, len(my_list) - 1)
if index4 != -1:
    print(f"Target {target4} found at index {index4}")
else:
    print(f"Target {target4} not found in the list")

# Search for target5 (15 - not in list)
index5 = recursive_binary_search(my_list, target5, 0, len(my_list) - 1)
if index5 != -1:
    print(f"Target {target5} found at index {index5}")
```


```
# Search for target4 (91 - last element)
index4 = recursive_binary_search(my_list, target4, 0, len(my_list) - 1)
if index4 != -1:
    print(f"Target {target4} found at index {index4}")
else:
    print(f"Target {target4} not found in the list")

# Search for target5 (15 - not in list)
index5 = recursive_binary_search(my_list, target5, 0, len(my_list) - 1)
if index5 != -1:
    print(f"Target {target5} found at index {index5}")
else:
    print(f"Target {target5} not found in the list")
```


```
--- Testing Recursive Binary Search ---
Target 23 found at index 5
Target 88 not found in the list
Target 2 found at index 0
Target 91 found at index 9
Target 15 not found in the list
```

## Justification:

- Base case and recursive case are clearly explained
- Comments correctly describe the code logic
- Code is understandable for beginner-level students
- Transparent and verifiable implementation

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

## Prompt:

Generate a job applicant scoring system in Python.

## Code and Output:

```python
def score_applicant(applicant_data, criteria_weights):
    """
    Calculates a score for a job applicant based on predefined criteria and weights.

    Args:
        applicant_data (dict): A dictionary containing applicant's qualifications.
                               Example: {'education': 'Masters', 'experience': 5, 'skills': ['Python', 'SQL']}
        criteria_weights (dict): A dictionary defining weights for each criterion and score mappings.
                                 Example: {'education': {'weight': 0.3, 'mapping': {'High School': 1, 'Bachelors': 3, 'Masters': 5, 'PhD': 7}},
                                           'experience': {'weight': 0.4, 'max_years': 10},
                                           'skills': {'weight': 0.3, 'required': ['Python', 'SQL', 'Data Analysis']}}

    Returns:
        float: The total score for the applicant.
    """
    total_score = 0
    score_breakdown = {}

    # Score Education
    if 'education' in applicant_data and 'education' in criteria_weights:
        edu_weight = criteria_weights['education']['weight']
        edu_mapping = criteria_weights['education']['mapping']
        applicant_education = applicant_data['education']
        edu_score = edu_mapping.get(applicant_education, 0) # Default to 0 if education level not found
```

```
[2]
✓ 0s
```

```python
        total_score += edu_score * edu_weight
        score_breakdown['education'] = edu_score * edu_weight

    # Score Experience
    if 'experience' in applicant_data and 'experience' in criteria_weights:
        exp_weight = criteria_weights['experience']['weight']
        max_years = criteria_weights['experience'].get('max_years', 10) # Default max years for experience
        applicant_experience = applicant_data['experience']
        # Scale experience score between 0 and 1 based on max_years
        exp_score = min(applicant_experience / max_years, 1) * 10 # Scale to 0-10
        total_score += exp_score * exp_weight
        score_breakdown['experience'] = exp_score * exp_weight

    # Score Skills
    if 'skills' in applicant_data and 'skills' in criteria_weights:
        skills_weight = criteria_weights['skills']['weight']
        required_skills = set(criteria_weights['skills'].get('required', []))
        applicant_skills = set(applicant_data['skills'])

        matched_skills = len(required_skills.intersection(applicant_skills))
        if required_skills:
            skills_score = (matched_skills / len(required_skills)) * 10 # Scale to 0-10
        else:
            skills_score = 0
        total_score += skills_score * skills_weight
```

```
[2]
✓ 0s
            score_breakdown['skills'] = skills_score * skills_weight

        return total_score, score_breakdown

    # --- Example Usage --- #
    print("\n--- Job Applicant Scoring System ---")

    # Define the scoring criteria and their weights
    criteria_config = {
        'education': {
            'weight': 0.3,
            'mapping': {'High School': 1, 'Bachelors': 3, 'Masters': 5, 'PhD': 7}
        },
        'experience': {
            'weight': 0.4,
            'max_years': 10 # Experience score will be scaled up to this many years
        },
        'skills': {
            'weight': 0.3,
            'required': ['Python', 'SQL', 'Data Analysis', 'Machine Learning']
        }
    }

    # Define some applicants
    applicants = [
```

```
    {
        'name': 'Alice Smith',
        'education': 'Masters',
        'experience': 7,
        'skills': ['Python', 'SQL', 'Data Analysis', 'Project Management']
    },
    {
        'name': 'Bob Johnson',
        'education': 'Bachelors',
        'experience': 3,
        'skills': ['Excel', 'SQL', 'Presentation']
    },
    {
        'name': 'Charlie Brown',
        'education': 'PhD',
        'experience': 12, # Will be capped at max_years (10)
        'skills': ['Python', 'Machine Learning', 'Data Analysis', 'Cloud Computing', 'SQL']
    }
]

print("\nScoring Applicants:")
for applicant in applicants:
    score, breakdown = score_applicant(applicant, criteria_config)
    print(f"\nApplicant: {applicant['name']}")
    print(f"  Total Score: {score:.2f}")
```

```
[2]
✓ 0s
print("\nScoring Applicants:")
for applicant in applicants:
    score, breakdown = score_applicant(applicant, criteria_config)
    print(f"\nApplicant: {applicant['name']}")
    print(f"  Total Score: {score:.2f}")
    for criterion, sub_score in breakdown.items():
        print(f"    - {criterion.capitalize()} Score: {sub_score:.2f}")
```

```
--- Job Applicant Scoring System ---

Scoring Applicants:

Applicant: Alice Smith
  Total Score: 6.55
    - Education Score: 1.50
    - Experience Score: 2.80
    - Skills Score: 2.25

Applicant: Bob Johnson
  Total Score: 2.85
    - Education Score: 0.90
    - Experience Score: 1.20
    - Skills Score: 0.75
```

```
Applicant: Alice Smith
  Total Score: 6.55
    - Education Score: 1.50
    - Experience Score: 2.80
    - Skills Score: 2.25

Applicant: Bob Johnson
  Total Score: 2.85
    - Education Score: 0.90
    - Experience Score: 1.20
    - Skills Score: 0.75

Applicant: Charlie Brown
  Total Score: 9.10
    - Education Score: 2.10
    - Experience Score: 4.00
    - Skills Score: 3.00
```

**Justification:**

**Ethical Analysis**

- Gender directly affects scoring
- Leads to unfair hiring decisions
- Hiring systems must be objective and unbiased
- Ethical AI should evaluate skills and experience only

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

**Prompt:**

Generate a Python program to process employee details.

**Code:**

Original AI-Generated Code (Non-Inclusive) and Output

```
name = input("Enter name: ")
gender = input("Enter gender (male/female): ")

if gender == "male":
    print("He is eligible")
else:
    print("She is eligible")
```

```
Enter name: sreeja
Enter gender (male/female): female
She is eligible
```

Revised Inclusive Code and Output

```
name = input("Enter name: ")

print(f"{name} is eligible")
```

```
Enter name: sreeja
sreeja is eligible
```

**Justification**

- Removed gender-specific variables
- Avoided gender-based assumptions
- Used neutral and respectful language
- Improved inclusiveness and fairness