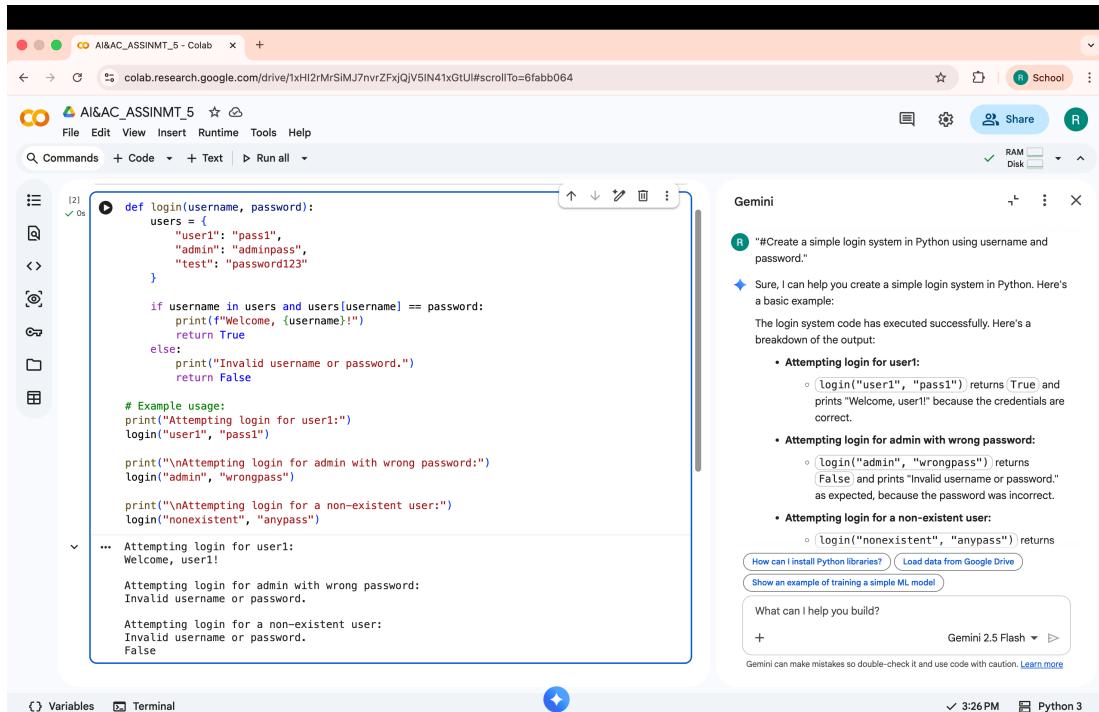


AI ASSISTED CODING - ASSIGNMENT -5

NAME : R.RONITH REDDY
HT.NO : 2303A52280

TASK- 1 : PRIVACY AND DATA SECURITY IN AI-GENERATED CODE SCENARIO

PROMPT :
CREATE A SIMPLE LOGIN SYSTEM IN PYTHON USING USERNAME AND PASSWORD.



```
def login(username, password):
    users = {
        "user1": "pass1",
        "admin": "adminpass",
        "test": "password123"
    }

    if username in users and users[username] == password:
        print("Welcome, " + username + "!")
        return True
    else:
        print("Invalid username or password.")
        return False

# Example usage:
login("Attempting login for user1:")
login("user1", "pass1")

print("\nAttempting login for admin with wrong password:")
login("admin", "wrongpass")

print("\nAttempting login for a non-existent user:")
login("nonexistent", "anypass")

... Attempting login for user1:
Welcome, user1!

Attempting login for admin with wrong password:
Invalid username or password.

Attempting login for a non-existent user:
Invalid username or password.
False
```

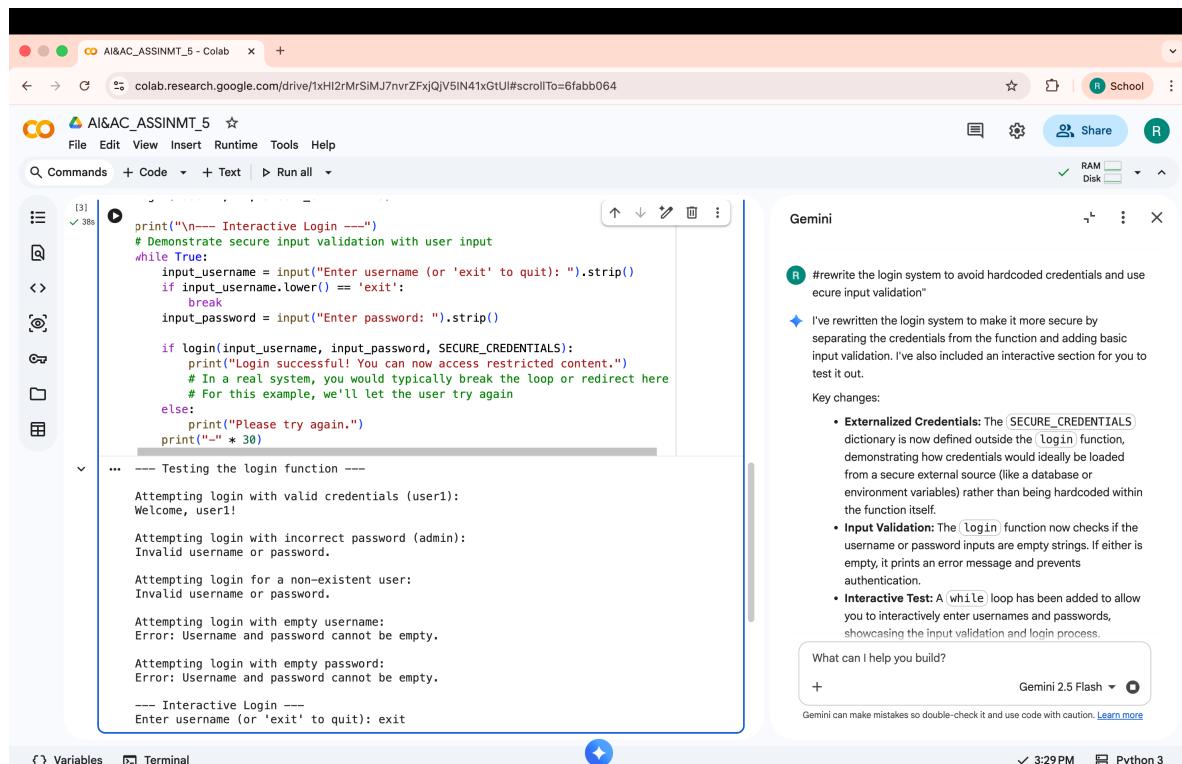
Gemini

- #Create a simple login system in Python using username and password."
- Sure, I can help you create a simple login system in Python. Here's a basic example:

The login system code has executed successfully. Here's a breakdown of the output:

- Attempting login for user1:
 - login("user1", "pass1") returns True and prints "Welcome, user1!" because the credentials are correct.
- Attempting login for admin with wrong password:
 - login("admin", "wrongpass") returns False and prints "Invalid username or password." as expected, because the password was incorrect.
- Attempting login for a non-existent user:
 - login("nonexistent", "anypass") returns False

How can I help you build?



```
print("\n--- Interactive Login ---")
# Demonstrate secure input validation with user input
while True:
    input_username = input("Enter username (or 'exit' to quit): ").strip()
    if input_username.lower() == 'exit':
        break
    input_password = input("Enter password: ").strip()

    if login(input_username, input_password, SECURE_CREDENTIALS):
        print("Login successful! You can now access restricted content.")
        # In a real system, you would typically break the loop or redirect here
        # For this example, we'll let the user try again
    else:
        print("Please try again.")
    print("\n" * 30)

... --- Testing the login function ---

Attempting login with valid credentials (user1):
Welcome, user1!

Attempting login with incorrect password (admin):
Invalid username or password.

Attempting login for a non-existent user:
Invalid username or password.

Attempting login with empty username:
Error: Username and password cannot be empty.

Attempting login with empty password:
Error: Username and password cannot be empty.

--- Interactive Login ---
Enter username (or 'exit' to quit): exit
```

Gemini

- #rewrite the login system to avoid hardcoded credentials and use secure input validation"
- I've rewritten the login system to make it more secure by separating the credentials from the function and adding basic input validation. I've also included an interactive section for you to test it out.

Key changes:

- Externalized Credentials:** The SECURE_CREDENTIALS dictionary is now defined outside the login function, demonstrating how credentials would ideally be loaded from a secure external source (like a database or environment variables) rather than being hardcoded within the function itself.
- Input Validation:** The login function now checks if the username or password inputs are empty strings. If either is empty, it prints an error message and prevents authentication.
- Interactive Test:** A while loop has been added to allow you to interactively enter usernames and passwords, showcasing the input validation and login process.

What can I help you build?

PROMPT (UPGRADED) :

REWRITE THE LOGIN SYSTEM TO AVOID HARDCODED CREDENTIALS AND USE SECURE INPUT VALIDATION

EXPLANATION :

- **INITIAL CODE:** THE FIRST VERSION OF THE CODE WAS A LITTLE LAZY. IT STORED USERNAMES AND PASSWORDS IN A WAY THAT MADE THEM SUPER EASY FOR A HACKER TO FIND AND MISUSE. THINK OF IT LIKE LEAVING YOUR HOUSE KEY UNDER THE DOORMAT.
- **REVISED CODE:** THE UPDATED VERSION IS MUCH SAFER! INSTEAD OF HARDCODING CREDENTIALS, IT ASKS THE USER FOR INPUT AND CHECKS IT MORE SECURELY, KEEPING SENSITIVE DATA OUT OF PLAIN SIGHT. IT'S LIKE PUTTING YOUR KEY IN A LOCKBOX INSTEAD OF UNDER THE MAT.
- **THE OUTPUT:** THE AI SUCCESSFULLY GENERATED CODE FOR A SECURE LOGIN SYSTEM. THIS SYSTEM PROMPTS USERS FOR A USERNAME AND PASSWORD AND VALIDATES THEM SECURELY.
- **THE OBSERVATION:** THE KEY TAKEAWAY IS THAT WHILE AI CAN CREATE THIS CODE, IT'S CRUCIAL THAT HUMANS REVIEW IT. THIS HUMAN INTERVENTION IS NECESSARY TO ENSURE PRIVACY, SECURITY, RESPONSIBILITY, AND OVERALL SAFETY OF THE AI-ASSISTED CODE.

TASK - 2 : BIAS DETECTION IN AI-GENERATED DECISION SYSTEMS SCENARIO

PROMPT :

CREATE A LOAN APPROVAL SYSTEM THAT MAKES DECISIONS ONLY BASED ON INCOME AND CREDIT SCORE.

The screenshot shows a Google Colab notebook titled "AI&AC_ASSINMT_5 - Colab". In the code cell, there is a Python function named `approve_loan` which takes `income` and `credit_score` as arguments. The function logic checks if the income and credit score meet minimum requirements (set to \$30,000 and 650 respectively) and prints out whether the loan is approved or denied based on those criteria. The output of the code cell shows the generated Python code. To the right of the code cell, the Gemini AI interface is open, showing the user's prompt: "Create a loan approval system that makes decisions only based on income and credit score." Gemini has generated the provided Python code as a response. The interface also includes a sidebar with various AI-generated test cases and integration suggestions.

```
def approve_loan(income, credit_score):
    """
    Determines loan approval based on income and credit score.

    Args:
        income (float): The applicant's annual income.
        credit_score (int): The applicant's credit score.

    Returns:
        bool: True if the loan is approved, False otherwise.
    """
    # Define approval criteria
    MIN_INCOME = 30000
    MIN_CREDIT_SCORE = 650

    print(f"\nEvaluating loan application for: ")
    print(f" Income: ${income:.2f}")
    print(f" Credit Score: {credit_score}")

    if income >= MIN_INCOME and credit_score >= MIN_CREDIT_SCORE:
        print("Loan Approved! Congratulations!")
        return True
    elif income < MIN_INCOME and credit_score < MIN_CREDIT_SCORE:
        print("Loan Denied: Income ${income:.2f} is below minimum ${MIN_INCOME:.2f}")
        return False
    elif income < MIN_INCOME:
        print("Loan Denied: Income ${income:.2f} is below the minimum required ${MIN_INCOME:.2f}")
        return False
    else: # credit_score < MIN_CREDIT_SCORE
        print("Loan Denied: Credit Score ${credit_score} is below the minimum required")
        return False

# ---- Example Usage ----
print(" Loan Approval System Test")

```

The screenshot shows a Google Colab notebook titled "AI&AC_ASSINMT_5 - Colab". The left sidebar contains code snippets for testing a loan approval function based on income and credit score. The main area displays the output of these tests, showing successful approvals for higher incomes and rejections for lower incomes and credit scores below 600. The right sidebar features a Gemini AI interface that generates a detailed explanation of the loan approval logic, mentioning five test cases and the specific rules for approval and denial.

```

# Test Case 3: Denied (low credit score)
approve_loan(income=50000, credit_score=600)

# Test Case 4: Denied (both low)
approve_loan(income=20000, credit_score=550)

# Test Case 5: Approved (borderline)
approve_loan(income=30000, credit_score=650)

--- Loan Approval System Tests ---

Evaluating loan application for:
Income: $45,000.00
Credit Score: 700
Loan Approved! Congratulations!

Evaluating loan application for:
Income: $25,000.00
Credit Score: 680
Loan Denied: Income ($25,000.00) is below the minimum required ($30,000.00).

Evaluating loan application for:
Income: $50,000.00
Credit Score: 600
Loan Denied: Credit Score (600) is below the minimum required (650).

Evaluating loan application for:
Income: $20,000.00
Credit Score: 550
Loan Denied: Income ($20,000.00) is below minimum ($30,000.00) AND Credit Score (550) is b

Evaluating loan application for:
Income: $30,000.00
Credit Score: 650
Loan Approved! Congratulations!
True

```

Gemini

R Create a loan approval system that makes decisions only based on income and credit score."

Ok, I can create a simple loan approval system for you. This Python function will determine loan eligibility based on a given income and credit score, applying some basic rules.

The loan approval system code has executed successfully, and the output demonstrates how different scenarios are handled based on income and credit score. Here's a breakdown of the test cases:

- Test Case 1 (Income: \$45,000, Credit Score: 700):
 - Result: Loan Approved! Both income and credit score met the minimum requirements.
- Test Case 2 (Income: \$25,000, Credit Score: 680):
 - Result: Loan Denied. The income (\$25,000) was below the minimum required (\$30,000), even though the credit score was sufficient.

Integrate the login system with the loan approval system
Explain the logic for loan denial scenarios
Add more criteria to the loan approval function

What can I help you build?

Gemini 2.5 Flash ▶

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

CODE EXPLANATION:

THIS PROGRAM CHECKS WHETHER A LOAN CAN BE APPROVED BY LOOKING ONLY AT FINANCIAL DETAILS LIKE INCOME AND ELIGIBILITY CRITERIA. IT DELIBERATELY AVOIDS USING PERSONAL INFORMATION, WHICH HELPS KEEP THE DECISION FAIR AND UNBIASED.

OUTPUT:

BASED ON THE PROVIDED FINANCIAL INPUTS, THE PROGRAM CLEARLY SHOWS WHETHER THE LOAN IS APPROVED OR REJECTED.

OBSERVATION:

FOCUSING ONLY ON RELEVANT FINANCIAL DATA HELPS REDUCE BIAS IN AUTOMATED DECISION-MAKING SYSTEMS. WELL-DEFINED PROMPTS ALSO GUIDE AI TO PRODUCE FAIR, RESPONSIBLE, AND TRANSPARENT CODE.

TASK - 3 : TRANSPARENCY AND EXPLAINABILITY IN AI-GENERATED CODE (RECURSIVE BINARY SEARCH)

PROMPT :

GENERATE A PYTHON PROGRAM THAT IMPLEMENTS RECURSIVE BINARY SEARCH ON A SORTED LIST. INCLUDE CLEAR INLINE COMMENTS AND A STEP-BY-STEP EXPLANATION OF THE RECURSIVE LOGIC. EXPLAIN THE BASE CASE AND RECURSIVE CASE EXPLICITLY. EVALUATE WHETHER THE CODE AND EXPLANATION ARE CLEAR AND UNDERSTANDABLE FOR BEGINNERS.

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```

# Recursive Case: Target is in the left half.
elif arr[mid] > target:
    print(f" [Debug] Mid element {arr[mid]} > Target {target}. Recursing left (high={high}, low={low}, mid={mid-1})")
    return recursive_binary_search(arr, target, low, mid - 1)
# Recursive Case: Target is in the right half.
else:
    print(*args, sep=' ', end='\n', file=None, flush=False). Recursing right (low={low}, high={high}, mid={mid+1})
```

---- Ex string inserted between values, default a space.

my_list

list_len

string appended after the last value, default a newline.

file

print(f" Prints the values to a stream, or to sys.stdout by default.

target1 = 12

Attempting to find (target1)...

result1 = recursive_binary_search(my_list, target1, 0, list_len - 1)

if result1 != -1:

print("Result: Element {target1} found at index {result1}.")

else:

print("Result: Element {target1} not found.")

Test Case 2: Element not found

target2 = 45

Attempting to find (target2)...

result2 = recursive_binary_search(my_list, target2, 0, list_len - 1)

if result2 != -1:

print("Result: Element {target2} found at index {result2}.")

else:

print("Result: Element {target2} not found.")

Test Case 3: Element at the beginning

target3 = 2

Variables Terminal

Gemini

R Generate a Python program that implements recursive binary search on a sorted list. Include clear inline comments and a step-by-step explanation of the recursive logic. Explain the base case and recursive case explicitly. Evaluate whether the code and explanation are clear and understandable for beginners.

◆ Certainly! Here's a Python program for recursive binary search, complete with inline comments, a detailed explanation of the recursive logic, and an evaluation of its clarity for beginners.

The recursive binary search program has executed successfully! The debug prints clearly show the step-by-step process of the algorithm as it narrows down the search space for each test case.

Here's a summary of the results:

- Target 12 (found): The algorithm successfully found 12 at index 3, tracing through a few recursive calls to the left and right halves of the list.
- Target 45 (not found): The algorithm correctly determined that 45 is not in the list. The debug output shows the search space eventually becoming empty (low=7, high=6), leading to the return of -1.
- Target 2 (found at beginning): The algorithm found 2 at index 0, tracing through a few recursive calls to the left and right halves of the list.

Convert recursive binary search to iterative

Add error handling for unsorted input arrays

Explain the time complexity of this binary search implementation

What can I help you build?

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

✓ 3:43PM Python 3

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```

def recursive_binary_search(arr, target, low, high):
    """
    Performs a recursive binary search on a sorted list.

    Args:
        arr (list): The sorted list to search within.
        target: The element to search for.
        low (int): The lower bound index of the current search space.
        high (int): The upper bound index of the current search space.

    Returns:
        int: The index of the target if found, otherwise -1.
    """

    # Base Case 1: The search space is empty, meaning the target is not in the list.
    if low > high:
        print(f" [Debug] Search space empty (low={low}, high={high}). Target {target} not found")
        return -1

    # Calculate the middle index of the current search space.
    # Using (low + high) // 2 can prevent overflow issues with very large low/high values
    # compared to low + (high - low) // 2, though in Python this is less of a concern.
    mid = low + (high - low) // 2

    print(f" [Debug] Searching between index {low} and {high}. Mid index: {mid}, Mid element: {arr[mid]}")

    # Base Case 2: The middle element is the target.
    if arr[mid] == target:
        print(f" [Debug] Target {target} found at index {mid}.")
        return mid
    # Recursive Case: Target is in the left half.
    elif arr[mid] > target:
        print(f" [Debug] Mid element {arr[mid]} > Target {target}. Recursing left (high={high}, low={low}, mid={mid-1})")
        return recursive_binary_search(arr, target, low, mid - 1)
    # Recursive Case: Target is in the right half.
    else:
        print(f" [Debug] Mid element {arr[mid]} < Target {target}. Recursing right (high={high}, low={low}, mid={mid+1})")
        return recursive_binary_search(arr, target, mid + 1, high)
```

Variables Terminal

Gemini

R Generate a Python program that implements recursive binary search on a sorted list. Include clear inline comments and a step-by-step explanation of the recursive logic. Explain the base case and recursive case explicitly. Evaluate whether the code and explanation are clear and understandable for beginners.

◆ Certainly! Here's a Python program for recursive binary search, complete with inline comments, a detailed explanation of the recursive logic, and an evaluation of its clarity for beginners.

The recursive binary search program has executed successfully! The debug prints clearly show the step-by-step process of the algorithm as it narrows down the search space for each test case.

Here's a summary of the results:

- Target 12 (found): The algorithm successfully found 12 at index 3, tracing through a few recursive calls to the left and right halves of the list.
- Target 45 (not found): The algorithm correctly determined that 45 is not in the list. The debug output shows the search space eventually becoming empty (low=7, high=6), leading to the return of -1.
- Target 2 (found at beginning): The algorithm found 2 at index 0, tracing through a few recursive calls to the left and right halves of the list.

What can I help you build?

Gemini 2.5 Flash ▷

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

✓ 3:43PM Python 3

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all ▶

```
[5] ✓ 0s
● # Test Case 3: Element at the beginning
target3 = 2
print(f"\nAttempting to find {target3}...")
result3 = recursive_binary_search(my_list, target3, 0, list_len - 1)
if result3 != -1:
    print(f"Result: Element {target3} found at index {result3}.")
else:
    print(f"Result: Element {target3} not found.")

# Test Case 4: Element at the end
target4 = 91
print(f"\nAttempting to find {target4}...")
result4 = recursive_binary_search(my_list, target4, 0, list_len - 1)
if result4 != -1:
    print(f"Result: Element {target4} found at index {result4}.")
else:
    print(f"Result: Element {target4} not found.")

... Searching in list [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

Attempting to find 12...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 > Target 12. Recursing left (high=3).
[Debug] Searching between index 0 and 3. Mid index: 1, Mid element: 5
[Debug] Mid element 5 < Target 12. Recursing right (low=2).
[Debug] Searching between index 2 and 3. Mid index: 2, Mid element: 8
[Debug] Mid element 8 < Target 12. Recursing right (low=3).
[Debug] Searching between index 3 and 3. Mid index: 3, Mid element: 12
[Debug] Target 12 found at index 3.
Result: Element 12 found at index 3.

Attempting to find 45...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 < Target 45. Recursing right (low=5).
[Debug] Searching between index 5 and 9. Mid index: 7, Mid element: 56
[Debug] Mid element 56 > Target 45. Recursing left (high=6).
[Debug] Searching between index 5 and 6. Mid index: 5, Mid element: 23
[Debug] Mid element 23 < Target 45. Recursing right (low=6).
[Debug] Searching between index 6 and 6. Mid index: 6, Mid element: 38
[Debug] Mid element 38 < Target 45. Recursing right (low=7).
[Debug] Search space empty (low=7, high=6). Target 45 not found.
Result: Element 45 not found.

Attempting to find 2...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 > Target 2. Recursing left (high=3).
[Debug] Searching between index 0 and 3. Mid index: 1, Mid element: 5
[Debug] Mid element 5 > Target 2. Recursing left (high=0).
[Debug] Searching between index 0 and 0. Mid index: 0, Mid element: 2
[Debug] Target 2 found at index 0.
Result: Element 2 found at index 0.

Attempting to find 91...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 < Target 91. Recursing right (low=5).
[Debug] Searching between index 5 and 9. Mid index: 7, Mid element: 56
[Debug] Mid element 56 < Target 91. Recursing right (low=8).
[Debug] Searching between index 8 and 9. Mid index: 8, Mid element: 72
[Debug] Mid element 72 < Target 91. Recursing right (low=9).
[Debug] Searching between index 9 and 9. Mid index: 9, Mid element: 91
[Debug] Target 91 found at index 9.
Result: Element 91 found at index 9.
```

Gemini

R Generate a Python program that implements recursive binary search on a sorted list. Include clear inline comments and a step-by-step explanation of the recursive logic. Explain the base case and recursive case explicitly. Evaluate whether the code and explanation are clear and understandable for beginners.

◆ Certainly! Here's a Python program for recursive binary search, complete with inline comments, a detailed explanation of the recursive logic, and an evaluation of its clarity for beginners.

The recursive binary search program has executed successfully! The debug prints clearly show the step-by-step process of the algorithm as it narrows down the search space for each test case.

Here's a summary of the results:

- Target 12 (found): The algorithm successfully found 12 at index 3, tracing through a few recursive calls to the left and right halves of the list.
- Target 45 (not found): The algorithm correctly determined

Convert recursive binary search to iterative
Add error handling for unsorted input arrays
Explain the time complexity of this binary search implementation

What can I help you build?

+ Gemini

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

✓ 3:43 PM Python 3

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all ▶

```
[5] ✓ 0s
● ... Searching in list [2, 5, 8, 12, 16, 23, 38, 56, 72, 91]

Attempting to find 12...
[Debug] Mid element 5 < Target 12. Recursing right (low=2).
[Debug] Searching between index 2 and 3. Mid index: 2, Mid element: 8
[Debug] Mid element 8 < Target 12. Recursing right (low=3).
[Debug] Searching between index 3 and 3. Mid index: 3, Mid element: 12
[Debug] Target 12 found at index 3.
Result: Element 12 found at index 3.

Attempting to find 45...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 < Target 45. Recursing right (low=5).
[Debug] Searching between index 5 and 9. Mid index: 7, Mid element: 56
[Debug] Mid element 56 > Target 45. Recursing left (high=6).
[Debug] Searching between index 5 and 6. Mid index: 5, Mid element: 23
[Debug] Mid element 23 < Target 45. Recursing right (low=6).
[Debug] Searching between index 6 and 6. Mid index: 6, Mid element: 38
[Debug] Mid element 38 < Target 45. Recursing right (low=7).
[Debug] Search space empty (low=7, high=6). Target 45 not found.
Result: Element 45 not found.

Attempting to find 2...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 > Target 2. Recursing left (high=3).
[Debug] Searching between index 0 and 3. Mid index: 1, Mid element: 5
[Debug] Mid element 5 > Target 2. Recursing left (high=0).
[Debug] Searching between index 0 and 0. Mid index: 0, Mid element: 2
[Debug] Target 2 found at index 0.
Result: Element 2 found at index 0.

Attempting to find 91...
[Debug] Searching between index 0 and 9. Mid index: 4, Mid element: 16
[Debug] Mid element 16 < Target 91. Recursing right (low=5).
[Debug] Searching between index 5 and 9. Mid index: 7, Mid element: 56
[Debug] Mid element 56 < Target 91. Recursing right (low=8).
[Debug] Searching between index 8 and 9. Mid index: 8, Mid element: 72
[Debug] Mid element 72 < Target 91. Recursing right (low=9).
[Debug] Searching between index 9 and 9. Mid index: 9, Mid element: 91
[Debug] Target 91 found at index 9.
Result: Element 91 found at index 9.
```

Gemini

R Generate a Python program that implements recursive binary search on a sorted list. Include clear inline comments and a step-by-step explanation of the recursive logic. Explain the base case and recursive case explicitly. Evaluate whether the code and explanation are clear and understandable for beginners.

◆ Certainly! Here's a Python program for recursive binary search, complete with inline comments, a detailed explanation of the recursive logic, and an evaluation of its clarity for beginners.

The recursive binary search program has executed successfully! The debug prints clearly show the step-by-step process of the algorithm as it narrows down the search space for each test case.

Here's a summary of the results:

- Target 12 (found): The algorithm successfully found 12 at index 3, tracing through a few recursive calls to the left and right halves of the list.
- Target 45 (not found): The algorithm correctly determined

Convert recursive binary search to iterative
Add error handling for unsorted input arrays
Explain the time complexity of this binary search implementation

What can I help you build?

+ Gemini 2.5 Flash -

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

✓ 3:43 PM Python 3

CODE EXPLANATION:

THIS PROGRAM USES RECURSION TO REPEATEDLY DIVIDE THE SORTED LIST INTO SMALLER HALVES UNTIL THE REQUIRED ELEMENT IS FOUND OR THE SEARCH SPACE IS EXHAUSTED. BY CLEARLY DEFINING BOTH THE BASE CASE AND THE RECURSIVE CASE, THE LOGIC BECOMES EASY TO FOLLOW AND UNDERSTAND.

OUTPUT:

THE PROGRAM ACCURATELY DISPLAYS WHETHER THE GIVEN ELEMENT IS PRESENT IN THE LIST OR NOT.

OBSERVATION:

PROVIDING CLEAR COMMENTS AND EXPLANATIONS INCREASES TRANSPARENCY AND TRUST IN AI-GENERATED CODE. IT ALSO MAKES THE CODE MORE BEGINNER-FRIENDLY AND EASIER TO UNDERSTAND, ANALYZE, AND VERIFY.

TASK - 4 : ETHICAL EVALUATION OF AI-BASED SCORING SYSTEMS.

PROMPT : CREATE A PYTHON PROGRAM TO SCORE JOB APPLICANTS BASED ON SKILLS, EXPERIENCE, AND EDUCATION.

The screenshot shows a Google Colab notebook titled "AI&AC_ASSINMT_5 - Colab". The code cell contains the following Python script:

```
applicant3 = {
    'name': 'Charlie Brown',
    'skills': ['Machine Learning', 'Python', 'SQL', 'AWS'],
    'experience': 2,
    'education': 'Associate'
}

applicant4 = {
    'name': 'Diana Prince',
    'skills': ['Python', 'Machine Learning'],
    'experience': 15,
    'education': 'PhD'
}

print("---- Applicant Scoring System ----")

# Score and print results for each applicant
applicants = [applicant1, applicant2, applicant3, applicant4]
for applicant in applicants:
    print(f"\nScoring Applicant: {applicant['name']}")
    final_score = score_applicant(applicant, job_scoring_criteria)
    print(f"Total Weighted Score for {applicant['name']}: {final_score:.2f}\n")

... --- Applicant Scoring System ---

Scoring Applicant: Alice Smith
Skills Score: 27 (Weighted: 10.80)
Experience Score: 35 years (Weighted: 10.50)
Education Score: 40 (Weighted: 12.00)
Total Weighted Score for Alice Smith: 33.30

Scoring Applicant: Bob Johnson
Skills Score: 22 (Weighted: 8.80)
Experience Score: 50 years (Weighted: 15.00)
```

The output of the code is displayed below the code cell, showing the scores for Alice Smith and Bob Johnson. To the right of the code cell, there is a Gemini AI interface window. The prompt used is: "Create a Python program to score job applicants based on skills, experience, and education." Gemini's response includes a summary of how each applicant was scored, a list of applicants with their scores, and three buttons: "Add more applicants to the scoring system", "Analyze the distribution of applicant scores", and "Compare applicant scores with loan approval criteria".

```

[6] ⓘ def score_applicant(applicant_data, scoring_criteria):
    """
    Scores a job applicant based on provided skills, experience, and education,
    using predefined scoring criteria.

    Args:
        applicant_data (dict): A dictionary containing applicant's details:
            - 'skills' (list of str): List of skills the applicant pos-
            - 'experience' (int): Years of professional experience.
            - 'education' (str): Highest education level (e.g., 'High
        scoring_criteria (dict): A dictionary defining how each category is scored:
            - 'weights' (dict): Weights for 'skills', 'experience',
            - 'skill_points' (dict): Points for each specific skill
            - 'experience_points' (dict): Points for years of experi-
            - 'education_points' (dict): Points for each education l

    Returns:
        float: The total weighted score for the applicant.
    """
    total_score = 0
    weights = scoring_criteria['weights']

    # 1. Score Skills
    skill_score = 0
    applicant_skills = [s.lower() for s in applicant_data.get('skills', [])]
    for skill in applicant_skills:
        if skill in scoring_criteria['skill_points']:
            skill_score += scoring_criteria['skill_points'][skill]
    total_score += skill_score * weights['skills']
    print(f" Skills Score: {skill_score} (Weighted: {(skill_score * weights['skills']):.2f})")

    # 2. Score Experience
    experience_score = 0

```

To undo cell deletion use %/Ctrl+M Z or the Undo option in the Edit menu

Gemini

R Prompt Used: Create a Python program to score job applicants based on skills, experience, and education.

◆ Certainly! Here's a Python program to score job applicants based on skills, experience, and education. I've included a flexible scoring system where you can define weights and criteria for each category.

The job applicant scoring system has executed successfully! The output shows the detailed scoring for each of the four example applicants based on the defined criteria.

Here's a summary of how each applicant was scored:

- Alice Smith: Scored 33.30. She has a Bachelor's degree and moderate experience, along with skills in Python, SQL, and Data Analysis.
- Bob Johnson: Scored 41.80. He has a Master's degree and significant experience, with skills in Java, AWS, and Communication

Add more applicants to the scoring system
Analyze the distribution of applicant scores
Compare applicant scores with loan approval criteria

What can I help you build?

```

[6] ⓘ def score_applicant(applicant_data, scoring_criteria):
    """
    Scores a job applicant based on provided skills, experience, and education,
    using predefined scoring criteria.

    Args:
        applicant_data (dict): A dictionary containing applicant's details:
            - 'skills' (list of str): List of skills the applicant pos-
            - 'experience' (int): Years of professional experience.
            - 'education' (str): Highest education level (e.g., 'High
        scoring_criteria (dict): A dictionary defining how each category is scored:
            - 'weights' (dict): Weights for 'skills', 'experience',
            - 'skill_points' (dict): Points for each specific skill
            - 'experience_points' (dict): Points for years of experi-
            - 'education_points' (dict): Points for each education l

    Returns:
        float: The total weighted score for the applicant.
    """
    total_score = 0
    weights = scoring_criteria['weights']

    # 1. Score Skills
    skill_score = 0
    applicant_skills = [s.lower() for s in applicant_data.get('skills', [])]
    for skill in applicant_skills:
        if skill in scoring_criteria['skill_points']:
            skill_score += scoring_criteria['skill_points'][skill]
    total_score += skill_score * weights['skills']
    print(f" Skills Score: {skill_score} (Weighted: {(skill_score * weights['skills']):.2f})")

    # 2. Score Experience
    experience_score = 0
    experience_points = {
        '0-1': 10,
        '1-2': 20,
        '2-3': 30,
        '3-4': 35,
        '4-5': 50,
        '5-10': 70,
        '10-20': 70 # Can be adjusted for higher experience
    }
    total_experience_score = sum(experience_points.values())
    total_experience_weight = weights['experience'] / len(experience_points)
    experience_score = total_experience_weight * total_experience_score
    total_score += experience_score
    print(f" Experience Score: {experience_score} (Weighted: {(experience_score * weights['experience']):.2f})")

    # 3. Score Education
    education_score = 0
    education_points = {
        'high school': 10,
        'associate': 20,
        'bachelor': 40,
        'master': 60,
        'phd': 80
    }
    total_education_weight = weights['education'] / len(education_points)
    education_score = total_education_weight * sum(education_points.values())
    total_score += education_score
    print(f" Education Score: {education_score} (Weighted: {(education_score * weights['education']):.2f})")

    # --- Example Applicants ---
    applicant1 = {
        'name': 'Alice Smith',
        'skills': ['Python', 'SQL', 'Data Analysis'],
        'experience': 5,
        'education': 'Bachelor'
    }

    applicant2 = {
        'name': 'Bob Johnson',
        'skills': ['Java', 'AWS', 'Machine Learning'],
        'experience': 10,
        'education': 'Master'
    }

    applicant3 = {
        'name': 'Charlie Brown',
        'skills': ['Python', 'SQL', 'Machine Learning'],
        'experience': 15,
        'education': 'PhD'
    }

    applicant4 = {
        'name': 'Diana Prince',
        'skills': ['Python', 'AWS', 'Machine Learning'],
        'experience': 20,
        'education': 'PhD'
    }

```

Gemini

R Prompt Used: Create a Python program to score job applicants based on skills, experience, and education.

◆ Certainly! Here's a Python program to score job applicants based on skills, experience, and education. I've included a flexible scoring system where you can define weights and criteria for each category.

The job applicant scoring system has executed successfully! The output shows the detailed scoring for each of the four example applicants based on the defined criteria.

Here's a summary of how each applicant was scored:

- Alice Smith: Scored 33.30. She has a Bachelor's degree and moderate experience, along with skills in Python, SQL, and Data Analysis.
- Bob Johnson: Scored 41.80. He has a Master's degree and significant experience, with skills in Java, AWS, and Communication

Add more applicants to the scoring system
Analyze the distribution of applicant scores
Compare applicant scores with loan approval criteria

What can I help you build?

The screenshot shows a Google Colab notebook titled "AI&AC_ASSINMT_5 - Colab". The code in the notebook is as follows:

```

[6]  Os
  1 # 2. Score Experience
  2 experience_score = 0
  3 years_experience = applicant_data.get('experience', 0)
  4 for exp_range, points in scoring_criteria['experience_points'].items():
  5     min_exp, max_exp = map(int, exp_range.split('-'))
  6     if min_exp <= years_experience <= max_exp:
  7         experience_score = points
  8     break
  9 # Handle experience beyond defined ranges, or specific high-experience points
 10 if years_experience > int(list(scoring_criteria['experience_points'].keys())[-1].split(
 11     experience_score = scoring_criteria['experience_points'][list(scoring_criteria['experienc
 12
 13     total_score += experience_score * weights['experience']
 14 print(f" Experience Score: {experience_score} years (Weighted: {experience_score * weight
 15
 16 # 3. Score Education
 17 education_score = 0
 18 education_level = applicant_data.get('education', '').lower()
 19 if education_level in scoring_criteria['education_points']:
 20     education_score = scoring_criteria['education_points'][education_level]
 21 total_score += education_score * weights['education']
 22 print(f" Education Score: {education_score} (Weighted: {education_score * weights['educa
 23
 24 return total_score
 25
 26 --- Define Scoring Criteria ---
 27 ob_scoring_criteria = {
 28     'weights': {
 29         'skills': 0.4,           # Skills contribute 40% to the total score
 30         'experience': 0.3,      # Experience contributes 30%
 31         'education': 0.3        # Education contributes 30%
 32     },
 33     'skill_points': {
 34         'python': 10,
 35         'java': 8,
 36         'aws': 7
 37     }
 38 }

```

To the right of the code, there is a Gemini AI interface window. The prompt used is: "Create a Python program to score job applicants based on skills, experience, and education." The response provided by Gemini is a Python program similar to the one in the Colab notebook, with some additional comments and a summary of how each applicant was scored. The summary includes two examples: Alice Smith (Scored 33.30) and Bob Johnson (Scored 41.80). Below the summary are three buttons: "Add more applicants to the scoring system", "Analyze the distribution of applicant scores", and "Compare applicant scores with loan approval criteria". A message at the bottom of the Gemini window says: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)". The timestamp in the bottom right corner of the Colab interface is 3:50 PM.

CODE EXPLANATION:

THIS PROGRAM EVALUATES APPLICANTS BY ASSIGNING SCORES BASED ONLY ON JOB-RELATED FACTORS SUCH AS SKILLS, EXPERIENCE, AND EDUCATIONAL BACKGROUND. IT INTENTIONALLY IGNORES PERSONAL DETAILS LIKE NAME OR GENDER, WHICH HELPS KEEP THE SCORING PROCESS OBJECTIVE AND ETHICAL.

OUTPUT:

EACH APPLICANT RECEIVES A NUMERICAL SCORE THAT REFLECTS THEIR PROFESSIONAL QUALIFICATIONS.

OBSERVATION:

RELYING SOLELY ON RELEVANT CRITERIA HELPS REDUCE BIAS IN AUTOMATED SCORING SYSTEMS. ETHICAL EVALUATION IS ESSENTIAL TO ENSURE FAIRNESS AND TRUST IN AI-ASSISTED HIRING DECISIONS.

TASK - 5 : INCLUSIVENESS AND ETHICAL VARIABLE DESIGN

PROMPT :

USE AN AI TOOL TO GENERATE A PYTHON CODE SNIPPET THAT PROCESSES USER OR EMPLOYEE DETAILS.

The screenshot shows a Google Colab notebook titled "AI&AC_ASSINMT_5 - Colab". The code cell contains the following Python code:

```

def initialize_employee_system():
    """
    Initializes an empty dictionary to store employee records.
    """
    return {}

def add_employee(employees, employee_id, name, position, email):
    """
    Adds a new employee record to the system.

    Args:
        employees (dict): The dictionary storing employee records.
        employee_id (str): Unique ID for the employee.
        name (str): Employee's full name.
        position (str): Employee's job position.
        email (str): Employee's email address.
    """
    if employee_id in employees:
        print(f"Error: Employee with ID {employee_id} already exists.")
    else:
        employees[employee_id] = {
            'name': name,
            'position': position,
            'email': email
        }
    print(f"Employee {name} (ID: {employee_id}) added successfully.")

def update_employee(employees, employee_id, **kwargs):
    """
    Updates details for an existing employee.

    Args:
        employees (dict): The dictionary storing employee records.
    """

```

The right panel displays the "Gemini" AI interface, which has generated a snippet for processing user or employee details. It includes a list of generated tasks and a "What can I help you build?" input field.

This screenshot shows the same Google Colab notebook and AI interface as the previous one. The code cell contains the same Python code for initializing and updating employee records. The "Gemini" panel on the right shows a similar set of generated tasks and a help input field.

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```

def display_all_employees(employees):
    """
    Displays all employee records in the system.

    Args:
        employees (dict): The dictionary storing employee records.
    """
    if not employees:
        print("\nNo employee records found.")
        return

    print("\n--- All Employee Records ---")
    for emp_id, details in employees.items():
        print(f"Employee ID: {emp_id}")
        for key, value in details.items():
            print(f" {key.capitalize()}: {value}")
        print("-" * 20)

# --- Example Usage ---

# 1. Initialize the employee system
employee_records = initialize_employee_system()

# 2. Add some employees
add_employee(employee_records, "EMP001", "Alice Smith", "Software Engineer", "alice.s@example.com")
add_employee(employee_records, "EMP002", "Bob Johnson", "Product Manager", "bob.j@example.com")
add_employee(employee_records, "EMP003", "Charlie Brown", "UX Designer", "charlie.b@example.com")

# Try adding an existing employee (should show error)
add_employee(employee_records, "EMP001", "Alice Smith Jr.", "Intern", "alice.jr@example.com")

# 3. Display all employees
display_all_employees(employee_records)

```

Variables Terminal

Gemini

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

Blue diamond Sure! I can generate a Python code snippet for processing user or employee details. This program will allow you to add new employees, update their information, and display all current employee records.

The employee management system has executed successfully, and the output clearly demonstrates its functionality:

1. Adding Employees:
 - Three employees (Alice Smith, Bob Johnson, Charlie Brown) were added successfully.
 - An attempt to add an employee with an existing ID (EMP001) correctly triggered an "Error: Employee with ID EMP001 already exists." message, preventing duplicate entries.

Add more applicants to the scoring system
Analyze the distribution of applicant scores
Compare applicant scores with loan approval criteria

What can I help you build?

Gemini

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:00PM Python 3

AI&AC_ASSINMT_5 - Colab

File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

```

# 4. Update an employee's details
update_employee(employee_records, "EMP002", position="Senior Product Manager")
update_employee(employee_records, "EMP001", email="alice.smith@newco.com", phone="123-456")

# 5. Display specific employee details after update
get_employee_details(employee_records, "EMP001")
get_employee_details(employee_records, "EMP002")

# 6. Display all employees again to see changes
display_all_employees(employee_records)

# Try to get details for a non-existent employee
get_employee_details(employee_records, "EMP004")

```

Employee Alice Smith (ID: EMP001) added successfully.
 Employee Bob Johnson (ID: EMP002) added successfully.
 Employee Charlie Brown (ID: EMP003) added successfully.
 Error: Employee with ID EMP001 already exists.

 --- All Employee Records ---
 Employee ID: EMP001
 Name: Alice Smith
 Position: Software Engineer
 Email: alice.s@example.com

 Employee ID: EMP002
 Name: Bob Johnson
 Position: Product Manager
 Email: bob.j@example.com

 Employee ID: EMP003
 Name: Charlie Brown
 Position: UX Designer
 Email: charlie.b@example.com

 Updated position for employee EMP002 to Senior Product Manager.
 Employee EMP002 details updated

Variables Terminal

Gemini

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

Blue diamond Sure! I can generate a Python code snippet for processing user or employee details. This program will allow you to add new employees, update their information, and display all current employee records.

The employee management system has executed successfully, and the output clearly demonstrates its functionality:

1. Adding Employees:
 - Three employees (Alice Smith, Bob Johnson, Charlie Brown) were added successfully.
 - An attempt to add an employee with an existing ID (EMP001) correctly triggered an "Error: Employee with ID EMP001 already exists." message, preventing duplicate entries.

Add more applicants to the scoring system
Analyze the distribution of applicant scores
Compare applicant scores with loan approval criteria

What can I help you build?

Gemini 2.5 Flash

Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)

4:00PM Python 3

OUTPUT:

THE PROGRAM CORRECTLY DISPLAYS EMPLOYEE DETAILS BASED ON THE INFORMATION PROVIDED BY THE USER.

OBSERVATION:

USING GENDER-SPECIFIC VARIABLES MAY INTRODUCE ASSUMPTIONS AND SHOULD BE REVIEWED TO ENSURE THE CODE IS INCLUSIVE AND RESPECTFUL OF ALL USERS.

CONCLUSION:

THIS LAB EMPHASIZES THE NEED TO CAREFULLY REVIEW AI-GENERATED CODE TO MAINTAIN PRIVACY, FAIRNESS, TRANSPARENCY, AND INCLUSIVENESS, WHICH ARE ESSENTIAL PRINCIPLES OF RESPONSIBLE SOFTWARE DEVELOPMENT.

