## TASK-5: Implement various Searching and Sorting Operations in python programming

**Aim:** To Implement various searching and sorting operations in python programming.

5.) A company stores employee records in a list of dictionaries, where each dictionary contains id, name and department. write a function find-employee-by-id that takes this list and a target employee ID as arguments and returns the dictionary of the employee with the matching ID, or None if no such employee is found..

**Algorithm:**

1. Input Definition:
2. Define the function find_employee_by-id that takes two parameters.
   a. A list of dictionaries (employees), where each dictionary represents an employee record with keys, id, name, and department
   b. An integer (target_id) representing the employee ID to be searched.
3. Iterate through the list.
   use a for loop to iterate through each dictionary in the employees list
4. Check for matching Record or ID
   within the loop, check if the id field of the current dictionary matches the target_id
5. Return Matching Record.
   If a matching is found, return the current dictionary.
6. Handle No match.
   If the loop completes without findings a match, return None.

Output :-

{'id': 2, 'name': 'Bob', 'department': 'Engineering'}

Program 5.1 :-

```
def find_employee_by_id(employees, target_id)
  for employee in employees
      if employee['id']==target_id
          return employee
  return None
# Test the function
employees = [
   {'id': 1, 'name': 'Alice', 'department': 'HR'},
   {'id': 2, 'name': 'Bob', 'department': 'Engineering'},
   {'id': 3, 'name': 'Charlie', 'department': 'Sales'},
]
Print(find_employee_by_id(employees, 2))#Output: {'id': 2,
'name': 'Bob', 'department': 'Engineering'}
```

program 5.1

```
def find_employee_by_id(employees, target_id):
    for employee in employees:
        if employee['id'] == target_id:
            return employee
    return None
# Test the function
employees = [
    {'id': 1, 'name': 'Alice', 'department': 'HR'},
    {'id': 2, 'name': 'Bob', 'department': 'Engineering'},
    {'id': 3, 'name': 'Charlie', 'department': 'Sales'}
]
print(find_employee_by_id(employees, 2))# output:
{'id': 2, 'name': 'Bob', 'department': 'Engineering'}
```

5.2 You are developing a grade management system for a school. The system maintains a list of student records, where each record is represented as a dictionary containing a student's name and score.

Algorithm:-

1. Initialization
   * Get the length of the student's list and store it in n
2. Outer Loop:
   * Iterate from i=0 to n-1 (inclusive). This loop represents the number of passes through the list.
3. Track swaps:
   * Initialize a boolean variable swapped to

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
        # Track if any swap is made in this pass
        swapped = False
        for j in range(0, n-i-1):
            if students[j]['score'] > students[j+1]['score']:
                # Swap if the score of the current student
                is greater than the next students[j],
                students[j+1] = students[j+1], students[j]
                swapped = True

        # If no two elements were swapped, the
        list is already sorted if not swapped:
            break
# Example usage
students = [
    {'name': 'Alice', 'score': 88},
    {'name': 'Bob', 'score': 95},
    {'name': 'Charlie', 'score': 75},
    {'name': 'Diana', 'score': 85}
]
print("Before sorting:")
for student in students:
    print(student)
bubble_sort_scores(students)
print("\nAfter sorting:")
for student in students:
    print(student)
```

False. This variable will track if any swaps are made in the current pass.

4. Inner LOOP:
   * Iterate from i=0 to n-i-2 (inclusive). This loop compares adjacent elements in the list and performs swaps if necessary.

5. Compare and swap:
   * For each pair of adjacent elements (i.e., students[j] and students[j+1]):
     - compare their score values.
     - If students[j]['score'] > students[j+1]['score'], swap the two elements.
     - Set swapped to True to indicate that a swap was made.

6. Early termination:
   - After each pass of the inner loop, check if swapped is False. If no swaps were made during the pass, the list is already sorted, and you can break out of the outer loop early.

7. Completion
   * The function modifies the students list in place, sorting it by score.

Program 5.7:

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
#Track if any swap is made in this pass
    swapped = False
```

Output :-

Before sorting :

{'name': 'Alice', 'score': 88}
{'name': 'Bob', 'score': 95}
{'name': 'charlie', 'score': 75}
{'name': 'Diana', 'score': 85}

After sorting :

{'name': 'charlie', 'score': 75}
{'name': 'Diana', 'score': 85}
{'name': 'Alice', 'score': 88}
{'name': 'Bob', 'score': 95}

```
for j in range (0, n-i-1):
  if students[j]['score'] > students[j+1]['score']:
# swap if the score of the current student
  is greater than the next students[j],
students[j+1] = students[j+1], students[j]
  swapped = True
# If no two elements were swapped, the
list is already sorted
  if not swapped:
    break
# Example usage
students = [
  {'name': 'Alice', 'score': 88},
  {'name': 'Bob', 'score': 95},
  {'name': 'Charlie', 'score': 75},
  {'name': 'Diana', 'score': 85}
]
print ("Before sorting:")
for student in students:
    print (student)
bubble_sort_scores (students)
print ("After sorting:")
for student in students:
    print (student)
```

| VEL TECH | |
|---|---|
| EX No. | 5 |
| PERFORMANCE (5) | 5 |
| RESULT AND ANALYSIS (5) | 5 |
| VIVA VOCE (5) | 5 |
| RECORD (5) | 5 |
| TOTAL (20) | 20 |
| SIGN WITH DATE | 10/9 |

Result: Thus, the program for various searching and sorting operations is executed and verified successfully.

| VEL TECH | |
|---|---|
| EX No. | 5 |
| PERFORMANCE (5) | |
| RESULT AND ANALYSIS (3) | |
| VIVA VOCE (3) | |
| RECORD (4) | |
| TOTAL (15) | |
| SIGN WITH DATE | |