25/8/25

TASK-5 IMPLEMENTING VARIOUS SEARCHING AND SORTING OPERATIONS IN PYTHON PROGRAMMING

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

5.1: 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 the 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 the definition

2. Define the function find employee by id that takes two parameters.

a. A list of dictionaries where each dictionary represents an employee record by keys id, name and department.

b. An integer (target-id) represents the employee id to be searched.

3. Iterate through the list

4. Within the loop, check if the id field to be current dictionary matches the target id.

5. Return the matching record: if the match is not found return current dictionary.

6. Handle No Match:

If the loop complete without finding a match return none.

5.2 You are developing a grade management system for a school. The system maintains a list of student records where each records as a dictio -nary containing a students name and score the school needs to generate that displays stu -dents. score in ascending order your task is to implement a feature that sorts the student reco -ds by their scores using the bubble sort algorithm.

ALGORITHM :-

1. Intialization : Get the length of the students list and store it in n.
2. outer loop : Iterate from i=0 to n-1 (inclusive). The loop represents of passes through the list.
3. Track swaps : Intialize a boolean variable 'swapped' to false variable will track if any swaps made in current
4. Inner loop: Iterate from j=0 to n-i-2. The loop compares adjacent elements in the list and perform swaps if necessary.
5. Compare and swap : For each pair of adjacent elements (i.e. students[j] and students[j+1]):
   - compare their scored values.
   - if students[j]['score'] > students[j+1]['score']
6. Early Termination : After each pass of the inner loop, check if swapped is false if no swaps user made during the pass, the list is already stopped and you can break out of the outer loop easily

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 }

7. Competition : The function modifies the student list in place . storing its state

PROGRAM:

```
def bubble sort scores (students):
    n= len (students)
    for  i in range (n)
  # Track if any swap mode in this pass.
        swapped false
    for j in range (0, n-i-1):
        if students [j] ['score'] ; students [j+1] ['score']:
  # if no elements were swapped the list is already
   sorted if not swapped.
  # Example usage
       students -[
    { name : 'Alice' : 'score' :88 }
    { name : 'bob' : 'score' : 95 }
    { name : 'Charlie' : 'score' : 75 }
    { name : 'diana' : 'score' : 85 }
    ]
    Printf (" Before sorting :" )
    for students in students
  Print (student)
bubble . sort . scores (students):
   Printf ("\n After sorting :")
   for student in students
       Print (student)
```

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

RESULT: Thus the program for various searching and sorting algorithms was executed and verified suressfully.