* Program:-

```
def find_employee_by_id(employees, target_id):
    for employee in employees:
        if employee['id'] == target_id:
            return employee
    return None

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'}.

Date:- 25/8/25

TASK:- 5 Implement 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 dictinaries, where each dictionary 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:
3. Iterate Through the list:
   use a for loop to iterate through each dictionary in the employees list.
4. Check for matching ID:
   within the loop, check if the id field of the current dictionary matches the target_id.
5. Return Matching Record:
   If a match is found, return the current dictionary.
6. Handle No Match:

* Program 5.2:

```
def bubble_sort_scores(students):
    n = len(students)
    for i in range(n):
        swapped = False
        for j in range(0, n-i-1):
            if students[j]['score'] > students[j+1]['score']:
                students[j], students[j+1] = students[j+1],
                students[j].
                swapped = True
        if not swapped:
            break

students = [{'name': 'Alice', 'score': 88},
            {'name': 'Bob', 'score': 95},
            {'name': 'Charlie', 'score': 75},
            ]

print("Before Sorting:")
for student in students:
    print(student)
bubble_sort_scores(students)
print("\n After Sorting:")
for student in students:
    print(student)
```

output:

```
Before Sorting:
{'name': 'Alice', 'score', 88}
{'name': 'Bob', 'score', 95}
{'name': 'Charlie', 'score', 75}
```

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 students 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 false. This variable will track if any swaps are made in the current pass.
4. Inner loop:
   - Iterate from j=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.
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

{'name': 'Diana', 'score': 85}.

After Sorting:

{'name': 'charlie', 'score': 75}

{'name': 'Diana', 'score': 85}

{'name': 'Alice', 'score': 88}

{'name': 'Bob', 'score': 95}

7. Completion:

- The function modifies the Students list in place, sorting it by score.

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

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

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