Choosing and using data structures

This reading illustrates the importance of chosing the correct data structure for the task at hand.

Which data structure to choose?

The tricky part for new developers is understanding which data structure is suited to the required solution. Each data structure offers a different approach to storing, accessing and updating the information stored inside it. There can be many factors to select from, including **size**, **speed** and **performance**. The best way to try and understand which one is more suitable is through an example.

Example: Employees list

In this example, there's a list of employees that work in a restaurant. You need to be able to find an employee by their employee ID - an integer-based numeric ID. The function `get_employee` contains a `for` loop to iterate over the list of employees and returns an employee object if the ID matches.

```
employee_list = [{"id": 12345, "name": "John", "department": "Kitchen"}, {"id": 12458, "name": "Paul", "department": "House Floor"}]

def get_employee(id):
    for employee in employee_list:
        if employee['id'] == id:
            return employee

print(get_employee(12458));
## OUTPUT
{'id': 12458, 'name': 'Paul', 'department': 'House Floor'}

```

Run

Reset

The code runs well and will return the user Paul, as its ID, 12458, is matched. The challenge comes when the list gets bigger.

Instead of two employees, you may have 2000 or even 20,000. The code will have to iterate over the list sequentially until the number is matched.

You could optimize the code to split the search, but even with this, it still lacks in performance in comparison to other data structures, such as the dictionary.

```
employee_dict = {
    12345: {
        "id": "12345",
        "name": "John",
        "department": "Kitchen"
    },
    12458: {
        "id": "12458",
        "name": "Paul",
        "department": "House Floor"
    }
}

def get_employee_from_dict(id):
    return employee_dict[id];


print(get_employee_from_dict(12458));
## OUTPUT
{'id': 12458, 'name': 'Paul', 'department': 'House Floor'}

```

Run

Reset

Notice how, in this code block, if you change the data structure to use a dictionary, it will allow you to find the employee. The main difference is that you no longer need to iterate over the list to locate them. If the list expands to a much larger size, the seek time to find the employee stays the same.

This is a prime example of how to choose the right data structure to suit the solution.

Both work well, but the trade-off to be considered is that of time and scale. The first solution will work well for smaller amounts of data, but loses performance as the data scales.

The second solution is better suited to large amounts of data as its structure alows for a constant seek time allowing large amounts of data to be accessed at a constant rate.

This example shows that there is no one size fits all solution and careful consideration should be given to the choice of data structure to be used depending on the constraints of the solution.

Mark as completed