CHITHA SHIVA KUMAR

MINI PROJECT

Develop A Simple to Do List Application Using Python with An Emphasis Functions and Data Structures

OBJECTIVE:

What is a To-Do list?

A **To-Do list** is a list of tasks we require to complete or things we would like to do.

Most typically, the tasks in this list are arranged in order of priority. By tradition, they are written on a piece of paper or post-it notes and act as a memory aid. Due to the evolution in technology, we have been able to create a To-Do lists with Excel Spreadsheets, Word Documents, E-mail Lists, and To-Do List applications like Microsoft to-do and Google to-do lists. We can utilize a to-do list in our home, workplace, or personal life.

Having a list of everything we are supposed to do written down in one place implies that we should not forget anything significant. By prioritizing the tasks in the list, we can play the order of doing them and quickly observe what requires our immediate attention and what tasks can be delayed for a while.

The objective of a simple to-do list program in Python is to create a tool that allows users to manage their tasks efficiently. This typically involves functionalities such as:

- **Add_task**: Adds a task to the list of tasks.
- **View_tasks**: Displays all tasks in the list.
- **Marking tasks as complete**: Users should be able to mark tasks as completed once they're done.
- **Display_tasks()**. This function iterates through the list of tasks and prints each task along with its status. Here's how you can implement it:

- **Complete_task**: Marks a task as completed and removes it from the list.
- **Delete_task**: Deletes a task from the list.
- **Main**: The main function where the user interacts with the application.
- **Saving tasks**: Optionally, the program can allow users to save their to-do list to a file so they can access it later.

Overall, the goal is to provide a simple and intuitive interface for users to manage their tasks effectively.

Add_task: Adds a task to the list of tasks.

```
class ToDoList:

    def __init__(self):

    self.tasks = [ ]

    def add_task(self, task):

    self.tasks.append(task)

    printf ("Task '{task}' added.")
```

Delete_task: Deletes a task from the list.

```
def delete_task(self, index):

    if 1 <= index <= len(self.tasks):

    deleted_task = self.tasks.pop(index - 1)

    printf("Deleted task: {deleted_task}")
```

```
    else:
     print("Invalid task index!")
```

Display_tasks(). This function iterates through the list of tasks and prints each task along with its status. Here's how you can implement it:

```
def display_tasks():
    if tasks:
    print("Tasks:")
    for i, task in enumerate(tasks, 1):
    status = "Done" if task["completed"] else "Not Done"
    print(f"{i}. {task['task']} - {status}")
    else:
     print ("No tasks yet!")
```

Complete_task: Marks a task as completed and removes it from the list.

```
def complete_task(index):
    if 1 <= index <= len(tasks):
        tasks [index –1] ["completed"] = True
        print ("Task marked as completed.")
```

```
else:
    print ("Invalid task index!")
```

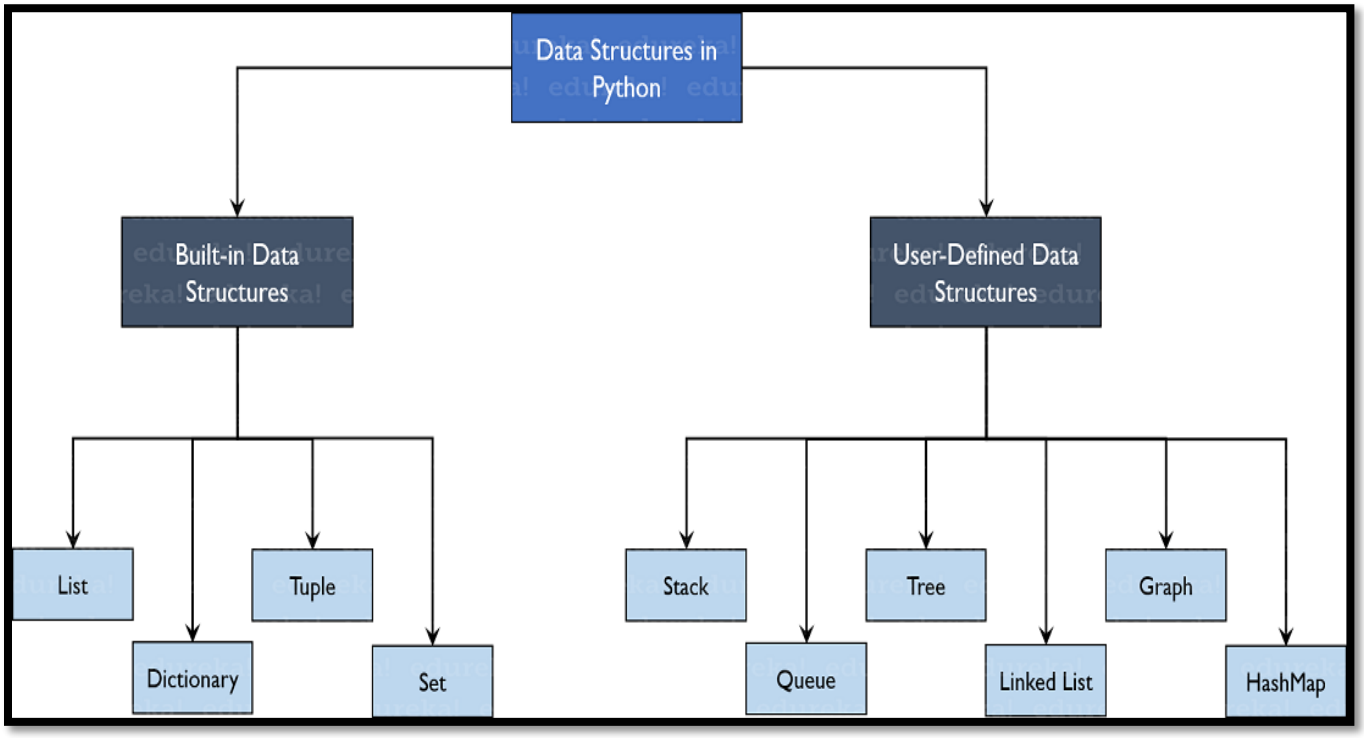
Data Structures:

Data structure is helps in Organizing, **managing** and **storing** data is important as it enables easier access and efficient modifications. Data Structures allows you to organize your data in such a way that enables you to store collections of data, relate them and perform operations on them accordingly.

Types of Data Structures in Python

Python has **implicit** support for Data Structures which enable you to store and access data. These structures are called List, Dictionary, Tuple and Set.

Python allows its users to create their own Data Structures enabling them to have **full control** over their functionality. The most prominent Data Structures are Stack, Queue, Tree, Linked List and so on which are also available to you in other programming languages. So now that you know



what are the types available to you, why don't we move ahead to the Data Structures and implement them using Python.

Built-in Data Structures:

As the name suggests, these Data Structures are built-in with Python which makes programming easier and helps programmers use them to obtain solutions faster. Let's discuss each of them in detail.

Lists:

Lists are used to store data of different data types in a sequential manner. There are addresses assigned to every element of the list, which is called as Index. The index value starts from 0 and goes on until the last element called the **positive index**. There is also **negative indexing** which starts from -1 enabling you to access elements from the last to first.

Creating a list:

To create a list, you use the square brackets and add elements into it accordingly. If you do not pass any elements inside the square brackets, you get an empty list as the output.

```
my_list = [ ] #create empty list

print(my_list)

my_list = [1, 2, 3, 'example', 3.132] #creating list with data

print(my_list)
```

Output:

```
[]
[1, 2, 3, 'example', 3.132]
```

Dictionary:

Dictionaries are used to store **key-value** pairs. To understand better, think of a phone directory where hundreds and thousands of names and their corresponding numbers have been added. Now the constant values here are Name and the Phone Numbers which are called as the keys. And the various names and phone numbers are the values that have been fed to the keys. If you access the values of the keys, you will obtain all the names and phone numbers.

Creating a Dictionary:

Dictionaries can be created using the flower braces or using the dict() function. You need to add the key-value pairs whenever you work with dictionaries.

```
my_dict = {} #empty dictionary

print(my_dict)

my_dict = {1: 'Python', 2: 'Java'} #dictionary with elements

print(my_dict)
```

Output:

```
{}

{1: ‘Python’, 2: ‘Java’}
```

Tuple:

Tuples are the same as lists are with the exception that the data once entered into the tuple cannot be changed no matter what. The only exception is when the data inside the tuple is mutable, only then the tuple data can be changed. The example program will help you understand better.

Creating a Tuple:

You create a tuple using parenthesis or using the tuple () function.

```
my_tuple = (1, 2, 3) #create tuple

print(my_tuple)
```

Output: (1, 2, 3)