Data Structures and Algorithms Assignment 1

1. Describe Python's built-in data structure?
ANS:
The Python's built-in data structure include list, set, tuples, and dictionary.

Lists
A list is defined as an ordered collection of items, and it is one of the essential data structures when using Python to create a project. The term “ordered collections” means that each item in a list comes with an order that uniquely identifies them. The order of elements is an inherent characteristic that remains constant throughout the life of the list.

Since everything in Python is considered an object, creating a list is essentially creating a Python object of a specific type. When creating a list, all the items in the list should be put in square brackets and separated by commas to let Python know that a list has been created. A sample list can be written as follows:

List_A = [item 1, item 2, item 3….., item n]

Tuples
A tuple is a built-in data structure in Python that is an ordered collection of objects. Unlike lists, tuples come with limited functionality.

The primary differing characteristic between lists and tuples is mutability. Lists are mutable, whereas tuples are immutable. Tuples cannot be modified, added, or deleted once they’ve been created. Lists are defined by using parentheses to enclose the elements, which are separated by commas.

The use of parentheses in creating tuples is optional, but they are recommended to distinguish between the start and end of the tuple. A sample tuple is written as follows:

tuple_A = (item 1, item 2, item 3,…, item n)

Sets
A set is defined as a unique collection of unique elements that do not follow a specific order. Sets are used when the existence of an object in a collection of objects is more important than the number of times it appears or the order of the objects. Unlike tuples, sets are mutable – they can be modified, added, replaced, or removed. A sample set can be represented as follows:

set_a = {“item 1”, “item 2”, “item 3”,….., “item n”}

2. Describe the Python user data structure?
ANS:
Data structures that aren’t supported by python but can be programmed to reflect the same functionality using concepts supported by python are user-defined data structures. There are many data structure that can be implemented this way:

- Linked list
- Stack
- Queue

- Tree
- Graph
- Hashmap

3. Describe the stages involved in writing an algorithm?
ANS:
Step – 1 : Obtain detailed information on the problem.
It is a very important step in writing an algorithm. Before starting with an algorithm the programmer must obtain maximum information about the problem that needs to be solved.

This step will help the programmer to get a better understanding of the problem which will surely prove to be useful while solving the problem.

Step – 2 : Analyze the problem.
Proper Analysis of the problem must be done including the data that must be gained, processed and retrieved for generating a valid output.

This step helps the programmer with various processes that must be attained while generating the output along with structure and type of data.

Step – 3 : Think of a problem solving approach.
This is a very important and the most difficult step in writing an algorithm. The programmer has to come up with a problem solving approach that will help us to build the model to solve the given problem.

Experience and practice is an important factor in thinking the problem solving approach. So make sure you try writing different algorithm and reading algorithms that will assist you for better understanding.

Step – 4 : Review the problem solving approach and try to think of a better Alternative.
A high quality Algorithm must contain the best approach to solve a problem which will help in reducing the effort in coding as well as decrease time complexity and size of the program.

Step – 5 : Develop a basic structure of the Algorithm.
Develop a basic structure of the problem solving approach; explain the approach step-by-step with short and effective description.

Step – 6 : Optimize, Improve and refine.
After developing an Algorithm try optimizing the explanation to increase readability and accessibility of it.

4. Outline the components of a good algorithm?

- Input and Output must be specified

An Input is the data transferred by the user to the program to produce certain output.

An Algorithm can have 0 to more inputs from the user. If Inputs must be taken from user the details of the data must be specified in the algorithm.

An Output is the data transferred by the program to the user results the computations.

An Algorithm must have at least 1 well-defined and desired output.

- All important steps must be mentioned

An Algorithm comprises of all short step-by-step processes that is performed in a program, thus every important step must be present in the Algorithm in considerate details.

The steps mentioned in an algorithm must lack grammatical mistakes to avoid misunderstanding and confusion.

- Instructions must be perfectly ordered

An Algorithm is a very important step in programming thus, it must be perfectly ordered to lack severe errors and confusion while coding.

An Algorithm helps and supports a programmer while coding to avoid errors, therefore a well ordered algorithm will provide better assistance.

- Short and effective descriptions

An Algorithm must contain short but effective description of the process meant to be conducted, to increase reliability and efficiency of it.

- The Algorithm must contain finite number of steps

An Algorithm must conclude after performing certain operation and generating an output. The loops mentioned in the algorithm must terminate after performing the operations.

The Algorithm must contain finite amount of steps to generate a valid output.

5. Describe the Tree traversal method?
ANS:

Traversal is a process to visit all the nodes of a tree and may print their values too. Because, all nodes are connected via edges (links) we always start from the root (head) node. That is, we cannot randomly access a node in a tree. There are three ways which we use to traverse a tree

- In-order Traversal
- Pre-order Traversal
- Post-order Traversal

Generally, we traverse a tree to search or locate a given item or key in the tree or to print all the values it contains.

In-order Traversal
In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.
Algorithm
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Pre-order Traversal
In this traversal method, the root node is visited first, then the left subtree and finally the right subtree.
Algorithm

Until all nodes are traversed −
Step 1 − Visit root node.
Step 2 − Recursively traverse left subtree.
Step 3 − Recursively traverse right subtree.

Post-order Traversal
In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.
Algorithm
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.

6. Explain the difference between inorder and postorder tree traversal?

ANS:

In-order Traversal
In this traversal method, the left subtree is visited first, then the root and later the right sub-tree. We should always remember that every node may represent a subtree itself.

If a binary tree is traversed in-order, the output will produce sorted key values in an ascending order.
Algorithm
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Visit root node.
Step 3 − Recursively traverse right subtree.

Post-order Traversal
In this traversal method, the root node is visited last, hence the name. First we traverse the left subtree, then the right subtree and finally the root node.
Algorithm
Until all nodes are traversed −
Step 1 − Recursively traverse left subtree.
Step 2 − Recursively traverse right subtree.
Step 3 − Visit root node.