Data Structures and Algorithms  **HEAP DS1**

1. Describe Python's built-in data structure?

* Python has four basic inbuilt data structures namely Lists, Dictionary, Tuple and Set. These almost cover 80% of the our real world data structures.

**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. Let us now understand lists better with the help of an example program.

my\_list = [] #create empty list

print(my\_list)

my\_list = [1, 2, 3, 'example', 3.132] #creating list with data

print(my\_list)

**Dictionary**

* Dictionaries are used to store key-value pairs.
* You can 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.
* So that is what a key-value pair is.
* in Python, this structure is stored using Dictionaries.
* Below is the eg.

my\_dict = {'First': 'Python', 'Second': 'Java'}

print(my\_dict)

my\_dict['Second'] = 'C++' #changing element

print(my\_dict)

my\_dict['Third'] = 'Ruby' #adding key-value pair

print(my\_dict)

Output:

{‘First’: ‘Python’, ‘Second’: ‘Java’}

{‘First’: ‘Python’, ‘Second’: ‘C++’}

{‘First’: ‘Python’, ‘Second’: ‘C++’, ‘Third’: ‘Ruby’}

**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.
* 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)

**Sets**

* Sets are a collection of unordered elements that are unique. Meaning that even if the data is repeated more than one time, it would be entered into the set only once. It resembles the sets that you have learnt in arithmetic. The operations also are the same as is with the arithmetic sets
* Sets are created using the flower braces but instead of adding key-value pairs, you just pass values to it.

my\_set = {1, 2, 3, 4, 5, 5, 5} #create set

print(my\_set)

Output:

{1, 2, 3, 4, 5}

Adding elements

To add elements, you use the add() function and pass the value to it.

my\_set = {1, 2, 3}

my\_set.add(4) #add element to set

print(my\_set)

Output:

{1, 2, 3, 4}

2. Describe the Python user data structure?

We have multiple user defined structures

**Arrays and Lists**

* Arrays and lists are the same structure with one difference. Lists allow heterogeneous data element storage whereas Arrays allow only homogenous elements to be stored within them.

**Stack**

* Stacks are linear Data Structures
* It follows Last in first out (LIFO) where data which is entered last will be the first to get accessed.
* It is built using the array structure and has operations like pushing (adding) elements, popping (deleting) elements and accessing elements only from one point in the stack called as the TOP.
* This TOP is the pointer to the current position of the stack.
* Stacks are prominently used in applications such as Recursive Programming, reversing words, undo mechanisms in word editors and so forth.

**Queue**

* A queue is also a linear data structure which is based on the principle of First in first out (FIFO) where the data entered first will be accessed first. It is built using the array structure and has operations which can be performed from both ends of the Queue, that is, head-tail or front-back.
* we can do like add as En-Queue and Delete as De-Queue and accessing the elements can be performed.
* Queues are used as Network Buffers for traffic congestion management, used in Operating Systems for job scheduling and many more.

**Tree**

* Trees are non-linear Data Structures which have root and nodes.
* The root is the node from where the data originates and the nodes are the other data points that are available to us.
* The node that precedes is the parent and the node after is called the child.

**Linked List**

* Linked lists are linear Data Structures which are not stored consequently but are linked with each other using pointers.
* The node of a linked list is composed of data and a pointer called next.
* These structures are most widely used in image viewing applications, music player applications and so forth.

**Graph**

* Graphs are used to store data collection of points called vertices (nodes) and edges (edges).
* Graphs can be called as the most accurate representation of a real-world map.
* They are used to find the various cost-to-distance between the various data points called as the nodes and hence find the least path. Many applications such as Google Maps, Uber, and many more use Graphs to find the least distance and increase profits in the best ways.

**HashMaps**

* HashMaps are the same as what dictionaries are in Python. They can be used to implement applications such as phonebooks, populate data according to the lists and much more.

3. Describe the stages involved in writing an algorithm?

* An algorithm is defined as “a step-by-step procedure of accomplishing some task
* below are few stages
* Begin/start : An algorithm will be enclosed by some start or being
* Input : We need some input either from user dynamically or user provided in code to accept data from user, generally used statements are INPUT, READ, GET or OBTAIN.
* Print/display : To display result or any message, generally used statements are PRINT, DISPLAY, or WRITE.
* COMPUTE or CALCULATE is used while describing mathematical expressions and based on situation relevant operators can be used.
* Display output -the final result which we need to achieve.
* In comparison to above we also have few other steps in which we can consider the stages like below:
* **Design**

The first stage is to identify the problem and thoroughly understand it.

After you obtain the input, break out the problem into stages and calculate what happens at each step so the next step can occur. This is also the point where you are going to flowchart and/or use pseudo code to work out the specific problems of solving the flow of operations within the code

* **Analyze**

Once you have the basic framework of the algorithm it’s time to start analyzing how efficient the code is in solving the problem. Algorithm design is fluid and subject to individual plans. This is a step that some programmers like to attack after they have coded the algorithm and run it through the compiler. Others prefer to examine it prior to writing the code and analyze results based on their expectations from the design stage

* **Implement**

Writing and coding the algorithm is the next step i-n the process. If you are the one writing the algorithm, then you need to write it in the coding language you understand the best. In order for you to know how to write the algorithm efficiently you have to know exactly what each line of code is going to accomplish when the program is executed. Write the code to execute quickly but can also handle the input data that it will receive

* **Experiment**

Once the algorithm is designed and coded go back and experiment with different variables in the algorithm. Try and enter data that will make it fail or try and re-write the code to work it out most efficiently. Experimentation in algorithmic design is really just another step of the analyzing of the algorithm. Keep attacking the efficiency aspect until it executes as much data as necessary in the smallest amount of time. When you find flaws in what you have written or ways to write the code better, then go back to the design step and redesign the algorithm. The design and analysis of algorithms is a circular process. You may find yourself becoming involved in any one of the steps. An experiment on an existing algorithm might lead to a new design. Or a re-coding of an algorithm might lead to a more efficient execution. Wherever you find yourself, keep working towards the goal of efficiency of the algorithm.

4. Outline the components of a good algorithm?

* An algorithm is defined as a step by step procedure of accomplishing some task
* below are the components of some good algorithms
* **Finiteness**: An algorithm should have finite number of steps and it should end after a finite time.
* **Input**: An algorithm may have many inputs or no inputs at all.
* **Output**: It should result at least one output.
* **Definiteness**: Each step must be clear, well-defined and precise. There should be no any ambiguity.
* **Effectiveness**: Each step must be simple and should take a finite amount of time.

eg below

Algorithm : Calculation of Simple Interest

Step 1: Start

Step 2: Read principle (P), time (T) and rate (R)

Step 3: Calculate I = P\*T\*R/100

Step 4: Print I as Interest

Step 5: Stop

5. Describe the Tree traversal method?

* Let us first see traversal meaning..It 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**
* **Level order Traversal**
* Tree Traversal Algorithms can be classified two categories by the order in which the nodes are visited
* **Depth-First Search (DFS) Algorithm**: It starts with the root node and first visits all nodes of one branch as deep as possible of the chosen Node and before backtracking, it visits all other branches in a similar fashion.
* **Breadth-First Search (BFS) Algorithm**: It also starts from the root node and visits all nodes of current depth before moving to the next depth in the tree.
* Depth-First Search (DFS) Algorithms have three variants:
* **Preorder Traversal** (current-left-right)— Visit the current node before visiting any nodes inside left or right subtrees.
* **Inorder Traversal** (left-current-right)— Visit the current node after visiting all nodes inside left subtree but before visiting any node within the right subtree.
* **Postorder Traversal** (left-right-current) — Visit the current node after visiting all the nodes of left and right subtrees.
* Breadth-First Search (BFS) Algorithm has one variant:
* **Level Order Traversal** — Visit nodes level-by-level and left-to-right fashion at the same level.

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

* Inorder in this it will first focus on the depth of the chosen Node and then go to the breadth at that level.
* It will start from the root node of the tree and go deeper-and-deeper into the left subtree with recursive manner.
* When we will reach to the left-most node with the above steps, then we will visit that current node and go to the left-most node of its right subtree(if exists).
* Same steps should be followed in a recursive manner to complete the inorder traversal. Order of those steps will be like (in recursive function)…
* **Go to left-subtree**
* **Visit Node**
* **Go to right-subtree**
* Inorder Traversal of Binary Search Tree will always give you Nodes in sorted manner.

**Postorder Traversal**

* here we visit the left subtree and the right subtree before visiting the current node in recursion.
* the sequence of the steps will be…
* **Go to left-subtree**
* **Go to right-subtree**
* **Visit Node**