**Software Design Laboratory Laboratory Report #3:** Data Structures (PYTHON) **Content:**

# PreLab

* **Readings**
  1. **Python DS**:
     + Chapters 7 ,8 ,10 ,12 : Ref Lab3 Fundamentals Of Python Data Structures , Kenneth A. Lambert

**Observation**

Based on my observation on Fundamentals of Python Data Structures by Kenneth A. Lambert in the seventh chapter is that a stack is a linear collection that only allows access to one end, known as the top. Elements are pushed onto or popped from the top. Other operations on stacks include peeking at the top element, determining the number of elements, determining whether the stack is empty, and returning a string representation. Stacks are also used in applications that manage data items in a last-in-first-out fashion. These applications include matching bracket symbols in expressions and evaluating postfixes expressions, backtracking algorithms, and memory management for subroutine calls on a virtual machine Arrays and singly linked structures both support simple stack implementations.

Moreover, in the eighth chapter, it discussed a queue is a linear collection that adds elements to one end, referred to as the rear, and removes them from the other end, referred to as the front. As a result, they are accessed in the first-in-first-out (FIFO) order. Other queue operations include peeking at the top element, calculating the number of elements, determining whether the queue is empty, and returning a string representation. Queues are also used in applications that manage data items in FIFO order. These applications include items for processing or access to resources that must be scheduled. Arrays and singly linked structures also support simple queue implementations. It was also mentioned that the priority queues are use a rating scheme as well as a FIFO order to schedule their elements. If two elements have the same priority, they are scheduled in the first-in-first-out (FIFO) order. Elements are otherwise ranked from smallest to largest based on some attribute, such as a number or alphabetical content. In general, no matter when they are added to the priority queue, elements with the lowest priority values are removed first.

In addition, the tenth chapter’s main topic is trees which is trees are hierarchical collections of objects. The root of a tree is the topmost node. Each node below the root in a general tree has at most one predecessor, or parent node, and zero or more successors, or child nodes. Leaves are nodes that do not have children. Interior nodes are nodes that have children. Also, a node in a binary tree can have no more than two children. Before proceeding to the next level, a complete binary tree fills each level of nodes. A complete binary tree contains all the possible nodes at each level. In addition, there are four types of tree traversals: preorder, inorder, postorder, and level order. Furthermore, an expression tree is a type of binary tree in which the interior nodes are operators, and the successor nodes are their operands. The leaf nodes contain atomic operands. Expression trees are used in programming language parsers and interpreters to represent the structure of expressions. A binary search tree is a type of binary tree in which each nonempty left subtree contains data that is less than or equal to the datum in its parent node, and each nonempty right subtree contains data that is greater than or equal to the datum in its parent node. If a binary search tree is nearly complete, it can support logarithmic searches and insertions. Also, a heap is a type of binary tree in which smaller data items are kept close to the root. A heap can be used to implement the n log n heap sort algorithm as well as a priority queue.

Apart from this, the twelfth chapter discussed about Graphs which have a wide range of applications. They are frequently used to represent networks of items that can be linked by various paths. Also, a graph is made up of one or more vertices (items) linked together by one or more edges. If there is an edge connecting two vertices, one is adjacent to the other. These two vertices are also referred to as neighbors. A path is a series of edges that connects one vertex to another in the graph. A vertex can be reached from another vertex if and only if a path exists between the two. The number of edges in a path determines its length. A graph is said to be connected if there is a path connecting each vertex to every other vertex. A graph is complete if an edge connects each vertex to every other vertex. A subgraph is made up of a subset of a graph's vertices and edges. A connected component is a subgraph made up of the vertices that can be reached from a given vertex. Furthermore, directed graphs allow only one-way travel along an edge, whereas undirected graphs allow two-way travel. Weights can be assigned to edges to indicate the cost of traveling along them and graph traversals investigate tree-like structures within a graph, beginning with a distinct start vertex. A depth-first traversal first visits all descendants on a given path, whereas a breadth-first traversal first visits all children of each vertex. A spanning tree has the fewest number of edges while still connecting all of the vertices in a graph. A minimum spanning tree is a spanning tree with the fewest weights possible on its edges. A topological sort produces a set of vertices in a directed acyclic graph. Furthermore, the single-source shortest-path problem requires a solution that includes the shortest paths from a given vertex to all other vertices.

**Conclusion**

A stack is an example of a linear data structure. Linear data structures are groups of components that are arranged in a straight line. Linear data structures grow and shrink as we add and remove components. A stack is formed by restricting the growth of a linear data structure so that new components can only be added or removed at one end. Stacks are useful data structures that can be used in many different ways in computer science. You already know that stacks are used to implement functions, and you've seen how each running program has a stack, as well as how the stack of a program grows and shrinks during function/procedure calls. This is especially important when it comes to comprehending how recursion works. Stacks are generally useful for processing nested structures or functions that call other functions (or themselves). A nested structure is one that can contain instances of itself within instances of itself. Algebraic expressions, for example, can be nested because a subexpression of one algebraic expression can be another algebraic expression. Functions, parsers, expression evaluation, and backtracking algorithms are all implemented using stacks.

Moreover, a queue is a first-in, first-out (FIFO) abstract data type that is widely used in computer programming. Queues can be used for anything where you want things to happen in the order, they were called but the computer can't keep up. Any queue of consumers for a resource in which the consumer who arrived first is served first is an example of a queue. The distinction between stacks and queues is in the removal process. Also, a double-ended queue, or deque, has the feature of adding and removing elements from either end. The Deque module is a part of collections library. It has the methods for adding and removing elements which can be invoked directly with arguments. In the below program we import the collections module and declare a deque.

In addition to this, trees are ADTs (Abstract Data Types) that use a hierarchical pattern for data allocation. A tree is essentially a collection of multiple nodes connected by edges. These 'trees' form a tree-like data structure, with the 'root' node leading to 'parent' nodes, which lead to 'children' nodes. The connections formed by lines are referred to as 'edges.' Endpoints with no children nodes are known as 'leaf' nodes. Because of their non-linear structure, trees play an important role in data structures. This results in a faster response time during a search and greater convenience during the design process.

In summary of the last discussion, A graph is an abstract data type composed of a finite set of vertices and a set of edges that represent links between the vertices. Graphs are used for a variety of purposes, including representing communication networks, data organization, computational devices, computation flow, and so on. It also need storage for your graph. There are several ways to represent a graph data structure, but you decide on a list that will store each person as a key and all of their connections as associated values.

* 1. **Python DS**: Chapters 7, 10: Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

**Observation**

Based on my observation on chapter 7, it discussed/ introduced the linked list as a data structure that can be used instead of an array-based sequence (such as a Python list). Array-based sequences and linked lists both keep elements in a specific order, but in very different ways. An array is a more centralized representation, with a single large chunk of memory capable of storing references to many elements. A linked list, on the other hand, uses a more distributed representation in which each element is assigned a lightweight object known as a node. Each node retains a reference to its element as well as one or more references to neighboring nodes in order to collectively represent the sequence's linear order. It also mentioned and discussed the singly linked list, circularly linked list, doubly linked list, and the positional list ADT. The positional list ADT is useful in a variety of contexts. A program that simulates a card game, for example, could model each player's hand as a positional list. Because most people keep cards of the same suit together, inserting and removing cards from a person's hand could be accomplished using positional list ADT methods, with the positions determined by a natural order of the suits. Similarly, the concept of positional insertion and deletion is embedded in a simple text editor because such editors typically perform all updates relative to a cursor, which represents the current position in the list of characters of text being edited. Furthermore, a Linked List is a collection of nodes that are arranged in an ordered fashion. Each node in the list contains one item and points to the next and previous node in the list. The list contains a link to the first and last node, and it can be traversed by beginning at one of these nodes and then following the links to more nodes.

Moreover, in the tenth chapter of Data Structures and Algorithms in Python is that I’ve noticed that it discussed and introduced the topics of Maps, Hash Tables, and Skip Lists which explains the dict class in Python is arguably the most important data structure in the language. It notable that represents a dictionary, which is an abstraction in which unique keys are mapped to associated values. Dictionaries are also known as associative arrays or maps because of the relationship they express between keys and values. In this book, we use the term dictionary to refer to Python's dict class, and the term map to refer to the more general concept of the abstract data type. I’ve also noticed that consider the problem of counting the number of occurrences of words in a document as an example of how to use a map because it can use words as keys and word counts as values, a map is an ideal data structure to use here. It disassembles the original document using a combination of file and string methods, resulting in a loop over all whitespace separated pieces of the document in lowercase. We omit all nonalphabetic characters so that parentheses, apostrophes, and other punctuation do not count as part of a word. Furthermore, it’s one of the most useful data structures for implementing a map, which is also used by Python's own implementation of the dict class. This is referred to as a hash table. it then pass the key-value pair to the hash table, which stores the data for later retrieval. One of the standard hash table operations is the use of a hash function to determine how the table should store the data. Also, The skip list is an intriguing data structure for realizing the sorted map ADT that a sorted array enables O(logn)-time binary search. Because of the need to shift elements, update operations on a sorted array take O(n) worst-case time. We demonstrated in Chapter 7 that linked lists can support very efficient update operations if the position within the list is known. Unfortunately, we cannot perform fast searches on a standard linked list; for example, the binary search algorithm requires an efficient means of directly accessing a sequence element by index.

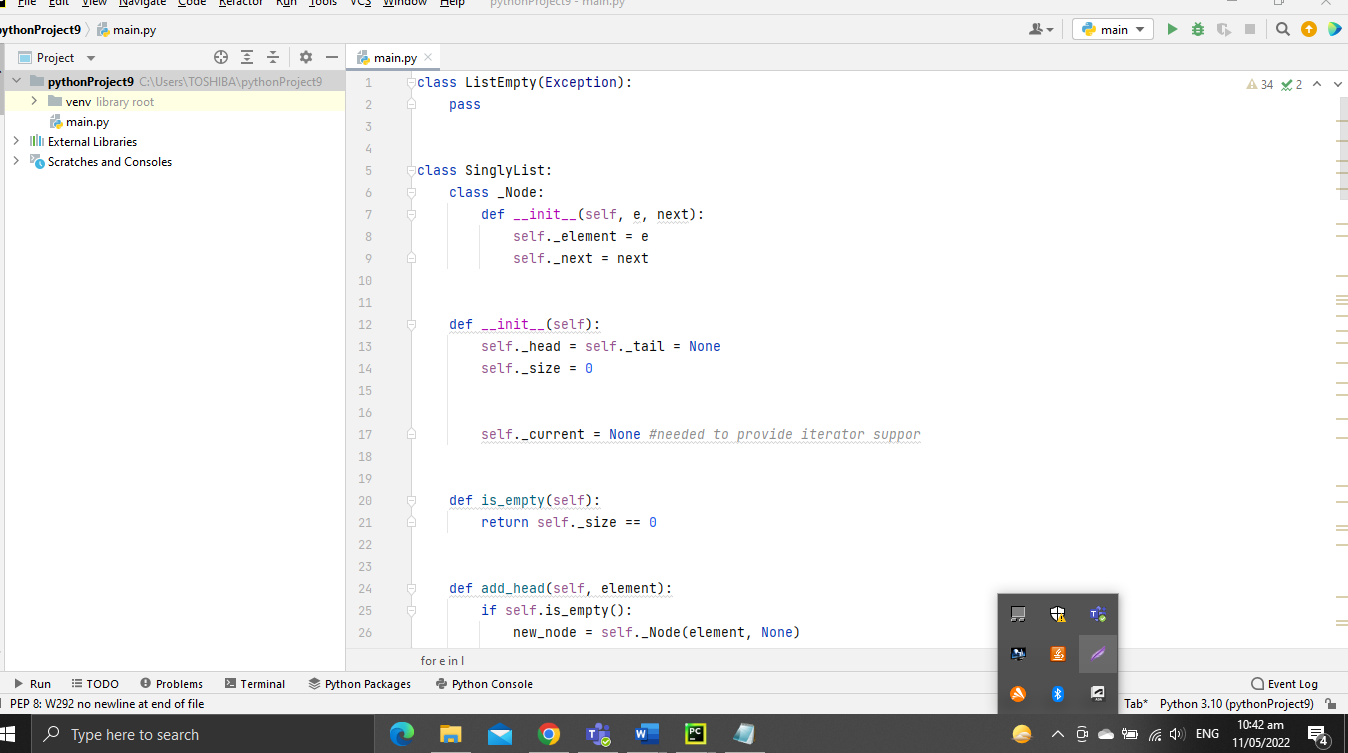
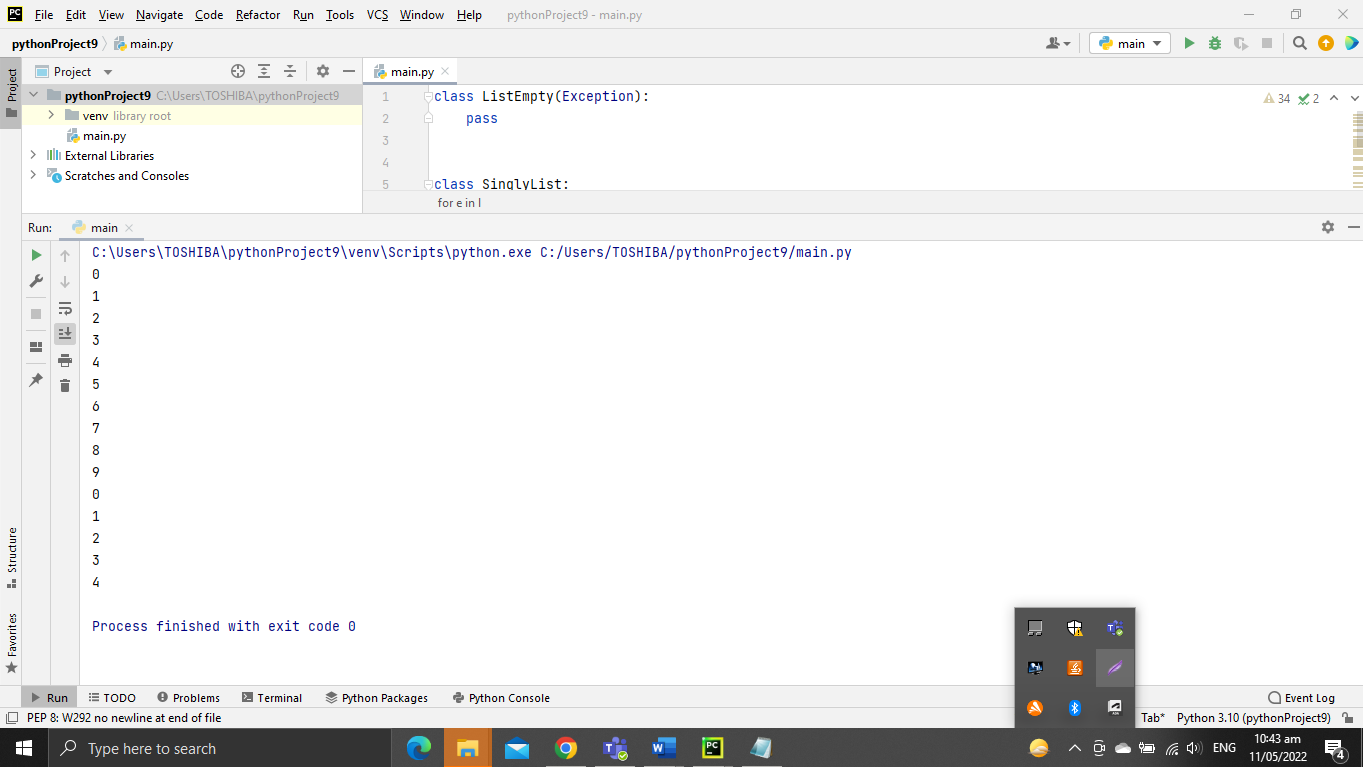
**Conclusion**

In conclusion at the chapter 7, Linked lists are linear data structures that store information in individual objects known as nodes. These nodes contain data as well as a reference to the next node in the list. Because of their ease of insertion and deletion, linked lists are frequently used. Stacks, queues, and other abstract data types can be implemented using them. Linked lists have a number of significant advantages over other linear data structures. Unlike arrays, they are a dynamic data structure that can be resized at runtime. Furthermore, the insertion and deletion operations are efficient and simple to implement.

Moreover, in the chapter 10- mapping can assist you in adjusting the range of values or preparing the values for specific types of analysis. Data mapping originated in functional languages, but it is now used in almost all programming languages that support first-class functions. Further, Hash tables have a significant impact due to their constant time complexity of O(1), which means they scale very well when used in algorithms. The linear time complexity of searching over a data structure such as an array is O. (n). Also, It enables the efficient viewing of elements or data. It skips several elements of the entire list in a single step, which is why it is called a skip list. The skip list is a more elaborate version of the linked list. It enables the user to quickly search for, remove, and insert the element.

Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al

**R-7.1** Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a next reference of None.



**R-7.2** Describe a good algorithm for concatenating two singly linked lists L and M, given only references to the first node of each list, into a single list L’ that contains all the nodes of L followed by all the nodes of M.

Algorithm concatenate(L,M):

Node n  L.getHead()

While (n.getNext()!=null)do

n  n.getNext()

n.setNext(M.getHead())

L’  L

Head tail

L ф ф

30

40

20

10

Head tail

M ф ф

30

40

20

10

* Doubly linked list
* Each node has two references, one for next and the other for previous.
* DLL has “header” and “trailer” nodes called dummy or sentinel nodes.
* An empty DLL has header and trailer only and its size is zero (not counting sentinel nodes).

Header Tailer

•

•

40

30

20

10

Ф

**R-7.3** Describe a recursive algorithm that counts the number of nodes in a singly linked list.

*# S = LinkedStack()  
# for i in range(5):  
# S.push(i)*def count\_nodes(S):  
 if len(S) != 0:  
 S.pop()  
 return 1 + count\_nodes(S)  
 else:  
 return 0  
*# print(count\_nodes(S))*

**R-7.4** Describe in detail how to swap two nodes x and y (and not just their contents) in a singly linked list L given references only to x and y. Repeat this exercise for the case when L is a doubly linked list. Which algorithm takes more time?

* **In Single Linked List**

N x y v

40

70

60

10

20

30

50

**ф**

Algorithm swap(x, y):

Node n  head

while( n.getNext() != x ) do

n  n.getNext()

Node v  y.getNext()

n.setNext(y)

y.setNext(x)

x.setNext(v)

**In Double Link List**

**N x y v**

•

•

60

50

40

30

20

10

ф

Algorithm swapDoubly(x, y):

DNode n  x.getPrev()

DNode v  y.getPrev()

n.setNext(y)

y.setPrev(n)

y.setNext(x)

x.setPrev(y)

x.setNext(v)

v.setPrev(x)

Swap in singly linked list take more time because we have to move from head to the node

before x.

1. **InLab**

* Write your Objectives (you can have your own objectives)

Objectives:

* To know how store a polynomial expression in memory using Single / Circular / Doubly Linked List.
* To know how to Implement List, Tree, etc.
* To know how to implements the concept of hashing using separate chaining.

Implementation of Single / Circular / Doubly Linked List

1. Write Python programs for the following operations on Single / Circular / Doubly Linked List.

**Creation**

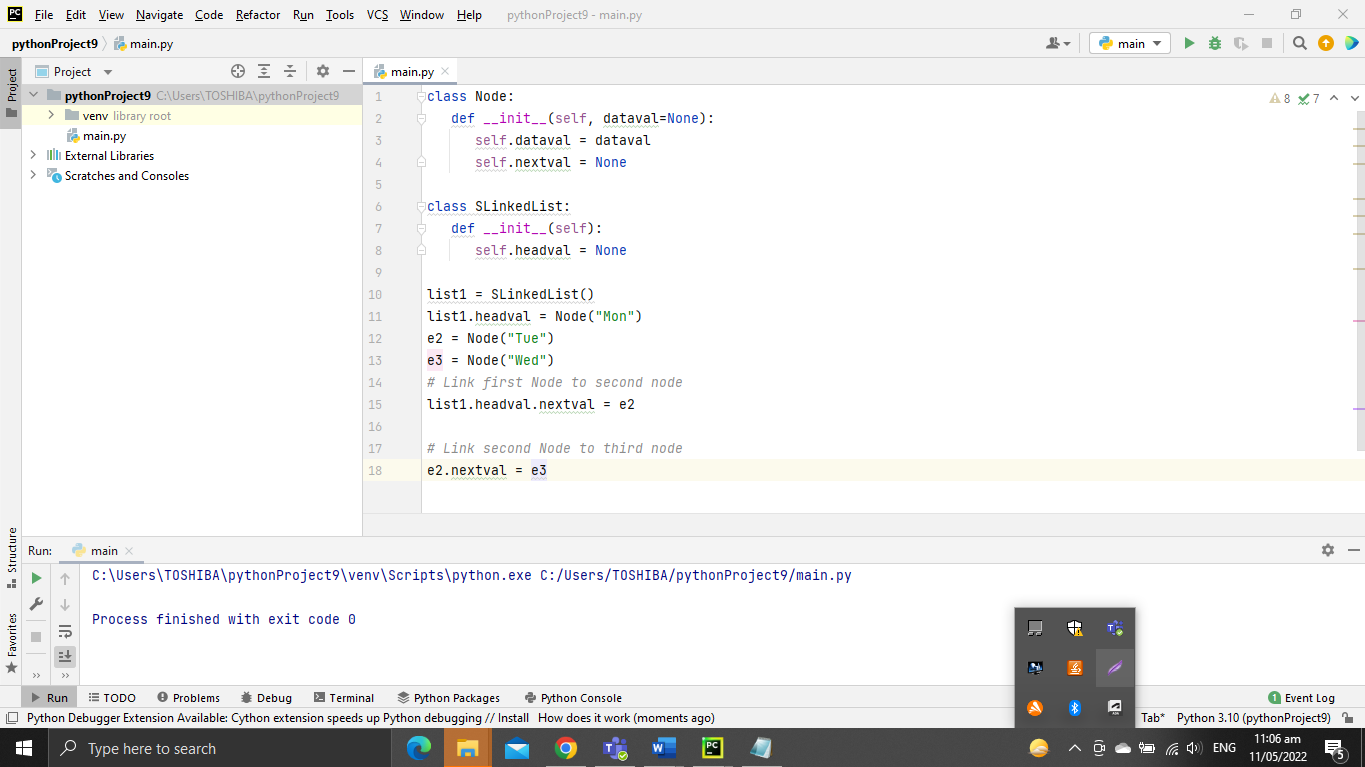


Figure 1

A linked list is created by using the node class we studied in the last chapter. We create a Node object and create another class to use this ode object. We pass the appropriate values through the node object to point the to the next data elements. The below program creates the linked list with three data elements. In the next section we will see how to traverse the linked list.

**Insertion**

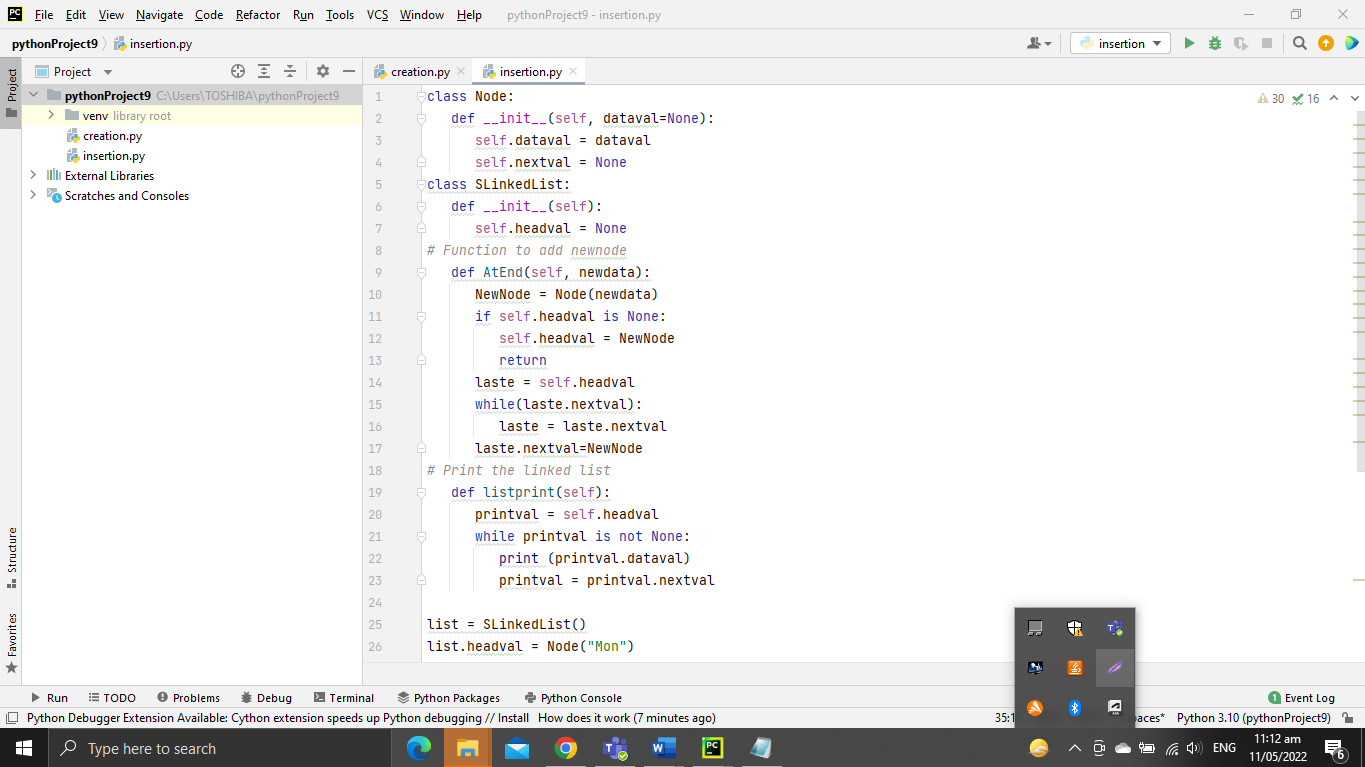


Figure 2

This involves pointing the next pointer of the the current last node of the linked list to the new data node. So the current last node of the linked list becomes the second last data node and the new node becomes the last node of the linked list.

**Deletion**

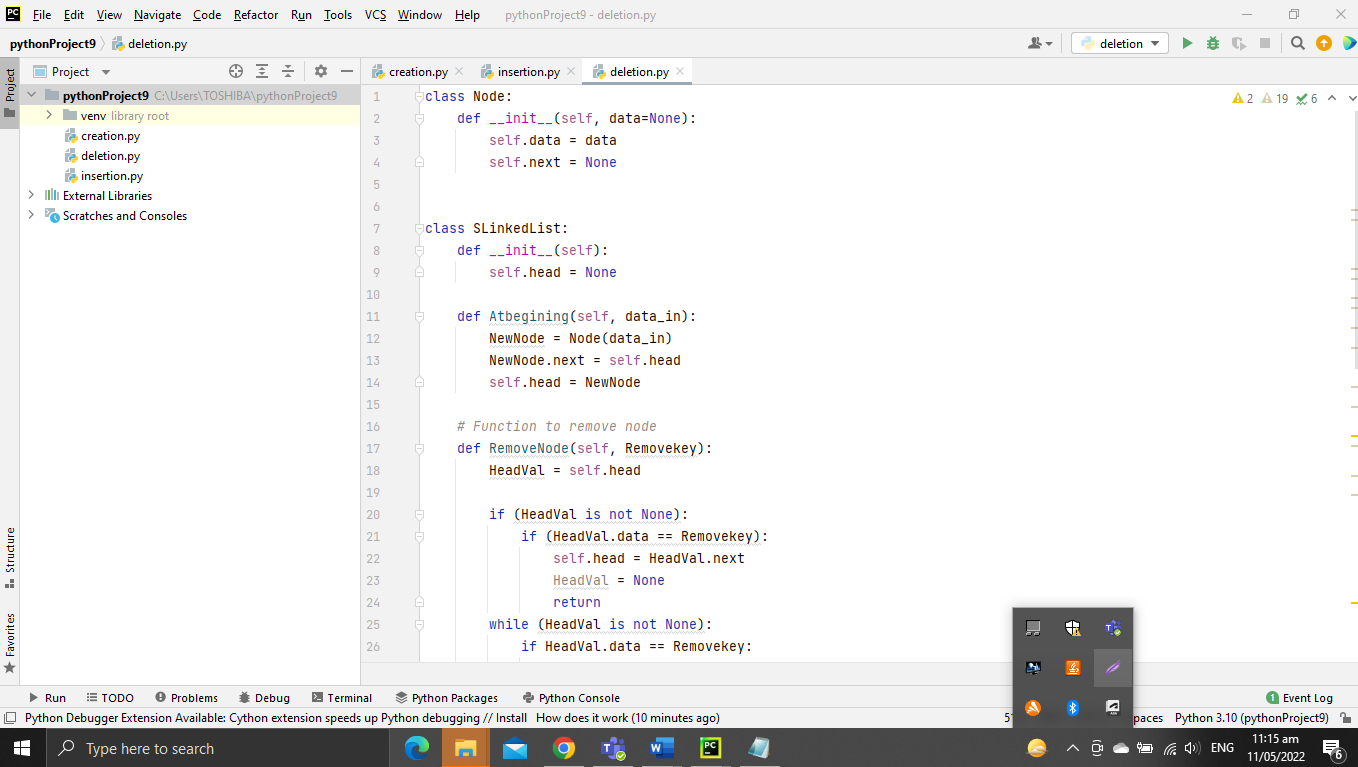


Figure 3

We can remove an existing node using the key for that node. In the below program we locate the previous node of the node which is to be deleted.Then, point the next pointer of this node to the next node of the node to be deleted.

**Traversal**

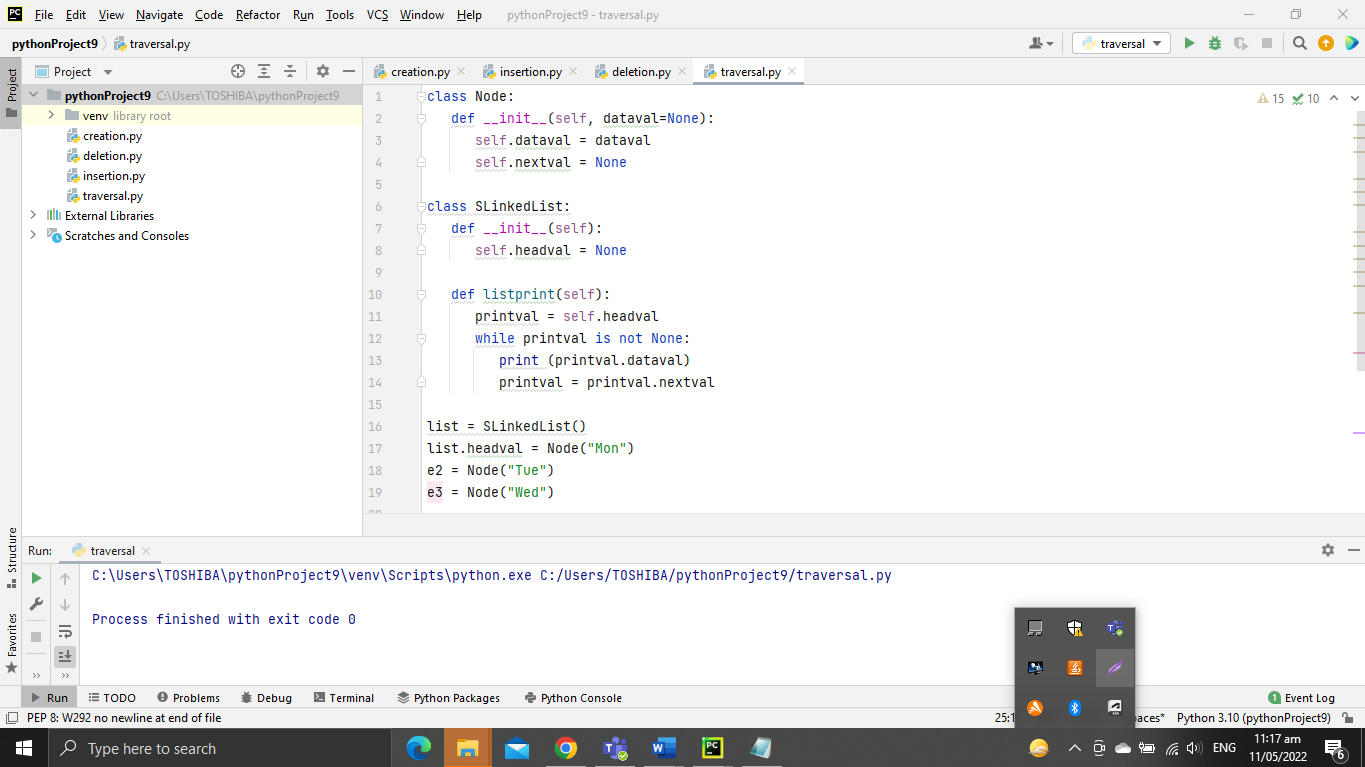


Figure 4

Singly linked lists can be traversed in only forward direction starting form the first data element. We simply print the value of the next data element by assigning the pointer of the next node to the current data element.

1. To store a polynomial expression in memory using Single / Circular / Doubly Linked List.

Implementation of Stack and Queue

1. Design and implement Stack and its operations using List.
2. Design and implement Queue and its operations using List

Implementation of Binary Search Tree

1. Create a binary search tree.

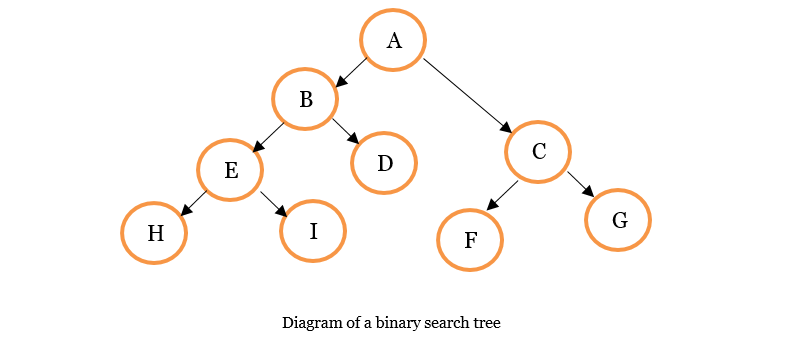


Figure 5

Let's us look at the relationship between the nodes.

* **A** is the root node.
* The left subtree begins at B while the right subtree begins at C.
* Node A has two child nodes – B and C.
* Node C is the parent node to F and G. F and G are siblings.
* Node F and G are know as leaf nodes because they do not have children.
* Node B is the parent node to D and E.
* Node D is the parent node to H and I.
* D and E are siblings as well as H and I.
* Node E is a leaf node.

So here are some important terms that we just used to describe the tree above:

**Root:** The topmost node in the tree.

**Parent:** A node with a child or children.

**Child:** A node extended from another node (parent node).

**Leaf:** A node without a child.

1. Traverse the above binary search tree recursively in pre-order, post-order and in-order.

PRE – ORDER

**How to Traverse a Tree Using Preorder Traversal**

The order here is Root, Left, Right.

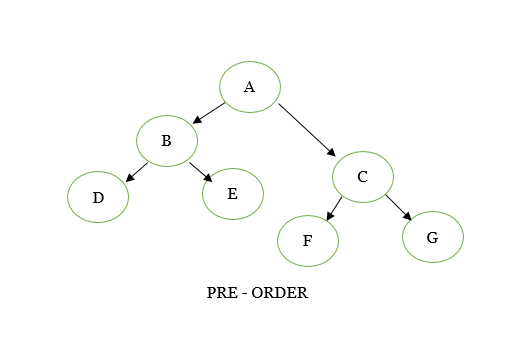


Figure 6

**How to Traverse a Tree Using Post - order Traversal**

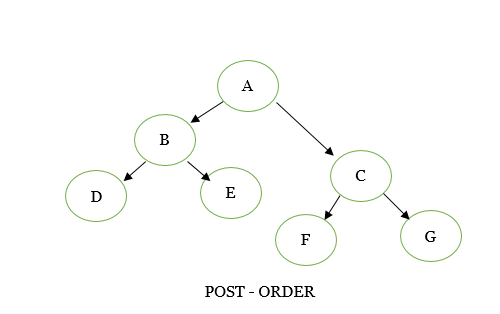
****

Figure 7

**How to Traverse a Tree Using In- order Traversal**

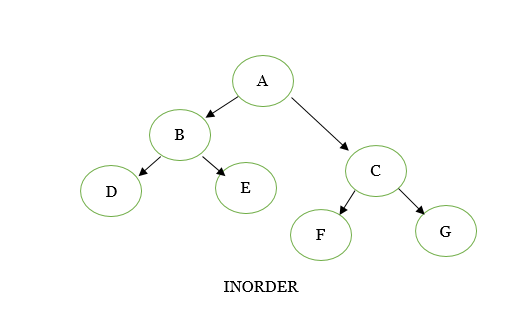


Figure 8

1. Count the number of nodes in the binary search tree.

* A is the root node.
* The left subtree begins at B while the right subtree begins at C.
* Node A has two child nodes – B and C.
* Node C is the parent node to F and G. F and G are siblings.
* Node F and G are know as leaf nodes because they do not have children.
* Node B is the parent node to D and E.
* Node D is the parent node to H and I.
* D and E are siblings as well as H and I.
* Node E is a leaf node.

Implementation of Traversal Algorithm for Breadth first traversal

1. Create a traversal algorithm for Breadth first traversal

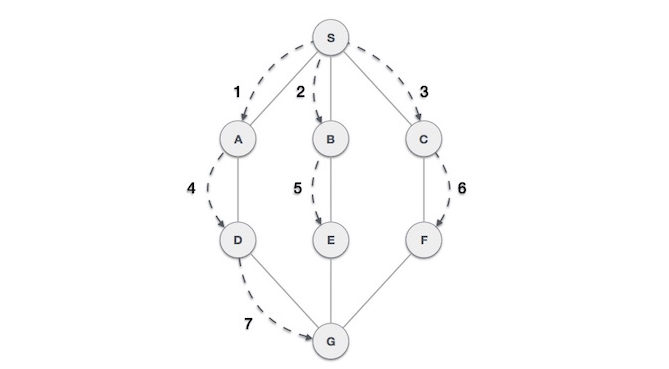


Figure 9

Breadth First Search (BFS) algorithm traverses a graph in a breadthward motion and uses a queue to remember to get the next vertex to start a search, when a dead end occurs in any iteration.

As in the example given above, BFS algorithm traverses from A to B to E to F first then to C and G lastly to D. It employs the following rules.

* Rule 1 − Visit the adjacent unvisited vertex. Mark it as visited. Display it. Insert it in a queue.
* Rule 2 − If no adjacent vertex is found, remove the first vertex from the queue.
* Rule 3 − Repeat Rule 1 and Rule 2 until the queue is empty.

Implementation of Traversal Algorithm for Breadth first traversal

1. Create a program that implements the concept of hashing using separate chaining.

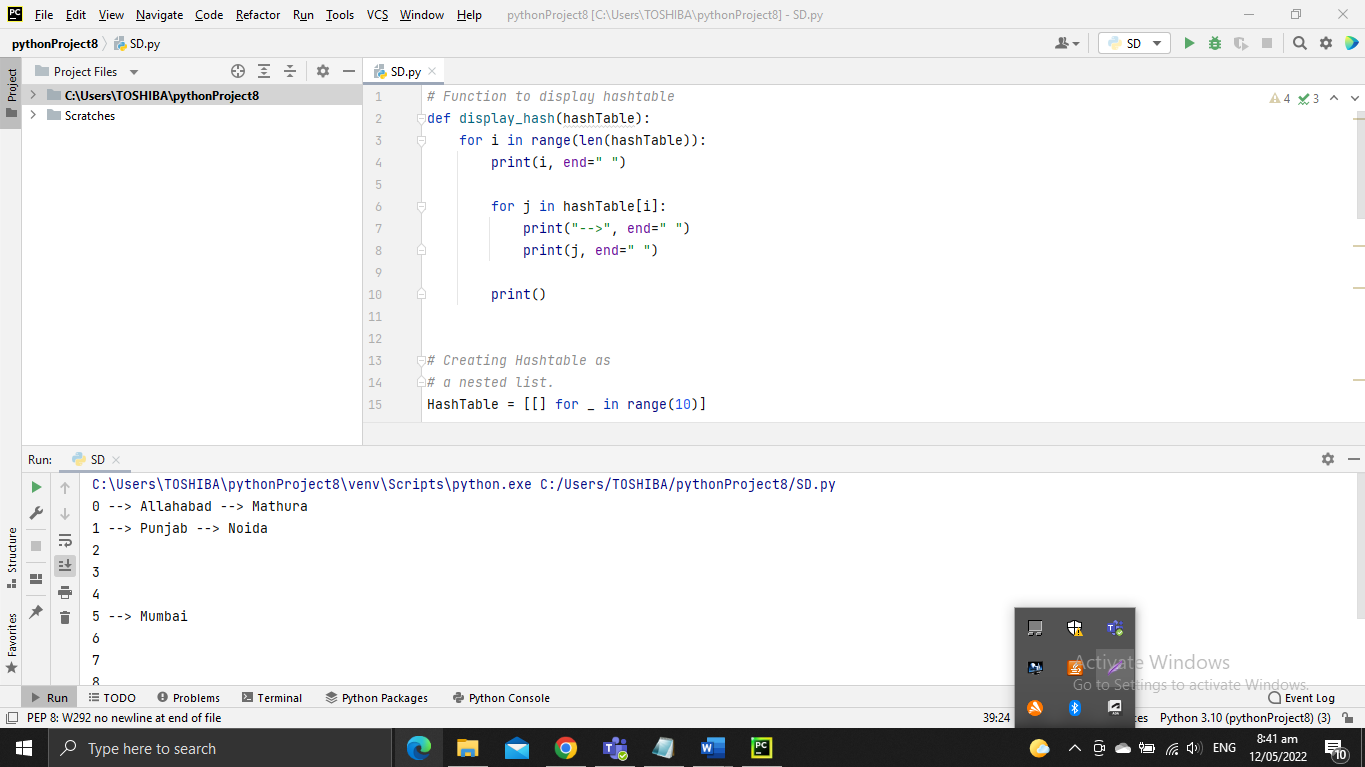


Figure 10

The performance of hashing is evaluated on the basis that each key is equally likely to be hashed for any slot of the hash table.

* Steps performed with screenshots of tools used, sample run with

**DISCUSSIONS** (DONT copy and paste from the e-book)**.**

* Edit your figures (screenshot), highlight by putting a box, give a figure number and brief description.

**Write your own Observation and Conclusion from what you have executed/read.**

**Observation**

The basis of our approach for analyzing the performance of algorithms is the scientific method. We begin by performing computational experiments to measure the running times of our programs. We use these measurements to develop hypotheses about performance. Next, we create mathematical models to explain their behavior. Finally, we consider analyzing the memory usage of our Java programs.

**Conclusion**

A conceptual design algorithm of the hybrid RD column is developed to obtain the desired selectivity design in case of a series reaction scheme with a pseudo-homogeneous concentration-based kinetic model. Presented methodology generates multiple feasible designs which is reliant on feed location, number of moles involved in the reaction, volumetric flow rate, number of stages etc., and can be accessed in terms of operating and capital cost to determine a good/optimal design for the given set of design goals. It is applicable to single feed configurations with reactant(s)/intermediate boiling in residue curve map. The developed method provides good initialization for rigorous simulation and optimization for industrially relevant multi reaction systems.

1. **PostLab**

* **Project**

**Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al: Chapter 7 Projects page 297**

**Do Projects P-7.44, P-7.45, P-7.46**

* **Debugging and Sample Run** (with screenshots and Discussion)



# Write your own Observation and Conclusion from what you have executed/read.

**Observation**

I conclude that When many individual operations are part of a larger computation, and we only care about the total time of that computation, an amortized bound is as good as a worst-case bound precisely because it gives a guarantee on the sum of the time spent on the individual operations. However, if data structure operations are used in a real-time system that is de-signed to provide more immediate responses (e.g., an operating system, Webserver, air traffic control system), a long delay caused by a single (amortized)operation may have an adverse effect.

**Conclusion**

Link-based structures support O(1)-time insertions and deletions at arbitrary positions. The ability to perform a constant-time insertion or deletion with the Positional List class, by using a Position to efficiently describe the location of the operation, is perhaps the most significant advantage of the linked list. This is in stark contrast to an array-based sequence. Ignoring the issue of resizing an array, inserting or deleting an element from the end of an array-based list can be done in constant time. However, more general insertions and deletions are expensive. For example, with Python’s array-based list class, a call to insert or pop with index k uses O(n−k+1) time because of the loop to shift all subsequent elements (see Section 5.4). As an example application, consider a text editor that maintains a document as a sequence of characters. Although users often add characters to the end of the document, it is also possible to use the cursor to insert or delete one or more characters at an arbitrary position within the document. If the character sequence were stored in an array-based sequence (such as a Python list), each such edit operation may require linearly many characters to be shifted, leading to O(n)performance for each edit operation. With a linked-list rep-resentation, an arbitrary edit operation (insertion or deletion of a character at the cursor) can be performed in O(1)worst-case time, assuming we are given a position that represents the location of the cursor.

**Note:**

1. Save all files (.py) in one folder.

Name the folder: <Course and Section>\_LastName,FirstName\_LAB REPORT#X. Example: **SOFTWARE DESIGN-2A\_PENTECOSTES,JAY-AR\_LAB REPORT#3.**

1. Commit all Python source codes to Github **(individual Github account**) Github Repository Name: Software Design Lab Exercises and put a URL of your Github repository.