```python
"""
    Author: Steven Ebreo
    Implementation of several abstract data types as a module
"""
class Node:
    def __init__(self, value):
        self.value = value
        self.next = None

class BinaryNode:
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self, value):
        self.root = BinaryNode(value)

    def inorder(self, node, l:list):
        if node:
            self.inorder(node.left, l)
            l.append(node.value)
            self.inorder(node.right, l)

    def preorder(self, node, l:list):
        if node:
            l.append(node.value)
            self.preorder(node.left, l)
            self.preorder(node.right, l)

    def postorder(self, node, l:list):
        if node:
            self.postorder(node.left, l)
            self.postorder(node.right,l)
            l.append(node.value)


    def insert(self, root, value):
        if root == None:
            return BinaryNode(value)
        elif value > root.value:
            root.right = self.insert(root.right, value)
        elif value < root.value:
            root.left = self.insert(root.left, value)
        return root

class LinkedList:
    def __init__(self, value):
        self.head = Node(value)

    def to_list(self) -> list:
        l = []
        current_node = self.head
        while current_node:
            l.append(str(current_node.value))
            current_node = current_node.next
        return l

    def search(self, value) -> Node:
        current_node = self.head
        while current_node:
            if current_node.value == value:
                return current_node
            current_node = current_node.next
        raise ValueError("Value not found in list")

    def insert_after(self, first_value, insert_value) -> bool:
        specific_node = self.search(first_value)
        if specific_node:
            inserted_node = Node(insert_value)
            inserted_node.next = specific_node.next
            specific_node.next = inserted_node
            return True
        else:
            return False

class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, value):
        self.queue.append(value)

    def dequeue(self):
        if self.is_empty():
            return "Queue is empty!"
        return self.queue.pop(0)

    def peek(self):
        if self.is_empty():
            return "Queue is empty!"
        return self.queue[0]

    def is_empty(self) -> bool:
        return self.get_len() == 0

    def get_len(self) -> int:
        return len(self.queue)
```

array.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Array
    Create an array of 5 student names
    Add one new student
    Remove one student
    Print updated list
"""

def main():
    students = ["Harry", "Hermione", "Ron", "Neville", "Draco"]
    print("Before update:")
    print(*students, sep=", ")

    # Inserts onto array and prints
    students.append("Luna")
    print("After insertion: ")
    print(*students, sep=", ")

    # Removes a student
    students.remove("Draco")
    print("After deletion: ")
    print(*students, sep=", ")

if __name__ == "__main__":
    main()
```

binarytree.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Binary Tree
    Create a binary tree with at least 7 nodes.
    Implement inorder, preorder, and postorder traversals
"""

from adt import BinaryTree

def main():
    tree = BinaryTree(50)
    root = tree.root
    values = [30, 70, 20, 40, 60, 80]
    for value in values:
        tree.insert(root, value)

    preorder_list = []
    tree.preorder(root, preorder_list)
    print("Preorder Traversal: ", end="")
    print(*preorder_list, sep=", ")

    inorder_list = []
    tree.inorder(root, inorder_list)
    print("Inorder Traversal: ", end="")
    print(*inorder_list, sep=", ")

    postorder_list = []
    tree.postorder(root, postorder_list)
    print("Postorder Traversal: ", end="")
    print(*postorder_list, sep=", ")

if __name__ == "__main__":
    main()
```

graph.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Graph
    Represent a friendship network using a graph
    Implement BFS and DFS traversals to explore the network
"""

def main():
    friendship_graph = {
        'Steven': ['Skealla', 'Faye', 'Oxy', 'Jup(iter)'],
        'Skealla': ['Steven', 'Oxy', 'Ralph'],
        'Ralph' : ['Skealla'],
        'Faye': ['Steven', 'Geanne'],
        'Geanne': ['Faye'],
        'Oxy': ['Steven', 'Jup(iter)'],
        'Jup(iter)': ['Steven', 'Oxy']
    }

    bfs_list = []
    bfs(friendship_graph, "Steven", bfs_list)
    print("Breadth First Search: ", end="")
    print(*bfs_list, sep=", ")

    dfs_list = []
    dfs(friendship_graph, "Steven", dfs_list)
    print("Depth First Search: ", end="")
    print(*dfs_list, sep=", ")

def bfs(graph:dict, start:str, l:list):
    visited = []
    queue = [start]

    while queue:
        node = queue.pop(0)
        if node not in visited:
            l.append(node)
            visited.append(node)
            queue.extend(graph[node])

def dfs(graph:dict, start:str, l:list):
    visited = []
    stack = [start]

    while stack:
        node = stack.pop()
        if node not in visited:
            l.append(node)
            visited.append(node)
            stack.extend(graph[node])


if __name__ == "__main__":
    main()
```

linkedlist.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Linked List
    Build a linked list of 3 integers, and display
    Insert a new number, and then display
"""
from adt import LinkedList

def main():
    sllist = LinkedList(1871)
    try:
        sllist.insert_after(1871, 1917)
        sllist.insert_after(1917, 1945)
    except False:
        raise AttributeError("Nonexistent insertion position")

    list = sllist.to_list()
    print(f"Initial list: {", ".join(list)}")

    # Insert a new number in between
    sllist.insert_after(1917, 1941)
    new_list = sllist.to_list()
    print(f"Mutated list: {", ".join(new_list)}")

if __name__ == "__main__":
    main()
```

queue.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Queue
    Simulate a line of people waiting:
    Add 3 names then remove the first one
"""

from adt import Queue

def main():
    queue = Queue()
    print("Queue is now open!")
    print("Current Queue: ", end="")

    queue.enqueue("Steven")
    queue.enqueue("Gea")
    queue.enqueue("Faye")

    for i in range(queue.get_len()):
        print(queue.queue[i], end=" ")

    print(f"\n{queue.dequeue()} has exited the queue!")

    print("New queue: ", end="")
    for i in range(queue.get_len()):
        print(queue.queue[i], end=" ")
    print()

if __name__ == "__main__":
    main()
```

stack.py

```python
"""
    Author: Steven Ebreo
    Problem Set 1 : Stack
    Simulate an "Undo" feature, pushing 3 actions then popping 1
"""

def main():
    stack = []

    # Simulation of searching using a keyboard
    search = "Search: "
    skibidi = "Skibidi"
    stack.append(skibidi)

    print(join_strlist(search, stack))

    toilet = "Toilet"
    stack.append(toilet)

    print(join_strlist(search, stack))

    ohio = "Ohio"
    stack.append(ohio)

    print(join_strlist(search, stack))

    print(f"CTRL + Z: {stack.pop()}")

    print(join_strlist(search, stack))

def join_strlist(header:str, prompt:list) -> str:
    return header + " ".join(prompt)


if __name__ == "__main__":
    main()
```