Double-click (or enter) to edit

1. Write a Python program to find a target values in a list using linear search with following steps:

a. Initialize the list to store the input elements.

b. Initialize found-False.

c. Enter the item to be searched (match_item).

d. For each element in the list

```
 1. if match item = value
```

```
 a. return match item's position.
```

e. If the match item is not in the list, display an error message that the item is not found in the list.

```
def linear_search(input_list, match_item):
    found = False

    for index, value in enumerate(input_list):
        if value == match_item:
            found = True
            return f"{match_item} found at position {index + 1}"

    if not found:
        return f"{match_item} not found in the list"

input_list = [10, 25, 5, 15, 30, 20]
match_item = int(input("Enter the item to search for: "))

result = linear_search(input_list, match_item)
print(result)
```

```
Enter the item to search for: 5
5 found at position 3
```

2. Write a Python program to implement binary search to

find the target values from the list:

a. Create a separate function to do binary search.

b. Get the number of inputs from the user.

c. Store the inputs individually in a list.

d. In binary search function at first sort the list in order to start the search from middle of the list.

e. Compare the middle element to right and left elements to search target element.

f. If greater, move to right of list or else move to another side of the list.

g. Print the result along with the position of the element.

```
def binary_search(arr, target):
    arr.sort()
    left = 0
    right = len(arr) - 1

    while left <= right:
        mid = (left + right) // 2

        if arr[mid] == target:
            return f"{target} found at position {mid + 1}"
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return f"{target} not found in the list"

num_inputs = int(input("Enter the number of elements: "))

input_list = []
for i in range(num_inputs):
    element = int(input(f"Enter element {i + 1}: "))
    input_list.append(element)

target_value = int(input("Enter the target value to search for: "))

result = binary_search(input_list, target_value)
print(result)
```

```
Enter the number of elements: 4
Enter element 1: 11
Enter element 2: 22
Enter element 3: 33
Enter element 4: 44
Enter element 4: 44
Enter the target value to search for: 22
22 found at position 2
```

6. Write a Python script to perform the following operations on a singly linked list

a. Create a list

b. Find the smallest element from the list

C. Insert an element if it is not a duplicate element

d. Display the elements in reverse order

```
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def insert(self, data):
        new_node = Node(data)
        if self.head is None:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def find_smallest(self):
        if self.head is None:
            return None
        current = self.head
        smallest = current.data
        while current:
            if current.data < smallest:
                smallest = current.data
            current = current.next
        return smallest

    def insert_unique(self, data):
        current = self.head
        while current:
            if current.data == data:
                print(f"{data} is a duplicate element and will not be inserted.")
                return
            current = current.next
        self.insert(data)

    def display_reverse(self):
        if self.head is None:
            print("The list is empty.")
            return
        stack = []
        current = self.head
        while current:
            stack.append(current.data)
            current = current.next
        while stack:
            print(stack.pop(), end=" -> ")
        print("None")

my_list = LinkedList()

my_list.insert(10)
my_list.insert(5)
my_list.insert(15)
my_list.insert(2)
my_list.insert(8)

smallest_element = my_list.find_smallest()
print(f"The smallest element in the list is: {smallest_element}")

my_list.insert_unique(5)
my_list.insert_unique(20)

print("Elements in reverse order:")
my_list.display_reverse()
```

```
The smallest element in the list is: 2
5 is a duplicate element and will not be inserted.
Elements in reverse order:
20 -> 8 -> 2 -> 15 -> 5 -> 10 -> None
```

7.Write a python program to implement the various operations for Stack ADT i.) Push ii.) Pop iii.) Display.

```
class Stack:
    def __init__(self):
```

```
    def __init__(self):
        self.items = []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        if not self.is_empty():
            return self.items.pop()
        else:
            print("Stack is empty. Cannot pop.")
            return None

    def display(self):
        if not self.is_empty():
            print("Stack elements:")
            for item in reversed(self.items):
                print(item)
        else:
            print("Stack is empty.")

    def is_empty(self):
        return len(self.items) == 0

my_stack = Stack()

my_stack.push(10)
my_stack.push(20)
my_stack.push(30)
my_stack.push(40)

my_stack.display()

popped_item = my_stack.pop()
if popped_item is not None:
    print(f"Popped item: {popped_item}")

my_stack.display()
```

```
Stack elements:
40
30
20
10
Popped item: 40
Stack elements:
30
20
10
```

8. Write a python script to implement the various

8. Write a python script to implement the various operations for Queue ADT i.) Insert ii.) Delete iii.) Display.

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

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)
        else:
            print("Queue is empty. Cannot dequeue.")
            return None

    def display(self):
        if not self.is_empty():
            print("Queue elements:")
            for item in self.items:
                print(item)
        else:
            print("Queue is empty.")

    def is_empty(self):
        return len(self.items) == 0

my_queue = Queue()

my_queue.enqueue(10)
my_queue.enqueue(20)
my_queue.enqueue(30)
my_queue.enqueue(40)

my_queue.display()

dequeued_item = my_queue.dequeue()
if dequeued_item is not None:
    print(f"Dequeued item: {dequeued_item}")

my_queue.display()
```

```
Queue elements:
10
20
30
40
```

```
40
Dequeued item: 10
Queue elements:
20
30
40
```

9. Write a program in python to convert the following infix expression to its postfix form using push and pop operations of a Stack

a. A/B^C+D E-F*G

b. (B^2-4AC)^(1/2) (100)

```
def infix_to_postfix(infix_expression):

    stack = []
    postfix_expression = ""

    for token in infix_expression:
        if token in ["A", "B", "C", "D", "E", "F", "G"]:
            postfix_expression += token
        elif token == "+" or token == "-":
            while stack and stack[-1] != "(":
                postfix_expression += stack.pop()
            stack.append(token)
        elif token == "*" or token == "/":
            while stack and stack[-1] != "(" and (stack[-1] == "*" or stack[-1] == "/"):
                postfix_expression += stack.pop()
            stack.append(token)
        elif token == "(":
            stack.append(token)
        elif token == ")":
            while stack and stack[-1] != "(":
                postfix_expression += stack.pop()
            stack.pop()

    while stack:
        postfix_expression += stack.pop()

    return postfix_expression

infix_expression_a = "A/B^C+D E-F*G"
postfix_expression_a = infix_to_postfix(infix_expression_a)

infix_expression_b = "(B^2-4*A*C)^(1/2) (100)"
infix_expression_b = "(B^2 - A*C)/(1/2)/(100)"
postfix_expression_b = infix_to_postfix(infix_expression_b)

print(postfix_expression_a)
print(postfix_expression_b)
```

```
ABC/DE+FG*-
BA*C*-/
```