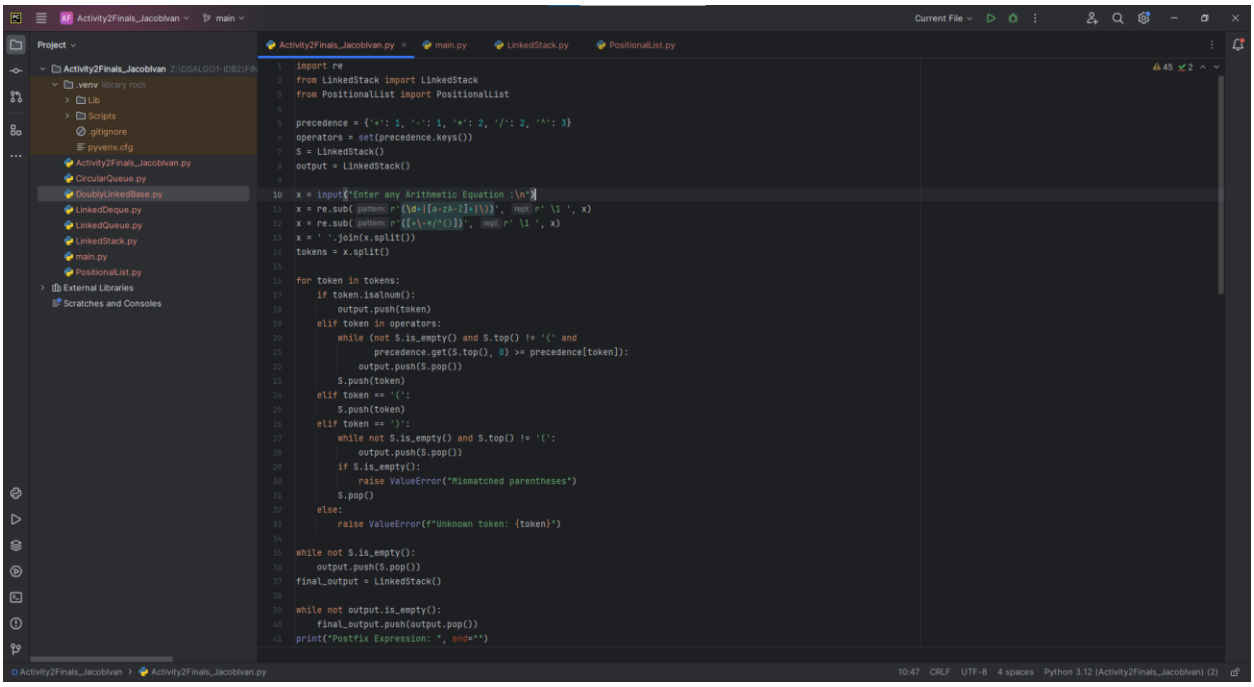
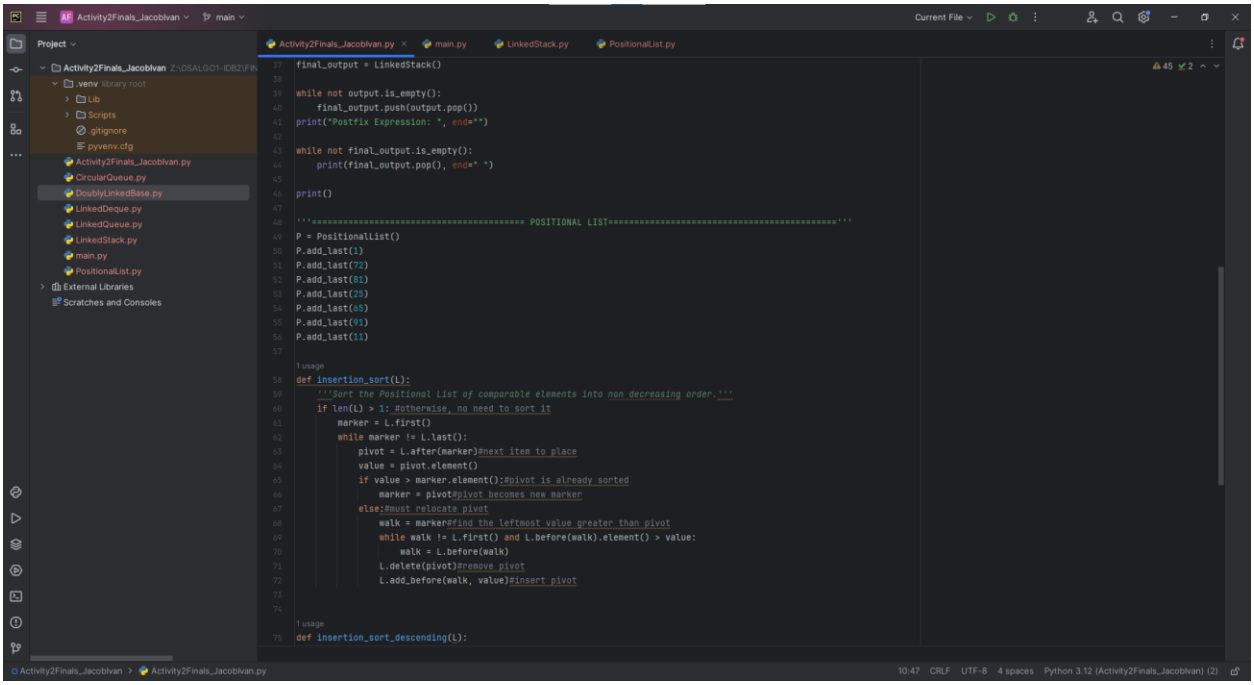
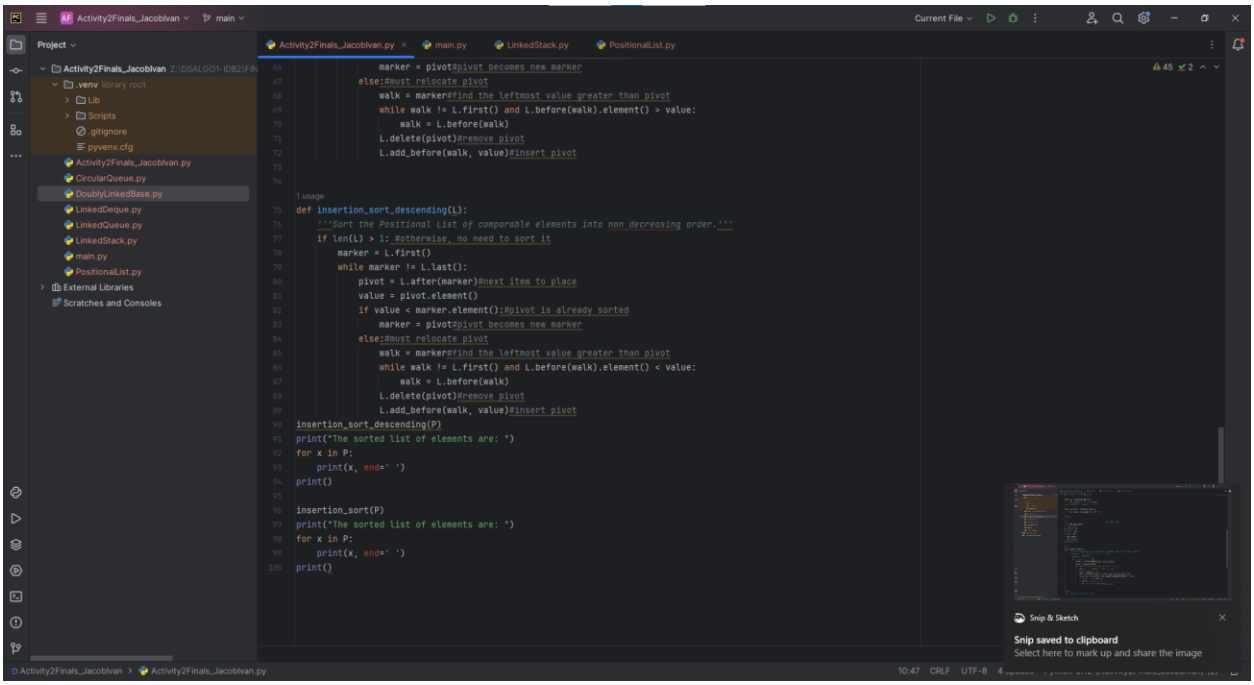
CODE:



```
import re
from LinkedStack import LinkedStack
from PositionalList import PositionalList

precedence = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
operators = set(precedence.keys())
S = LinkedStack()
output = LinkedStack()

x = input("Enter any Arithmetic Equation :\n")
x = re.sub( pattern: r'(\d+|[a-zA-Z]+|\))', repl: r' \1 ', x)
x = re.sub( pattern: r'([+\-*/^()])', repl: r' \1 ', x)
x = ' '.join(x.split())
tokens = x.split()

for token in tokens:
    if token.isalnum():
        output.push(token)
    elif token in operators:
        while (not S.is_empty() and S.top() != '(' and
               precedence.get(S.top(), 0) >= precedence[token]):
            output.push(S.pop())
        S.push(token)
    elif token == '(':
        S.push(token)
    elif token == ')':
        while not S.is_empty() and S.top() != '(':
            output.push(S.pop())
        if S.is_empty():
            raise ValueError("Mismatched parentheses")
        S.pop()
    else:
        raise ValueError(f"Unknown token: {token}")

while not S.is_empty():
    output.push(S.pop())
final_output = LinkedStack()

while not output.is_empty():
    final_output.push(output.pop())
print("Postfix Expression: ", end="")
```



```
final_output = LinkedStack()

while not output.is_empty():
    final_output.push(output.pop())
print("Postfix Expression: ", end="")

while not final_output.is_empty():
    print(final_output.pop(), end=" ")

print()

'''======================================== POSITIONAL LIST========================================'''
P = PositionalList()
P.add_last(1)
P.add_last(72)
P.add_last(81)
P.add_last(25)
P.add_last(65)
P.add_last(91)
P.add_last(11)

1 usage
def insertion_sort(L):
    '''Sort the Positional List of comparable elements into non decreasing order.'''
    if len(L) > 1: #otherwise, no need to sort it
        marker = L.first()
        while marker != L.last():
            pivot = L.after(marker)#next item to place
            value = pivot.element()
            if value > marker.element():#pivot is already sorted
                marker = pivot#pivot becomes new marker
            else:#must relocate pivot
                walk = marker#find the leftmost value greater than pivot
                while walk != L.first() and L.before(walk).element() > value:
                    walk = L.before(walk)
                L.delete(pivot)#remove pivot
                L.add_before(walk, value)#insert pivot


1 usage
def insertion_sort_descending(L):
```


Activity2Finals_Jacoblvan
main
Current File
Project
Activity2Finals_Jacoblvan Z:\DSALGO1-IDB2\FIN
.venv library root
Lib
Scripts
.gitignore
pyvenv.cfg
Activity2Finals_Jacoblvan.py
CircularQueue.py
DoublyLinkedBase.py
LinkedDeque.py
LinkedQueue.py
LinkedStack.py
main.py
PositionalList.py
External Libraries
Scratches and Consoles
Activity2Finals_Jacoblvan.py
main.py
LinkedStack.py
PositionalList.py
marker = pivot#pivot becomes new marker
else:#must relocate pivot
walk = marker#find the leftmost value greater than pivot
while walk != L.first() and L.before(walk).element() > value:
walk = L.before(walk)
L.delete(pivot)#remove pivot
L.add_before(walk, value)#insert pivot
1 usage
def insertion_sort_descending(L):
'''Sort the Positional List of comparable elements into non decreasing order.'''
if len(L) > 1: #otherwise, no need to sort it
marker = L.first()
while marker != L.last():
pivot = L.after(marker)#next item to place
value = pivot.element()
if value < marker.element():#pivot is already sorted
marker = pivot#pivot becomes new marker
else:#must relocate pivot
walk = marker#find the leftmost value greater than pivot
while walk != L.first() and L.before(walk).element() < value:
walk = L.before(walk)
L.delete(pivot)#remove pivot
L.add_before(walk, value)#insert pivot
insertion_sort_descending(P)
print("The sorted list of elements are: ")
for x in P:
print(x, end=' ')
print()
insertion_sort(P)
print("The sorted list of elements are: ")
for x in P:
print(x, end=' ')
print()
Snip & Sketch
Snip saved to clipboard
Select here to mark up and share the image
Activity2Finals_Jacoblvan
Activity2Finals_Jacoblvan.py
10:47
CRLF
UTF-8

Activity2Finals_Jacoblvan.py | main.py | LinkedStack.py | PositionalList.py

```python
class LinkedStack:
    '''LIFO STack implementation using a singly linked list for storage.'''

    #---------------- nested _Node class ----------------
    class _Node:
        '''Lightweight non public class for storing a singly linked node.'''
        __slots__ = '_element', '_next' #streamline memory usage

        def __init__(self, element, next):
            self._element = element
            self._next = next

    #---------------- stack methods ----------------
    def __init__(self):
        '''Create an empty Stack'''
        self._head = None
        self._size = 0
    def __len__(self):
        '''Return the number of elements in the stack'''
        return self._size
    8 usages
    def is_empty(self):
        '''Return True if the stack is empty.'''
        return self._size == 0

    7 usages
    def push(self, e):
        '''Add element e to the top of the stack.'''
        self._head = self._Node(e, self._head)
        self._size += 1
    3 usages
    def top(self):
        '''Return but do not remove the element at the top of the stack'''
        '''Raise empty exception if the stack is empty!'''
        if self.is_empty():
            raise Exception('Stack is empty')
        return self._head._element #top of the stack is the head of the list
    6 usages
    def pop(self):
        '''Remove and return the elements fro mthe top of the stack (LIFO)'''
        '''Raise Empty exception if the stack is empty!'''
```

```python
    def pop(self):
        '''Remove and return the elements fro mthe top of the stack (LIFO)'''
        '''Raise Empty exception if the stack is empty!'''
        if self.is_empty():
            raise Exception("The stack is empty!")
        answer = self._head._element
        self._head = self._head._next
        self._size -=1
        return answer
```

Activity2Finals_Jacobivan.py | main.py | LinkedStack.py | PositionalList.py

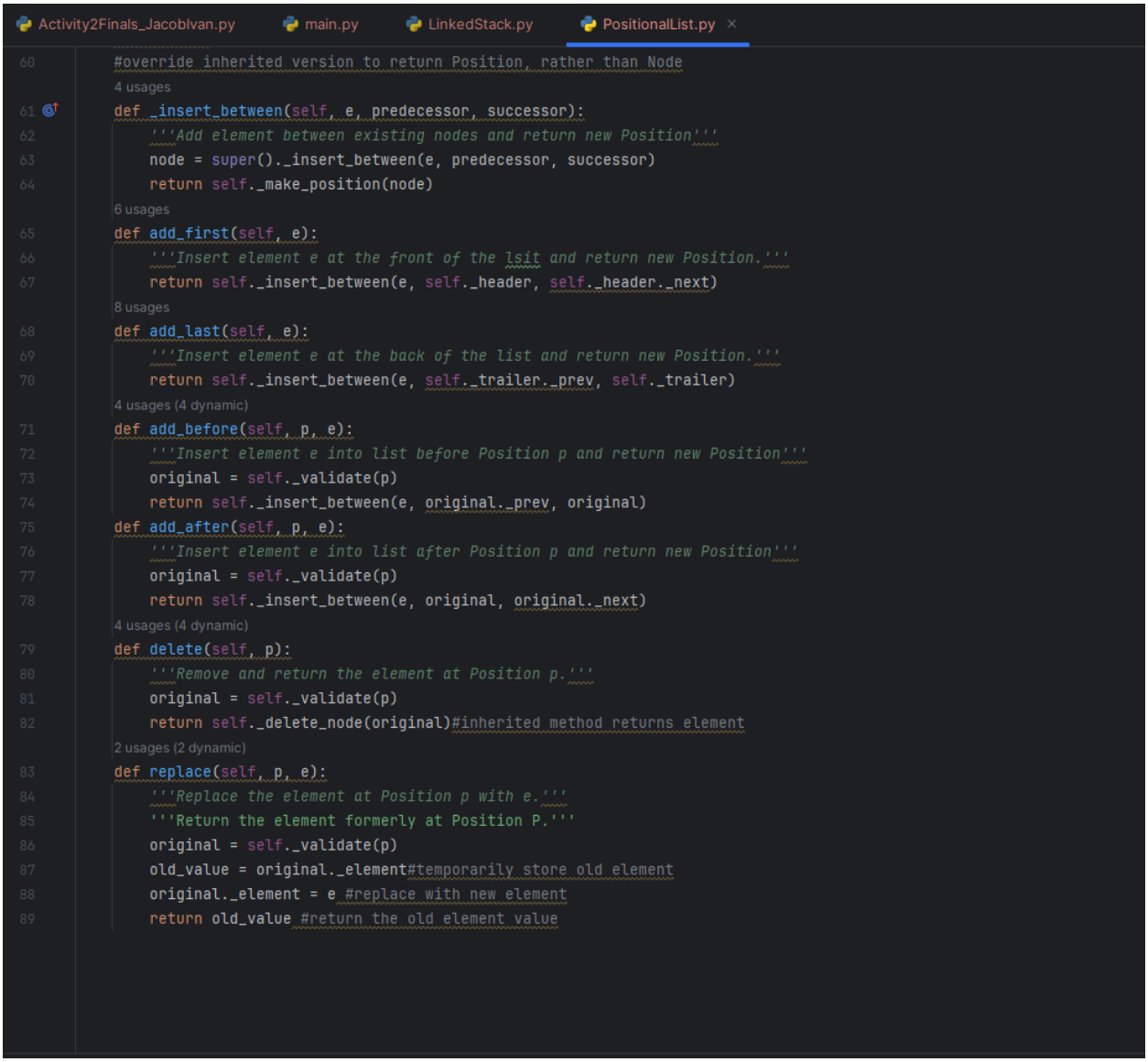
```python
from DoublyLinkedBase import _DoublyLinkedBase
4 usages
class PositionalList(_DoublyLinkedBase):
    '''A sequential container of elements allowing positional access.'''
    #---Positional list class
    class Position:
        '''An abstraction representing the location of a single element.'''
        def __init__(self, container, node):
            '''Constructor should not be invoked by the user.'''
            self._container = container
            self._node = node
        def element(self):
            '''Return the element stored at this Position'''
            return self._node._element
        def __eq__(self, other):
            '''Return True if other is a Position representing the same location.'''
            return type(other) is type(self) and other._node is self._node
        def __ne__(self,other):
            '''Return True if other does not represent the same location.'''
            return not (self == other) #opposite of __eq__

    #-- utility method
    6 usages
    def _validate(self, p):
        '''Return postiion's node or raise appropriate error if invalid'''
        if not isinstance(p, self.Position):
            raise TypeError('p must be proper Position type')
        if p._container is not self:
            raise ValueError('p does not belong to this container')
        if p._node._next is None:#convention for deprecated nodes
            raise ValueError('p is no longer valid')
        return p._node
    #-- utility method
    5 usages
    def _make_position(self, node):
        '''Return Position instance for given node (or None if sentinel).'''
        if node is self._header or node is self._trailer:
            return None #boudnary violation
        else:
            return self.Position(self, node) #legitimate position
    #-- accessors
```

PositionalList › after()

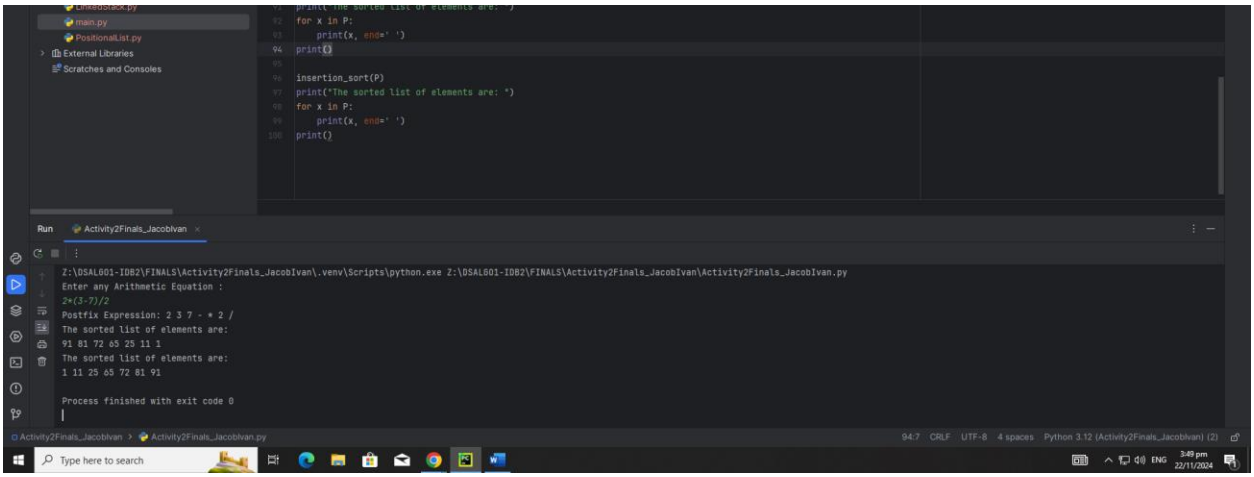
Activity2Finals_Jacoblvan.py | main.py | LinkedStack.py | PositionalList.py

```
37                  return self.Position(self, node) #legitimate position
38          #-- accessors
            10 usages (9 dynamic)
39          def first(self):
40              '''Return the first Position in the list (or None if list is empty.)'''
41              return self._make_position(self._header._next)
            4 usages (4 dynamic)
42          def last(self):
43              '''Return the last Position in the list (or None if list is empty)'''
44              return self._make_position(self._trailer._prev)
            8 usages (8 dynamic)
45          def before(self, p):
46              '''Return the Position just before Position P (or None if p is first)'''
47              node = self._validate(p)
48              return self._make_position(node._prev)
            6 usages (5 dynamic)
49          def after(self, p):
50              '''Return the Position just after Position p (or None if p is last.)'''
51              node = self._validate(p)
52              return self._make_position(node._next)
53          def __iter__(self):
54              '''Generate forward iteration of the elements of the list'''
55              cursor = self.first()
56              while cursor is not None:
57                  yield cursor.element()
58                  cursor = self.after(cursor)
59          #--mutators
60          #override inherited version to return Position, rather than Node
            4 usages
61          def _insert_between(self, e, predecessor, successor):
62              '''Add element between existing nodes and return new Position'''
63              node = super()._insert_between(e, predecessor, successor)
64              return self._make_position(node)
            6 usages
65          def add_first(self, e):
66              '''Insert element e at the front of the lsit and return new Position.'''
67              return self._insert_between(e, self._header, self._header._next)
            8 usages
68          def add_last(self, e):
69              '''Insert element e at the back of the list and return new Position.'''
70              return self._insert_between(e, self._trailer._prev, self._trailer)
```

PositionalList › after()


Activity2Finals_Jacoblvan.py
main.py
LinkedStack.py
PositionalList.py
#override inherited version to return Position, rather than Node
4 usages
def _insert_between(self, e, predecessor, successor):
'''Add element between existing nodes and return new Position'''
node = super()._insert_between(e, predecessor, successor)
return self._make_position(node)
6 usages
def add_first(self, e):
'''Insert element e at the front of the lsit and return new Position.'''
return self._insert_between(e, self._header, self._header._next)
8 usages
def add_last(self, e):
'''Insert element e at the back of the list and return new Position.'''
return self._insert_between(e, self._trailer._prev, self._trailer)
4 usages (4 dynamic)
def add_before(self, p, e):
'''Insert element e into list before Position p and return new Position'''
original = self._validate(p)
return self._insert_between(e, original._prev, original)
def add_after(self, p, e):
'''Insert element e into list after Position p and return new Position'''
original = self._validate(p)
return self._insert_between(e, original, original._next)
4 usages (4 dynamic)
def delete(self, p):
'''Remove and return the element at Position p.'''
original = self._validate(p)
return self._delete_node(original)#inherited method returns element
2 usages (2 dynamic)
def replace(self, p, e):
'''Replace the element at Position p with e.'''
'''Return the element formerly at Position P.'''
original = self._validate(p)
old_value = original._element#temporarily store old element
original._element = e #replace with new element
return old_value #return the old element value

OUTPUT:


main.py
PositionalList.py
External Libraries
Scratches and Consoles
for x in P:
print(x, end=' ')
print()
insertion_sort(P)
print("The sorted list of elements are: ")
for x in P:
print(x, end=' ')
print()
Run
Activity2Finals_JacobIvan
Z:\DSALGO1-IDB2\FINALS\Activity2Finals_JacobIvan\.venv\Scripts\python.exe Z:\DSALGO1-IDB2\FINALS\Activity2Finals_JacobIvan\Activity2Finals_JacobIvan.py
Enter any Arithmetic Equation :
2+(3-7)/2
Postfix Expression: 2 3 7 - * 2 /
The sorted list of elements are:
91 81 72 65 25 11 1
The sorted list of elements are:
1 11 25 65 72 81 91
Process finished with exit code 0
Activity2Finals_JacobIvan > Activity2Finals_JacobIvan.py
94:7 CRLF UTF-8 4 spaces Python 3.12 (Activity2Finals_JacobIvan) (2)
Type here to search
ENG
3:49 pm
22/11/2024