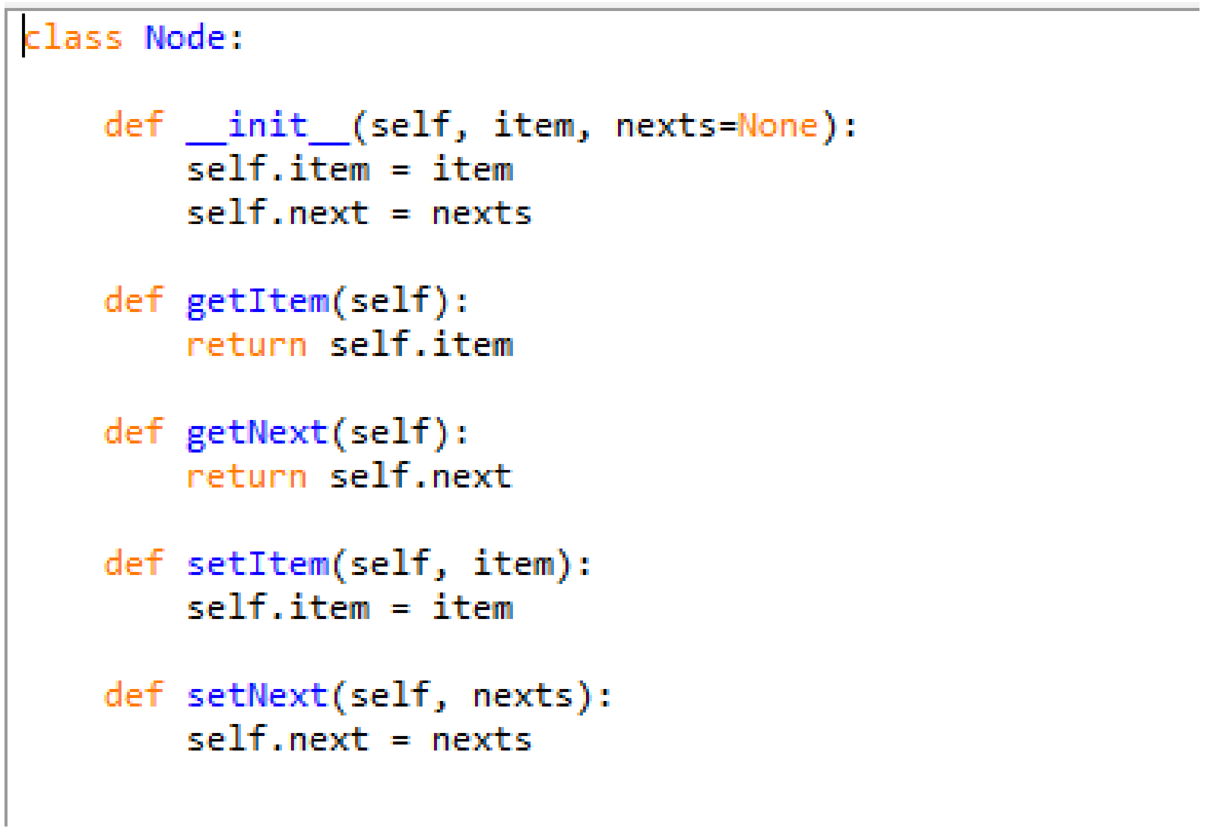
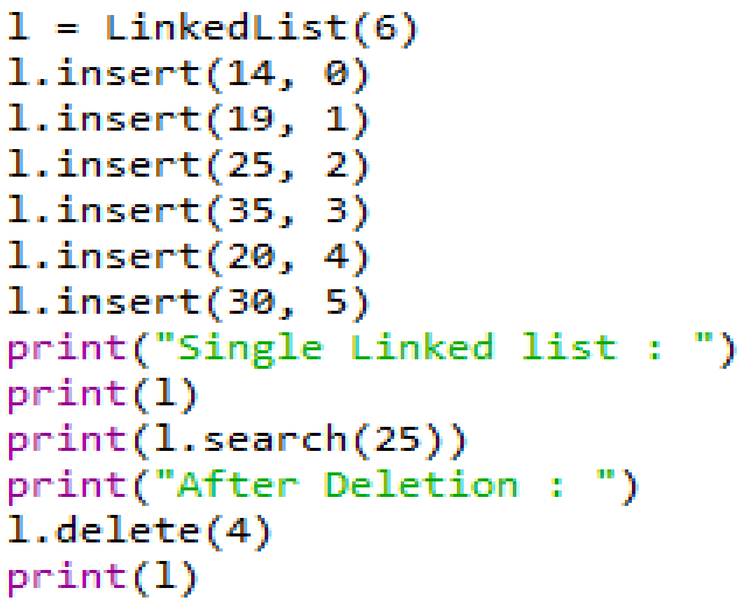
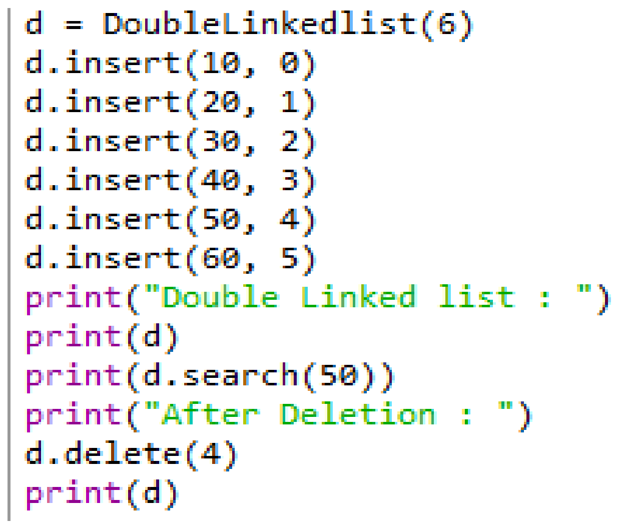


```python
class Node:

    def __init__(self, item, nexts=None):
        self.item = item
        self.next = nexts

    def getItem(self):
        return self.item

    def getNext(self):
        return self.next

    def setItem(self, item):
        self.item = item

    def setNext(self, nexts):
        self.next = nexts
```

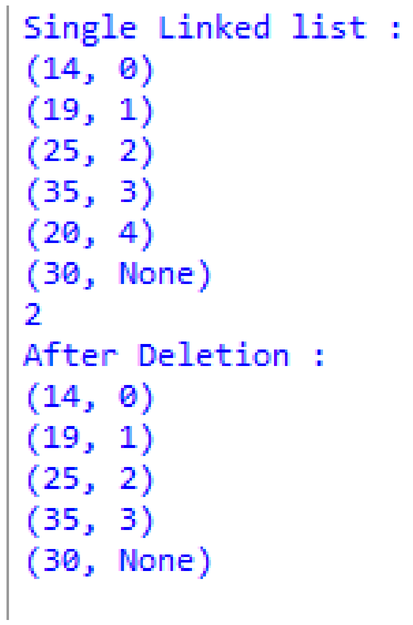


```python
l = LinkedList(6)
l.insert(14, 0)
l.insert(19, 1)
l.insert(25, 2)
l.insert(35, 3)
l.insert(20, 4)
l.insert(30, 5)
print("Single Linked list : ")
print(l)
print(l.search(25))
print("After Deletion : ")
l.delete(4)
print(l)
```

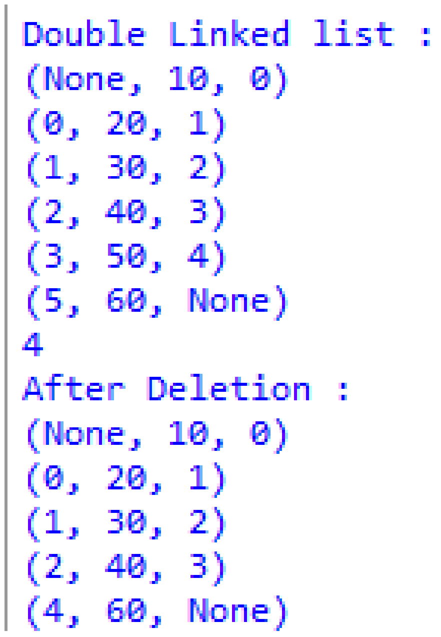
```python
class LinkedList:

    def __init__(self, nodes):
        self.linkedL = []
        for i in range(nodes):
            self.linkedL.append(0)

    def insert(self, item, pos):
        self.cursor = 0
        index = pos
        if index < len(self.linkedL):
            self.cursor = index
            self.node = Node(item, self.cursor + 1)
            self.linkedL[self.cursor] = item

    def search(self, item):
        self.item = item
        n = 0
        while n != len(self.linkedL):
            if self.item == self.linkedL[n]:
                return n
            else:
                n += 1
        return None

    def delete(self, n):
        if len(self.linkedL) == 0:
            return None
        else:
            for i in range(len(self.linkedL)):
                if i == n:
                    del self.linkedL[i]

    def __str__(self):
        st = ""
        for i in range(len(self.linkedL)):
            if i == len(self.linkedL) - 1:
                st += f"{(self.linkedL[i], None)}"
            else:
                st += f"{(self.linkedL[i], i)}\n"
        return st
```



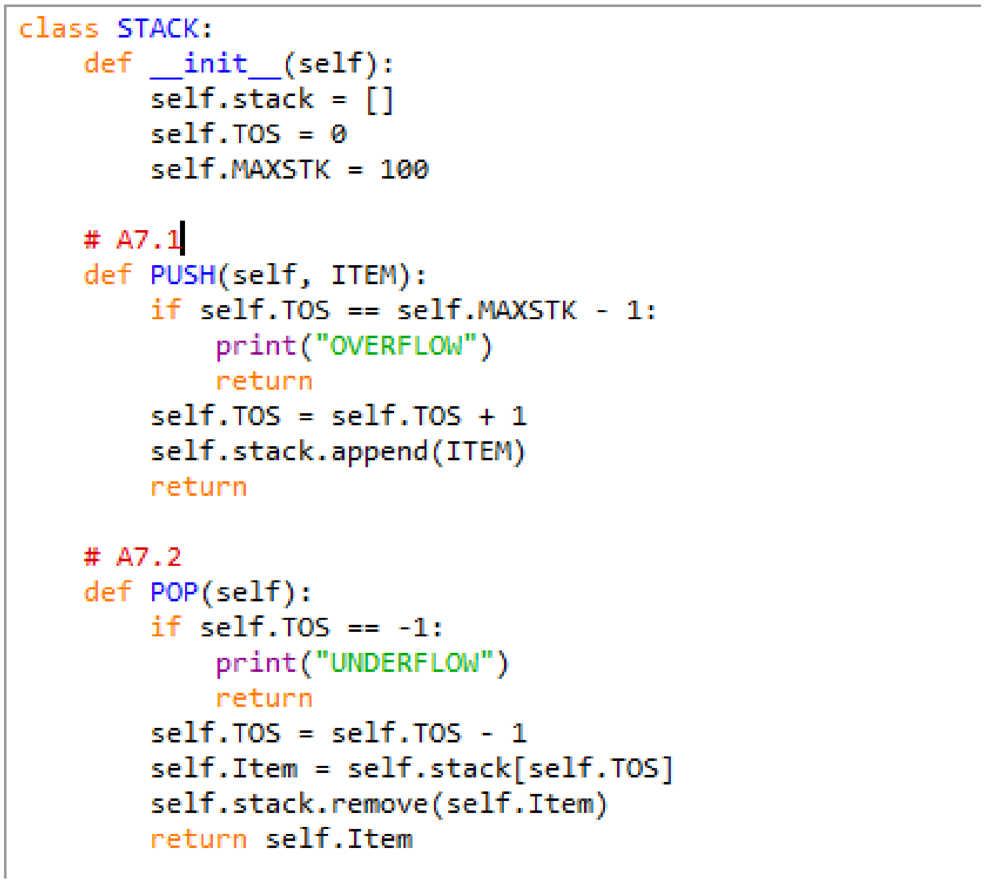
```
d = DoubleLinkedlist(6)
d.insert(10, 0)
d.insert(20, 1)
d.insert(30, 2)
d.insert(40, 3)
d.insert(50, 4)
d.insert(60, 5)
print("Double Linked list : ")
print(d)
print(d.search(50))
print("After Deletion : ")
d.delete(4)
print(d)
```



```
Single Linked list :
(14, 0)
(19, 1)
(25, 2)
(35, 3)
(20, 4)
(30, None)
2
After Deletion :
(14, 0)
(19, 1)
(25, 2)
(35, 3)
(30, None)
```



```
Double Linked list :
(None, 10, 0)
(0, 20, 1)
(1, 30, 2)
(2, 40, 3)
(3, 50, 4)
(5, 60, None)
4
After Deletion :
(None, 10, 0)
(0, 20, 1)
(1, 30, 2)
(2, 40, 3)
(4, 60, None)
```



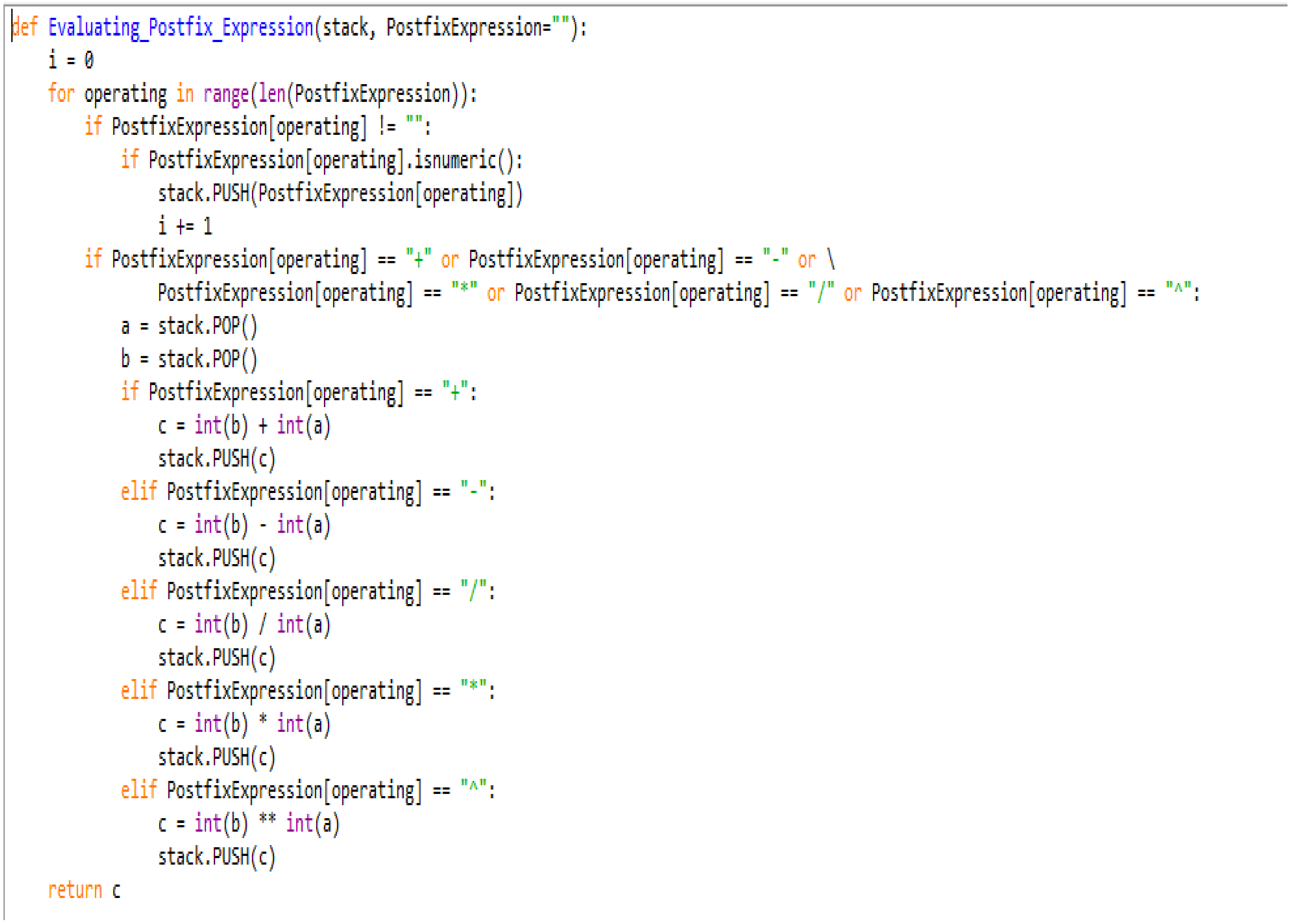
```python
class STACK:
    def __init__(self):
        self.stack = []
        self.TOS = 0
        self.MAXSTK = 100

    # A7.1
    def PUSH(self, ITEM):
        if self.TOS == self.MAXSTK - 1:
            print("OVERFLOW")
            return
        self.TOS = self.TOS + 1
        self.stack.append(ITEM)
        return

    # A7.2
    def POP(self):
        if self.TOS == -1:
            print("UNDERFLOW")
            return
        self.TOS = self.TOS - 1
        self.Item = self.stack[self.TOS]
        self.stack.remove(self.Item)
        return self.Item
```

```
s = STACK()
s.PUSH(44)
s.PUSH(55)
s.PUSH(66)
print("After Pushing : ", s.stack)
s.POP()
print("After Popping : ", s.stack)
```

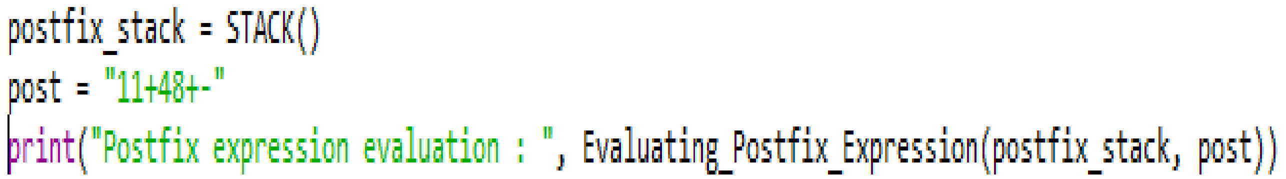
```
After Pushing :  [44, 55, 66]
After Popping :  [44, 55]
```

```
    def ClearStack(self):
        X = 4
        Z = 0
        Y = X + 1
        self.PUSH(Y)
        self.PUSH(Y+1)
        self.PUSH(X)
        self.POP(Y)
        X = Y + 1
        self.PUSH(X)
        self.PUSH(Z)
        while (self.empty()):
            self.POP(Z)
            print(Z)

        print("X = ", X)
        print("Y = ", Y)
        print("Z = ", Z)
```

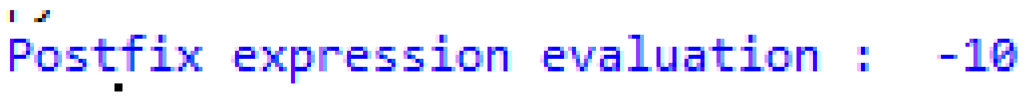
```python
class DoubleLinkedlist:

    def __init__(self, nodes):
        self.doubleLinkedL = []
        for i in range(nodes):
            self.doubleLinkedL.append(0)

    def insert(self, item, pos):
        self.cursor = 0
        index = pos
        if index < len(self.doubleLinkedL):
            self.cursor = index
            self.node = Node(item, self.cursor + 1)
            self.doubleLinkedL[self.cursor] = item

    def search(self, item):
        self.item = item
        n = 0
        while n != len(self.doubleLinkedL):
            if self.item == self.doubleLinkedL[n]:
                return n
            else:
                n += 1
        return None

    def delete(self, n):
        if len(self.doubleLinkedL) == 0:
            return None
        else:
            for i in range(len(self.doubleLinkedL)):
                if i == n:
                    del self.doubleLinkedL[i]

    def __str__(self):
        st = ""
        for i in range(len(self.doubleLinkedL)):
            if i==0:
                st+=f"{None, self.doubleLinkedL[i], i}\n"
            elif i == len(self.doubleLinkedL) - 1:
                st += f"{(i, self.doubleLinkedL[i], None)}"
            else:
                st += f"{(i-1,self.doubleLinkedL[i], i)}\n"
        return st
```



```
def Evaluating_Postfix_Expression(stack, PostfixExpression=""):
    i = 0
    for operating in range(len(PostfixExpression)):
        if PostfixExpression[operating] != "":
            if PostfixExpression[operating].isnumeric():
                stack.PUSH(PostfixExpression[operating])
                i += 1
        if PostfixExpression[operating] == "+" or PostfixExpression[operating] == "-" or \
                PostfixExpression[operating] == "*" or PostfixExpression[operating] == "/" or PostfixExpression[operating] == "^":
            a = stack.POP()
            b = stack.POP()
            if PostfixExpression[operating] == "+":
                c = int(b) + int(a)
                stack.PUSH(c)
            elif PostfixExpression[operating] == "-":
                c = int(b) - int(a)
                stack.PUSH(c)
            elif PostfixExpression[operating] == "/":
                c = int(b) / int(a)
                stack.PUSH(c)
            elif PostfixExpression[operating] == "*":
                c = int(b) * int(a)
                stack.PUSH(c)
            elif PostfixExpression[operating] == "^":
                c = int(b) ** int(a)
                stack.PUSH(c)
    return c
```



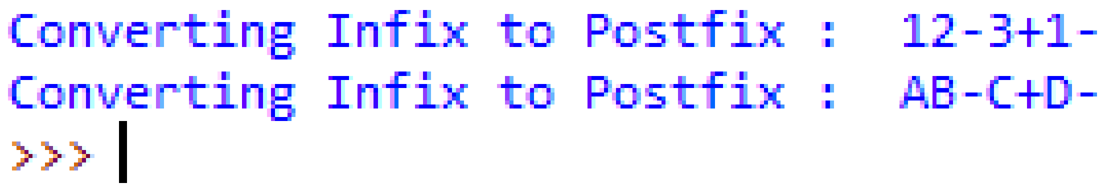
```
postfix_stack = STACK()
post = "11+48+-"
print("Postfix expression evaluation : ", Evaluating_Postfix_Expression(postfix_stack, post))
```



```
Postfix expression evaluation :  -10
```

```python
def convert_infix_into_postfix(stack, infix=""):
    operator = ""
    j=1
    infix += ")"
    postfix = ""
    stack.PUSH("(")
    for i in range(len(infix)):
        if infix[i] == "(":
            stack.PUSH(infix[i])
        elif infix[i].isnumeric() or infix[i].isalpha():
            postfix+=infix[i]
        elif infix[i] == ")":
            while True:
                num = stack.POP()
                if num == "(":
                    break
                postfix+=num
        elif infix[i] == "+" or infix[i] == "-" or \
                infix[i] == "*" or infix[i] == "/" or infix[i] == "^":
            if j==2:
                postfix+=operator
                j=1
            operator = infix[i]
            j+=1
        if i == len(infix) - 1:
            postfix+=operator
    return postfix
```

```python
inf = "1-2+3-1"
inf1 = "A-B+C-D"

converting_inf_to_post = STACK()
converting_inf_to_post1 = STACK()

print("Converting Infix to Postfix : ", convert_infix_into_postfix(converting_inf_to_post, inf))
print("Converting Infix to Postfix : ", convert_infix_into_postfix(converting_inf_to_post1, inf1))
```



```
Converting Infix to Postfix :  12-3+1-
Converting Infix to Postfix :  AB-C+D-
>>> 
```