Data Structures 2023-2

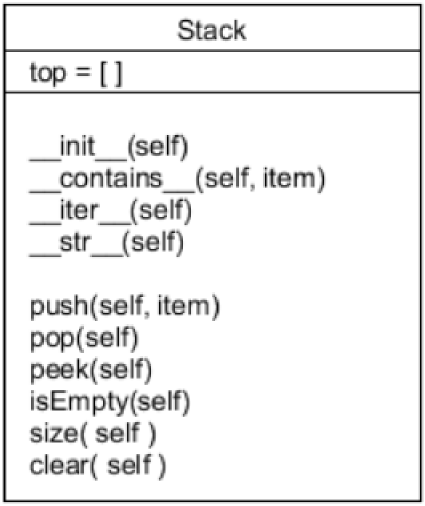
Lab 03: Stack Abstract Data Type

1. **Task-1: Stack Abstract Data Type**
   Write code for Stack abstract data type and test it

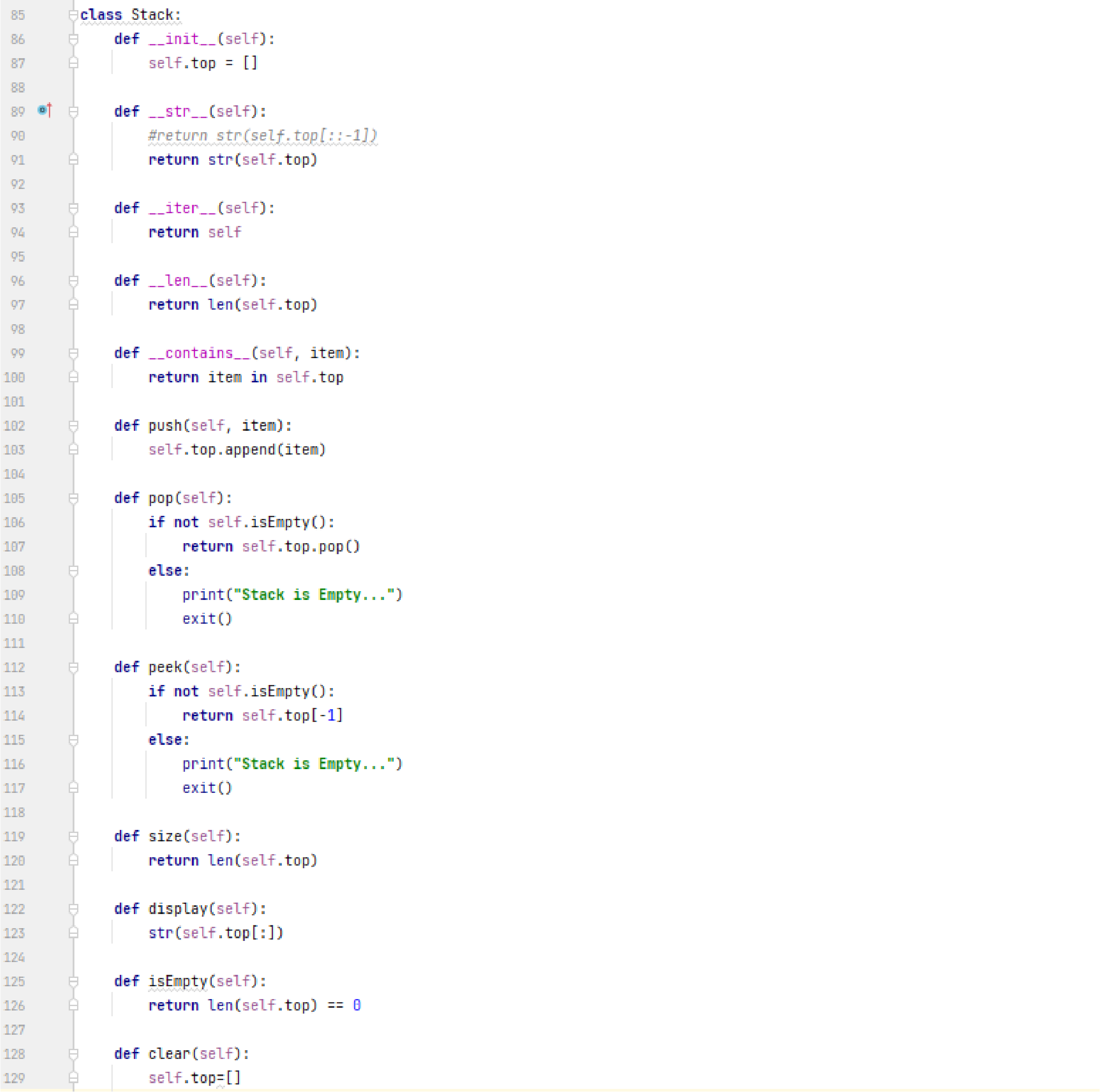
   1.1 Object: Collection of data items such that last-in first out (LIFO) mechanism is maintained

   1.2 Operations

   - push(x): adds an element x on the top of the stack
   - pop(): removes the top element of the stack. The next element will become the top element
   - isEmpty(): It returns true if the stack is empty, otherwise false
   - peek(): It returns the top element without removing it from the Stack
   - size(): It returns the number of items in the stack
   - display(): It displays all the elements stored in the stack
   - find(item): return true if item is found in the stack

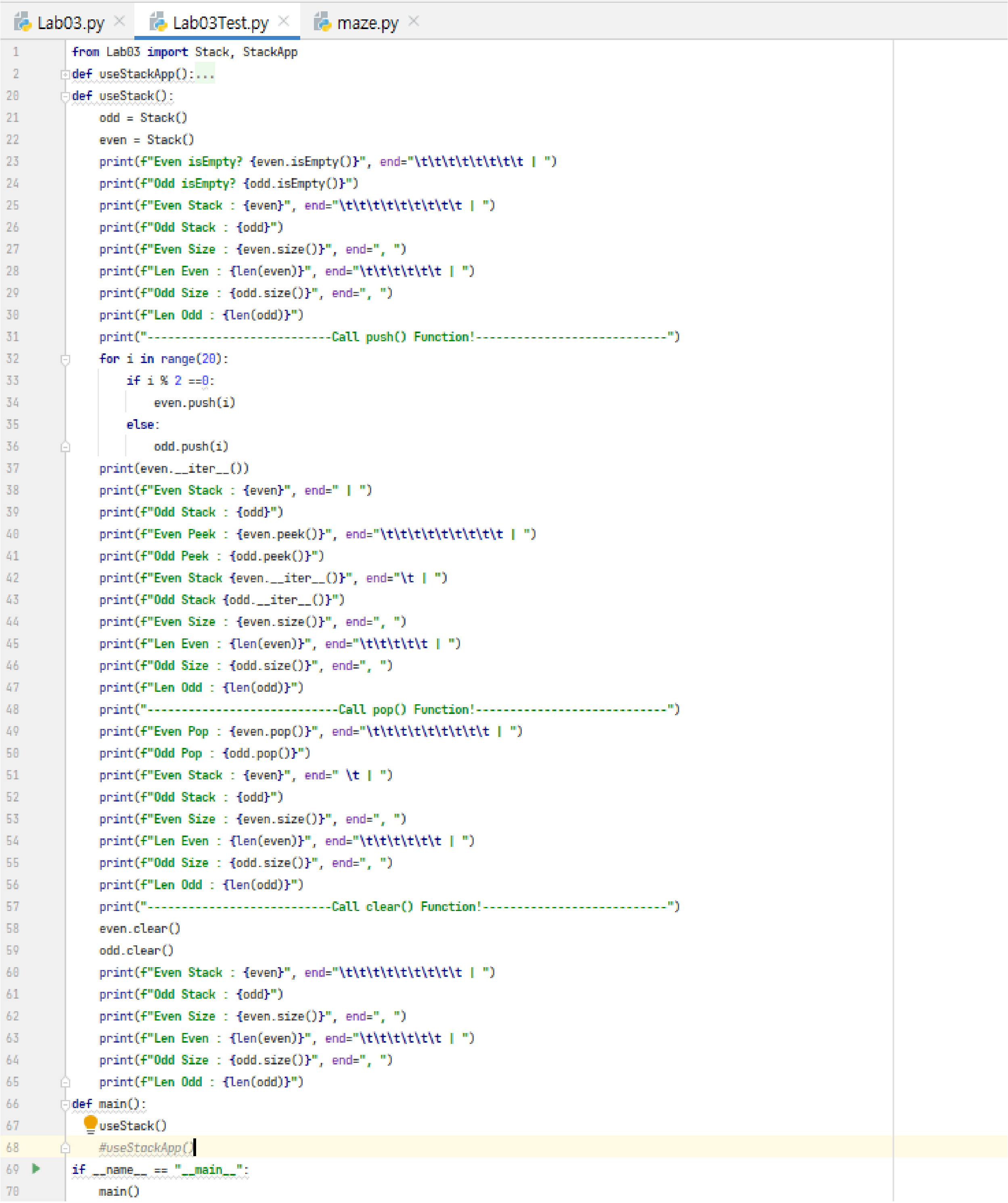


Lab03.py Code



```python
class Stack:
    def __init__(self):
        self.top = []

    def __str__(self):
        #return str(self.top[::-1])
        return str(self.top)

    def __iter__(self):
        return self

    def __len__(self):
        return len(self.top)

    def __contains__(self, item):
        return item in self.top

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

    def pop(self):
        if not self.isEmpty():
            return self.top.pop()
        else:
            print("Stack is Empty...")
            exit()

    def peek(self):
        if not self.isEmpty():
            return self.top[-1]
        else:
            print("Stack is Empty...")
            exit()

    def size(self):
        return len(self.top)

    def display(self):
        str(self.top[:])

    def isEmpty(self):
        return len(self.top) == 0

    def clear(self):
        self.top=[]
```

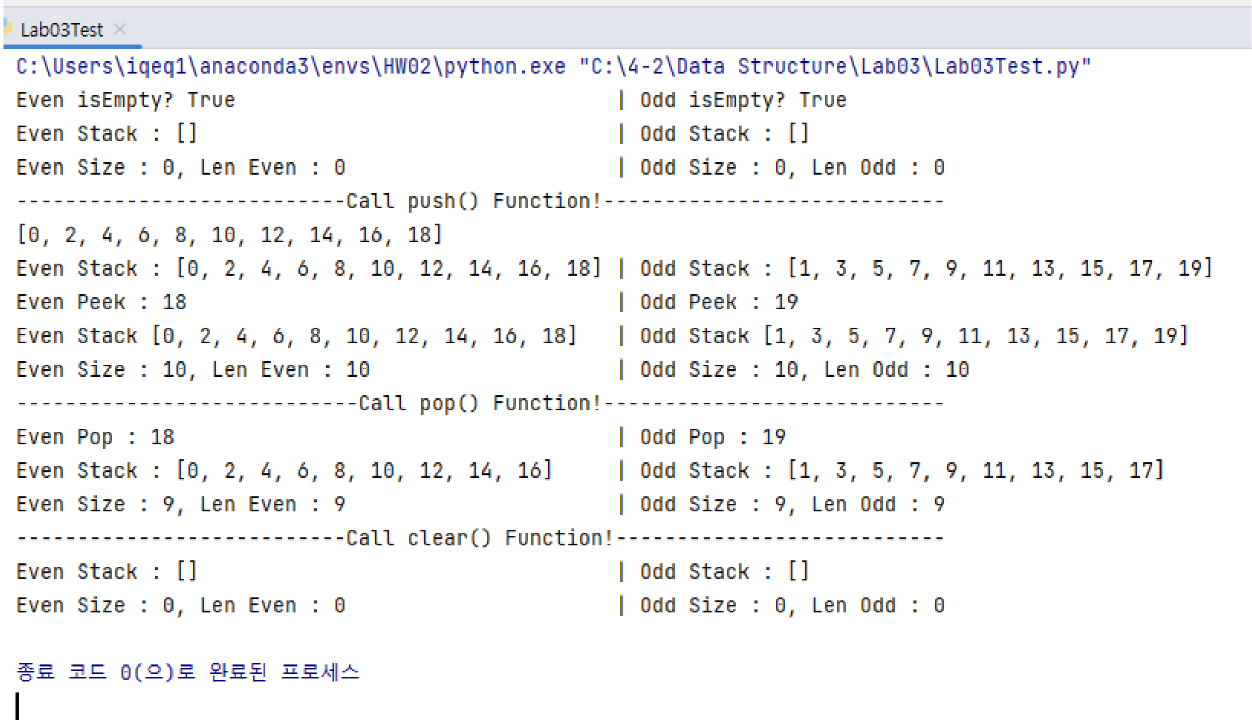
Lab03Test.py Code”



```
from Lab03 import Stack, StackApp
def useStackApp():...
def useStack():
    odd = Stack()
    even = Stack()
    print(f"Even isEmpty? {even.isEmpty()}", end="\t\t\t\t\t\t\t\t | ")
    print(f"Odd isEmpty? {odd.isEmpty()}")
    print(f"Even Stack : {even}", end="\t\t\t\t\t\t\t\t\t | ")
    print(f"Odd Stack : {odd}")
    print(f"Even Size : {even.size()}", end=", ")
    print(f"Len Even : {len(even)}", end="\t\t\t\t\t\t | ")
    print(f"Odd Size : {odd.size()}", end=", ")
    print(f"Len Odd : {len(odd)}")
    print("---------------------------Call push() Function!---------------------------")
    for i in range(20):
        if i % 2 ==0:
            even.push(i)
        else:
            odd.push(i)
    print(even.__iter__())
    print(f"Even Stack : {even}", end=" | ")
    print(f"Odd Stack : {odd}")
    print(f"Even Peek : {even.peek()}", end="\t\t\t\t\t\t\t\t\t | ")
    print(f"Odd Peek : {odd.peek()}")
    print(f"Even Stack {even.__iter__()}", end="\t | ")
    print(f"Odd Stack {odd.__iter__()}")
    print(f"Even Size : {even.size()}", end=", ")
    print(f"Len Even : {len(even)}", end="\t\t\t\t\t | ")
    print(f"Odd Size : {odd.size()}", end=", ")
    print(f"Len Odd : {len(odd)}")
    print("---------------------------Call pop() Function!---------------------------")
    print(f"Even Pop : {even.pop()}", end="\t\t\t\t\t\t\t\t\t | ")
    print(f"Odd Pop : {odd.pop()}")
    print(f"Even Stack : {even}", end=" \t | ")
    print(f"Odd Stack : {odd}")
    print(f"Even Size : {even.size()}", end=", ")
    print(f"Len Even : {len(even)}", end="\t\t\t\t\t\t | ")
    print(f"Odd Size : {odd.size()}", end=", ")
    print(f"Len Odd : {len(odd)}")
    print("---------------------------Call clear() Function!---------------------------")
    even.clear()
    odd.clear()
    print(f"Even Stack : {even}", end="\t\t\t\t\t\t\t\t\t | ")
    print(f"Odd Stack : {odd}")
    print(f"Even Size : {even.size()}", end=", ")
    print(f"Len Even : {len(even)}", end="\t\t\t\t\t\t | ")
    print(f"Odd Size : {odd.size()}", end=", ")
    print(f"Len Odd : {len(odd)}")
def main():
    useStack()
    #useStackApp()
if __name__ == "__main__":
    main()
```

Results/Output

Insert pictures for the output of the programs written for this task

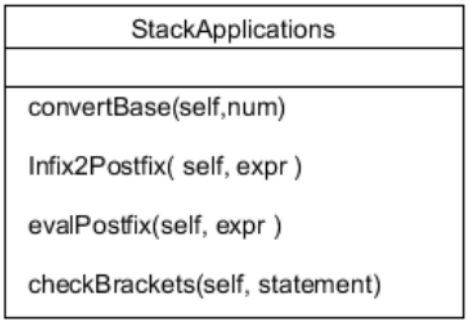


```
C:\Users\iqeq1\anaconda3\envs\HW02\python.exe "C:\4-2\Data Structure\Lab03\Lab03Test.py"
Even isEmpty? True                               | Odd isEmpty? True
Even Stack : []                                  | Odd Stack : []
Even Size : 0, Len Even : 0                      | Odd Size : 0, Len Odd : 0
---------------------------Call push() Function!---------------------------
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
Even Stack : [0, 2, 4, 6, 8, 10, 12, 14, 16, 18] | Odd Stack : [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
Even Peek : 18                                   | Odd Peek : 19
Even Stack [0, 2, 4, 6, 8, 10, 12, 14, 16, 18]   | Odd Stack [1, 3, 5, 7, 9, 11, 13, 15, 17, 19]
Even Size : 10, Len Even : 10                    | Odd Size : 10, Len Odd : 10
----------------------------Call pop() Function!---------------------------
Even Pop : 18                                    | Odd Pop : 19
Even Stack : [0, 2, 4, 6, 8, 10, 12, 14, 16]     | Odd Stack : [1, 3, 5, 7, 9, 11, 13, 15, 17]
Even Size : 9, Len Even : 9                      | Odd Size : 9, Len Odd : 9
---------------------------Call clear() Function!--------------------------
Even Stack : []                                  | Odd Stack : []
Even Size : 0, Len Even : 0                      | Odd Size : 0, Len Odd : 0

종료 코드 0(으)로 완료된 프로세스
```

2. Task-2: Applications of Stack data structure

Write code for the following applications of Stack data structure

- Convert Bases
- Evaluate postfix expressions
- Convert infix form to postfix form
- check brackets


StackApplications

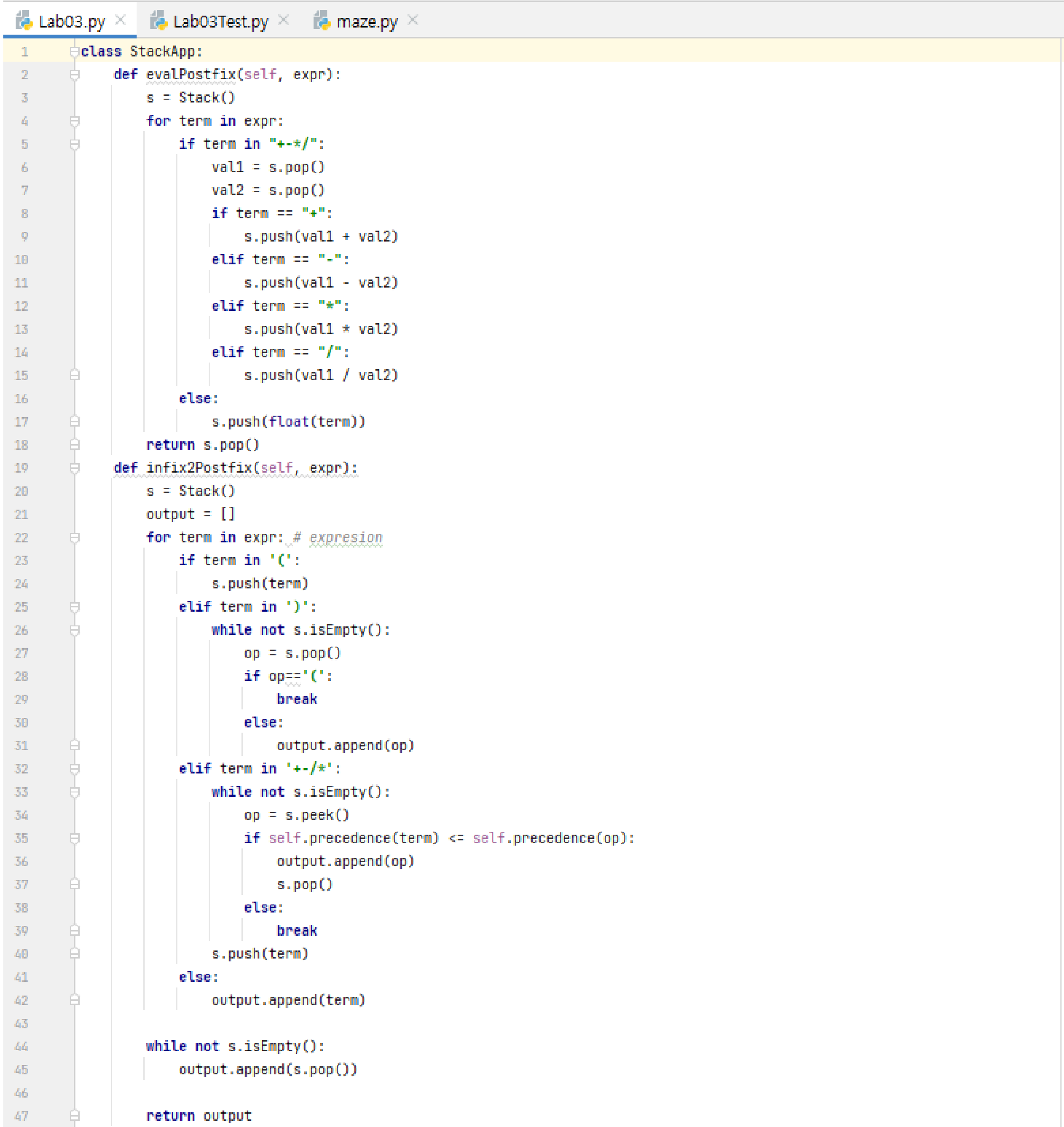
convertBase(self,num)

Infix2Postfix(self, expr)

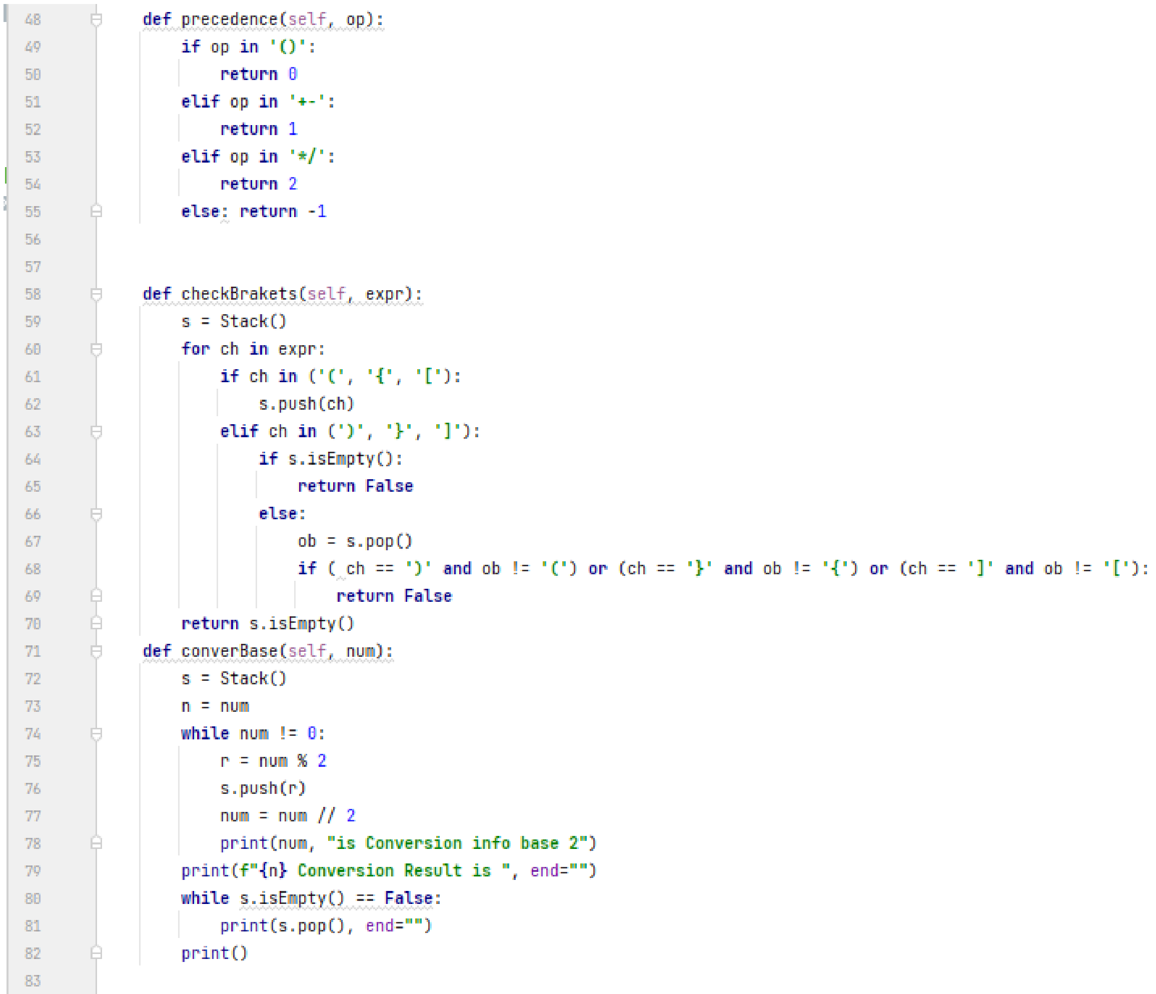
evalPostfix(self, expr)

checkBrackets(self, statement)

Lab03.py Code

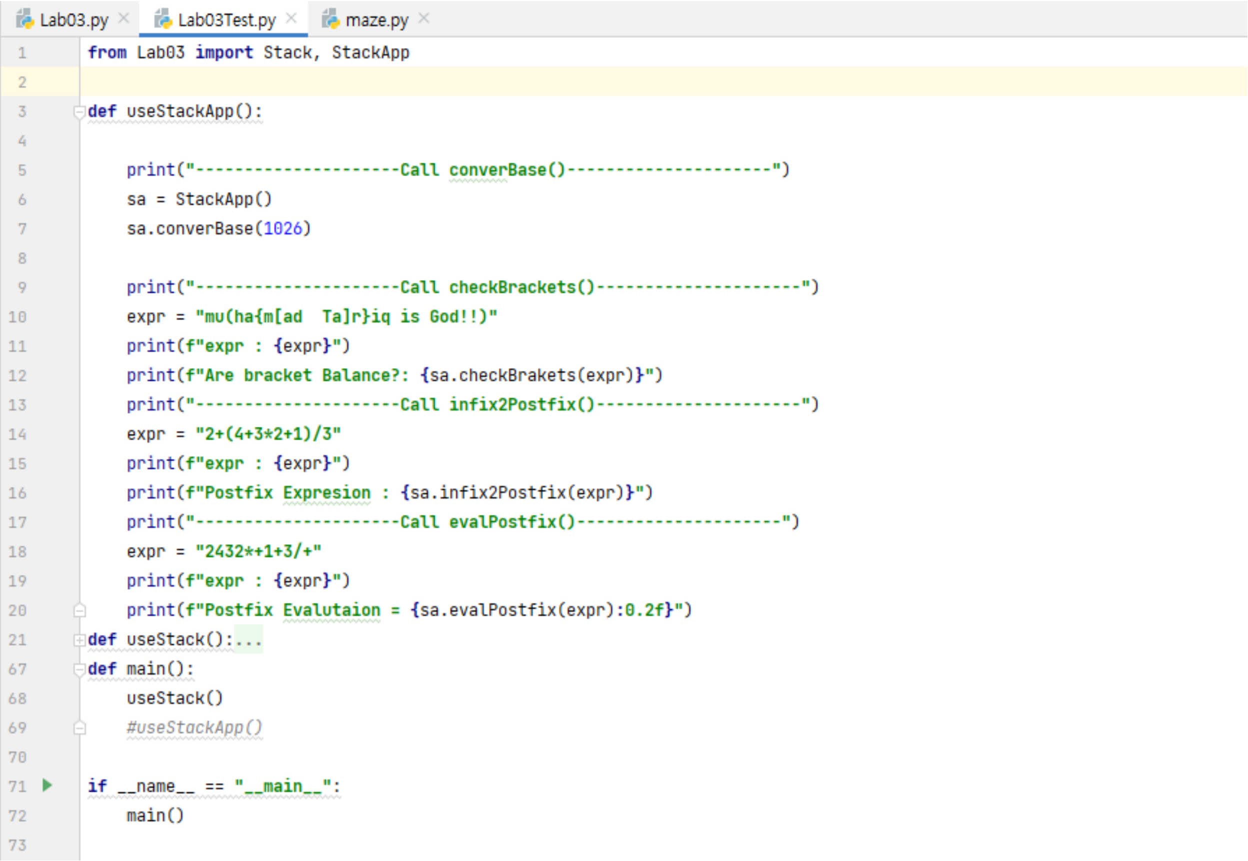


```
class StackApp:
    def evalPostfix(self, expr):
        s = Stack()
        for term in expr:
            if term in "+-*/":
                val1 = s.pop()
                val2 = s.pop()
                if term == "+":
                    s.push(val1 + val2)
                elif term == "-":
                    s.push(val1 - val2)
                elif term == "*":
                    s.push(val1 * val2)
                elif term == "/":
                    s.push(val1 / val2)
            else:
                s.push(float(term))
        return s.pop()
    def infix2Postfix(self, expr):
        s = Stack()
        output = []
        for term in expr: # expresion
            if term in '(':
                s.push(term)
            elif term in ')':
                while not s.isEmpty():
                    op = s.pop()
                    if op=='(':
                        break
                    else:
                        output.append(op)
            elif term in '+-/*':
                while not s.isEmpty():
                    op = s.peek()
                    if self.precedence(term) <= self.precedence(op):
                        output.append(op)
                        s.pop()
                    else:
                        break
                s.push(term)
            else:
                output.append(term)

        while not s.isEmpty():
            output.append(s.pop())

        return output
```



```
    def precedence(self, op):
        if op in '()':
            return 0
        elif op in '+-':
            return 1
        elif op in '*/':
            return 2
        else: return -1


    def checkBrakets(self, expr):
        s = Stack()
        for ch in expr:
            if ch in ('(', '{', '['):
                s.push(ch)
            elif ch in (')', '}', ']'):
                if s.isEmpty():
                    return False
                else:
                    ob = s.pop()
                    if ( ch == ')' and ob != '(') or (ch == '}' and ob != '{') or (ch == ']' and ob != '['):
                        return False
        return s.isEmpty()
    def converBase(self, num):
        s = Stack()
        n = num
        while num != 0:
            r = num % 2
            s.push(r)
            num = num // 2
            print(num, "is Conversion info base 2")
        print(f"{n} Conversion Result is ", end="")
        while s.isEmpty() == False:
            print(s.pop(), end="")
        print()

```

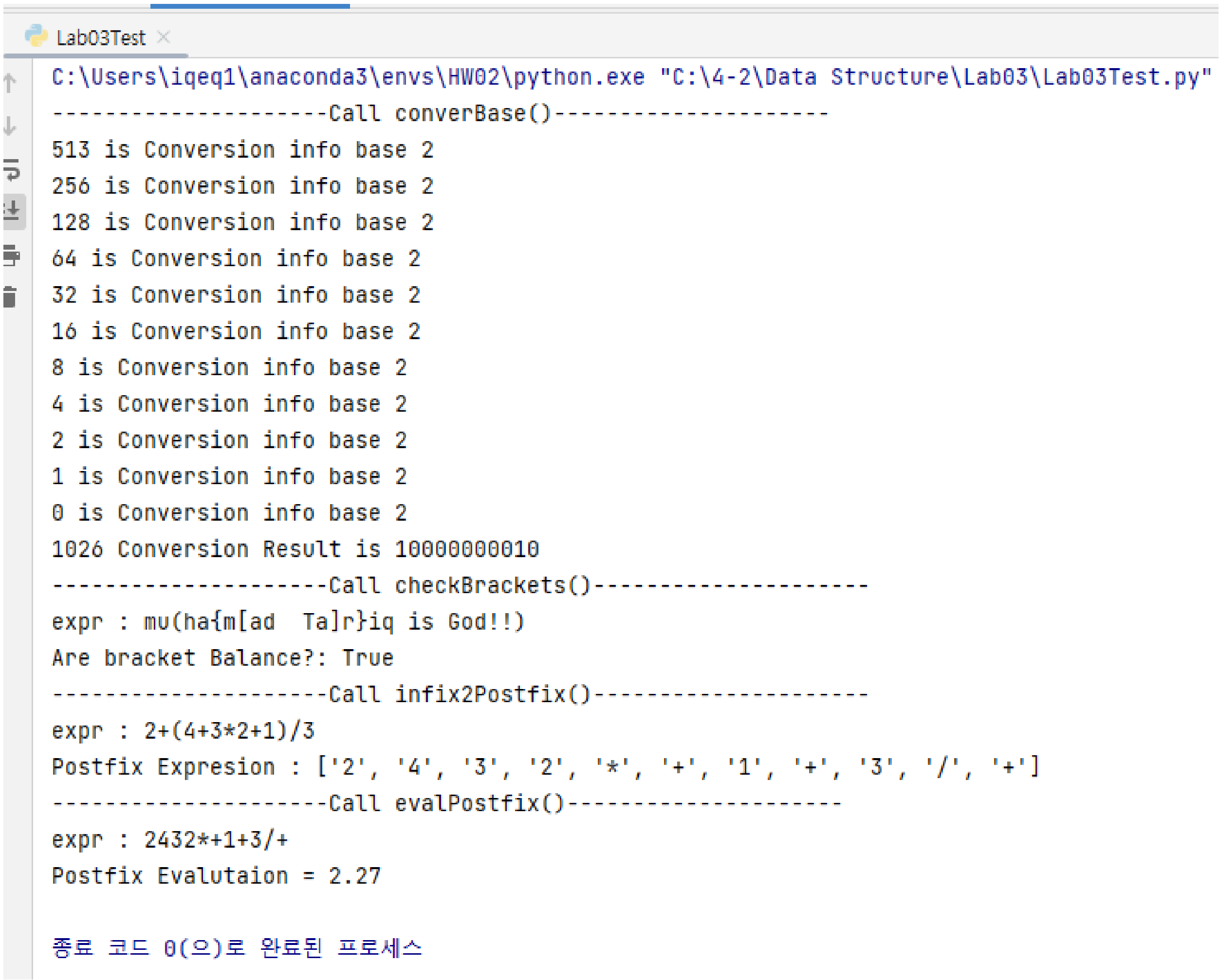
Lab03Test.py Code



```python
from Lab03 import Stack, StackApp

def useStackApp():

    print("---------------------Call converBase()---------------------")
    sa = StackApp()
    sa.converBase(1026)

    print("---------------------Call checkBrackets()---------------------")
    expr = "mu(ha{m[ad  Ta]r}iq is God!!)"
    print(f"expr : {expr}")
    print(f"Are bracket Balance?: {sa.checkBrakets(expr)}")
    print("---------------------Call infix2Postfix()---------------------")
    expr = "2+(4+3*2+1)/3"
    print(f"expr : {expr}")
    print(f"Postfix Expresion : {sa.infix2Postfix(expr)}")
    print("---------------------Call evalPostfix()---------------------")
    expr = "2432*+1+3/+"
    print(f"expr : {expr}")
    print(f"Postfix Evalutaion = {sa.evalPostfix(expr):0.2f}")
def useStack():...
def main():
    useStack()
    #useStackApp()

if __name__ == "__main__":
    main()

```

Results/Output

Insert pictures for the output of the programs written for this task



```
C:\Users\iqeq1\anaconda3\envs\HW02\python.exe "C:\4-2\Data Structure\Lab03\Lab03Test.py"
---------------------Call converBase()---------------------
513 is Conversion info base 2
256 is Conversion info base 2
128 is Conversion info base 2
64 is Conversion info base 2
32 is Conversion info base 2
16 is Conversion info base 2
8 is Conversion info base 2
4 is Conversion info base 2
2 is Conversion info base 2
1 is Conversion info base 2
0 is Conversion info base 2
1026 Conversion Result is 10000000010
---------------------Call checkBrackets()---------------------
expr : mu(ha{m[ad  Ta]r}iq is God!!)
Are bracket Balance?: True
---------------------Call infix2Postfix()---------------------
expr : 2+(4+3*2+1)/3
Postfix Expresion : ['2', '4', '3', '2', '*', '+', '1', '+', '3', '/', '+']
---------------------Call evalPostfix()---------------------
expr : 2432*+1+3/+
Postfix Evalutaion = 2.27

종료 코드 0(으)로 완료된 프로세스
```

3. Conclusion

Conclude the Lab. Write your views about it, i.e. what have you learned from this lab? It was helpful or difficult etc

Through the task, I studied the stack among the basic data structures. From Basic Stack datastructure to Prefix, Parenthesis Validation Check, and etc. It was good for me to learn many things. The queue data structure, which will be learned next week, is also expected. Thank you