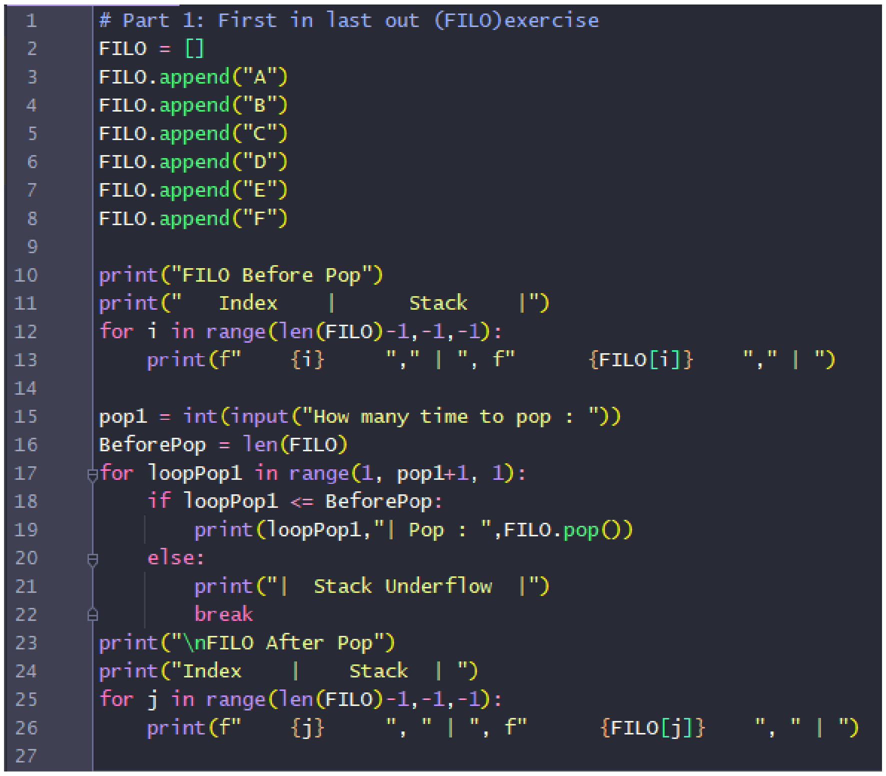
Stack, Queue

Exercise 1

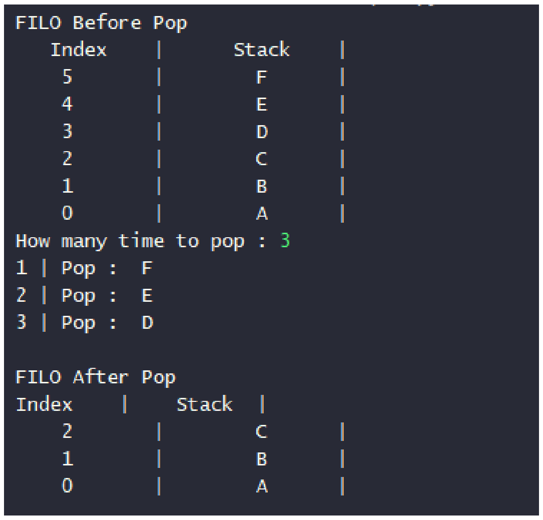
Part 1: First in last out (FILO) exercise

Code:

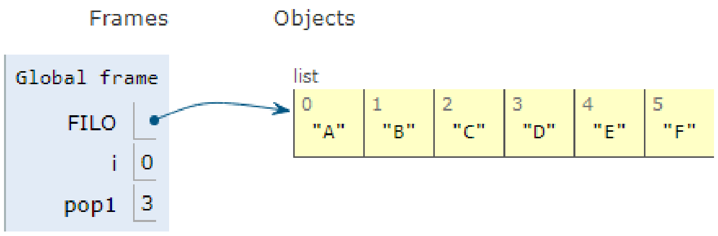


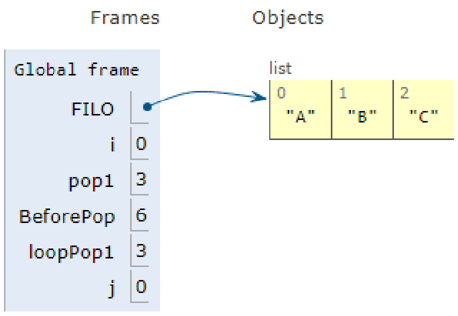
```python
# Part 1: First in last out (FILO)exercise
FILO = []
FILO.append("A")
FILO.append("B")
FILO.append("C")
FILO.append("D")
FILO.append("E")
FILO.append("F")

print("FILO Before Pop")
print("   Index     |       Stack     |")
for i in range(len(FILO)-1,-1,-1):
    print(f"    {i}     "," | ", f"      {FILO[i]}    "," | ")

pop1 = int(input("How many time to pop : "))
BeforePop = len(FILO)
for loopPop1 in range(1, pop1+1, 1):
    if loopPop1 <= BeforePop:
        print(loopPop1,"| Pop : ",FILO.pop())
    else:
        print("|  Stack Underflow  |")
        break
print("\nFILO After Pop")
print("Index    |    Stack   | ")
for j in range(len(FILO)-1,-1,-1):
    print(f"    {j}     ", " | ", f"      {FILO[j]}    ", " | ")
```

Result :



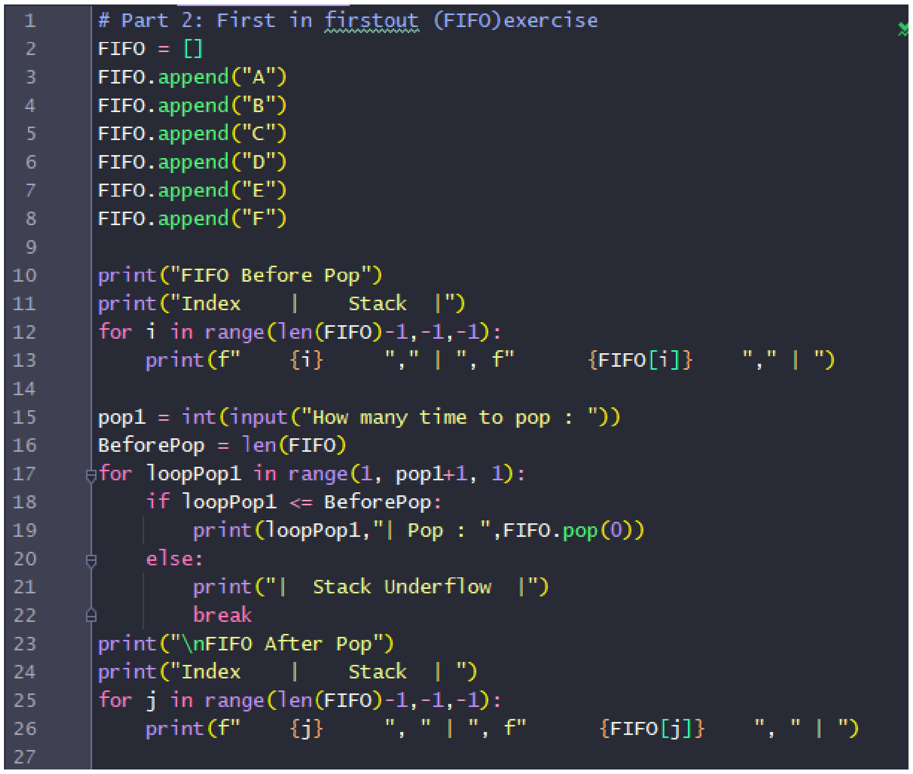
```
FILO Before Pop
   Index     |       Stack     |
    5        |         F       |
    4        |         E       |
    3        |         D       |
    2        |         C       |
    1        |         B       |
    0        |         A       |
How many time to pop : 3
1 | Pop :  F
2 | Pop :  E
3 | Pop :  D

FILO After Pop
Index    |    Stack   |
    2        |         C       |
    1        |         B       |
    0        |         A       |
```





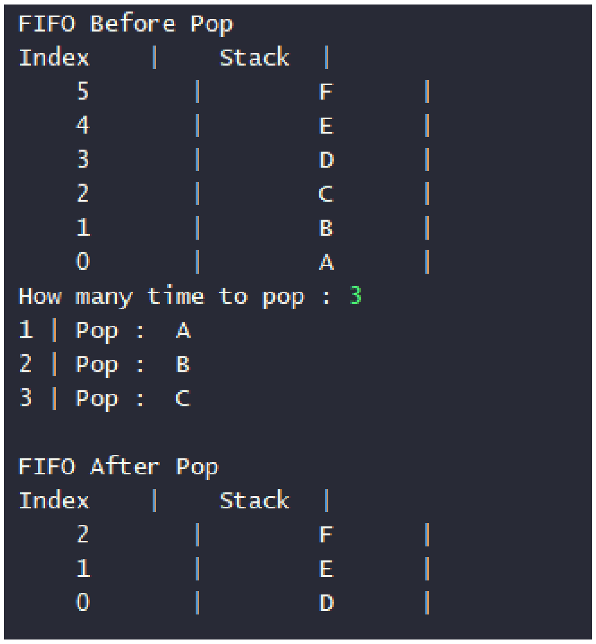
Part 2: First in firstout (FIFO)exercise

Code:

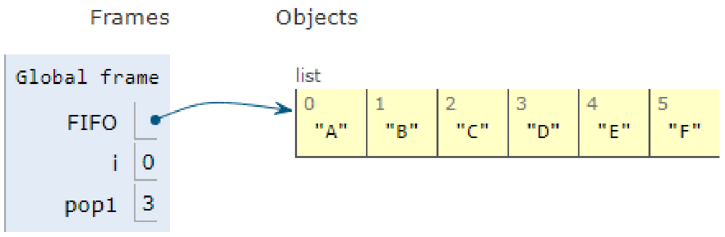


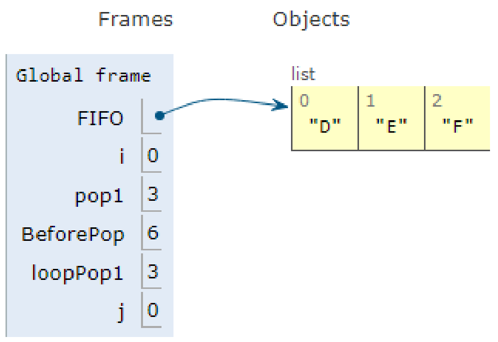
```python
# Part 2: First in firstout (FIFO)exercise
FIFO = []
FIFO.append("A")
FIFO.append("B")
FIFO.append("C")
FIFO.append("D")
FIFO.append("E")
FIFO.append("F")

print("FIFO Before Pop")
print("Index    |    Stack  |")
for i in range(len(FIFO)-1,-1,-1):
    print(f"    {i}     "," | ", f"      {FIFO[i]}    "," | ")

pop1 = int(input("How many time to pop : "))
BeforePop = len(FIFO)
for loopPop1 in range(1, pop1+1, 1):
    if loopPop1 <= BeforePop:
        print(loopPop1,"| Pop : ",FIFO.pop(0))
    else:
        print("|  Stack Underflow  |")
        break
print("\nFIFO After Pop")
print("Index    |    Stack  | ")
for j in range(len(FIFO)-1,-1,-1):
    print(f"    {j}     ", " | ", f"      {FIFO[j]}    ", " | ")
```

Result :



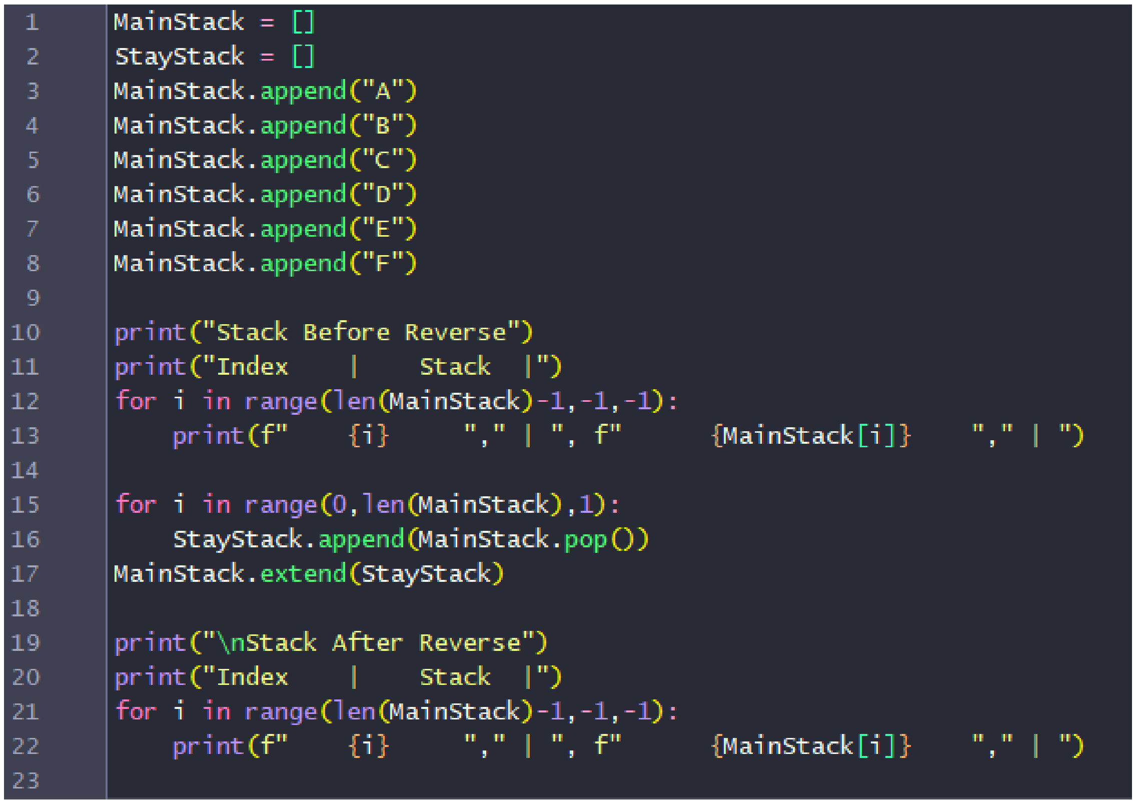
```
FIFO Before Pop
Index    |    Stack   |
    5       |         F       |
    4       |         E       |
    3       |         D       |
    2       |         C       |
    1       |         B       |
    0       |         A       |
How many time to pop : 3
1 | Pop :  A
2 | Pop :  B
3 | Pop :  C

FIFO After Pop
Index    |    Stack   |
    2       |         F       |
    1       |         E       |
    0       |         D       |
```





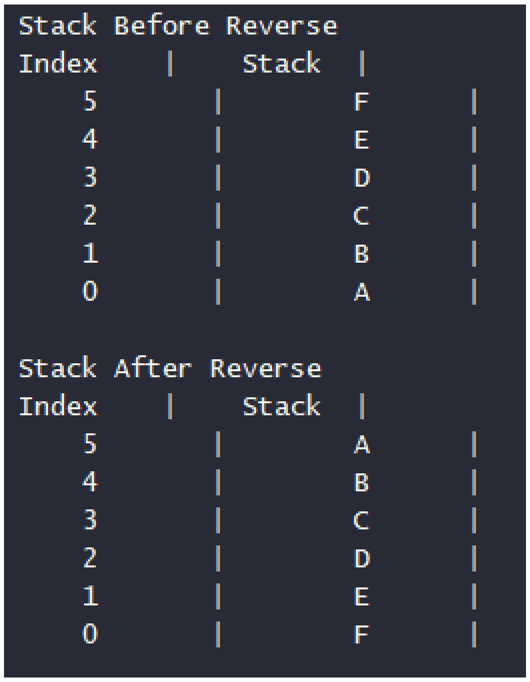
Exercise 2 : Reverse stack exercise

Code :

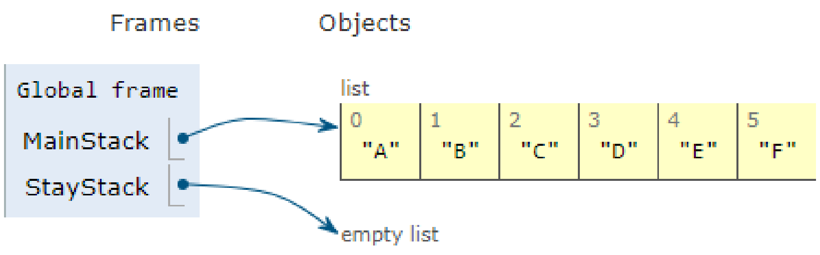


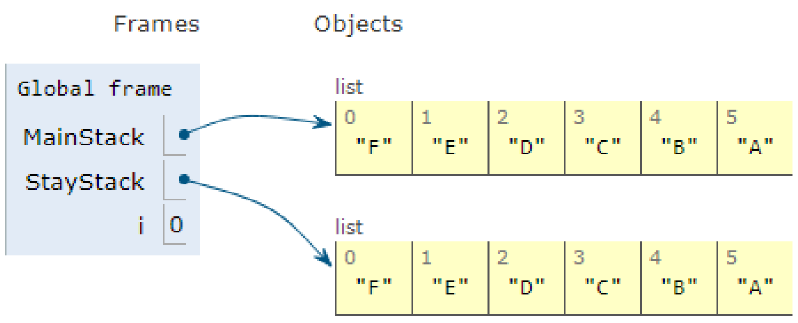
```python
MainStack = []
StayStack = []
MainStack.append("A")
MainStack.append("B")
MainStack.append("C")
MainStack.append("D")
MainStack.append("E")
MainStack.append("F")

print("Stack Before Reverse")
print("Index    |    Stack  |")
for i in range(len(MainStack)-1,-1,-1):
    print(f"    {i}     "," | ", f"      {MainStack[i]}    "," | ")

for i in range(0,len(MainStack),1):
    StayStack.append(MainStack.pop())
MainStack.extend(StayStack)

print("\nStack After Reverse")
print("Index    |    Stack  |")
for i in range(len(MainStack)-1,-1,-1):
    print(f"    {i}     "," | ", f"      {MainStack[i]}    "," | ")
```

Result :



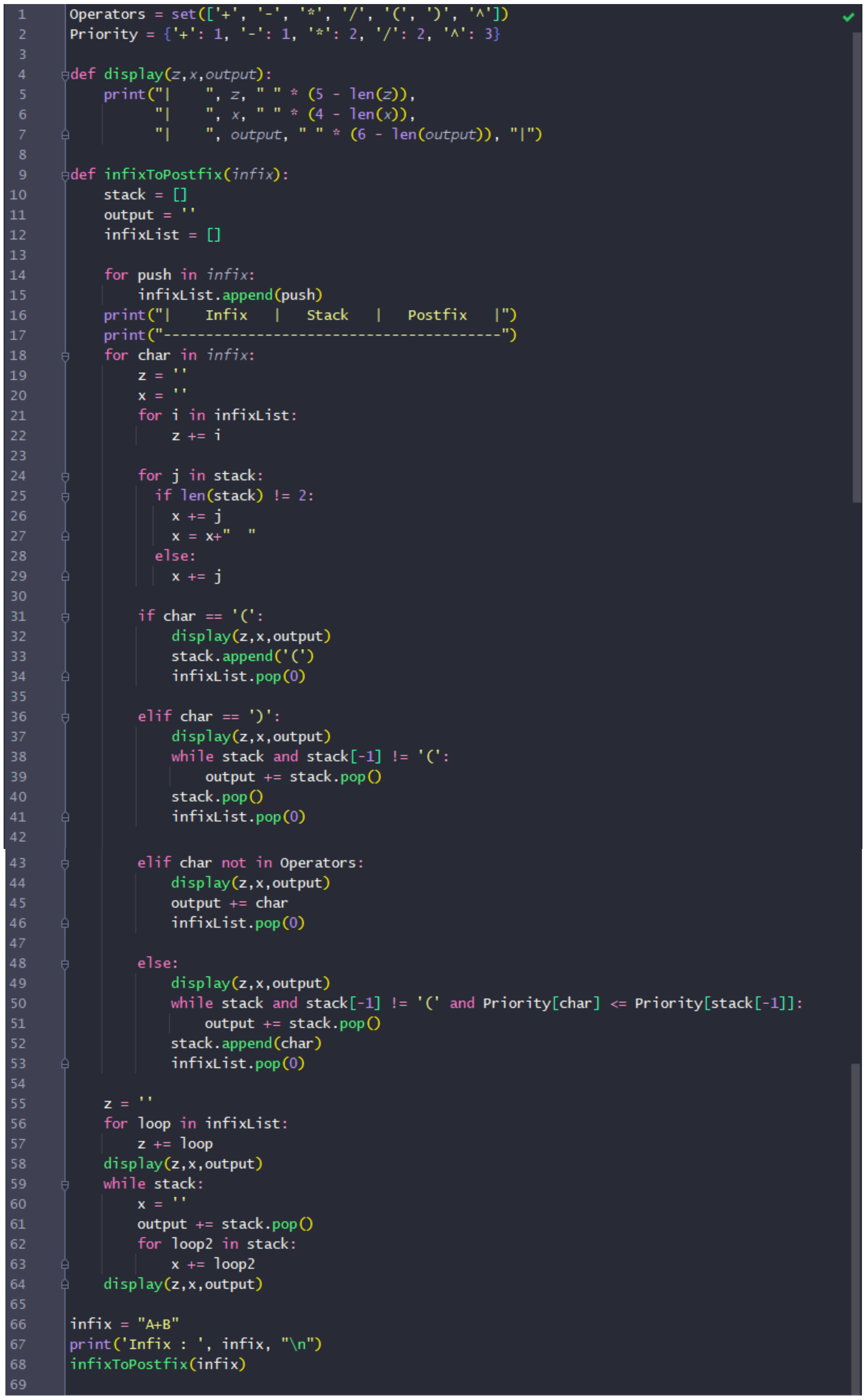
```
Stack Before Reverse
Index    |    Stack  |
    5      |        F      |
    4      |        E      |
    3      |        D      |
    2      |        C      |
    1      |        B      |
    0      |        A      |

Stack After Reverse
Index    |    Stack  |
    5      |        A      |
    4      |        B      |
    3      |        C      |
    2      |        D      |
    1      |        E      |
    0      |        F      |
```





Exercise 3 : Postfix math

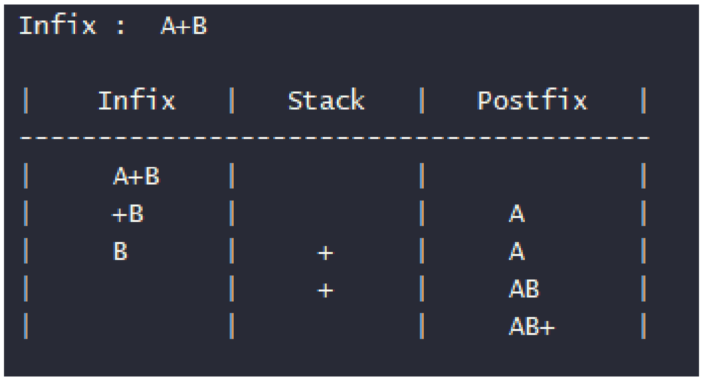
Code :



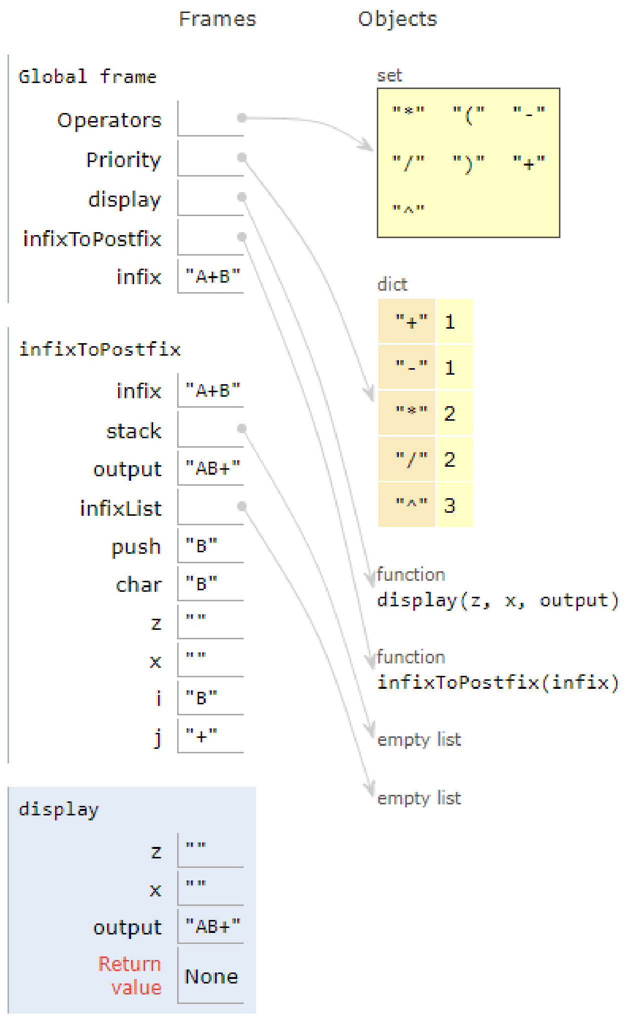
```python
Operators = set(['+', '-', '*', '/', '(', ')', '^'])
Priority = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}

def display(z,x,output):
    print("|    ", z, " " * (5 - len(z)),
          "|    ", x, " " * (4 - len(x)),
          "|    ", output, " " * (6 - len(output)), "|")

def infixToPostfix(infix):
    stack = []
    output = ''
    infixList = []

    for push in infix:
        infixList.append(push)
    print("|    Infix   |   Stack   |   Postfix   |")
    print("----------------------------------------")
    for char in infix:
        z = ''
        x = ''
        for i in infixList:
            z += i

        for j in stack:
          if len(stack) != 2:
            x += j
            x = x+"  "
          else:
            x += j

        if char == '(':
            display(z,x,output)
            stack.append('(')
            infixList.pop(0)

        elif char == ')':
            display(z,x,output)
            while stack and stack[-1] != '(':
                output += stack.pop()
            stack.pop()
            infixList.pop(0)

        elif char not in Operators:
            display(z,x,output)
            output += char
            infixList.pop(0)

        else:
            display(z,x,output)
            while stack and stack[-1] != '(' and Priority[char] <= Priority[stack[-1]]:
                output += stack.pop()
            stack.append(char)
            infixList.pop(0)

    z = ''
    for loop in infixList:
        z += loop
    display(z,x,output)
    while stack:
        x = ''
        output += stack.pop()
        for loop2 in stack:
            x += loop2
    display(z,x,output)

infix = "A+B"
print('Infix : ', infix, "\n")
infixToPostfix(infix)
```

Result :

A + B



```
Infix :  A+B

|    Infix   |   Stack   |   Postfix   |
-----------------------------------------
|     A+B    |           |             |
|     +B     |           |     A       |
|     B      |     +     |     A       |
|            |     +     |     AB      |
|            |           |     AB+     |
```



Frames Objects

Global frame

Operators	
Priority	
display	
infixToPostfix	
infix	"A+B"

infixToPostfix

infix	"A+B"
stack	
output	"AB+"
infixList	
push	"B"
char	"B"
z	""
x	""
i	"B"
j	"+"

display

z	""
x	""
output	"AB+"
Return value	None

set

"*" "(" "-"
"/" ")" "+"
"^"

dict

"+"	1
"-"	1
"*"	2
"/"	2
"^"	3

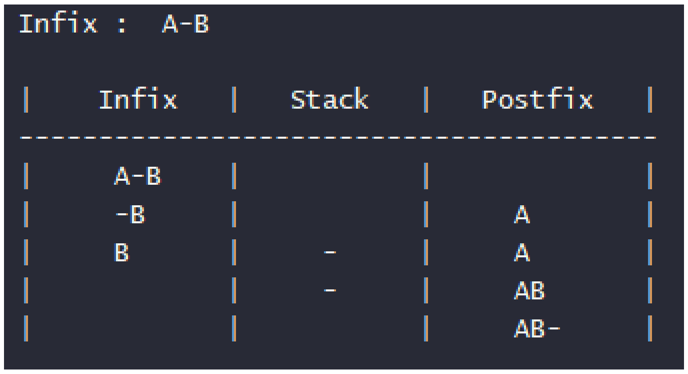
function
display(z, x, output)

function
infixToPostfix(infix)

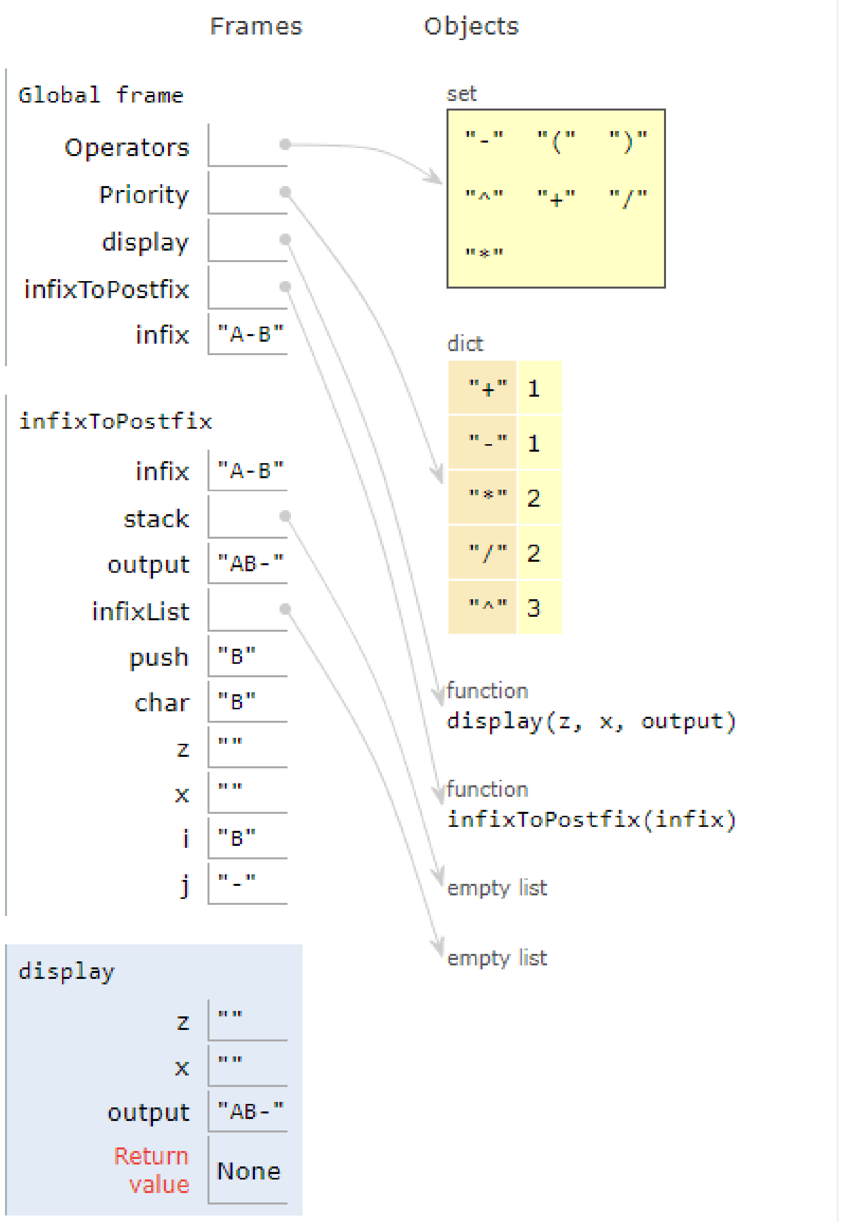
empty list

empty list

A - B



```
Infix :  A-B

|    Infix    |   Stack   |   Postfix   |
------------------------------------------
|     A-B     |           |             |
|     -B      |           |     A       |
|     B       |     -     |     A       |
|             |     -     |     AB      |
|             |           |     AB-     |
```



Frames   Objects

```
Global frame
    Operators    ●  →  set
     Priority    ●  →  dict
      display    ●  →  function display(z, x, output)
infixToPostfix   ●  →  function infixToPostfix(infix)
        infix    "A-B"

infixToPostfix
        infix    "A-B"
        stack    ●  →  empty list
       output    "AB-"
    infixList    ●  →  empty list
         push    "B"
         char    "B"
            z    ""
            x    ""
            i    "B"
            j    "-"

display
            z    ""
            x    ""
       output    "AB-"
 Return value    None
```

set

```
"-"   "("   ")"
"^"   "+"   "/"
"*"
```

dict

"+"	1
"-"	1
"*"	2
"/"	2
"^"	3

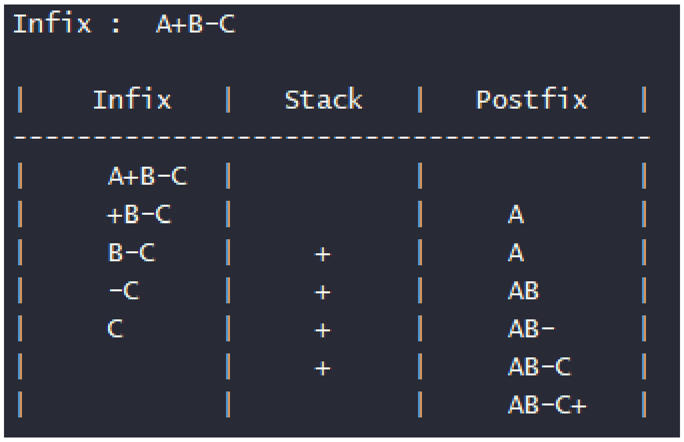
function
display(z, x, output)

function
infixToPostfix(infix)

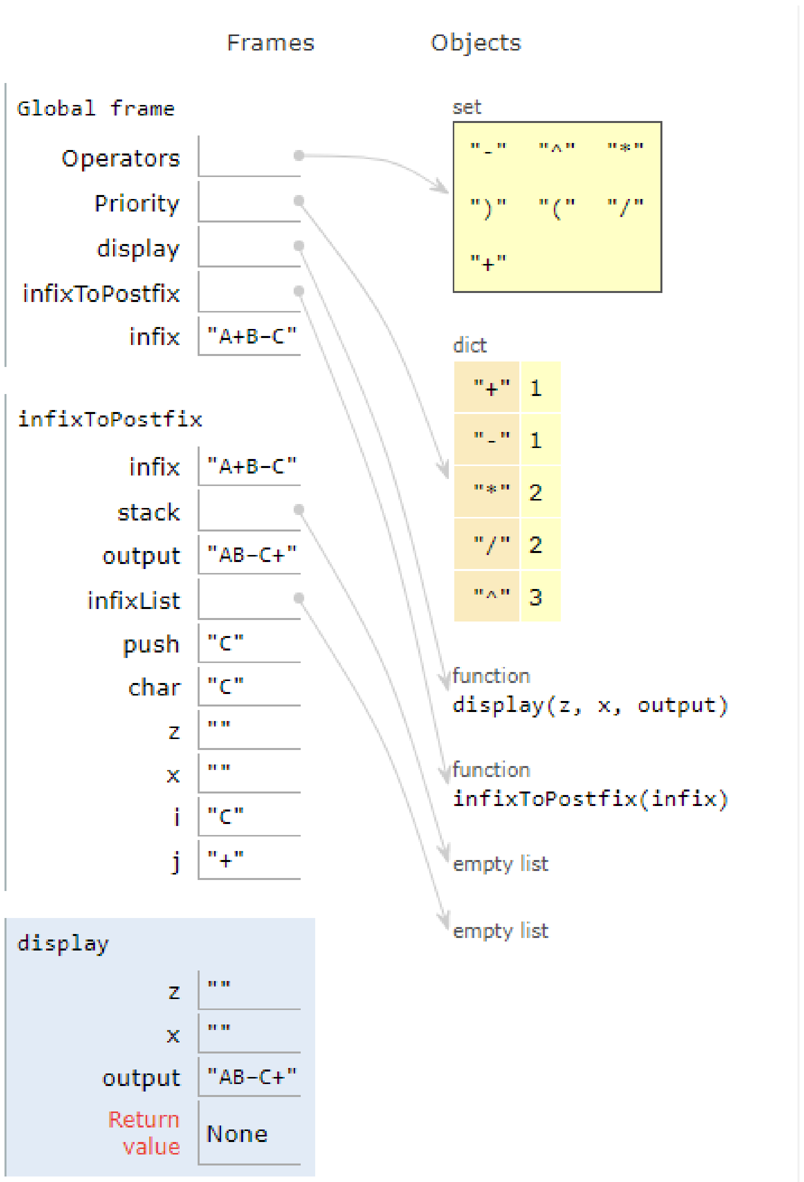
empty list

empty list

A + B - C

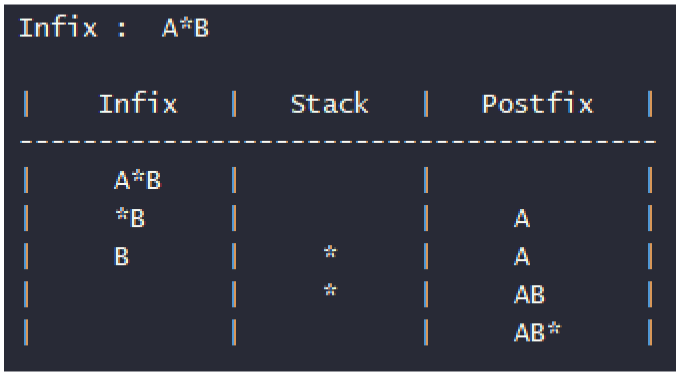


```
Infix :  A+B-C

|    Infix    |    Stack    |    Postfix    |
-------------------------------------------
|      A+B-C  |             |               |
|      +B-C   |             |     A         |
|      B-C    |      +      |     A         |
|      -C     |      +      |     AB        |
|      C      |      +      |     AB-       |
|             |      +      |     AB-C      |
|             |             |     AB-C+     |
```

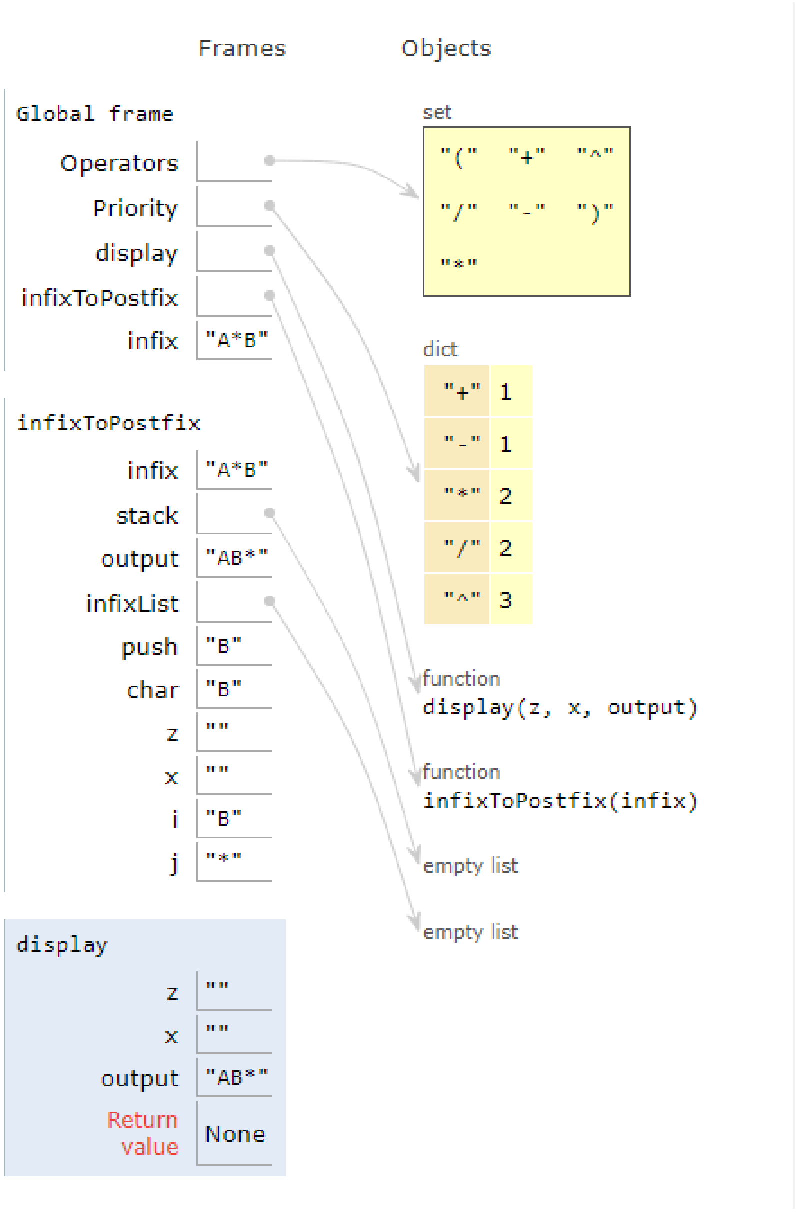


```
Frames                     Objects

Global frame               set
   Operators   •           "-"  "^"  "*"
   Priority    •           ")"  "("  "/"
   display     •           "+"
   infixToPostfix •
   infix  "A+B-C"          dict
                           "+"  1
infixToPostfix             "-"  1
   infix  "A+B-C"          "*"  2
   stack      •            "/"  2
   output  "AB-C+"         "^"  3
   infixList  •
   push  "C"               function
   char  "C"               display(z, x, output)
   z     ""
   x     ""                function
   i     "C"               infixToPostfix(infix)
   j     "+"
                           empty list
display
   z     ""                empty list
   x     ""
   output  "AB-C+"
   Return
    value  None
```

A * B



```
Infix :  A*B

|    Infix   |    Stack   |    Postfix   |
------------------------------------------
|     A*B    |            |              |
|     *B     |            |      A       |
|     B      |     *      |      A       |
|            |     *      |      AB      |
|            |            |      AB*     |
```



Frames Objects

Global frame

Operators	
Priority	
display	
infixToPostfix	
infix	"A*B"

infixToPostfix

infix	"A*B"
stack	
output	"AB*"
infixList	
push	"B"
char	"B"
z	""
x	""
i	"B"
j	"*"

display

z	""
x	""
output	"AB*"
Return value	None

set

"(" "+" "^"
"/" "-" ")"
"*"

dict

"+"	1
"-"	1
"*"	2
"/"	2
"^"	3

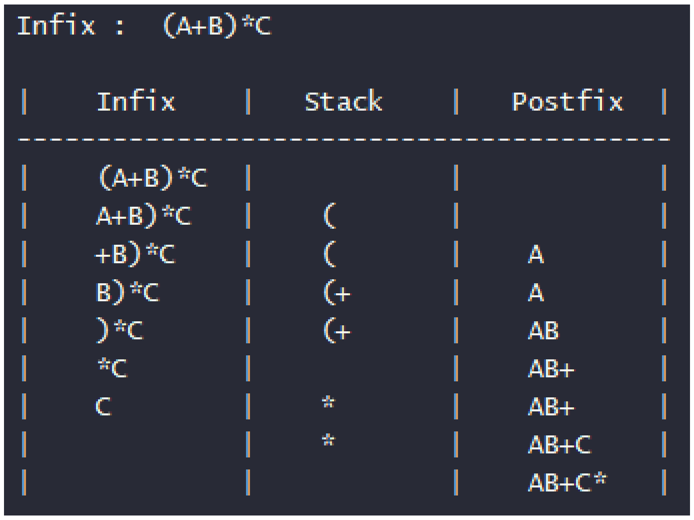
function
display(z, x, output)

function
infixToPostfix(infix)

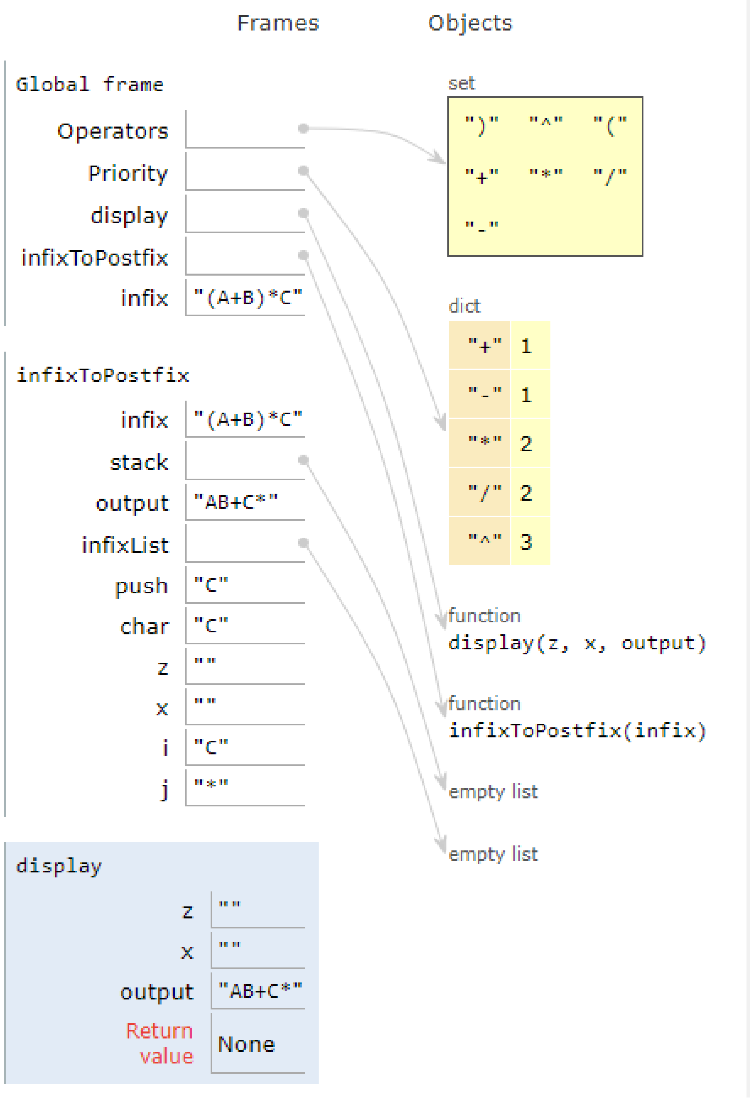
empty list

empty list

(A + B) * C

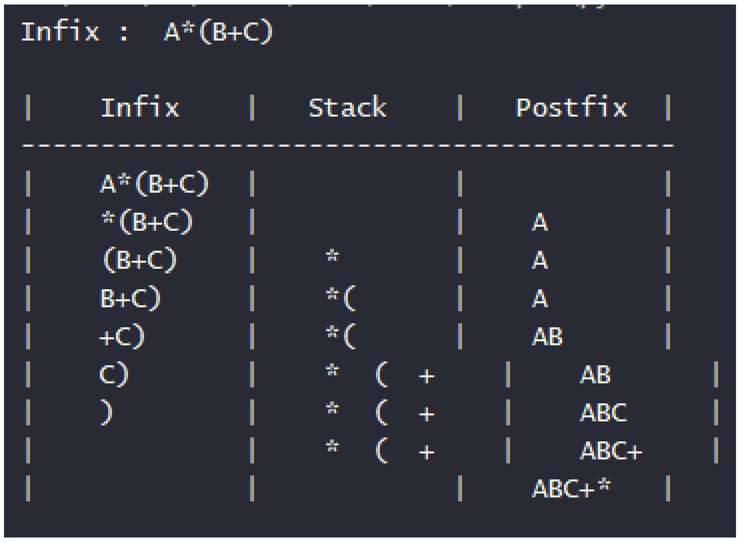


```
Infix :  (A+B)*C

|    Infix    |   Stack    |   Postfix  |
-----------------------------------------
|    (A+B)*C  |            |            |
|    A+B)*C   |   (        |            |
|    +B)*C    |   (        |    A       |
|    B)*C     |   (+       |    A       |
|    )*C      |   (+       |    AB      |
|    *C       |            |    AB+     |
|    C        |   *        |    AB+     |
|             |   *        |    AB+C    |
|             |            |    AB+C*   |
```

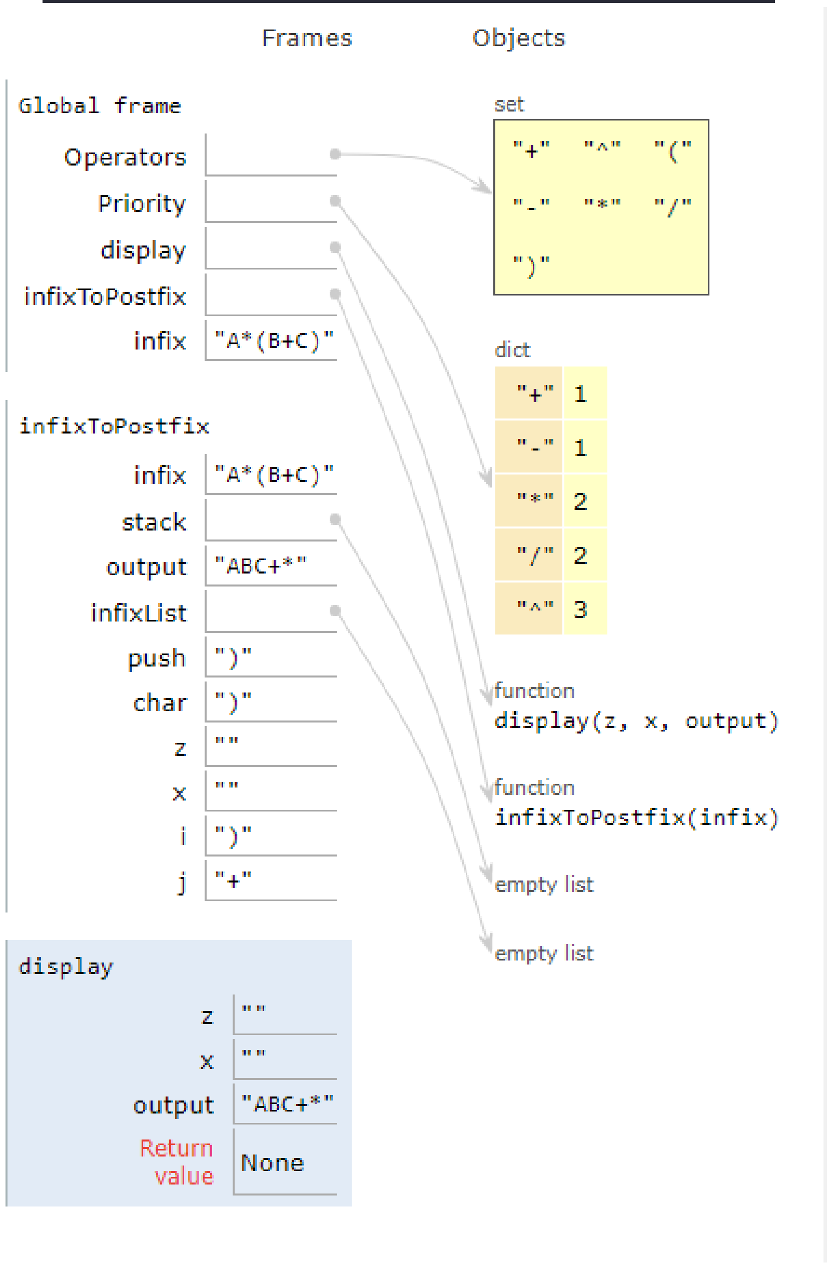


```
Global frame
   Operators   -> set {")", "^", "(", "+", "*", "/", "-"}
   Priority    -> dict {"+": 1, "-": 1, "*": 2, "/": 2, "^": 3}
   display     -> function display(z, x, output)
   infixToPostfix -> function infixToPostfix(infix)
   infix       "(A+B)*C"

infixToPostfix
   infix       "(A+B)*C"
   stack       -> empty list
   output      "AB+C*"
   infixList   -> empty list
   push        "C"
   char        "C"
   z           ""
   x           ""
   i           "C"
   j           "*"

display
   z           ""
   x           ""
   output      "AB+C*"
   Return value  None
```

A * (B + C)



```
Infix :  A*(B+C)

|    Infix    |    Stack    |    Postfix   |
-------------------------------------------
|    A*(B+C)  |             |              |
|    *(B+C)   |             |   A          |
|    (B+C)    |    *        |   A          |
|    B+C)     |    *(       |   A          |
|    +C)      |    *(       |   AB         |
|    C)       |    *  (  +     |   AB         |
|    )        |    *  (  +     |   ABC        |
|             |    *  (  +     |   ABC+       |
|             |             |   ABC+*      |
```



Frames Objects

Global frame

Operators	
Priority	
display	
infixToPostfix	
infix	"A*(B+C)"

infixToPostfix

infix	"A*(B+C)"
stack	
output	"ABC+*"
infixList	
push	")"
char	")"
z	""
x	""
i	")"
j	"+"

display

z	""
x	""
output	"ABC+*"
Return value	None

set

"+" "^" "("
"-" "*" "/"
")"

dict

"+"	1
"-"	1
"*"	2
"/"	2
"^"	3

function
display(z, x, output)

function
infixToPostfix(infix)

empty list

empty list