Lecture 5
Queue

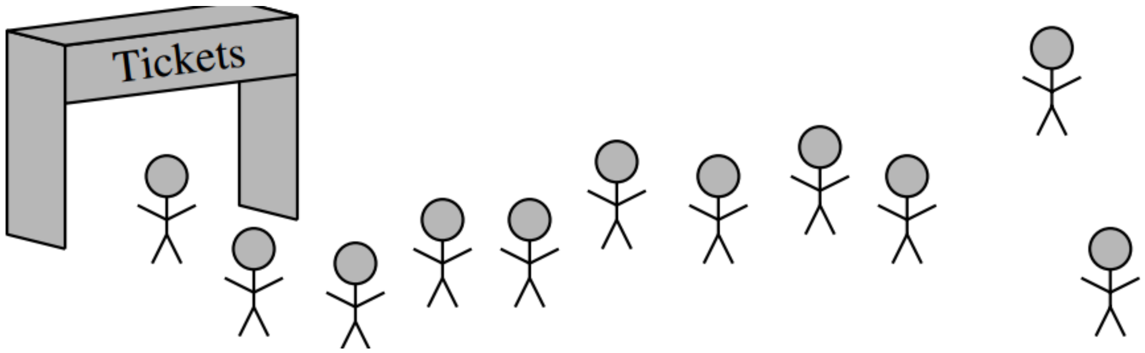
Dr. Sara S. Elhishi

Mansoura University

Sara_shaker2008@mans.edu.eg

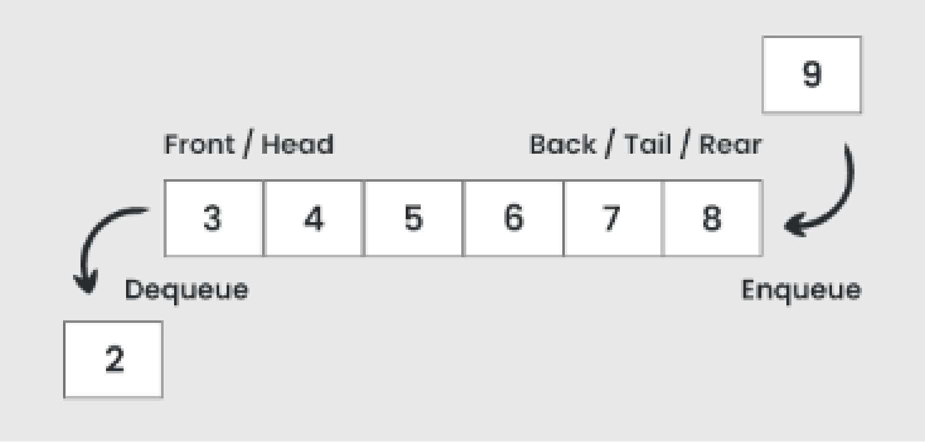
Queue

- A data that follows the principle of first-in, first-out (FIFO)
- The item that is placed first is the one that will be withdrawn first.
- A queue would, therefore, be a logical choice for a data structure to handle calls to a customer service center, a waitlist at a restaurant, or a networked printer.



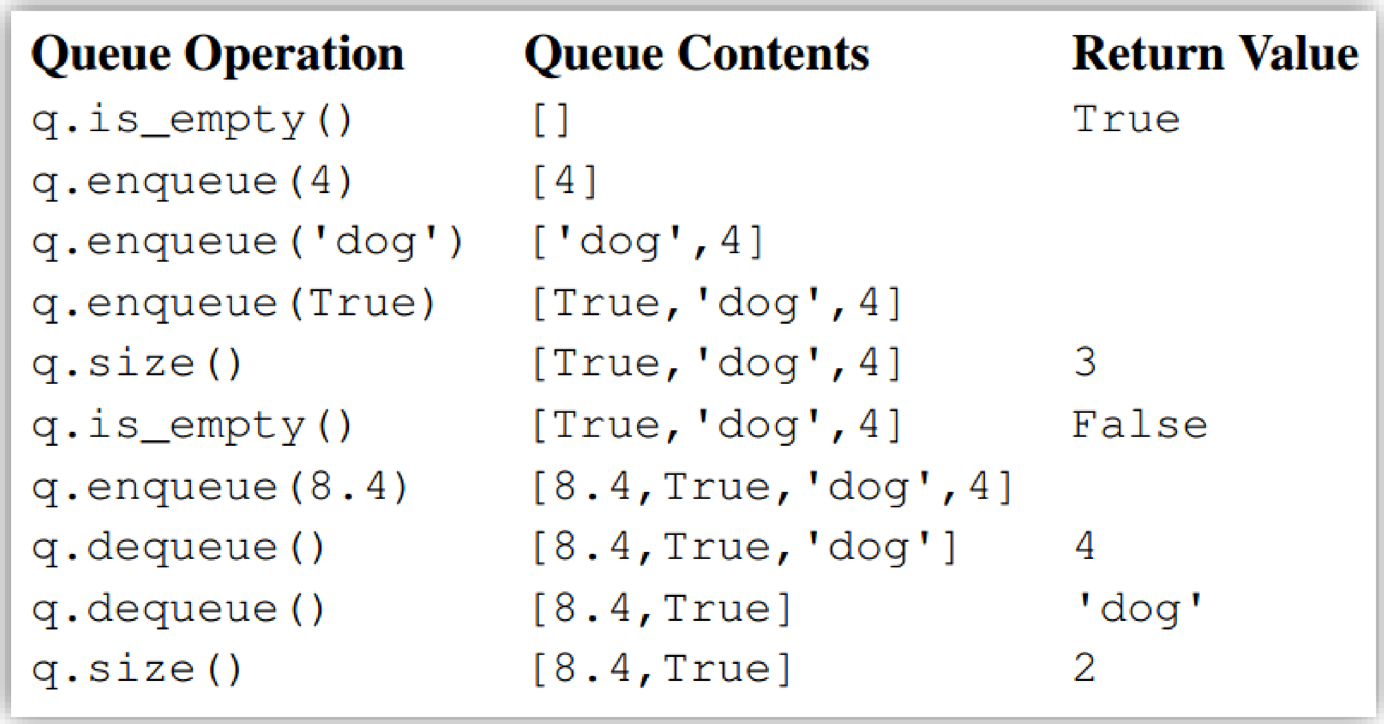
Queue Operations

- Enqueue: Inserting an element into the queue
- Dequeue: Removing an element from the queue



Queue ADT

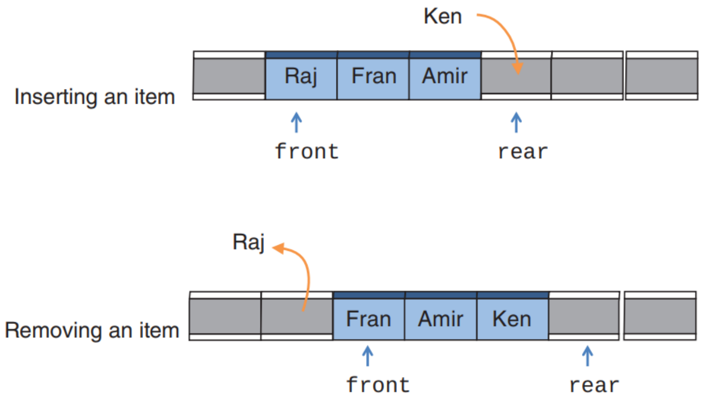
- queue()
- enqueue(item)
- dequeue()
- is_empty()
- size()



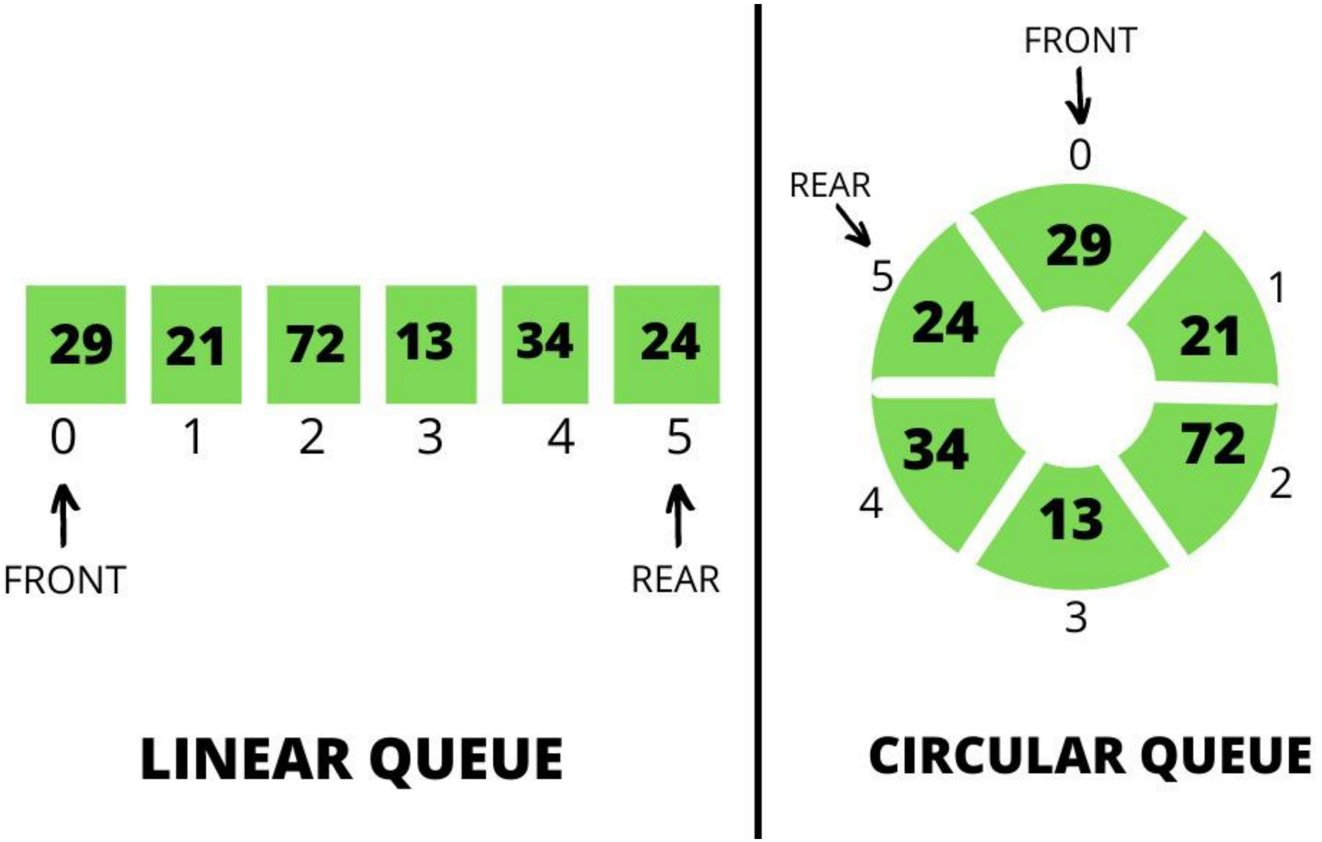
Queue Operation	Queue Contents	Return Value
`q.is_empty()`	`[]`	`True`
`q.enqueue(4)`	`[4]`	
`q.enqueue('dog')`	`['dog',4]`	
`q.enqueue(True)`	`[True,'dog',4]`	
`q.size()`	`[True,'dog',4]`	`3`
`q.is_empty()`	`[True,'dog',4]`	`False`
`q.enqueue(8.4)`	`[8.4,True,'dog',4]`	
`q.dequeue()`	`[8.4,True,'dog']`	`4`
`q.dequeue()`	`[8.4,True]`	`'dog'`
`q.size()`	`[8.4,True]`	`2`

Implementation Problem

- Implementing using arrays involves updating indices which is easy and efficient.
- What happens when you reach the end?
- O(N) instead of O(1)
- Is there a method to prevent the occurrence of shifts?



Circular Queue

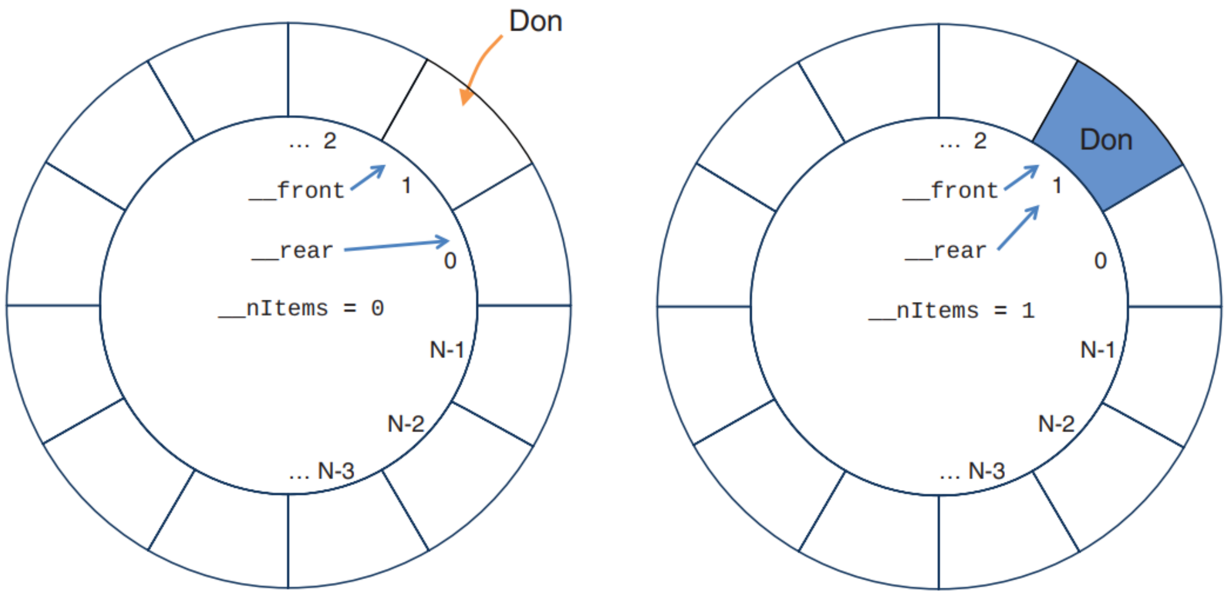


LINEAR QUEUE

CIRCULAR QUEUE

Circular Queue

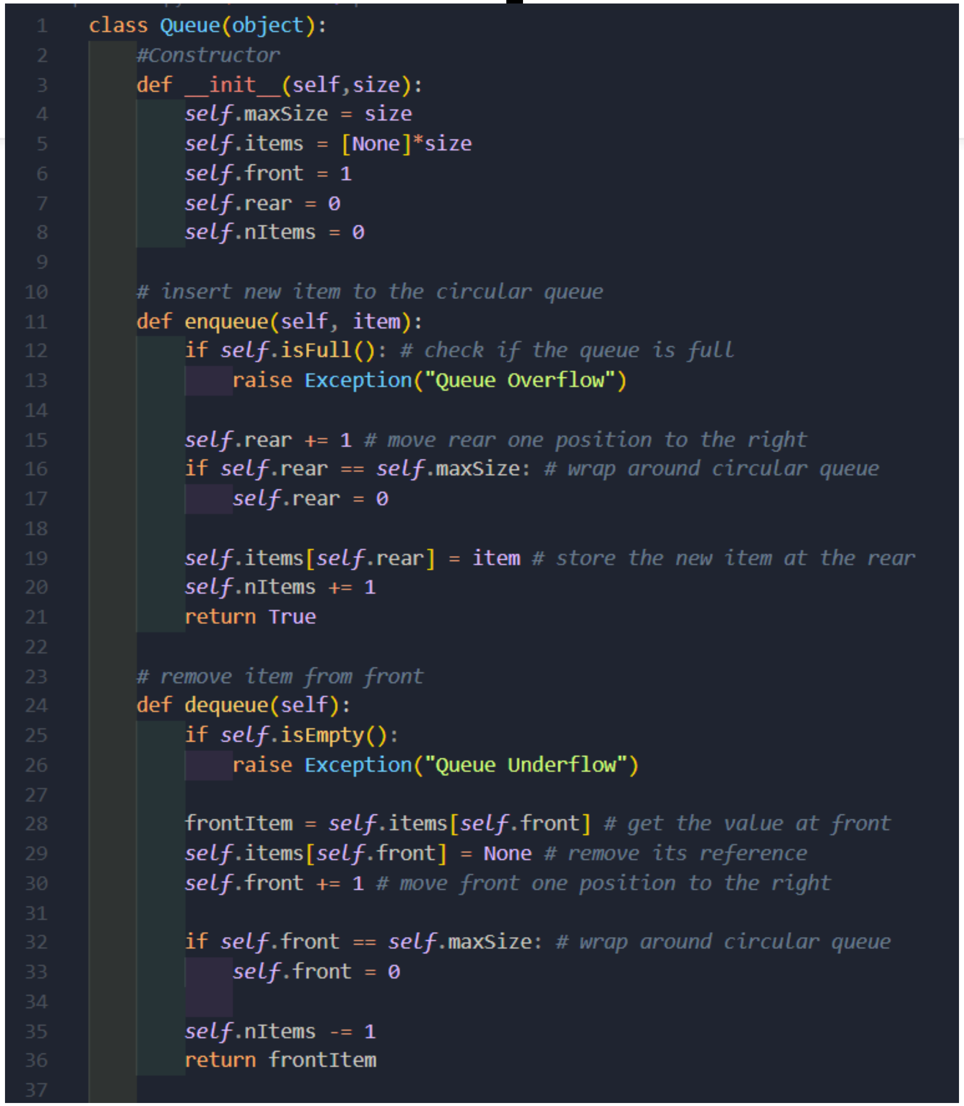
- Allow the front and rear pointers to wrap around to the start of the array
- Arrange a series of cells in a circular shape, such that the last cell and the first cell are positioned next to each other



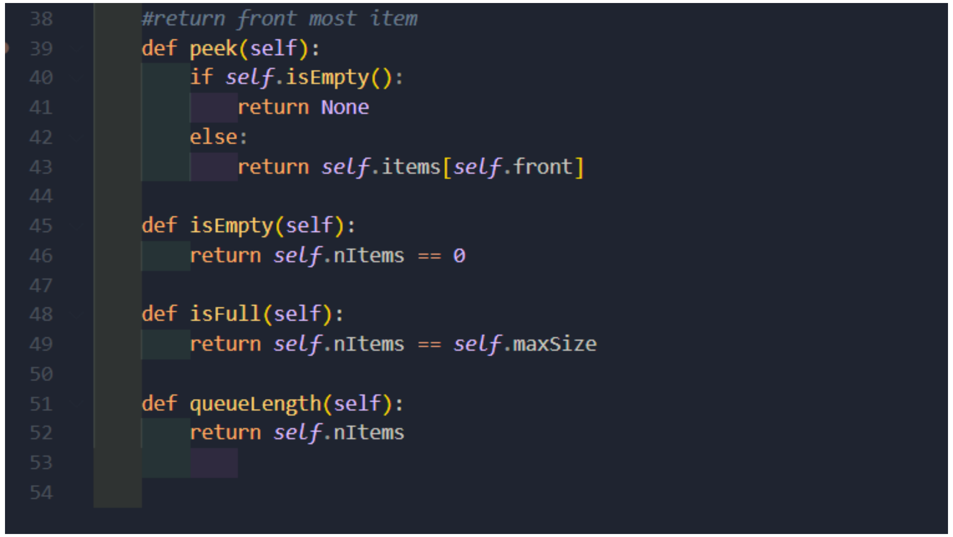
Empty queue before first insert

First item in queue

Queue Implementation

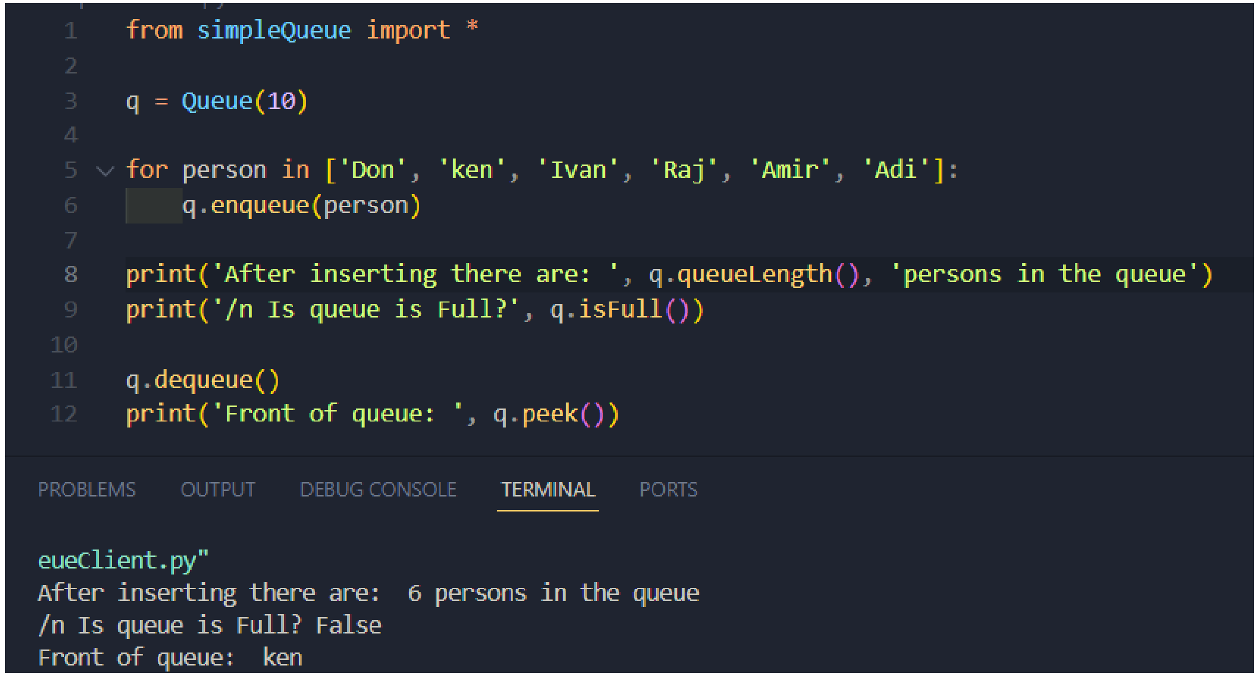


```python
class Queue(object):
    #Constructor
    def __init__(self,size):
        self.maxSize = size
        self.items = [None]*size
        self.front = 1
        self.rear = 0
        self.nItems = 0

    # insert new item to the circular queue
    def enqueue(self, item):
        if self.isFull(): # check if the queue is full
            raise Exception("Queue Overflow")

        self.rear += 1 # move rear one position to the right
        if self.rear == self.maxSize: # wrap around circular queue
            self.rear = 0

        self.items[self.rear] = item # store the new item at the rear
        self.nItems += 1
        return True

    # remove item from front
    def dequeue(self):
        if self.isEmpty():
            raise Exception("Queue Underflow")

        frontItem = self.items[self.front] # get the value at front
        self.items[self.front] = None # remove its reference
        self.front += 1 # move front one position to the right

        if self.front == self.maxSize: # wrap around circular queue
            self.front = 0

        self.nItems -= 1
        return frontItem

```



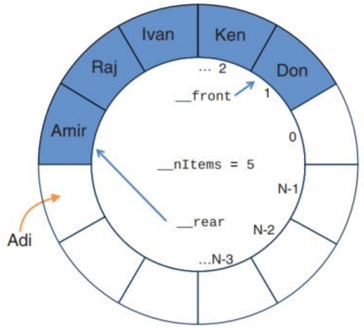
```python
    #return front most item
    def peek(self):
        if self.isEmpty():
            return None
        else:
            return self.items[self.front]

    def isEmpty(self):
        return self.nItems == 0

    def isFull(self):
        return self.nItems == self.maxSize

    def queueLength(self):
        return self.nItems

```

Test of the Queue

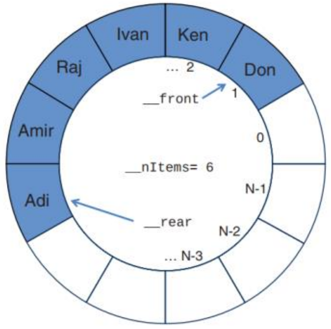


```python
from simpleQueue import *

q = Queue(10)

for person in ['Don', 'ken', 'Ivan', 'Raj', 'Amir', 'Adi']:
    q.enqueue(person)

print('After inserting there are: ', q.queueLength(), 'persons in the queue')
print('/n Is queue is Full?', q.isFull())

q.dequeue()
print('Front of queue: ', q.peek())
```

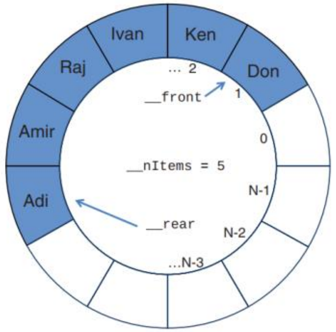
```
eueClient.py"
After inserting there are:  6 persons in the queue
/n Is queue is Full? False
Front of queue:  ken
```



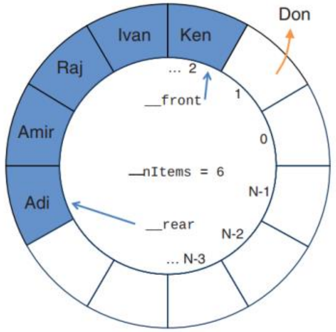
Before inserting last item in queue



Last item inserted in queue



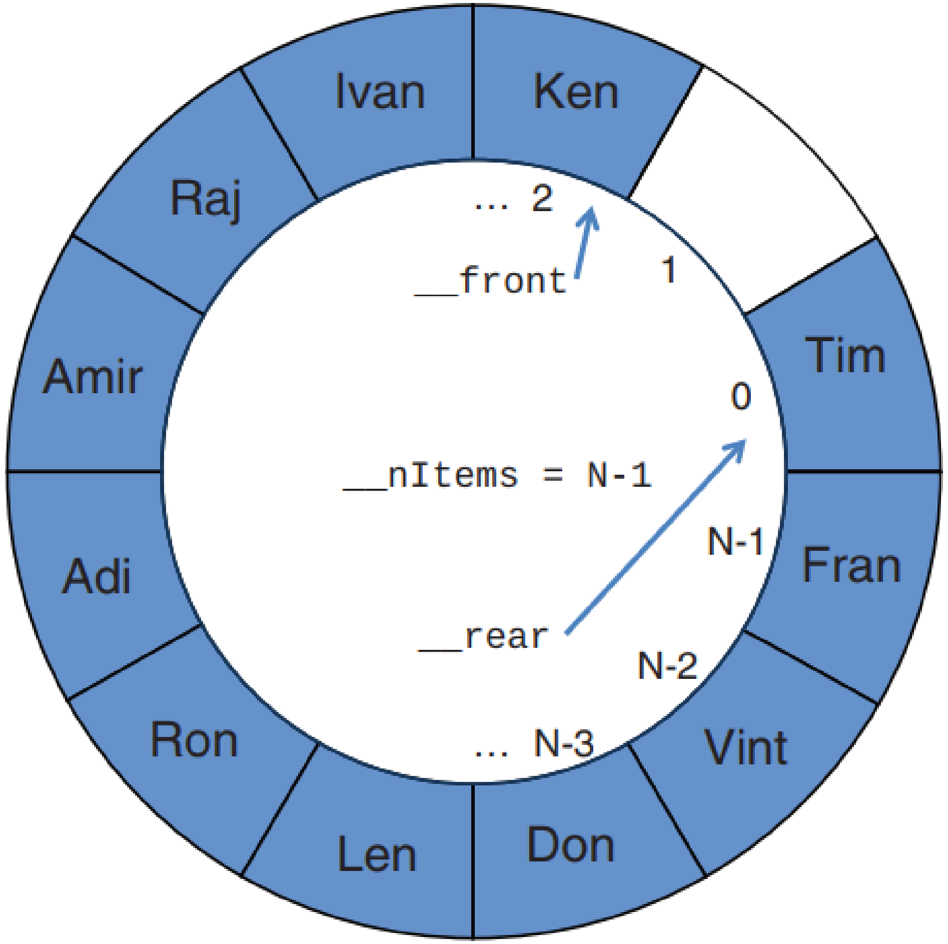
Before deleting item in queue



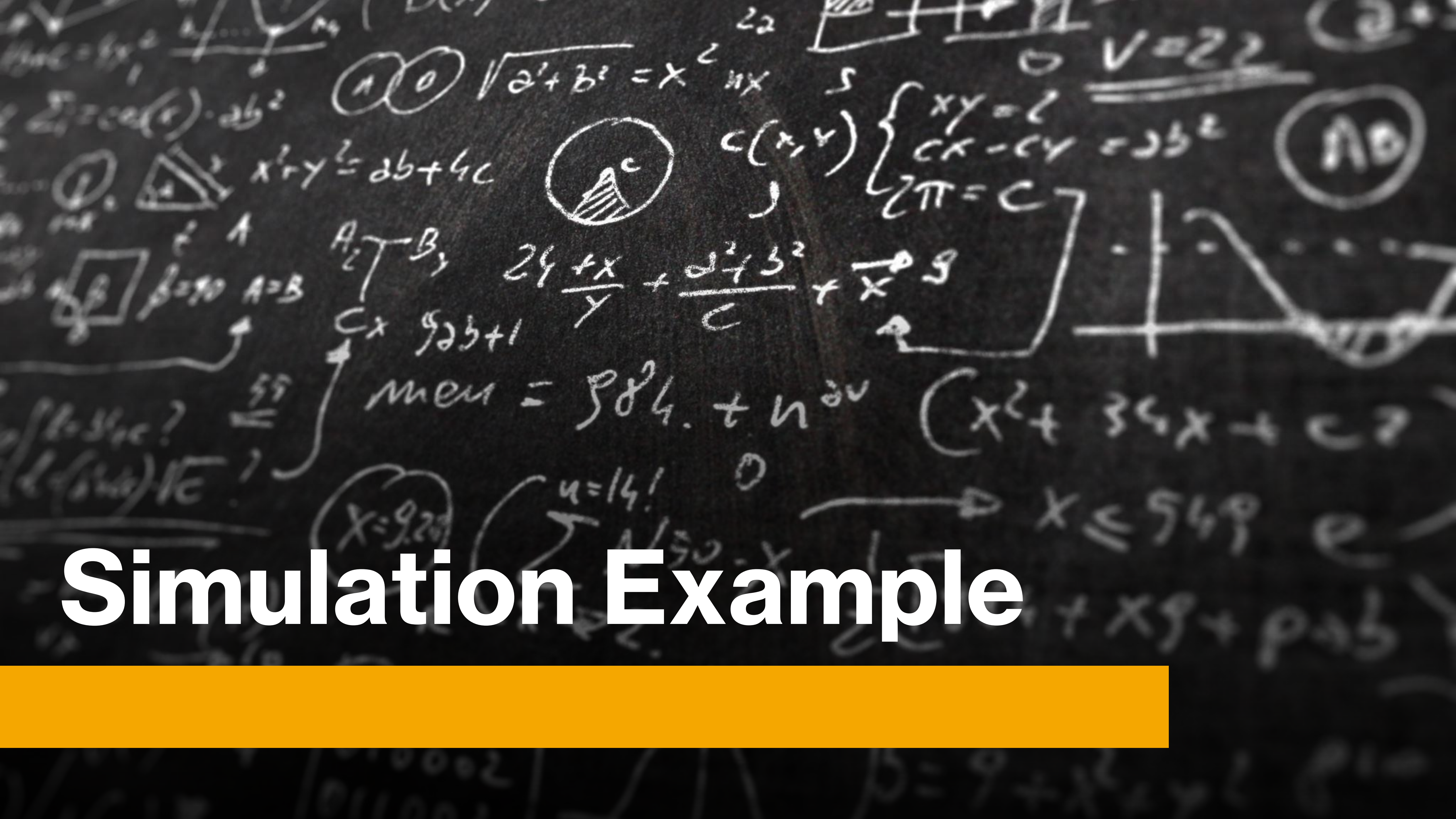
First item deleted from queue

Queue Wraps Around The Circular Array

- By removing a single name and subsequently adding more names to the queue.
- 'Tim', is positioned at the start of the underlying array
- One more item will be placed at 1! Same as the start empty
- To keep track of how many positions left: (front-rear) -1



Items in queue wrapping around

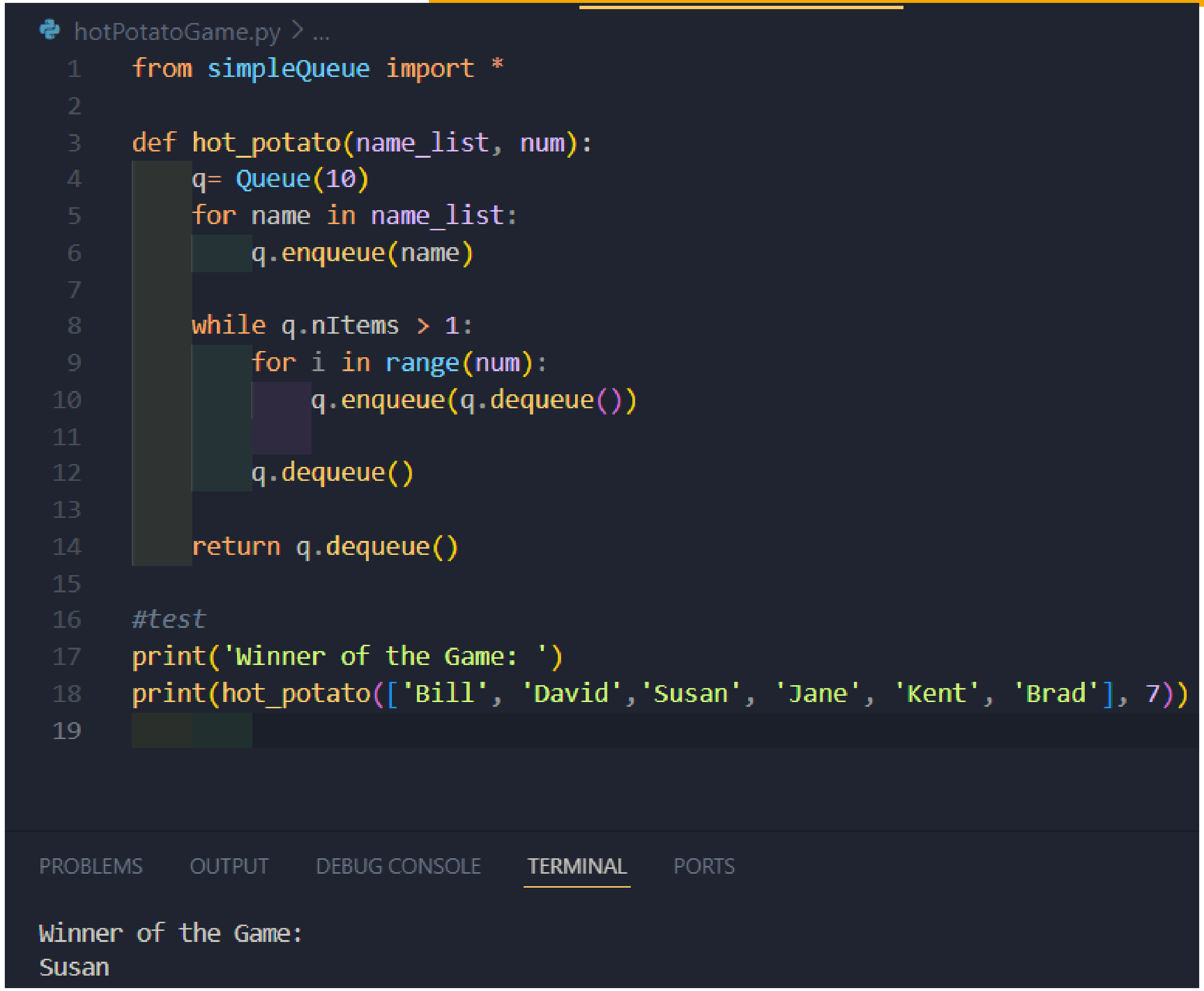


Simulation Example

Hot Potato Game

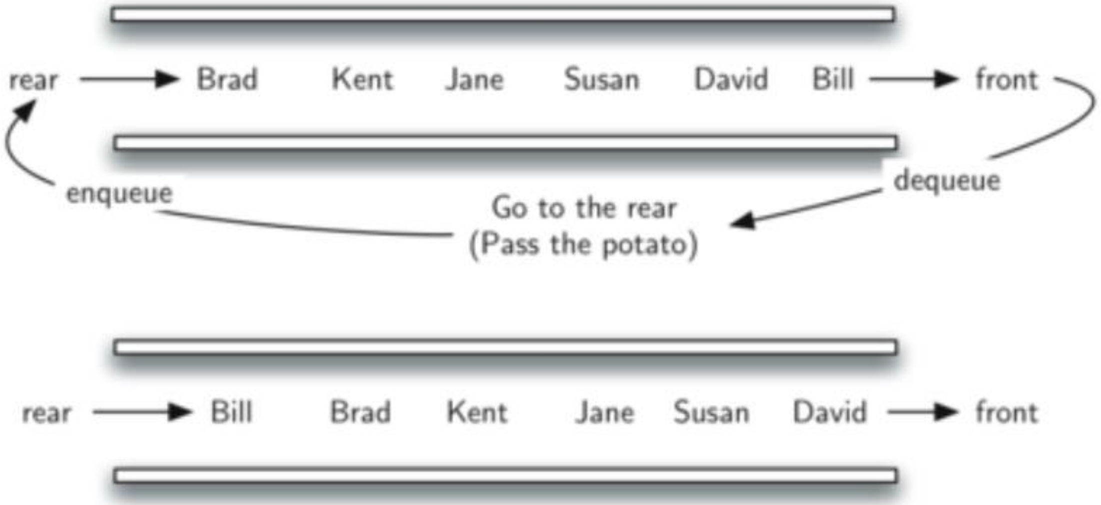
- In this children line up in a circle and pass an item from neighbour to neighbour as fast as they can.
- At a certain point in the game, the action is stopped and the child who has the item (the potato) is removed from the circle.

Hot Potato Game Using a Queue



```python
from simpleQueue import *

def hot_potato(name_list, num):
    q= Queue(10)
    for name in name_list:
        q.enqueue(name)

    while q.nItems > 1:
        for i in range(num):
            q.enqueue(q.dequeue())

        q.dequeue()

    return q.dequeue()

#test
print('Winner of the Game: ')
print(hot_potato(['Bill', 'David','Susan', 'Jane', 'Kent', 'Brad'], 7))
```

```
Winner of the Game:
Susan
```

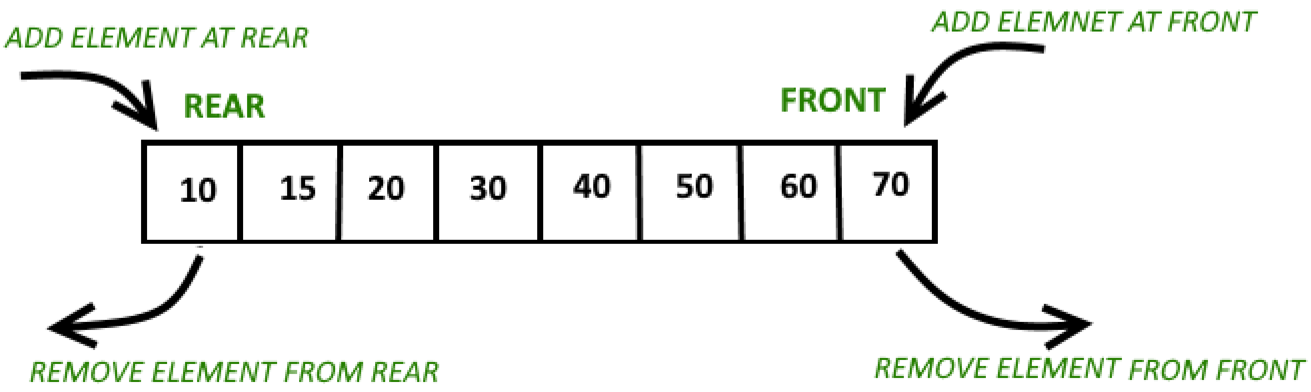


Queues Variations

Deque, Priority Queue

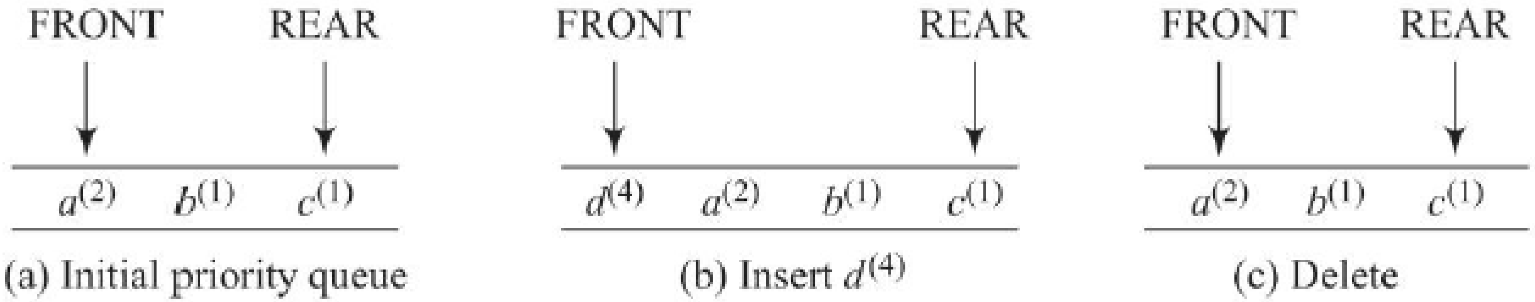
Deque (Double-Ended Queue)

- A **deque** is a data structure that allows the insertion and removal of elements from both ends.
- insertLeft(), insertRight(), removeLeft(), and removeRight().
- More flexible, but not utilized as much as Stack and Queues.



Priority Queue

- In a priority queue, things are arranged according to their priority value.
- First-In-First-Out (FIFO) ordering, while also possessing the sorting behavior
- Highest priority at the Front.
- Equal priorities follow a FIFO.
- Ex: Handle mail messages.


FRONT
REAR
$a^{(2)}$ $b^{(1)}$ $c^{(1)}$
FRONT
REAR
$d^{(4)}$ $a^{(2)}$ $b^{(1)}$ $c^{(1)}$
FRONT
REAR
$a^{(2)}$ $b^{(1)}$ $c^{(1)}$

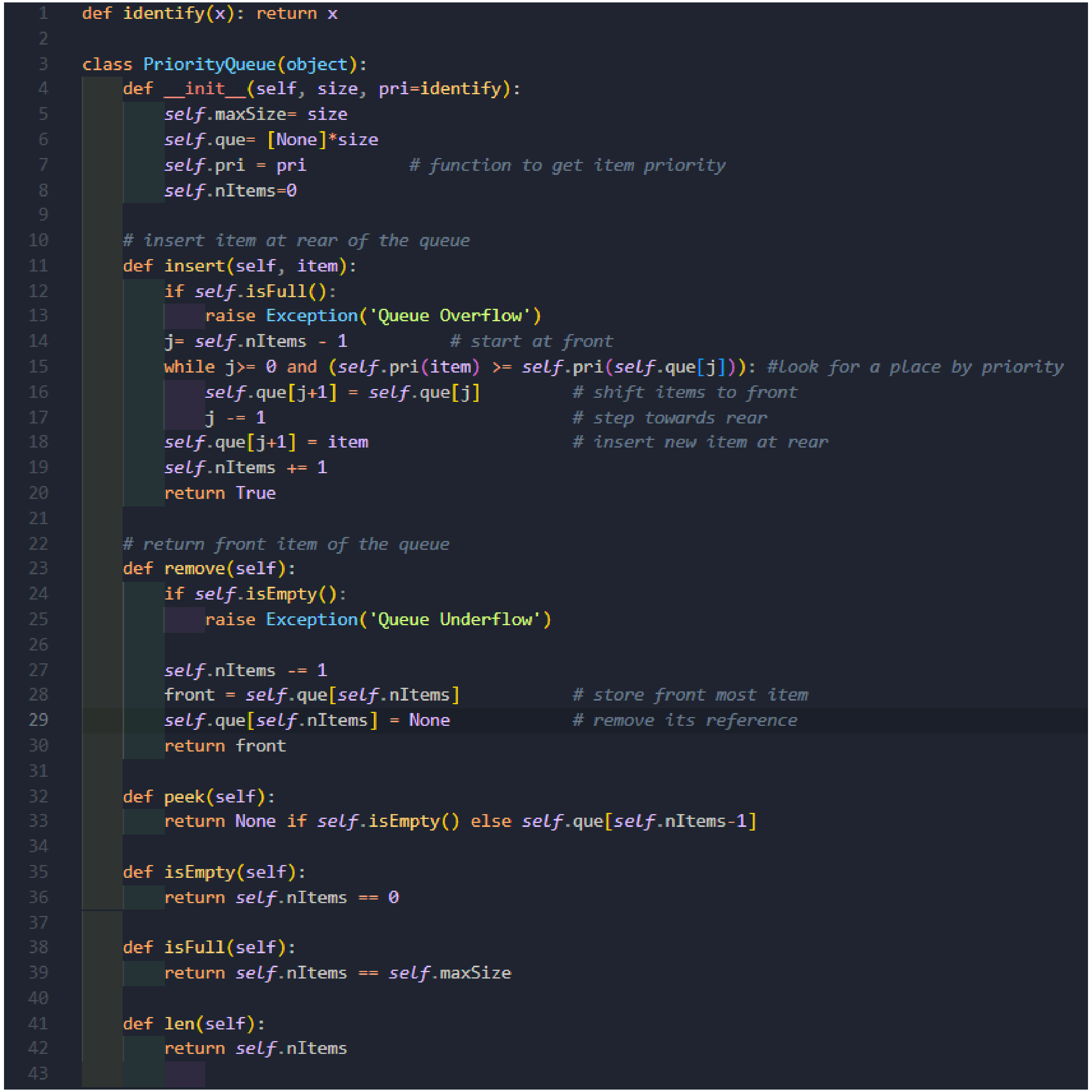
(a) Initial priority queue

(b) Insert $d^{(4)}$

(c) Delete

$x^{(y)}$: x is the element with priority y.

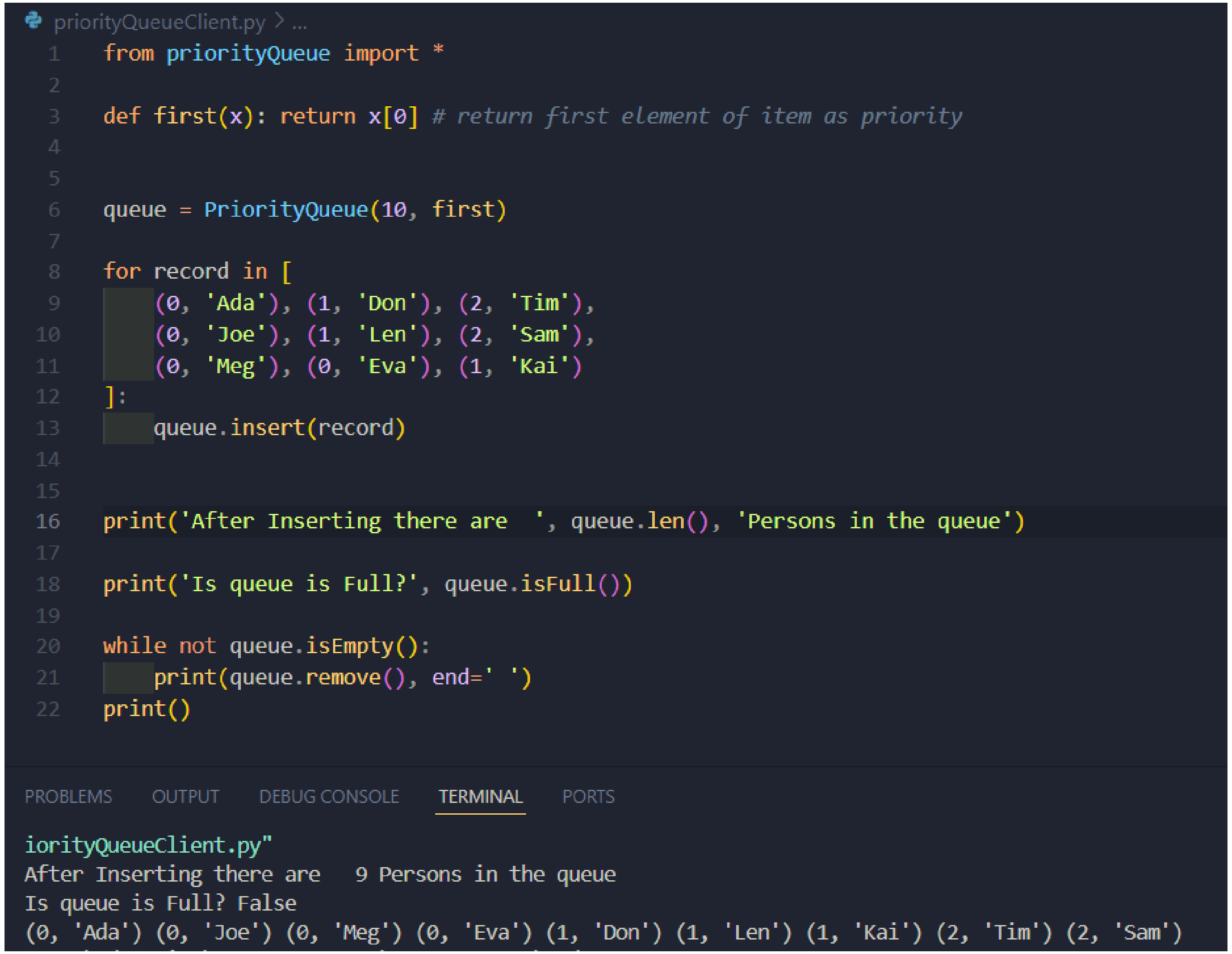
Priority Queue In Python



```python
def identify(x): return x

class PriorityQueue(object):
    def __init__(self, size, pri=identify):
        self.maxSize= size
        self.que= [None]*size
        self.pri = pri          # function to get item priority
        self.nItems=0

    # insert item at rear of the queue
    def insert(self, item):
        if self.isFull():
            raise Exception('Queue Overflow')
        j= self.nItems - 1          # start at front
        while j>= 0 and (self.pri(item) >= self.pri(self.que[j])): #look for a place by priority
            self.que[j+1] = self.que[j]         # shift items to front
            j -= 1                              # step towards rear
        self.que[j+1] = item                    # insert new item at rear
        self.nItems += 1
        return True

    # return front item of the queue
    def remove(self):
        if self.isEmpty():
            raise Exception('Queue Underflow')

        self.nItems -= 1
        front = self.que[self.nItems]           # store front most item
        self.que[self.nItems] = None            # remove its reference
        return front

    def peek(self):
        return None if self.isEmpty() else self.que[self.nItems-1]

    def isEmpty(self):
        return self.nItems == 0

    def isFull(self):
        return self.nItems == self.maxSize

    def len(self):
        return self.nItems

```

Test Priority Queue

- inserts tuples of the form (*priority, name*) into the PriorityQueue object, defining the first element of those tuples to be the priority



```python
from priorityQueue import *

def first(x): return x[0] # return first element of item as priority


queue = PriorityQueue(10, first)

for record in [
    (0, 'Ada'), (1, 'Don'), (2, 'Tim'),
    (0, 'Joe'), (1, 'Len'), (2, 'Sam'),
    (0, 'Meg'), (0, 'Eva'), (1, 'Kai')
]:
    queue.insert(record)


print('After Inserting there are  ', queue.len(), 'Persons in the queue')

print('Is queue is Full?', queue.isFull())

while not queue.isEmpty():
    print(queue.remove(), end=' ')
print()
```

```
iorityQueueClient.py"
After Inserting there are   9 Persons in the queue
Is queue is Full? False
(0, 'Ada') (0, 'Joe') (0, 'Meg') (0, 'Eva') (1, 'Don') (1, 'Len') (1, 'Kai') (2, 'Tim') (2, 'Sam')
```

Search and Traversal

- Shouldn’t we examine how search and traversal work on stacks, queues, and priority queues?
- Stacks and Queues are designed for insertion and removal
- If an application needs traversal, it’s likely to use another data structure.

Parsing Arithmetic Expressions

Usecase Example

Arithmetic Expression Recap

The DelimiterChecker.py program shows how a stack could be used to check whether delimiters were formatted correctly.

((())) √ while (())) X

Now, We need to upgrade our app to consider checking and evaluating a whole expression, such as:

(2+3)* 2 = 10

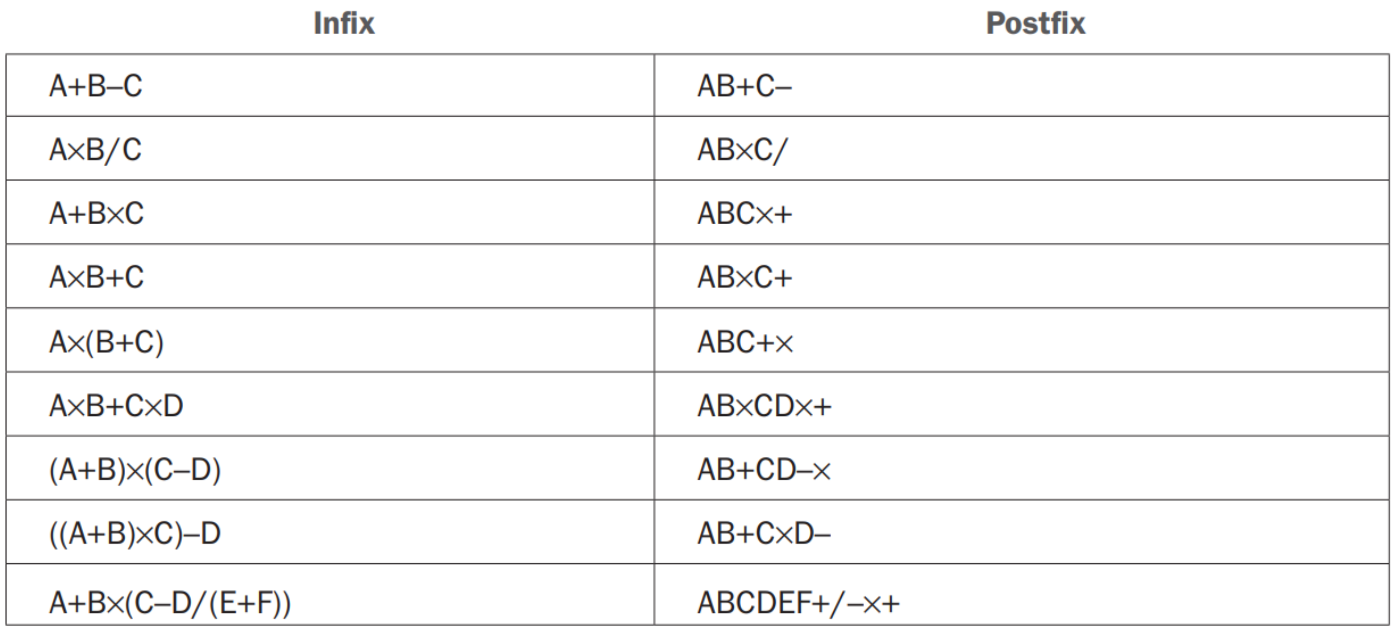
2-Step Approach

✓ Convert the arithmetic expression into postfix notation (e.g., 2+3 → 23+)

Determine the value of the postfix expression (e.g., 23+ → 5)

Step 1: Postfix Notation

- **Postfix** notation is a mathematical notation in which the operator comes after the two operands.
- The expression A+B is simplified to AB+



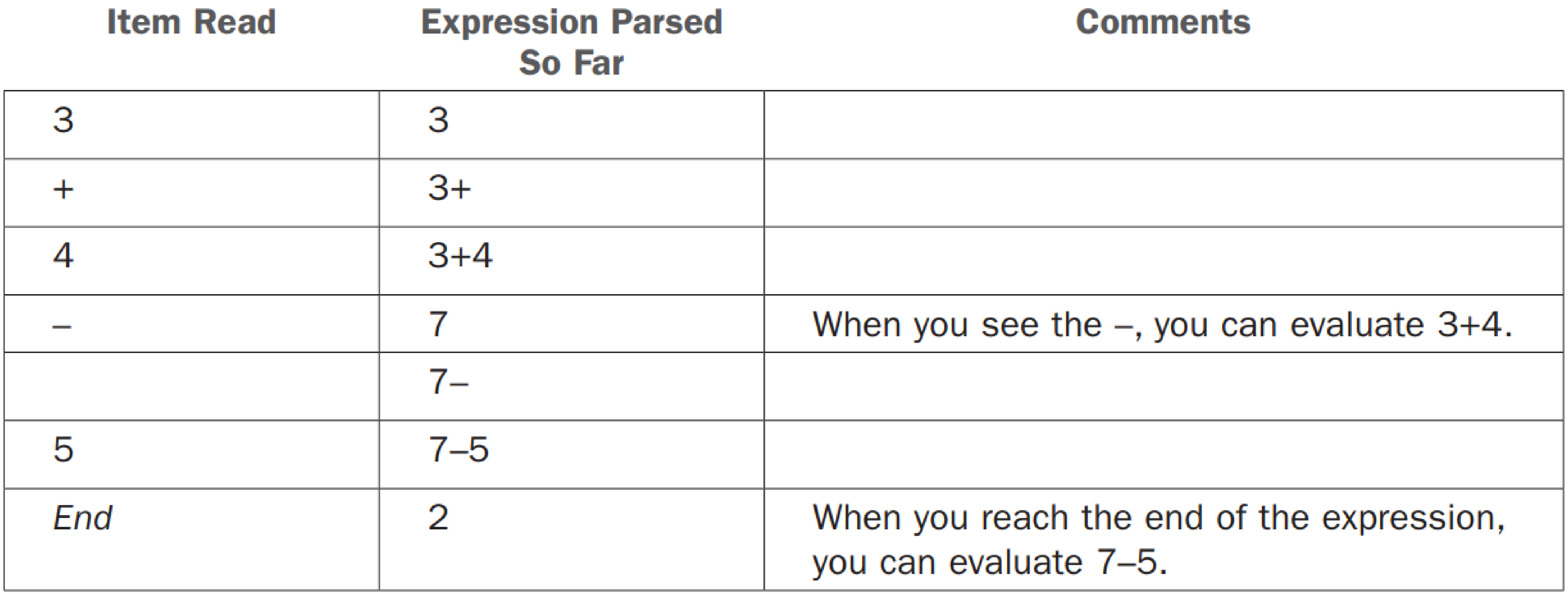
Infix	Postfix
A+B–C	AB+C–
A×B/C	AB×C/
A+B×C	ABC×+
A×B+C	AB×C+
A×(B+C)	ABC+×
A×B+C×D	AB×CD×+
(A+B)×(C–D)	AB+CD–×
((A+B)×C)–D	AB+C×D–
A+B×(C–D/(E+F))	ABCDEF+/–×+

Translating Infix To Postfix

How Human Evaluate Infix?

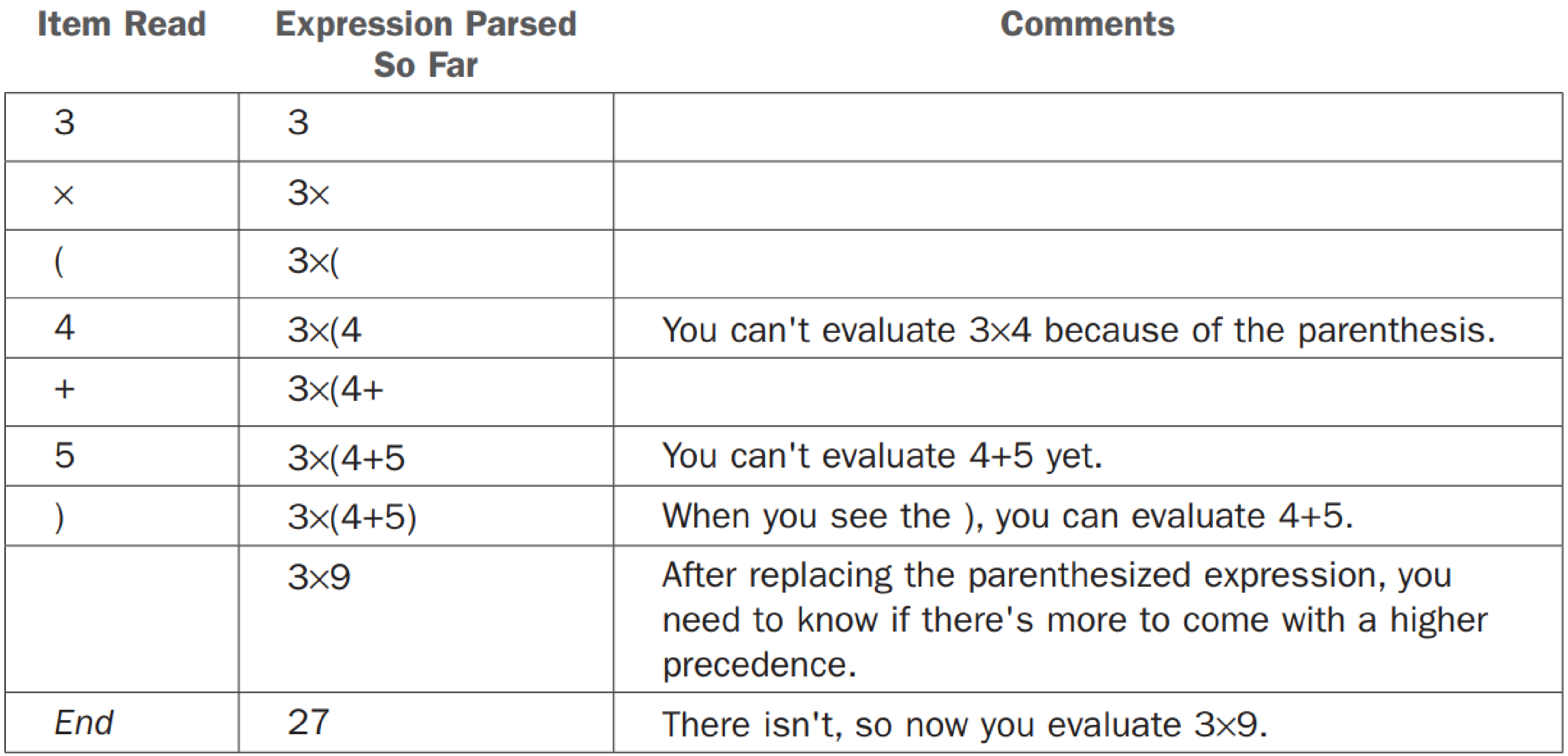
- Read sequentially from left to right.
- Once you have information on evaluating 2 operands and an operator, you replace them with the result
- Proceed with the same procedure until the end.

Evaluate 3 + 4 - 5



Item Read	Expression Parsed So Far	Comments
3	3	
+	3+	
4	3+4	
–	7	When you see the –, you can evaluate 3+4.
	7–	
5	7–5	
End	2	When you reach the end of the expression, you can evaluate 7–5.

Evaluate 3 * (4+5)

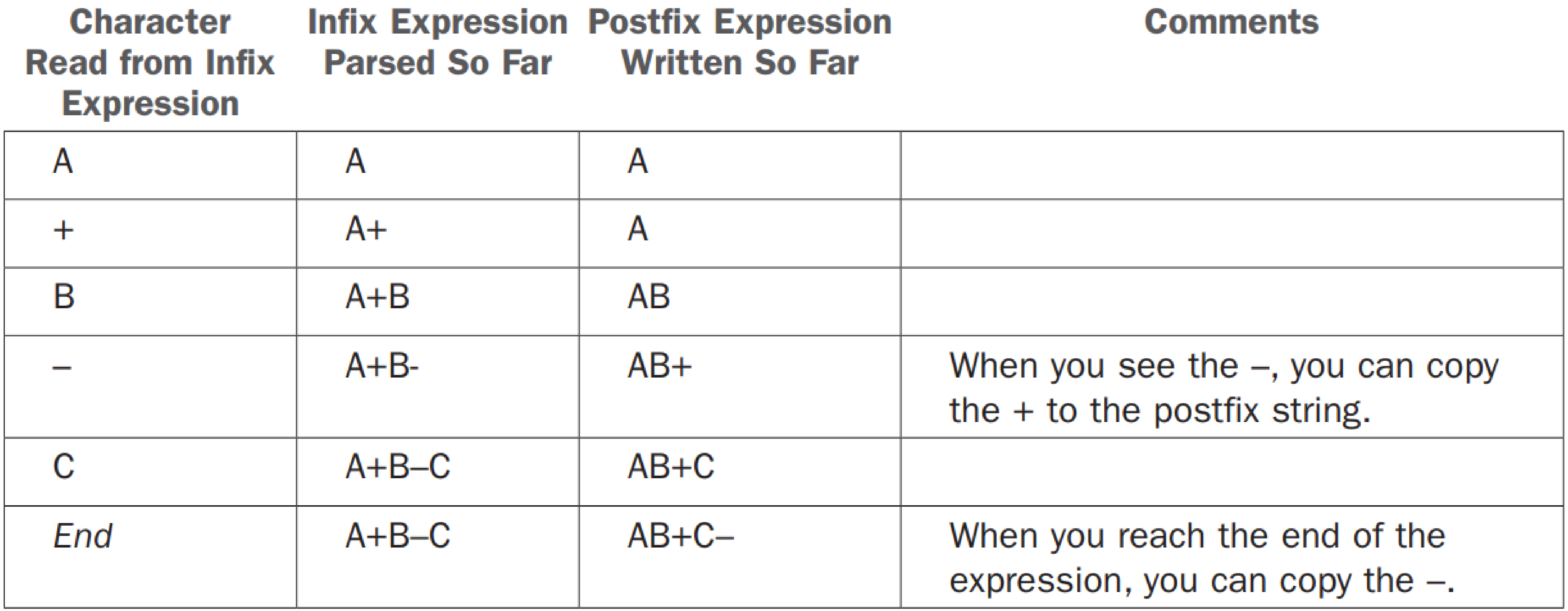


Item Read	Expression Parsed So Far	Comments
3	3	
×	3×	
(	3×(	
4	3×(4	You can't evaluate 3×4 because of the parenthesis.
+	3×(4+	
5	3×(4+5	You can't evaluate 4+5 yet.
)	3×(4+5)	When you see the), you can evaluate 4+5.
	3×9	After replacing the parenthesized expression, you need to know if there's more to come with a higher precedence.
End	27	There isn't, so now you evaluate 3×9.

How Computers Translate Infix To Prefix

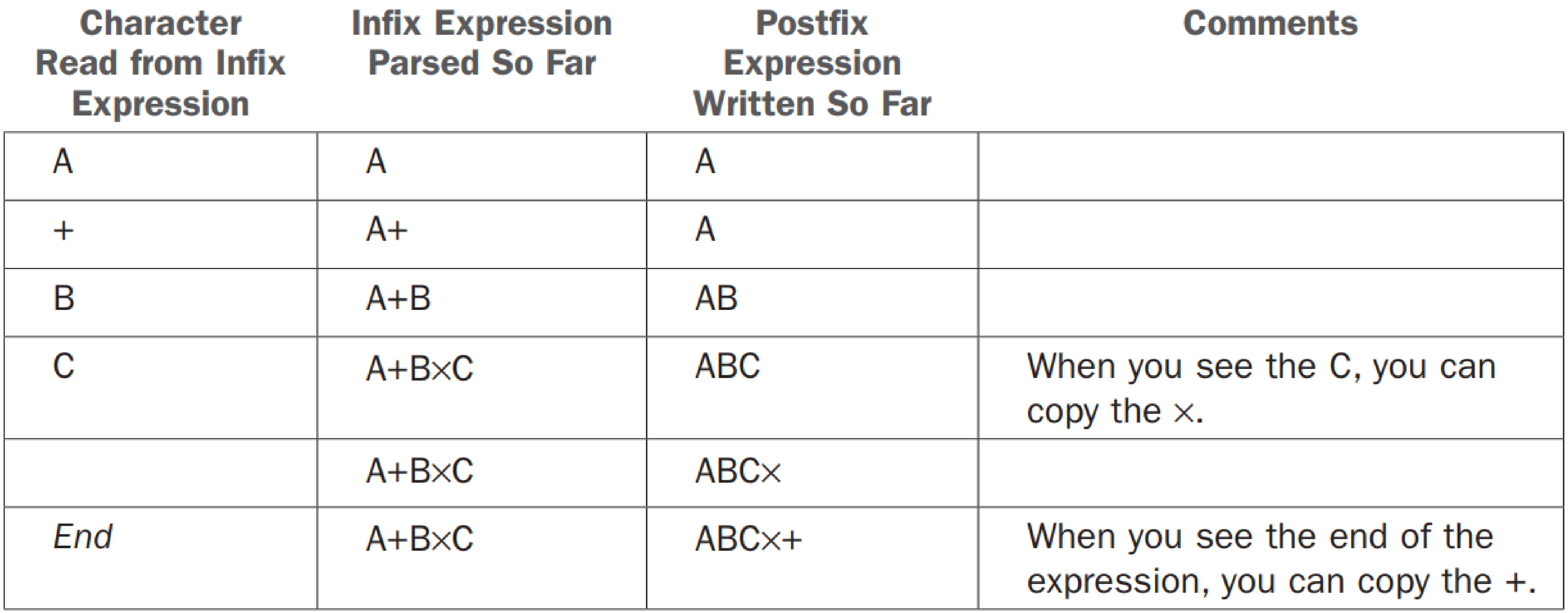
- Read sequentially from left to right.
- If the character is an operand, it is directly copied to the postfix string
- If the character is an operator, append it to the postfix string instead of evaluating it.

Evaluate A+B-C



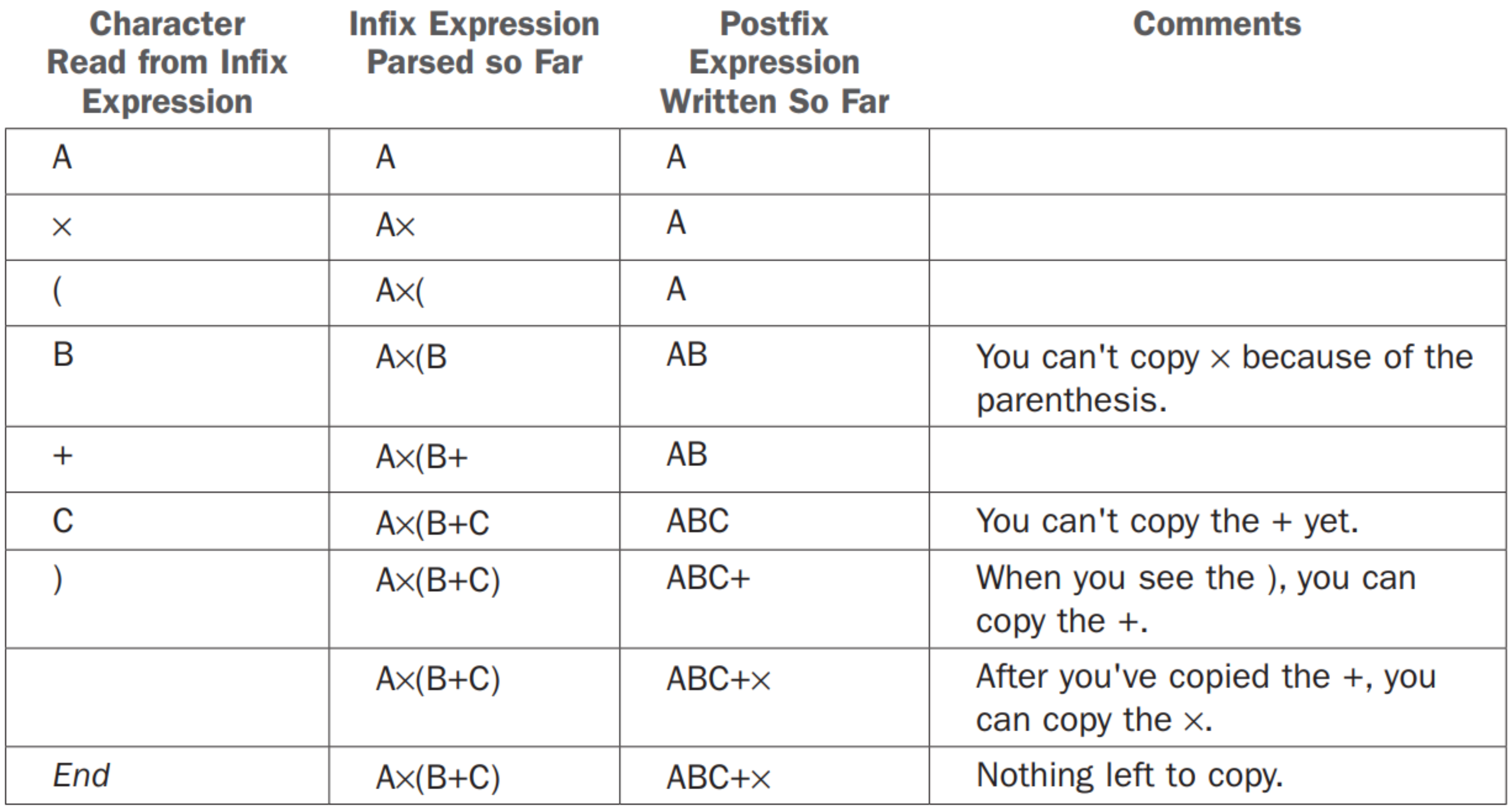
Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
–	A+B-	AB+	When you see the –, you can copy the + to the postfix string.
C	A+B–C	AB+C	
End	A+B–C	AB+C–	When you reach the end of the expression, you can copy the –.

Evaluate A+BxC



Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Comments
A	A	A	
+	A+	A	
B	A+B	AB	
C	A+B×C	ABC	When you see the C, you can copy the ×.
	A+B×C	ABC×	
End	A+B×C	ABC×+	When you see the end of the expression, you can copy the +.

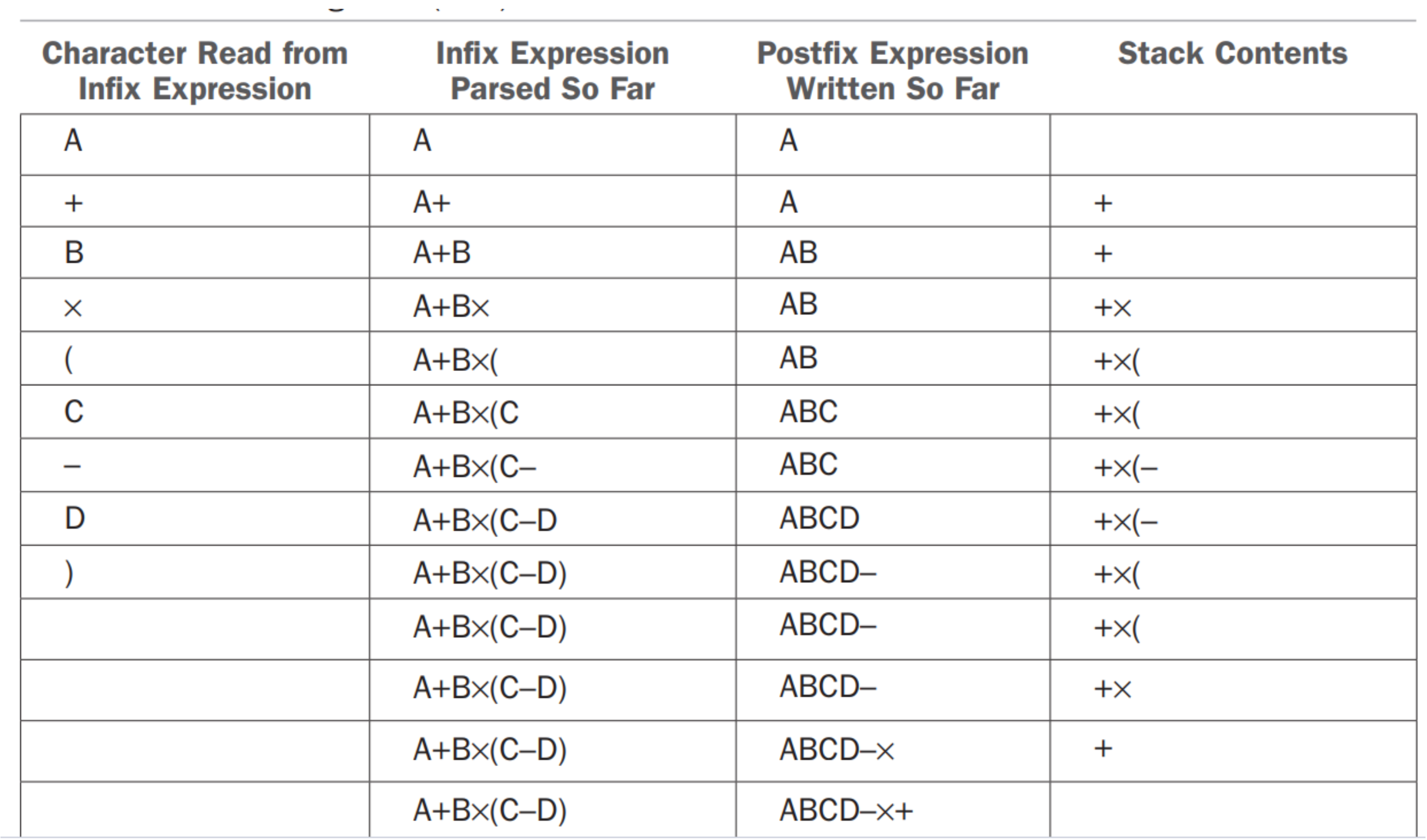
Evaluate Ax(B+C)



Character Read from Infix Expression	Infix Expression Parsed so Far	Postfix Expression Written So Far	Comments
A	A	A	
×	A×	A	
(	A×(	A	
B	A×(B	AB	You can't copy × because of the parenthesis.
+	A×(B+	AB	
C	A×(B+C	ABC	You can't copy the + yet.
)	A×(B+C)	ABC+	When you see the), you can copy the +.
	A×(B+C)	ABC+×	After you've copied the +, you can copy the ×.
End	A×(B+C)	ABC+×	Nothing left to copy.

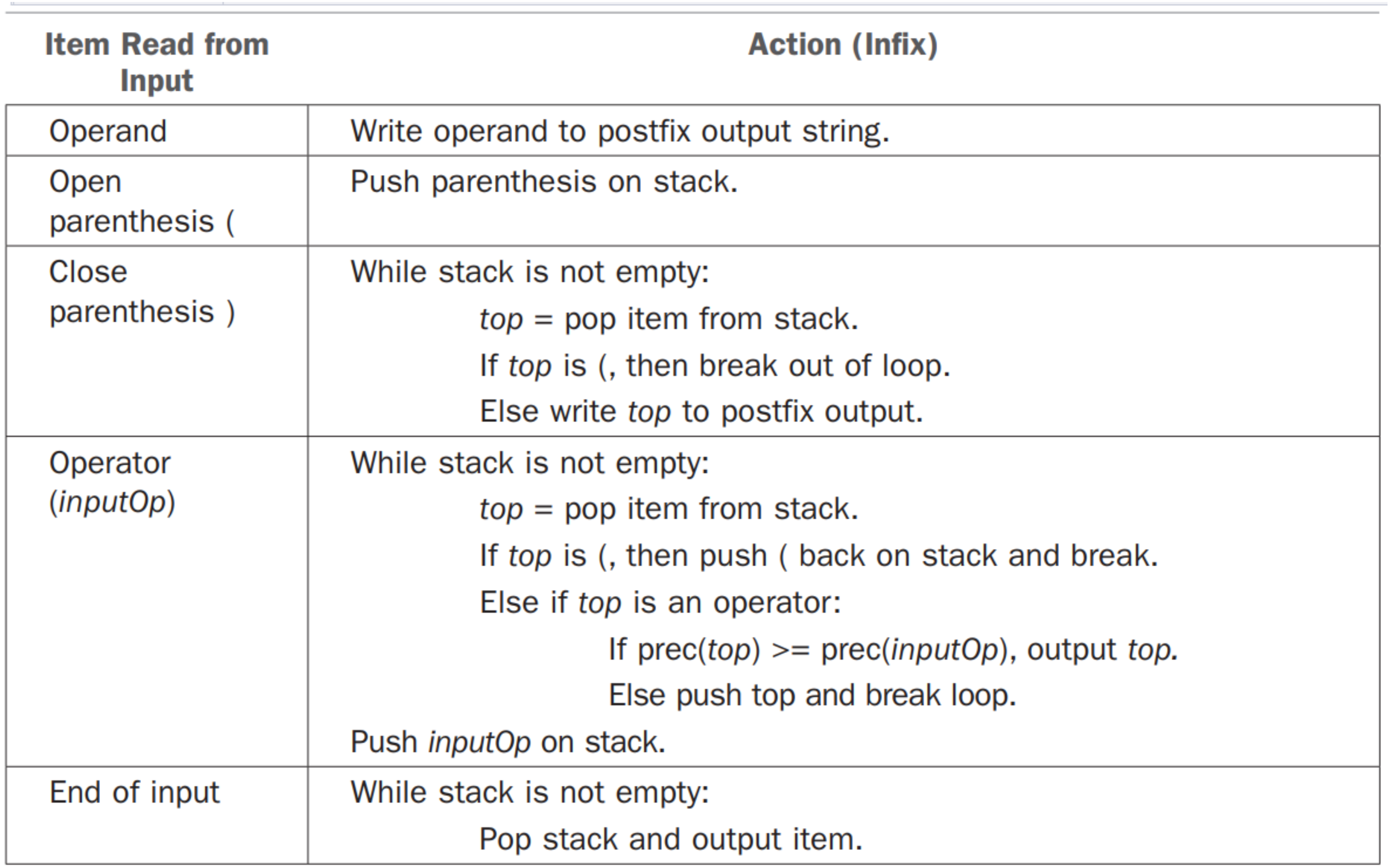
Saving Operators on a Stack

- The sequence of the operators is inverted while transitioning from infix to postfix notation.
- A+B×(C–D)



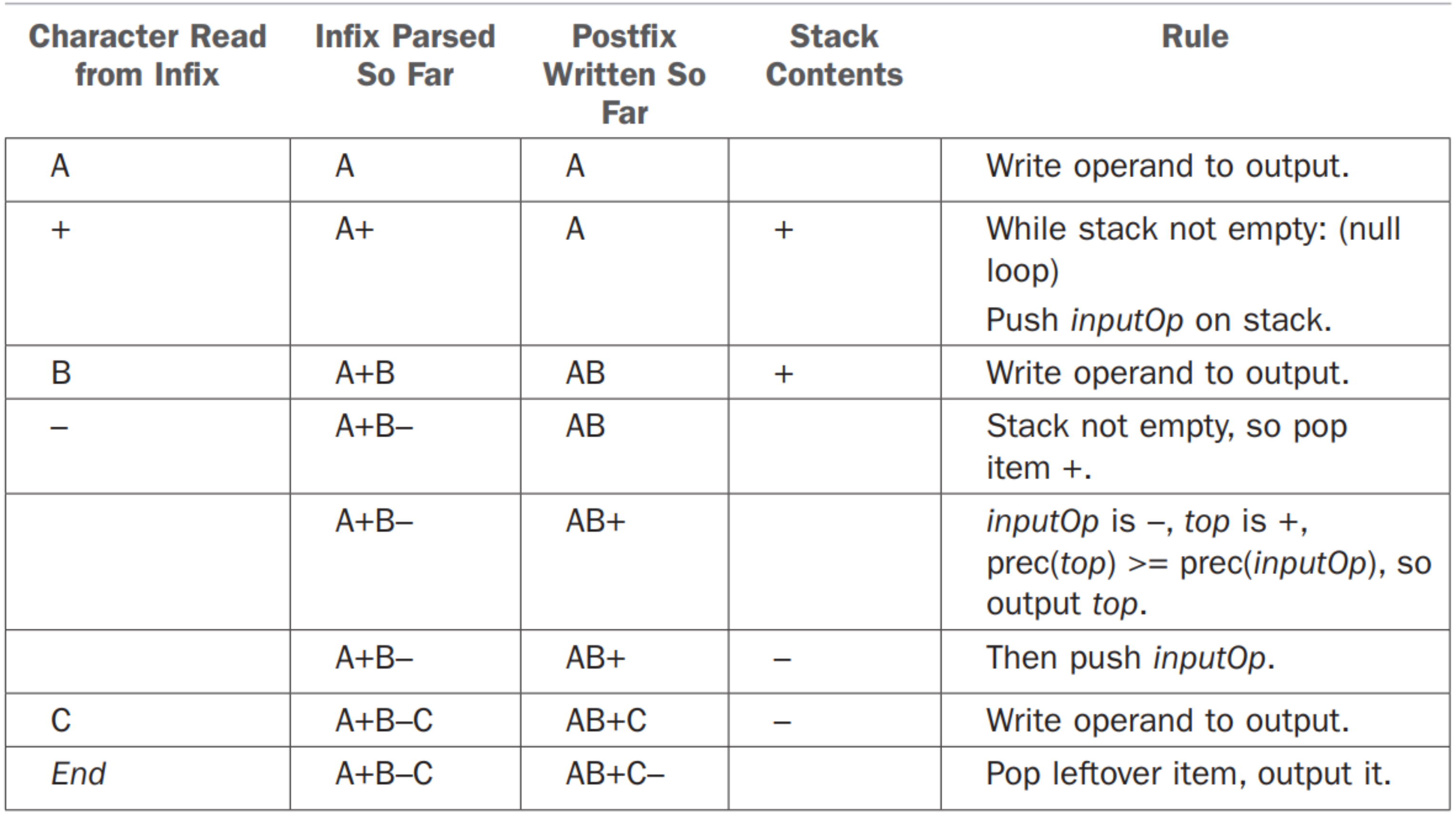
Character Read from Infix Expression	Infix Expression Parsed So Far	Postfix Expression Written So Far	Stack Contents
A	A	A	
+	A+	A	+
B	A+B	AB	+
×	A+B×	AB	+×
(	A+B×(	AB	+×(
C	A+B×(C	ABC	+×(
–	A+B×(C–	ABC	+×(–
D	A+B×(C–D	ABCD	+×(–
)	A+B×(C–D)	ABCD–	+×(
	A+B×(C–D)	ABCD–	+×(
	A+B×(C–D)	ABCD–	+×
	A+B×(C–D)	ABCD–×	+
	A+B×(C–D)	ABCD–×+	

Translation Rules



Item Read from Input	Action (Infix)
Operand	Write operand to postfix output string.
Open parenthesis (	Push parenthesis on stack.
Close parenthesis)	While stack is not empty: *top* = pop item from stack. If *top* is (, then break out of loop. Else write *top* to postfix output.
Operator (*inputOp*)	While stack is not empty: *top* = pop item from stack. If *top* is (, then push (back on stack and break. Else if *top* is an operator: If prec(*top*) >= prec(*inputOp*), output *top.* Else push top and break loop. Push *inputOp* on stack.
End of input	While stack is not empty: Pop stack and output item.

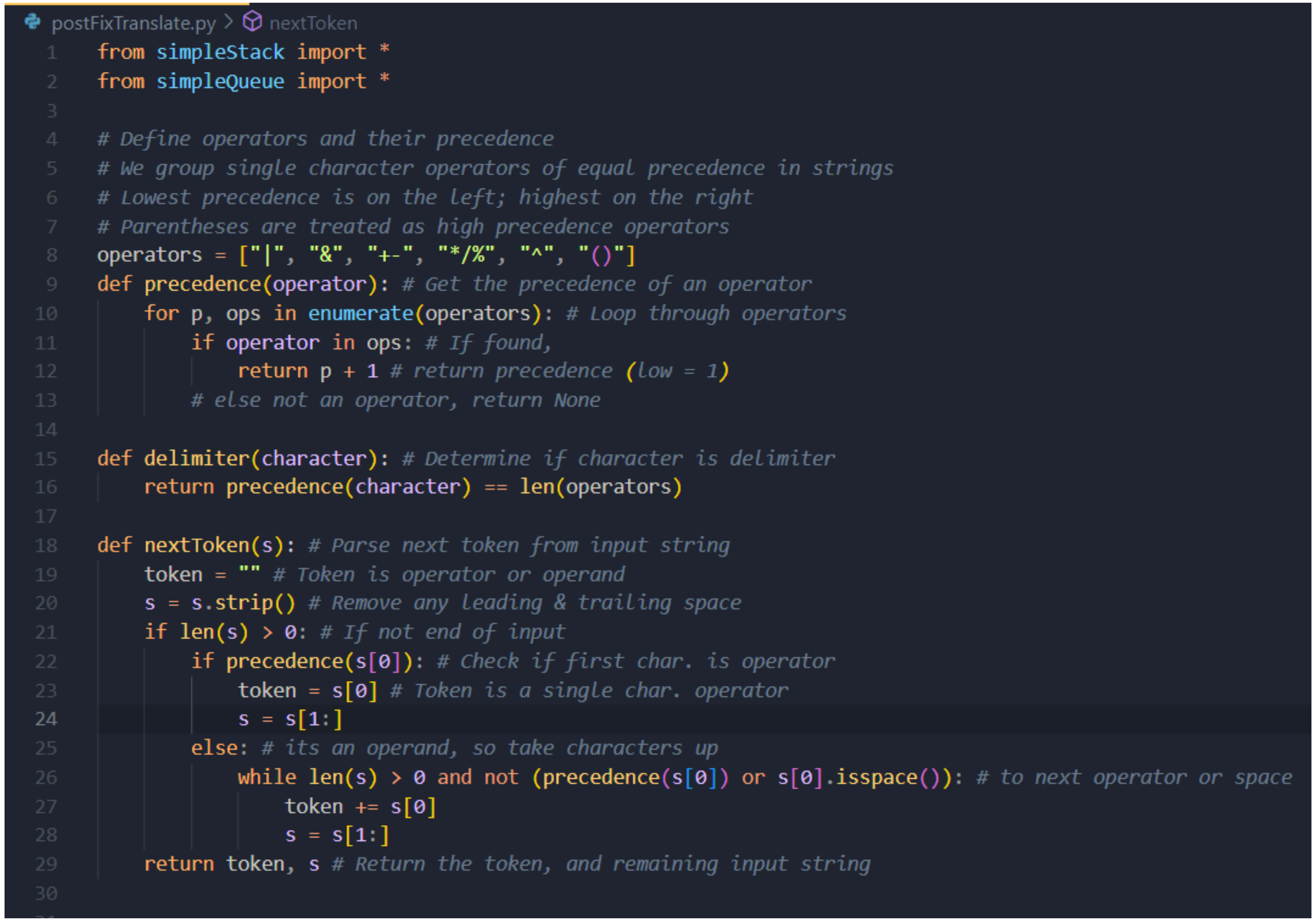
Translation Rules A+B-C



Character Read from Infix	Infix Parsed So Far	Postfix Written So Far	Stack Contents	Rule
A	A	A		Write operand to output.
+	A+	A	+	While stack not empty: (null loop) Push *inputOp* on stack.
B	A+B	AB	+	Write operand to output.
–	A+B–	AB		Stack not empty, so pop item +.
	A+B–	AB+		*inputOp* is –, *top* is +, prec(*top*) >= prec(*inputOp*), so output *top*.
	A+B–	AB+	–	Then push *inputOp*.
C	A+B–C	AB+C	–	Write operand to output.
End	A+B–C	AB+C–		Pop leftover item, output it.

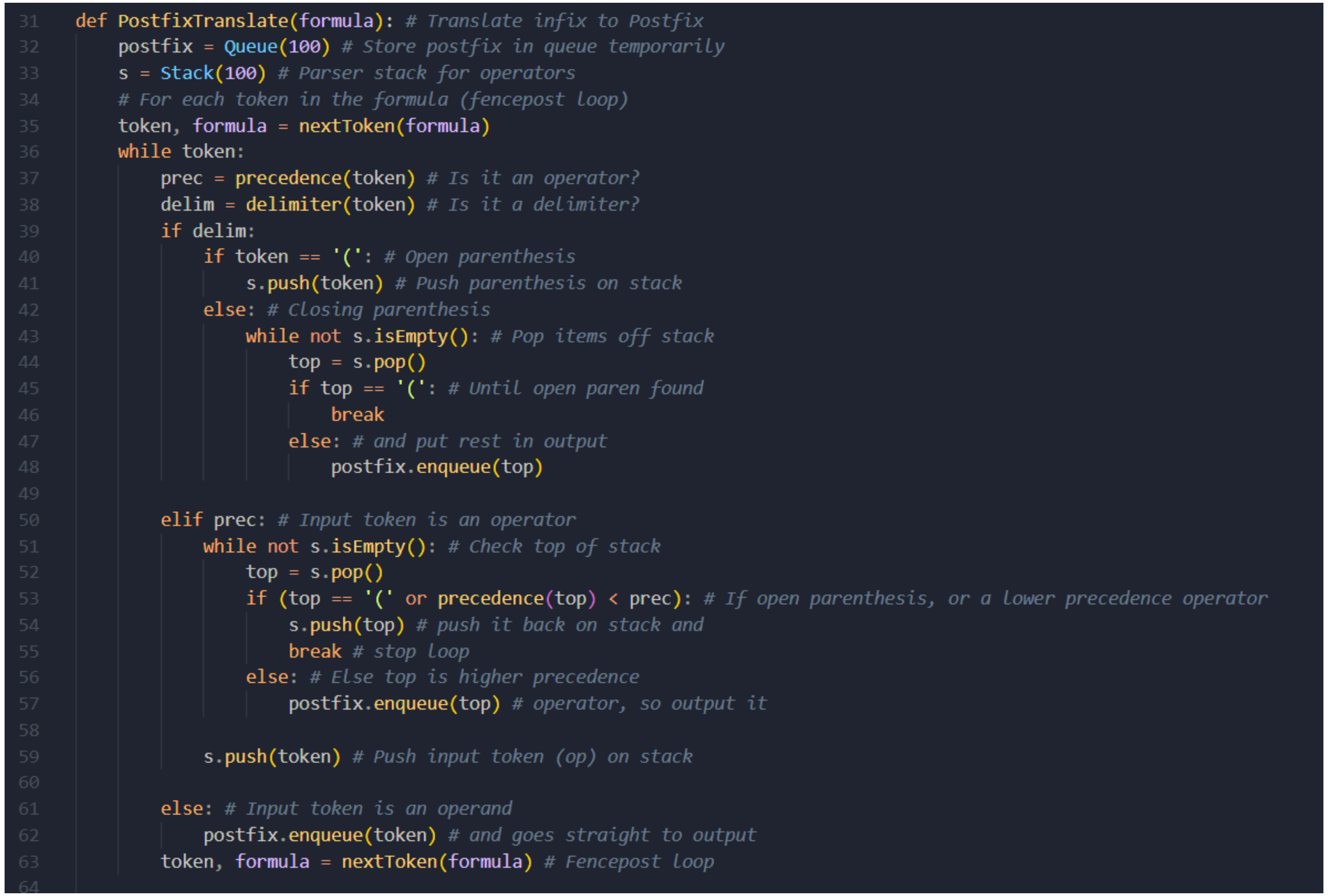
Infix To Postfix Implementation

postfixTranslate.py

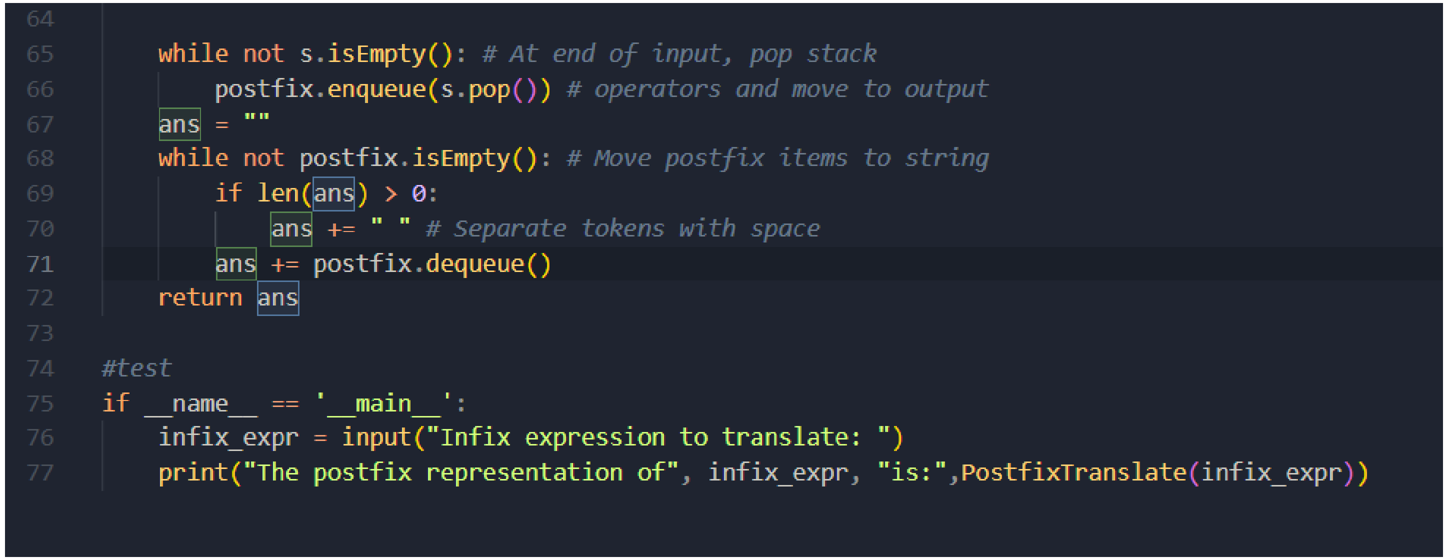


```python
from simpleStack import *
from simpleQueue import *

# Define operators and their precedence
# We group single character operators of equal precedence in strings
# Lowest precedence is on the left; highest on the right
# Parentheses are treated as high precedence operators
operators = ["|", "&", "+-", "*/%", "^", "()"]
def precedence(operator): # Get the precedence of an operator
    for p, ops in enumerate(operators): # Loop through operators
        if operator in ops: # If found,
            return p + 1 # return precedence (low = 1)
        # else not an operator, return None

def delimiter(character): # Determine if character is delimiter
    return precedence(character) == len(operators)

def nextToken(s): # Parse next token from input string
    token = "" # Token is operator or operand
    s = s.strip() # Remove any leading & trailing space
    if len(s) > 0: # If not end of input
        if precedence(s[0]): # Check if first char. is operator
            token = s[0] # Token is a single char. operator
            s = s[1:]
        else: # its an operand, so take characters up
            while len(s) > 0 and not (precedence(s[0]) or s[0].isspace()): # to next operator or space
                token += s[0]
                s = s[1:]
    return token, s # Return the token, and remaining input string

```

postfixTranslate.py

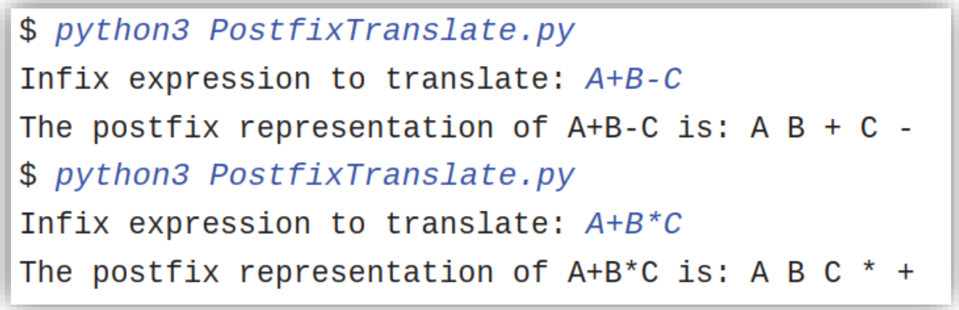


```python
def PostfixTranslate(formula): # Translate infix to Postfix
    postfix = Queue(100) # Store postfix in queue temporarily
    s = Stack(100) # Parser stack for operators
    # For each token in the formula (fencepost loop)
    token, formula = nextToken(formula)
    while token:
        prec = precedence(token) # Is it an operator?
        delim = delimiter(token) # Is it a delimiter?
        if delim:
            if token == '(': # Open parenthesis
                s.push(token) # Push parenthesis on stack
            else: # Closing parenthesis
                while not s.isEmpty(): # Pop items off stack
                    top = s.pop()
                    if top == '(': # Until open paren found
                        break
                    else: # and put rest in output
                        postfix.enqueue(top)

        elif prec: # Input token is an operator
            while not s.isEmpty(): # Check top of stack
                top = s.pop()
                if (top == '(' or precedence(top) < prec): # If open parenthesis, or a lower precedence operator
                    s.push(top) # push it back on stack and
                    break # stop loop
                else: # Else top is higher precedence
                    postfix.enqueue(top) # operator, so output it

            s.push(token) # Push input token (op) on stack

        else: # Input token is an operand
            postfix.enqueue(token) # and goes straight to output
        token, formula = nextToken(formula) # Fencepost loop
```



```python

    while not s.isEmpty(): # At end of input, pop stack
        postfix.enqueue(s.pop()) # operators and move to output
    ans = ""
    while not postfix.isEmpty(): # Move postfix items to string
        if len(ans) > 0:
            ans += " " # Separate tokens with space
        ans += postfix.dequeue()
    return ans

#test
if __name__ == '__main__':
    infix_expr = input("Infix expression to translate: ")
    print("The postfix representation of", infix_expr, "is:",PostfixTranslate(infix_expr))
```

postfixTranslate.py

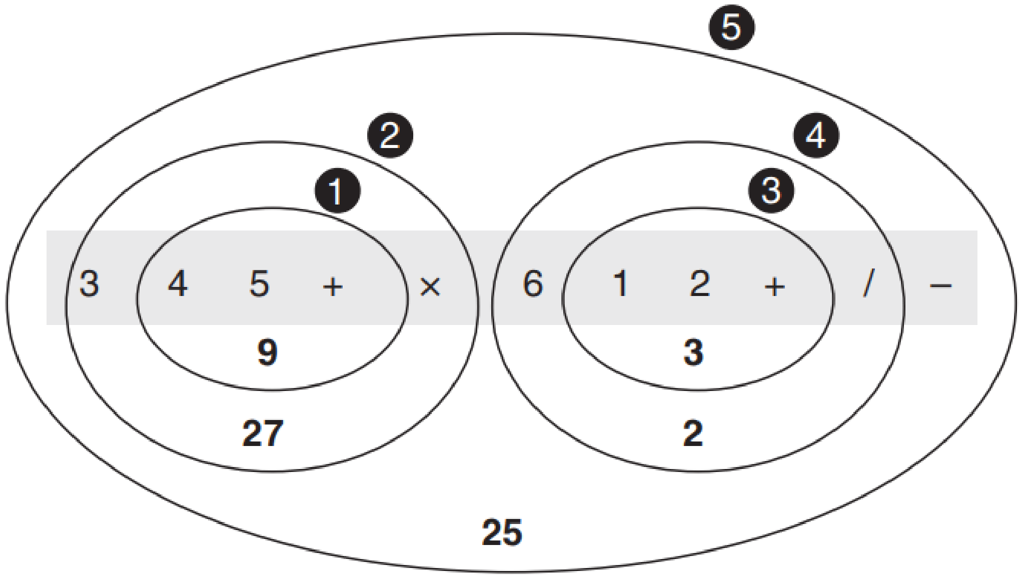


```
$ python3 PostfixTranslate.py
Infix expression to translate: A+B-C
The postfix representation of A+B-C is: A B + C -
$ python3 PostfixTranslate.py
Infix expression to translate: A+B*C
The postfix representation of A+B*C is: A B C * +
```

Evaluating Postfix Expression

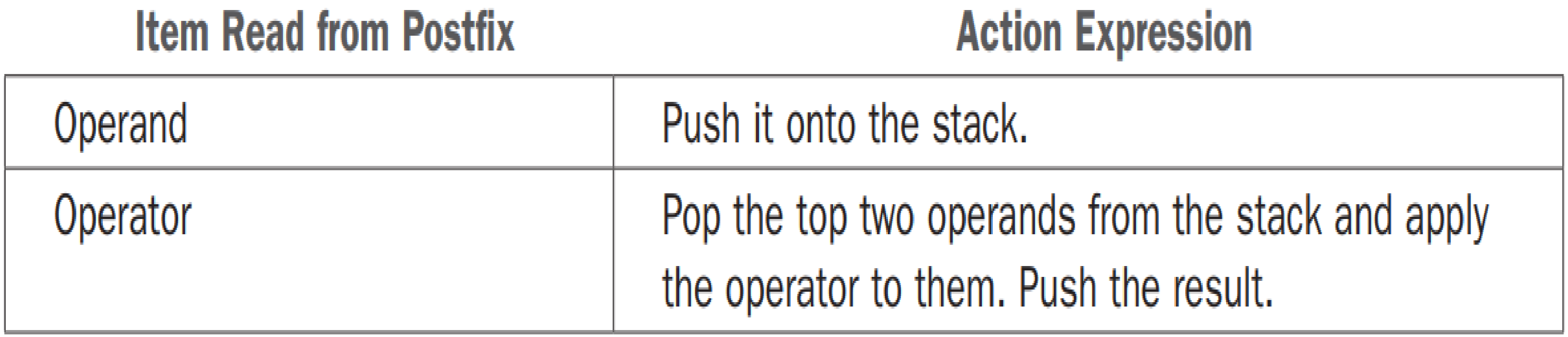
Evaluate 3*(4+5)-6/(1+2)

- The postfix expression is: 345+*612+/-



Rules for Postfix Evaluation

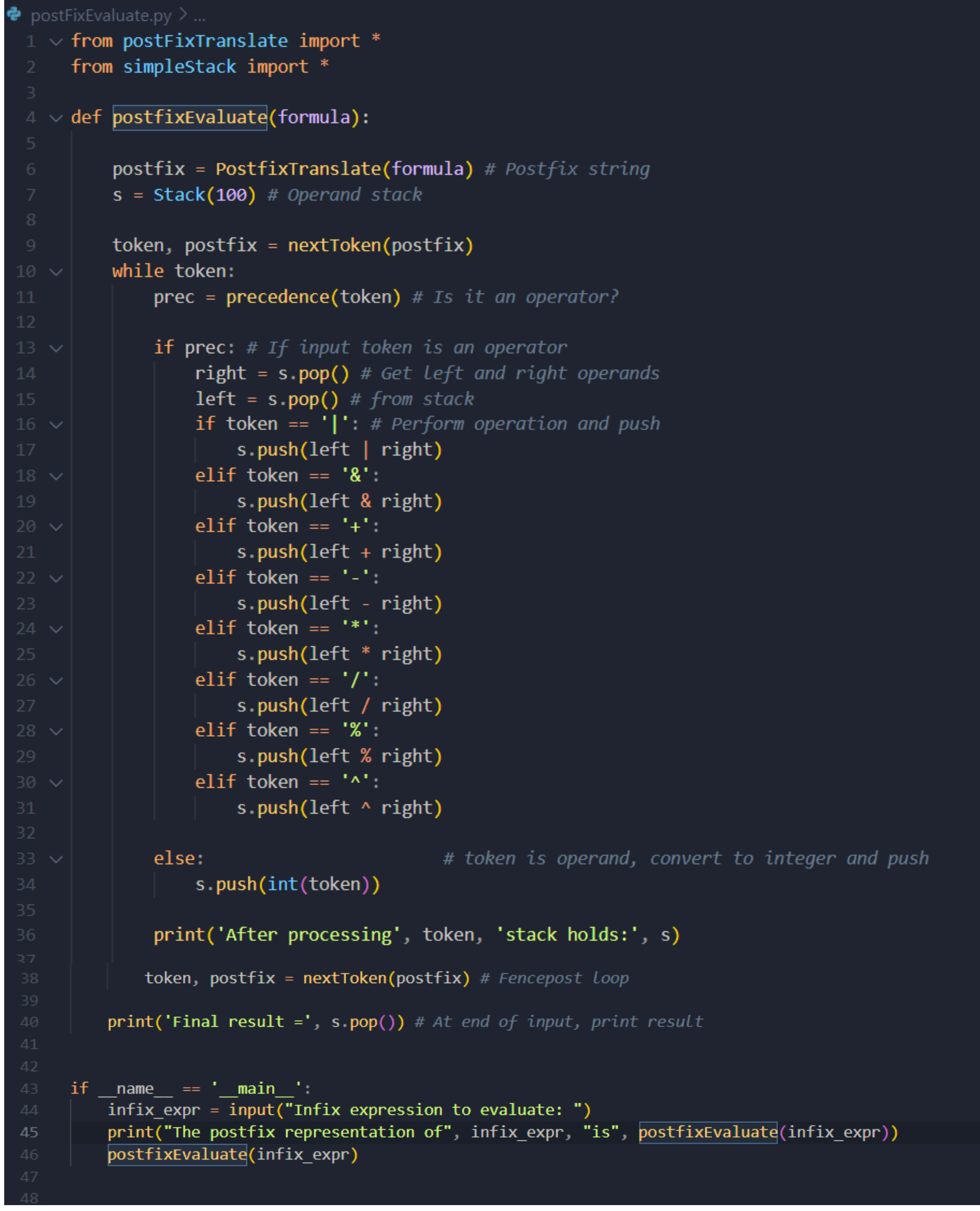
- Upon seeing an operator, it is evident that you must apply it to the most recent two operands that you have encountered.



Item Read from Postfix	Action Expression
Operand	Push it onto the stack.
Operator	Pop the top two operands from the stack and apply the operator to them. Push the result.

PostfixEvaluate.py

```
$ python3 PostfixEvaluate.py
Infix expression to evaluate: 3*(4+5)-6/(1+2)
The postfix representation of 3*(4+5)-6/(1+2) is 3 4 5 + * 6 1 2 + / -
After processing 3 stack holds: [3]
After processing 4 stack holds: [3, 4]
After processing 5 stack holds: [3, 4, 5]
After processing + stack holds: [3, 9]
After processing * stack holds: [27]
After processing 6 stack holds: [27, 6]
After processing 1 stack holds: [27, 6, 1]
After processing 2 stack holds: [27, 6, 1, 2]
After processing + stack holds: [27, 6, 3]
After processing / stack holds: [27, 2.0]
After processing - stack holds: [25.0]
Final result = 25.0
```



```python
from postFixTranslate import *
from simpleStack import *

def postfixEvaluate(formula):

    postfix = PostfixTranslate(formula) # Postfix string
    s = Stack(100) # Operand stack

    token, postfix = nextToken(postfix)
    while token:
        prec = precedence(token) # Is it an operator?

        if prec: # If input token is an operator
            right = s.pop() # Get left and right operands
            left = s.pop() # from stack
            if token == '|': # Perform operation and push
                s.push(left | right)
            elif token == '&':
                s.push(left & right)
            elif token == '+':
                s.push(left + right)
            elif token == '-':
                s.push(left - right)
            elif token == '*':
                s.push(left * right)
            elif token == '/':
                s.push(left / right)
            elif token == '%':
                s.push(left % right)
            elif token == '^':
                s.push(left ^ right)

        else:                           # token is operand, convert to integer and push
            s.push(int(token))

        print('After processing', token, 'stack holds:', s)

        token, postfix = nextToken(postfix) # Fencepost loop

    print('Final result =', s.pop()) # At end of input, print result


if __name__ == '__main__':
    infix_expr = input("Infix expression to evaluate: ")
    print("The postfix representation of", infix_expr, "is", postfixEvaluate(infix_expr))
    postfixEvaluate(infix_expr)

```

Thanks