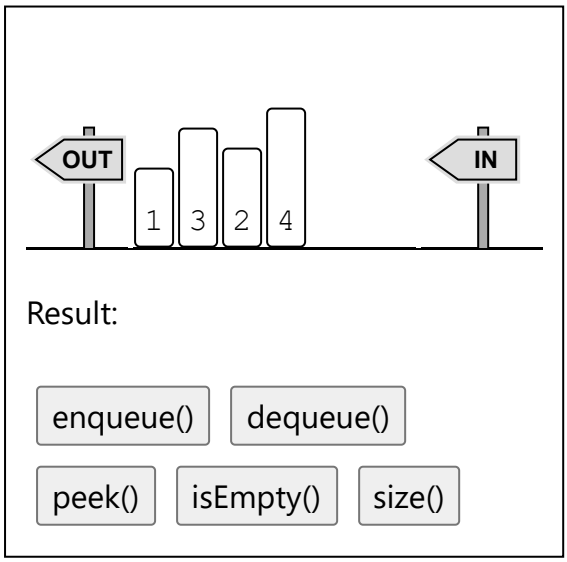
DSA Queues

‹ Previous Next ›

Queues

A queue is a data structure that can hold many elements.



Think of a queue as people standing in line in a supermarket.

The first person to stand in line is also the first who can pay and leave the supermarket. This way of organizing elements is called FIFO: First In First Out.

- **Peek:** Returns the first element in the queue.
- **isEmpty:** Checks if the queue is empty.
- **Size:** Finds the number of elements in the queue.

Experiment with these basic operations in the queue animation above.

Queues can be implemented by using arrays or linked lists.

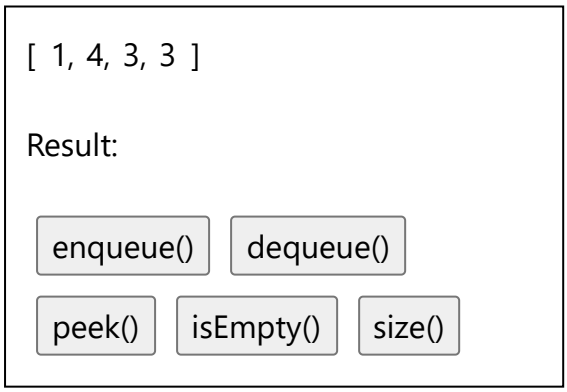
Queues can be used to implement job scheduling for an office printer, order processing for e-tickets, or to create algorithms for breadth-first search in graphs.

Queues are often mentioned together with Stacks, which is a similar data structure described on the previous page.

Queue Implementation using Arrays

To better understand the benefits with using arrays or linked lists to implement queues, you should check out this page that explains how arrays and linked lists are stored in memory.

This is how it looks like when we use an array as a queue:



Reasons to implement queues using arrays:

- **Memory Efficient:** Array elements do not hold the next elements address like linked list nodes do.
- **Easier to implement and understand:** Using arrays to implement queues require less code than using linked lists, and for this reason it is typically easier to understand as well.

Reasons for **not** using arrays to implement queues:

other elements must be shifted to take the removed elements' place. This is inefficient and can cause problems, especially if the queue is long.

- **Alternatives:** Some programming languages have built-in data structures optimized for queue operations that are better than using arrays.

Note: When using arrays in Python for this tutorial, we are really using the Python 'list' data type, but for the scope of this tutorial the 'list' data type can be used in the same way as an array. Learn more about Python lists here.

Since Python lists has good support for functionality needed to implement queues, we start with creating a queue and do queue operations with just a few lines:

Example

Python:

```
queue = []

# Enqueue
queue.append('A')
queue.append('B')
queue.append('C')
print("Queue: ", queue)

# Dequeue
element = queue.pop(0)
print("Dequeue: ", element)

# Peek
frontElement = queue[0]
print("Peek: ", frontElement)

# isEmpty
isEmpty = not bool(queue)
print("isEmpty: ", isEmpty)
```

Try it Yourself »

But to explicitly create a data structure for queues, with basic operations, we should create a queue class instead. This way of creating queues in Python is also more similar to how queues can be created in other programming languages like C and Java.

Example

Python:

```
class Queue:
    def __init__(self):
        self.queue = []

    def enqueue(self, element):
        self.queue.append(element)

    def dequeue(self):
        if self.isEmpty():
            return "Queue is empty"
        return self.queue.pop(0)

    def peek(self):
        if self.isEmpty():
            return "Queue is empty"
        return self.queue[0]

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

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

# Create a queue
myQueue = Queue()
```

```
print("Dequeue: ", myQueue.dequeue())

print("Peek: ", myQueue.peek())

print("isEmpty: ", myQueue.isEmpty())

print("Size: ", myQueue.size())
```

Try it Yourself »

Queue Implementation using Linked Lists

Reasons for using linked lists to implement queues:

- **Dynamic size:** The queue can grow and shrink dynamically, unlike with arrays.
- **No shifting:** The front element of the queue can be removed (dequeue) without having to shift other elements in the memory.

Reasons for **not** using linked lists to implement queues:

- **Extra memory:** Each queue element must contain the address to the next element (the next linked list node).
- **Readability:** The code might be harder to read and write for some because it is longer and more complex.

This is how a queue can be implemented using a linked list.

Example

Python:

```
class Node:
    def __init__(self, data):
        self.data = data
```

```
        self.front = None
        self.rear = None
        self.length = 0

    def enqueue(self, element):
        new_node = Node(element)
        if self.rear is None:
            self.front = self.rear = new_node
            self.length += 1
            return
        self.rear.next = new_node
        self.rear = new_node
        self.length += 1

    def dequeue(self):
        if self.isEmpty():
            return "Queue is empty"
        temp = self.front
        self.front = temp.next
        self.length -= 1
        if self.front is None:
            self.rear = None
        return temp.data

    def peek(self):
        if self.isEmpty():
            return "Queue is empty"
        return self.front.data

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

    def size(self):
        return self.length

    def printQueue(self):
        temp = self.front
        while temp:
            print(temp.data, end=" ")
```

```
myQueue = Queue()

myQueue.enqueue('A')
myQueue.enqueue('B')
myQueue.enqueue('C')
print("Queue: ", end="")
myQueue.printQueue()

print("Dequeue: ", myQueue.dequeue())

print("Peek: ", myQueue.peek())

print("isEmpty: ", myQueue.isEmpty())

print("Size: ", myQueue.size())
```

Try it Yourself »

DSA Exercises

Test Yourself With Exercises

Exercise:

The Array below is used as a Queue data structure:

`[5,11,8,3]`

Which indexes and values are affected by the `endueue` and `dedueue` operations?

```
        index    in the array.

    dequeue():
        value    is taken
        out of the queue.
```

Submit Answer »

Start the Exercise

‹ Previous Sign in to track progress Next ›

COLOR PICKER

PLUS SPACES

GET CERTIFIED FOR TEACHERS

FOR BUSINESS CONTACT US

Top Tutorials

HTML Tutorial
CSS Tutorial
JavaScript Tutorial
How To Tutorial
SQL Tutorial
Python Tutorial
W3.CSS Tutorial
Bootstrap Tutorial
PHP Tutorial
Java Tutorial
C++ Tutorial
jQuery Tutorial

Top References

HTML Reference
CSS Reference
JavaScript Reference
SQL Reference
Python Reference
W3.CSS Reference
Bootstrap Reference
PHP Reference
HTML Colors
Java Reference
AngularJS Reference
jQuery Reference