



# What is a Queue?

This lesson gives an introduction to the queue data structure, its various uses, and types. We will also go through the inner workings of a Queue by briefly discussing each of its functions.


## We'll cover the following

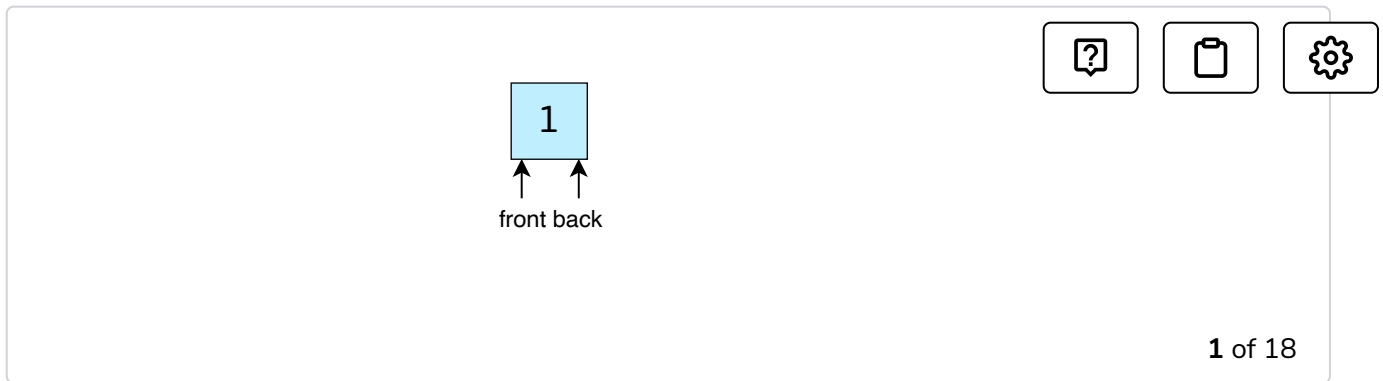


- Introduction
- What are Queues used for?
- How do Queues work?
- Example
- Types of Queues
  - Circular Queue
  - Priority Queue
  - Double-Ended Queue

## Introduction#

Similar to the stack, a queue is another linear data structure that stores the elements in a sequential manner. The only significant difference between stacks and queues is that instead of using the LIFO principle, queues implement the FIFO method which is short for **First in First Out**.

According to *FIFO*, the first element inserted is the one that comes out first. You can think of a queue as a pipe with two ends called **front** and **rear** / **back**. Elements from a queue are always deleted from the **front** and added at the **rear**. The following animation illustrates the structure of a queue. 



Queues are slightly trickier to implement as compared to stacks because we have to keep track of both ends of the array. The elements are inserted from the back and removed from the front.

A perfect real-life example of a queue is a line of people waiting to get a ticket from the booth. If a new person comes, he will join the line from the end, and the person standing at the front will be the first to get the ticket and hence leave the line.

## What are Queues used for?#

Most operating systems also perform operations based on a Priority Queue (a kind of queue) which allows operating systems to switch between appropriate processes. They are also used to store packets on routers in a certain order when a network is congested. Implementing a cache also heavily relies on queues. We generally use Queues when:

- We want to prioritize something over another
- A resource is shared between multiple devices

## How do Queues work?#



A typical queue needs the following set of functions to work. The operations are listed below in the table:



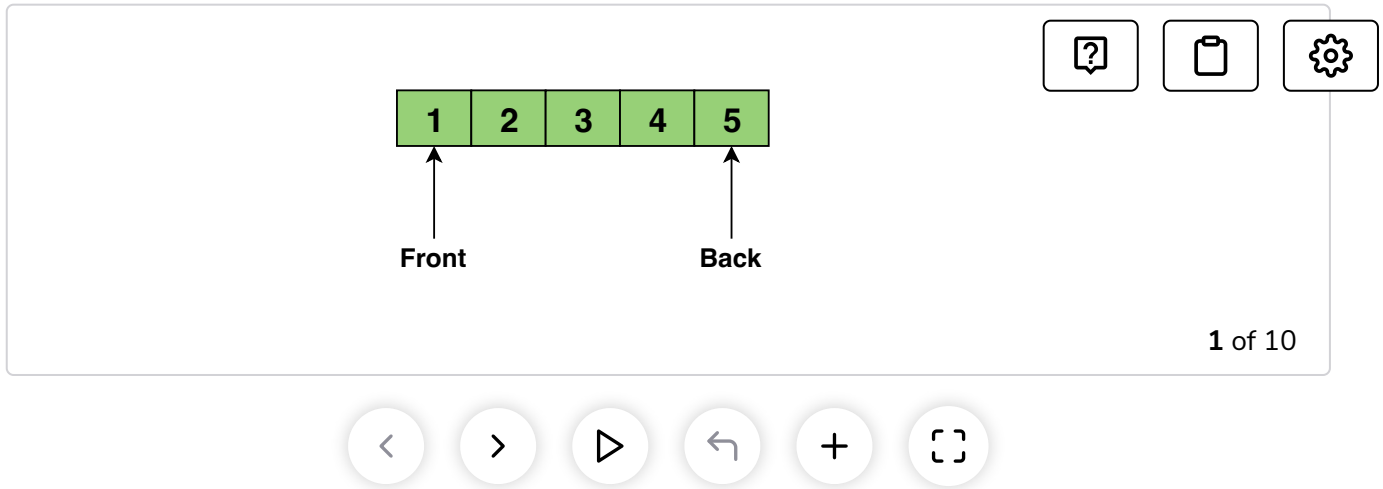
Function	What does it do?
<code>enqueue(element)</code>	inserts element at the end of the queue
<code>dequeue()</code>	removes an element from the start of the queue
<code>front()</code>	returns the first element of the queue
<code>rear()</code>	returns the last element inserted into the queue
<code>isEmpty()</code>	checks if the queue is empty
<code>size()</code>	returns the size of the queue

The entire functionality of queues depends on the first two functions and the rest are just helper functions to produce simple, and understandable code.

## Example #

See the animation below for a visual of how queues work. First, we will insert **8** and then remove **3** from the queue.





## Types of Queues#

Listed below are the four most common types of queues.

- *Linear Queue*
- *Circular Queue*
- *Priority Queue*
- *Double-ended Queue*

The queue that we have discussed so far was a *linear queue*. Let's look at the last two types and see how they are different from *linear queue*.

## Circular Queue#

Circular queues are almost similar to linear queues with only one exception. As the name itself suggests, circular queues are circular in structure which means that both ends are connected to form a circle. Initially, the front and rear part of the queue point to the same location. Eventually they move apart as more elements are inserted into the queue. Circular queues are generally used in:

- Simulation of objects
- Event handling (do something when a particular event occurs)



# Priority Queue#



In priority queues, all elements have a priority associated with them and are sorted such that the most prioritized object appears at the front and the least prioritized object appears at the end of the queue. These queues are widely used in most operating systems to determine which programs should be given more priority.

## Double-Ended Queue#

The double-ended queue acts as a queue from both ends(back and front). It is a flexible data structure that provides **enqueue** and **dequeue** functionality on both ends in  $O(1)$ . Hence, it can act like a normal linear queue if needed. Python has a built-in **deque** class that can be imported from the **collections** module. The class contains several useful methods such as **rotate**. Looking back, the **Right Rotate List** challenge can be solved easily with **deque.rotate()**. Try it out as a small exercise.

---

Now that we have covered all the basics of queues, let's try to implement them in Python. See you in the next lesson!

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#) ⓘ



← Back

Stack (Implementation)

Next →

Queue (Implementation)



Completed



Report an Issue

