# Stacks and Queues

Written by Rachel J. Morris, last updated Oct 9, 2017

## About

Stacks and Queues are data types that only allow access to specific items stored within them – either the very front or the very back items. Here we will discuss the implementation of these structures as well as when we would use them.

## Topics

1. Arrays & Lists vs. Stacks & Queues
2. How Queues work: Front
3. How Stacks work

## 1. Arrays & Lists vs. Stacks & Queues

## 1. Arrays & Lists vs. Stacks & Queues

What is the point of the restricted-access data types, Stacks and Queues, and why would we use them instead of just using a normal array or linked list where we can more freely access elements at whatever index we want?

Notes

## 1. Arrays & Lists vs. Stacks & Queues

What is the point of the restricted-access data types, Stacks and Queues, and why would we use them instead of just using a normal array or linked list where we can more freely access elements at whatever index we want?

The reason that we would choose a Stack or a Queue over other structures all has to do with **design choices**. We will also see this later on with other data structures, such as Dictionaries and Binary Search Trees.

Notes

## 1. Arrays & Lists vs. Stacks & Queues

Let’s say we’re implementing a system that takes orders and processes them. However, it cannot process all possible orders simultaneously.

In this case, we would need a structure to store the incoming data while we’re currently processing the most recently received task.

In this case, we can use a Queue, which is a first-in-first-out structure, to keep process the jobs in the order that they are received in.

Notes

Queue: FIFO

# 1. Arrays & Lists vs. Stacks & Queues

Or, let’s say we’re tracking input from the keyboard, and building a message string character-by-character, as each letter or symbol is typed.

If the user hits *backspace*, we need to erase the *most recently typed letter* from our message.

In this case, a last-in-first-out Stack would be a good option.

Notes

Queue: FIFO

Stack: LIFO

## 1. Arrays & Lists vs. Stacks & Queues

These data structures exist for some given design reason, and Stacks and Queues are restricted access data types specifically because their designs help the programmer achieve some kind of functionality.

Later on, we will also learn about **priority queues**, where items are stored in a queue structure, but when they're inserted, they're placed at a specific position based on their priority – then, we pull the highest-priority item from the front.

Notes

Queue: FIFO

Stack: LIFO

## 2. How Queues work

## 2. How Queues work

Queues can be implemented with an Array or with a Linked List type of structure.

Items are Pushed into a queue at the **back**, and items are **Popped** from a queue in the **front**.

Additionally, the only item you can access is at the **front**. Usually, the “get” function for a Queue is just called **Front**.

Notes

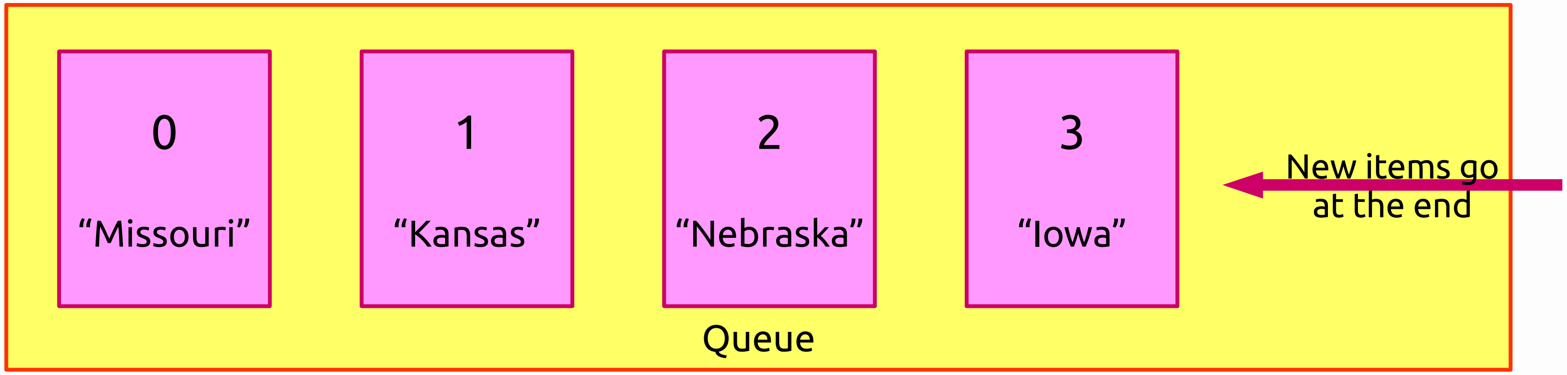
Queue: FIFO

Main functions:
**Push** (at back)
**Pop** (at front)
**Front**

# 2. How Queues work: Push

With the Push function, items are added at the back, with the *oldest* item standing up front. Like at a grocery store, whoever has been waiting longest gets to go first.

In this example, items have been added in the order:
**`Push( “Missouri” ), Push( “Kansas” ), Push( “Nebraska” ), Push( “Iowa” ).`**



Notes

Queue: FIFO

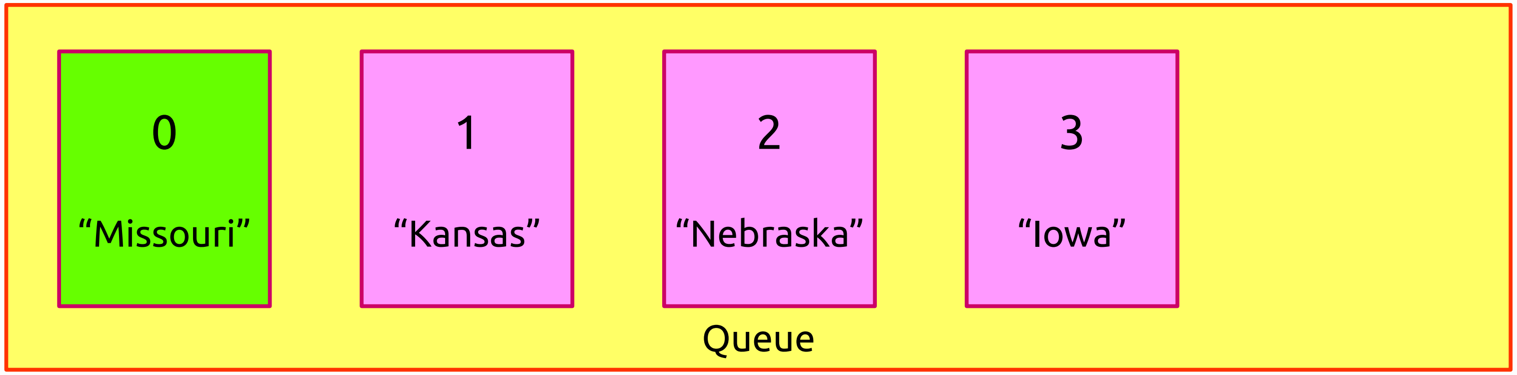
Main functions:
**Push** (at back)
**Pop** (at front)
**Front**

## 2. How Queues work: Front

When we need to work with the next item in the Queue, we can use the **Front()** function to access it.

In this example, items have been added in the order:
`Push( “Missouri” ), Push( “Kansas” ), Push( “Nebraska” ), Push( “Iowa” ).`

`Front() returns “Missouri”.`



Notes

Queue: FIFO

Main functions:
**Push** (at back)
**Pop** (at front)
**Front**

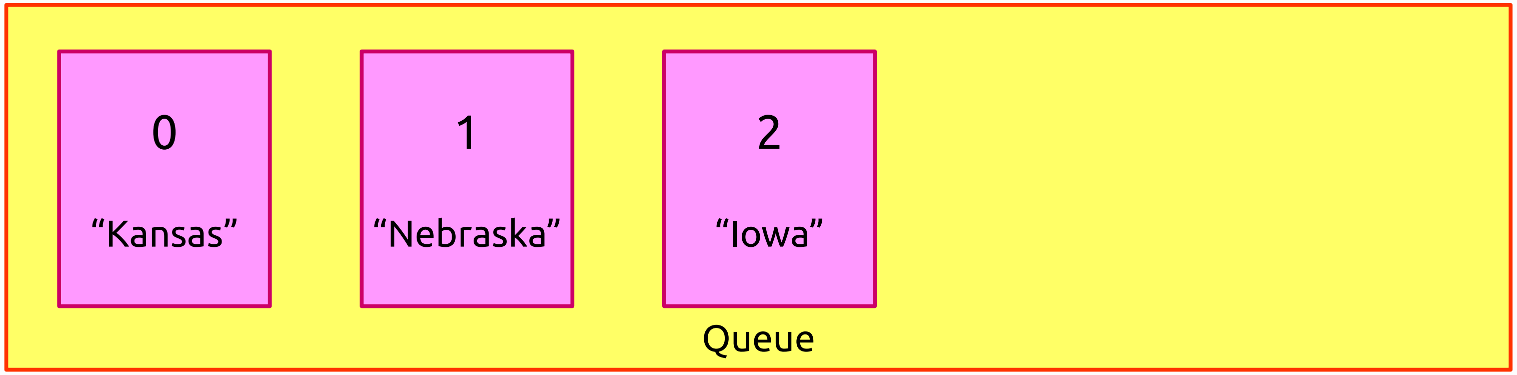
# 2. How Queues work: Pop

Once we're done processing the current item, **Pop()** will remove the front-most item.

In this example, items have been added in the order:
**`Push( "Missouri" ), Push( "Kansas" ), Push( "Nebraska" ), Push( "Iowa" ).`**

**`Front() returns "Missouri".`**

**`Pop() removes "Missouri". Now the Front() is "Kansas".`**



Notes

Queue: FIFO

Main functions:
**Push** (at back)
**Pop** (at front)
**Front**

## 3. How Stacks work

## 3. How Stacks work

Stacks can be implemented with an Array or with a Linked List type of structure.

Items are **Pushed** into a stack at the **back**, and items are **Popped** from a stack at the **back**.

Additionally, the only item you can access is at the **back**. Usually, the “get” function for a Stack is just called **Top**.

Notes

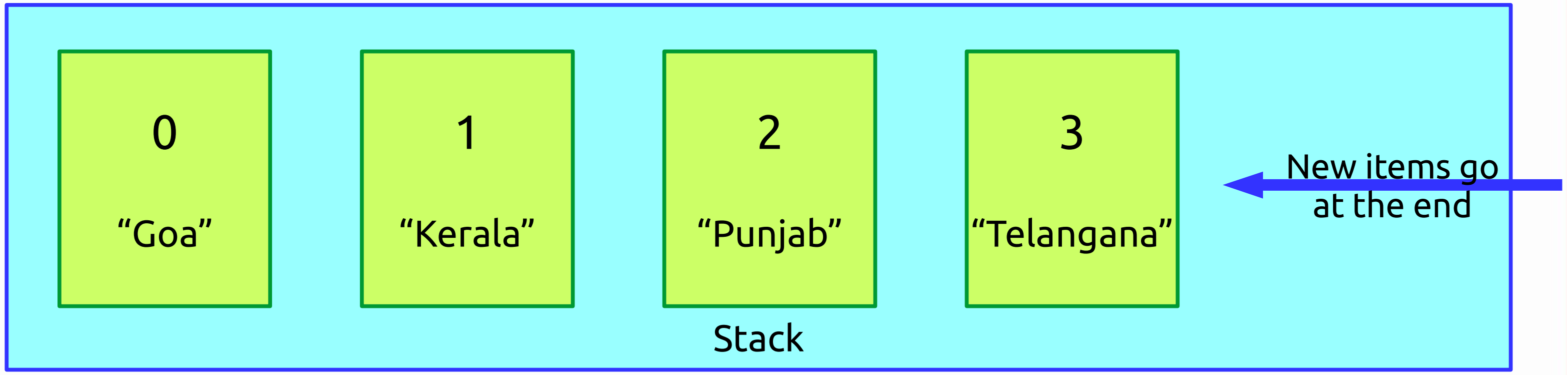
Stack: LIFO

Main functions:
**Push** (at back)
**Pop** (at back)
**Top**

# 3. How Stacks work: Push

With the Push function, items are added at the back, with the *oldest* item standing up front. The oldest item cannot be removed until all items that came before it are removed.

In this example, items have been added in the order:
`Push( “Goa” ), Push( “Kerala” ), Push( “Punjab” ), Push( “Telangana” ).`



Notes

Stack: LIFO

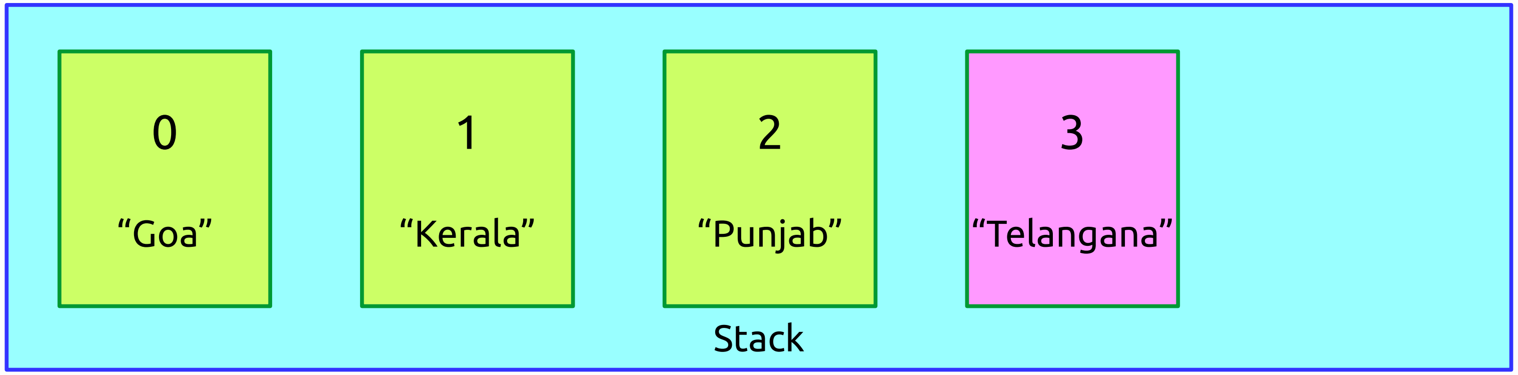
Main functions:
**Push** (at back)
**Pop** (at back)
**Top**

# 3. How Stacks work: Top

We are able to access the "top"-most item (the item at the back) with Top().

In this example, items have been added in the order:
`Push( "Goa" ), Push( "Kerala" ), Push( "Punjab" ), Push( "Telangana" ).`

`Top() returns "Telangana".`



Notes

Stack: LIFO

Main functions:
**Push** (at back)
**Pop** (at back)
**Top**

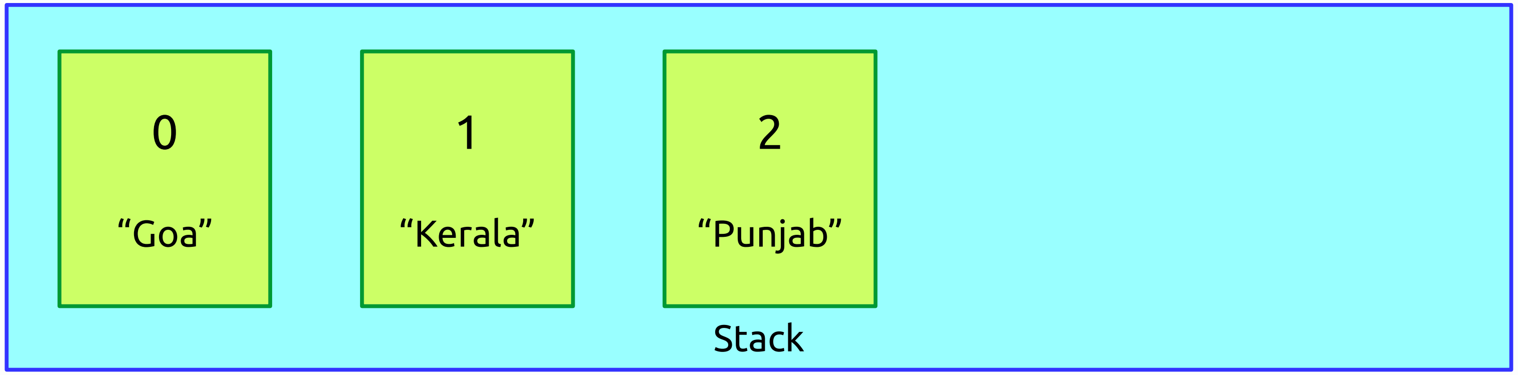
## 3. How Stacks work: Pop

And Pop() removes the “top”-most (last) item in the Stack.

In this example, items have been added in the order:
`Push( “Goa” ), Push( “Kerala” ), Push( “Punjab” ), Push( “Telangana” ).`

`Top() returns “Telangana”.`

`Pop() removes “Telangana”. Now the Top() is “Punjab”.`



Notes

Stack: LIFO

Main functions:
**Push** (at back)
**Pop** (at back)
**Top**

## Conclusion

Stacks and Queues are data structures that are pretty simple in concept, but have a lot of uses in computer science.

They can be implemented with arrays or with linked lists.