What are stacks?

Selecting transcript lines in this section will navigate to timestamp in the video

- Stacks are another container type we can use to store our data. A stack is an ordered series of objects just like a list, but its intended use is slightly different. We push objectives onto the stack and pop objects off of it. Think of a stack of pancakes. Each pancake is stacked on top of the other. To take a pancake off the stack, we remove from the top because it's difficult to remove from the bottom. To add a pancake to the stack, we place it on top of the stack because it's difficult to add to the bottom. Naturally, if you wanted to add or remove from the bottom of the stack, you would have to lift the entire stack in order to add that item. This is why for stacks we add and remove from the top. Stacks follow a last in, first out, or LIFO policy. We say LIFO because the last item on the stack will be the first item removed from the stack. Furthermore, the first item pushed on the stack will be the last item popped off of the stack. When adding an item to the stack, we use the terminology push because we're pushing an item onto the stack or adding it. When removing an item, we use the word pop because we're popping an item off the stack or removing it. We can push as many items as we want onto the stack, but in order to retrieve a specific item further down on the stack we must pop off each item to get to it. This makes stack especially useful for keeping track of state or the order of when things have occurred.Selecting transcript lines in this section will navigate to timestamp in the video

- It's important to understand the pros and cons of stacks and queues so you know when to use them. Stacks are great for programs where you need to reverse things. For example, let's say you push a string onto a stack one character at a time and then make a string from the members popped off the stack. The resulting string is reversed. Stacks are also good for keeping track of state as things are pushed on and popped off the stacks. Going deeper into the operations, if you need to add and remove from the back of a data structure a lot, stacks are a great option. Pushing, peeking and popping takes very little time, in fact, constant time because stacks often have a linked list implementation or dynamic array implementation. If you find yourself needing to index your data structure and get a specific item in the middle of your use case, stacks are not the solution. It would take O of N time in the worst case because you would need to pop off everything on the stack and essentially destroy the data structure in order to see each individual item. If you need to search for items or sort your items consistently then a stack is also not the solution. Stacks really just help you keep state. A stack only shows its advantages when you need to use it in a manner where the last item in is the first item out. Now let's talk about queues. Queues are similar to stacks in that they only show their advantages when you're using first in, first out or FIFO functionality. Enqueuing to the back and dequeuing from the front is also very quick and takes constant time because the queue has a doubly-linked list implementation. If you find yourself wanting to pull something from the middle of the queue consistently, a priority queue might be a good option to give your items priority and allow certain ones to skip ahead of the others. However, if you are pulling most items from the middle and not the front of your data structure, then queues are not your solution. Stacks and queues are great data structures to keep in your back pocket.