**Stacks & Queues**

* **Stacks & Queues Introduction:**

Both are linear data structures, both are allow you to traverse sequentially, in which only one data can be reached directly.

In stacks and queues there is no random access.

A picture containing timeline

Description automatically generated

* **Stacks:**

**Graphical user interface, application

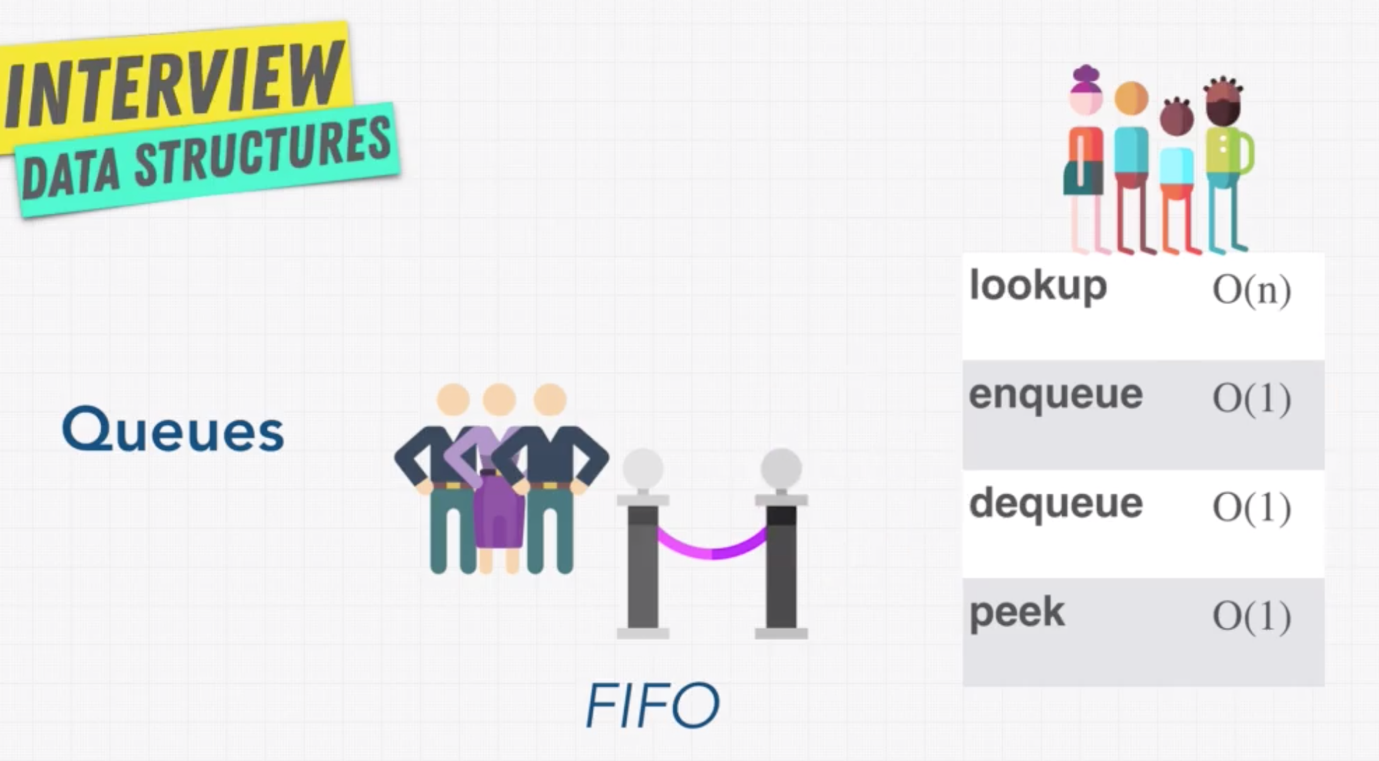
Description automatically generated**

This data structure is mainly used in programming language to store the function / method calls.

Another example for stack is browser front and back button to maintain the history.

And it is the same with undo and re do.

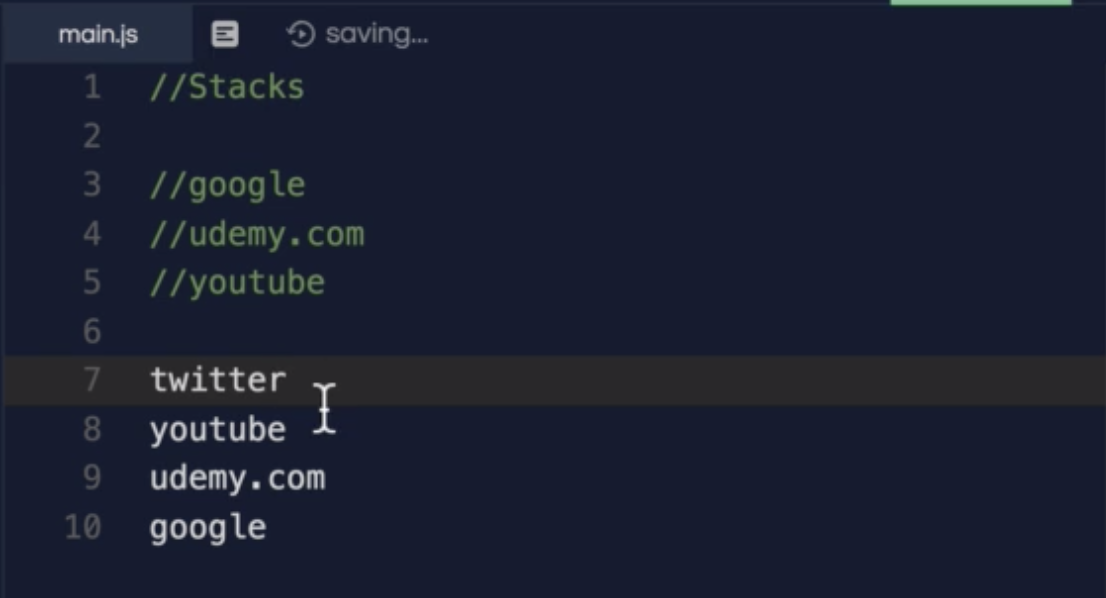
* **Queues:**

****

Example of queue is printer.

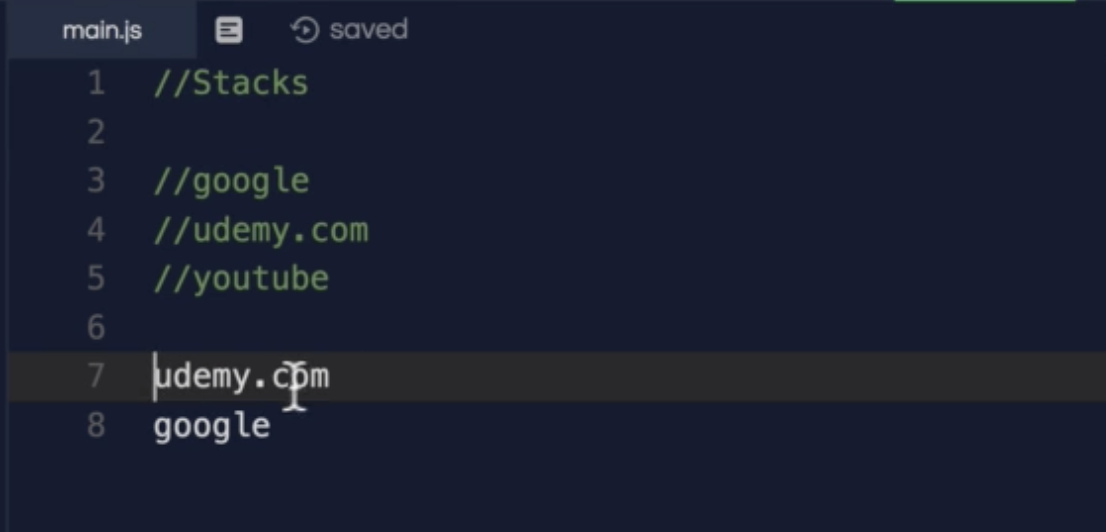
Generally we will not use arrays to build a queue. It is very inefficient, as we have to shift the indexes whenever there is a deletion I the array.

* **Exercise: Stacks vs Queues:**

****

**Graphical user interface, text, application

Description automatically generated**

****

**Text

Description automatically generated**

* **Solution: Stack vs Queue:**

**Stack:**

For stack we can use both (arrays or linked list). But the here are the pros and cons:

Arrays are much faster as all the elements are inside the same memory location (sequential). But Linked list are scattered.

Arrays need minimum memory when compared with LinkedList as it needs extra space to save the next elements pointer.

Another downside of dynamic arrays are if it reaches the maximum size, it will create a new array (in memory) with double the size of initial array.

**Queue:**

For Queue the best option is to use Linked List

Text

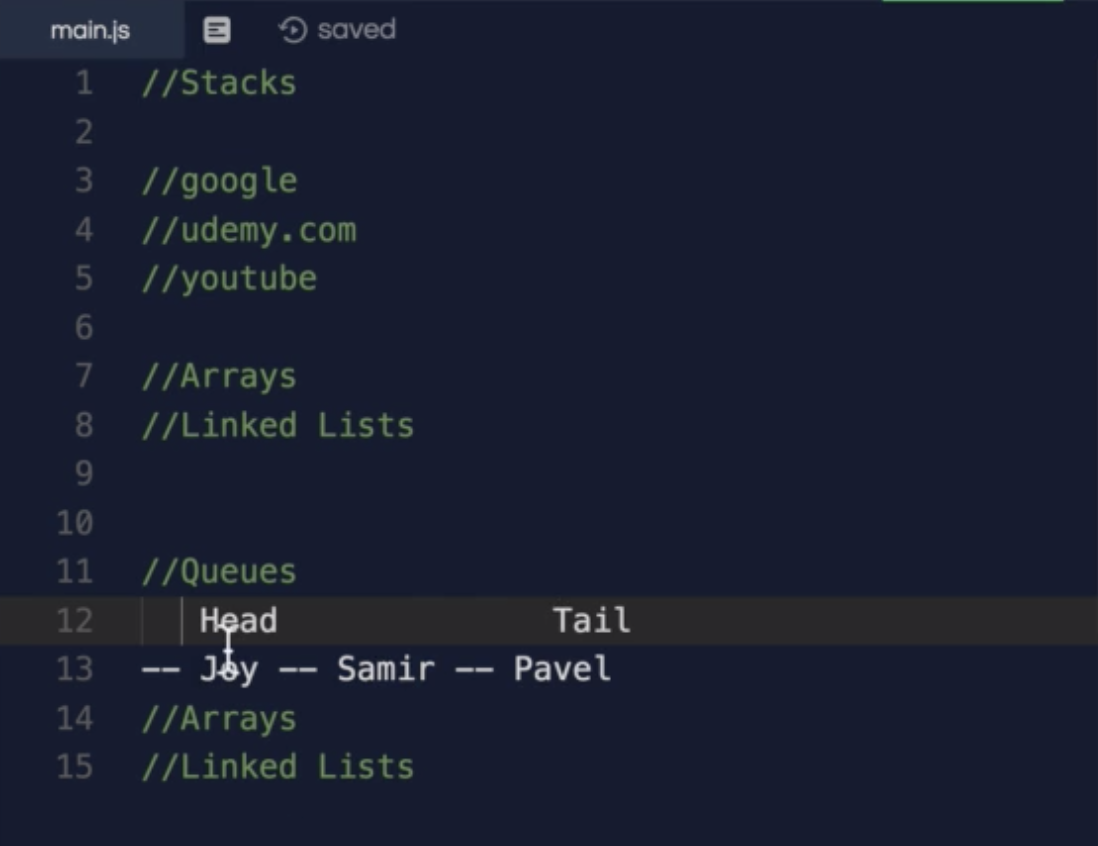
Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated



* **Stacks & Queue – Review:**

**A picture containing text

Description automatically generated**