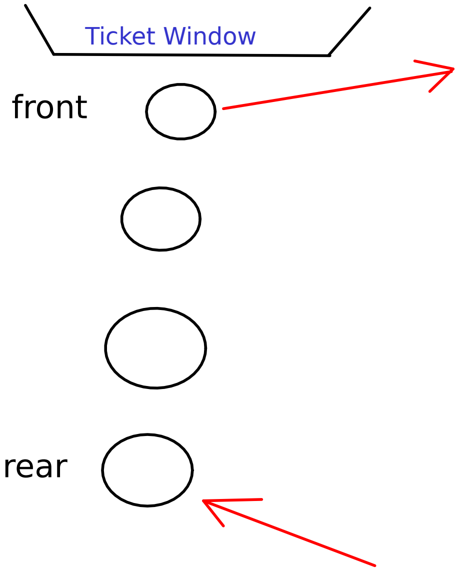
Abstract Data Type (ADT) --

In computer science, an abstract data type (ADT) is a mathematical model for data types.
An abstract data type is defined by its behavior (semantics) from the point of view of a user, of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations.

Eg. - Queue, Stack, Hash Table, Linked List, Binary Search Tree etc.

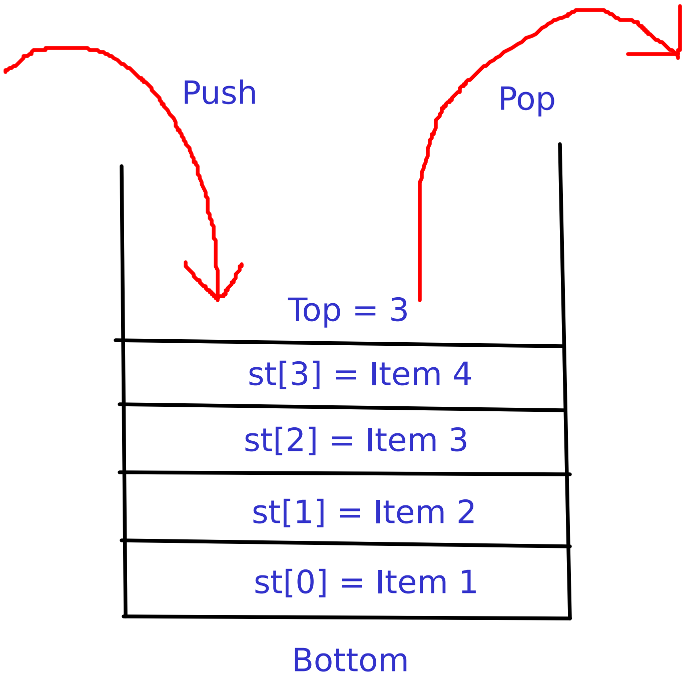
Queue<type> : FIFO (First In First Out) linear data structure,
it supports the operations of enqueue (at rear end) and dequeue (at front end)

Operations:
void enqueue(<Type> key)
<Type> dequeue()
int size()
<Type> peek()
bool is_empty()
bool is_full()



Stack<type> : LIFO (Last In First Out) linear data structure,
it supports the operations of push (insertion) and pop (deletion), both at the same end

Operations:
void push(<Type> key)
<Type> pop()
int size()
<Type> top() / peek()
bool is_empty()
bool is_full()



st[3] = Item 4
st[2] = Item 3
st[1] = Item 2
st[0] = Item 1

Array Implementation of Queue:

```
class Queue<Type T>
{
    T q[MAXSIZE];
    int front, rear;

    Queue() { rear = -1; front = 0; }

    void enqueue(T data) {
        if( is_full() == true )
            return "Error: Queue is full!";
        rear = (rear + 1) % MAXSIZE;
        q[rear] = data;
    }

    T dequeue() {
        if( is_empty() == true )
            return "Error: Queue is empty!";
        tmp = q[front];
        front = (front + 1) % MAXSIZE;
        return tmp;
    }

    T peek() { return q[front]; }

    int size() {
        int size = (rear - front + 1);
        if(size < 0)
            size = size + MAXSIZE;
        return size;
    }

    bool is_empty() { return ( size()==0 ); }

    bool is_full() { return ( size()==MAXSIZE ); }
}
```

Array Implementation of Stack:

```
class Stack<Type T>
{
    T st[MAXSIZE];
    int top;

    Stack() { top = -1; }

    void push(T data) {
        if( is_full() == true )
            return "Error: Stack is full!";
        top++;
        st[top] = data;
    }

    T pop() {
        if( is_empty() == true )
            return "Error: Stack is empty!";
        tmp = st[top];
        top--;
        return tmp;
    }

    T top() { return st[top]; }

    int size() { return (top+1); }

    bool is_empty() { return (top==-1); }

    bool is_full() { return (top==MAXSIZE-1); }
}
```

Operations to be performed (in sequence):
enqueue(44), enqueue(52), enqueue(69), enqueue(72), dequeue(), dequeue(), dequeue(), enqueue(37), dequeue(), enqueue(28), enqueue(13), dequeue(), dequeue()

q:	13	~~52~~	~~69~~	~~72~~	~~37~~	~~28~~
	0	1	2	3	4	5

rear = ~~-1~~ ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 0

front = ~~0~~ ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ 0