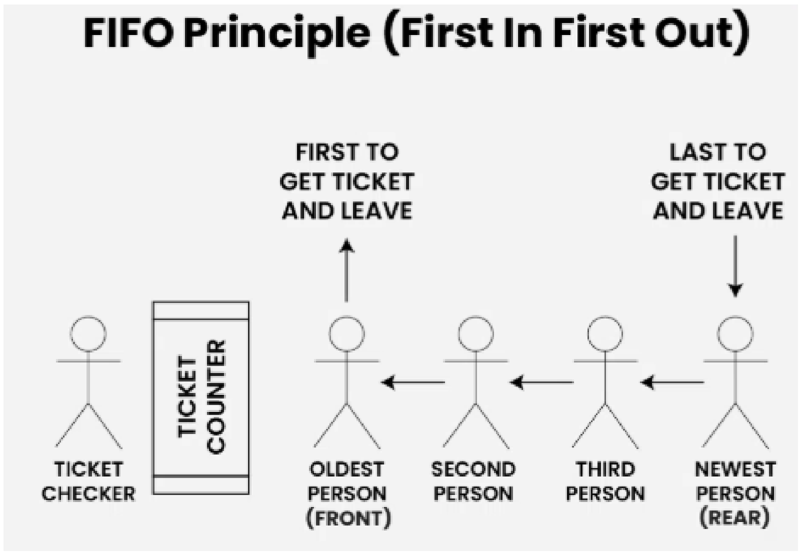
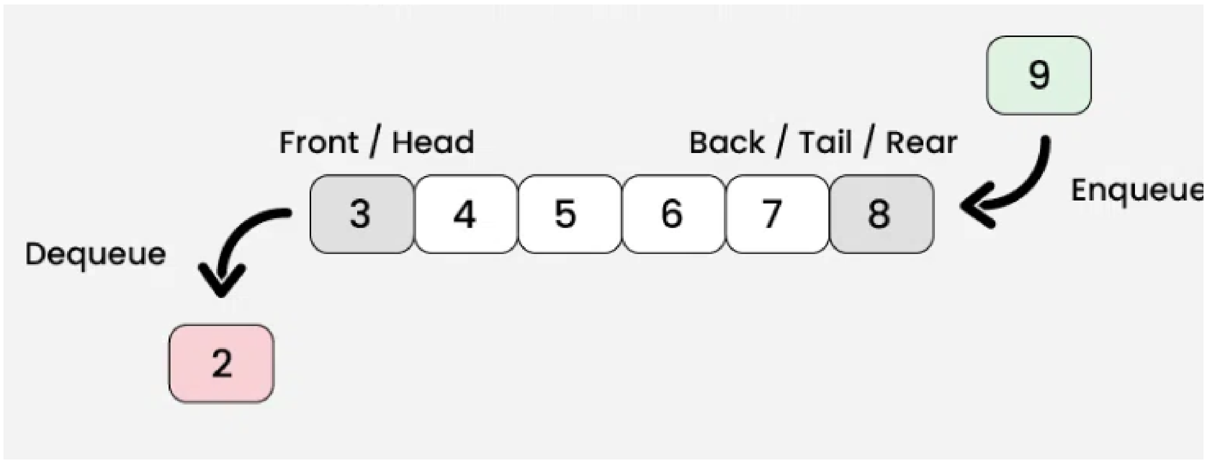
Queues

What is queue?

A queue is a data structure used for storing and managing data in specific order.
It follows the principle of “First in First out” (FIFO), where the first element to be added is the first one to remove.



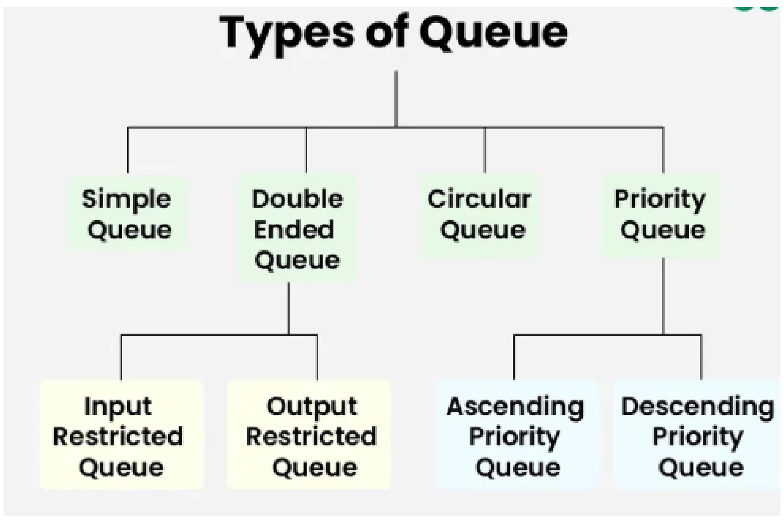
Representation of queue



1. Enqueue: adds an element to the end of the queue.
2. Dequeue: removes the element from the front of the queue.
3. peek/front: returns the element at the front without removing it.

4. Size: returns the number of elements in the queue.
5. isEmpty: returns true if the queue is empty, else returns false.
6. isFull: returns true if the queue is full, else returns false.

Types of queues



1. Simple queue: it follows FIFO structure. We can only insert elements at the back and remove the element from the front of the queue.

2. Double-ended queue (Deque): the insertion and deletion operations can be performed from both ends. They have two types:
    - Input restricted queue: in this type of queue, input can be taken from only one end but deletion can be done from any ends.
    - Output restricted queue: in this type of queue, deletion can be done from only one end but input can be taken from any ends.

3. Priority queue: In a priority queue elements are accessed based on the priority assigned to them. They have two types:
    - Ascending priority queue: elements are arranged in the increasing order of their priority values. The element with the smallest priority value is popped first.
    - Descending priority queue: elements are arranged in the decreasing order of their priority values. The element with the largest priority value is popped first.

4. Circular queue: the end of the queue is connected to the front of the queue forming a circular queue.

Basic operations on queue

- enqueue() - Insertion of elements to the queue.
- dequeue() - Removal of elements from the queue.
- getFront()- returns the element at the front end without removing it.
- getRear() - This operation returns the element at the rear end without removing it.
- isFull() - checks if the queue is full.
- isEmpty() - Checks if the queue is empty.
- size() - returns the total number of elements in the queue.

Initializing queue

In python

```python
from collections import deque
q = deque()
```

In C++

```cpp
#include <queue>
queue<int> q;
```

In javascript

Simple queue initialization:

```javascript
let queue = [];
```

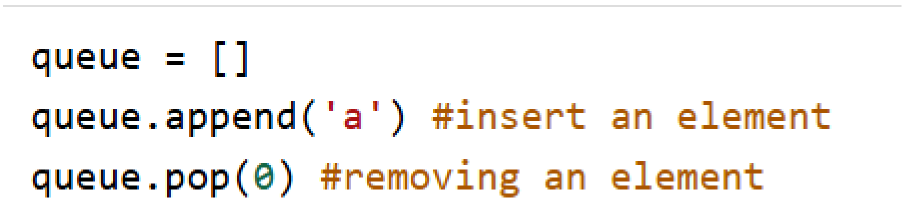
Queue as a class:

```javascript
class Queue {
  constructor() {
    this.items = [];
  }
```

Implementation of queue

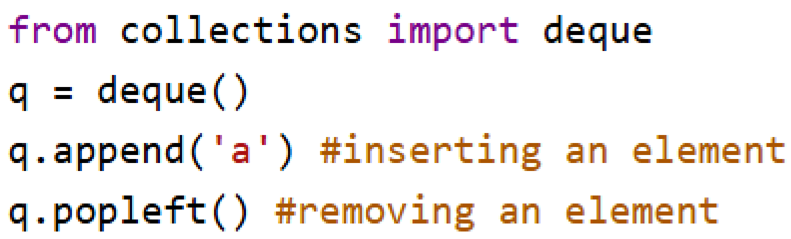
In python

Using list



```
queue = []
queue.append('a') #insert an element
queue.pop(0) #removing an element
```

Using deque

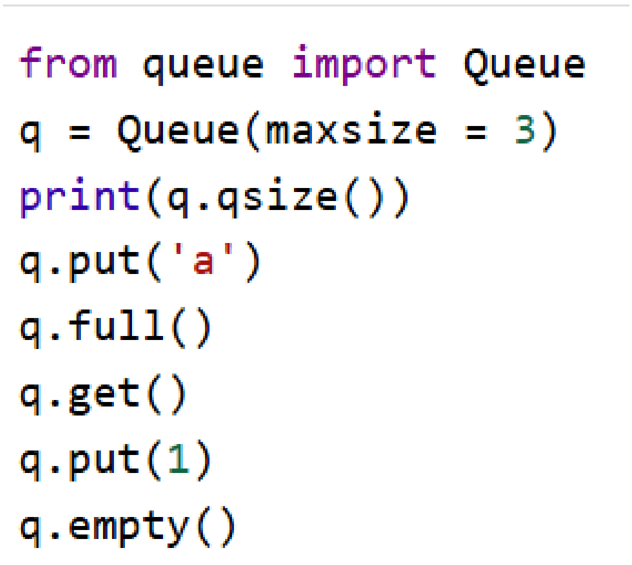


```
from collections import deque
q = deque()
q.append('a') #inserting an element
q.popleft() #removing an element
```

Using queue

There are various functions in this module:

- maxsize : number of elements allowed in the queue.
- empty() : return true if the queue is empty, else false.
- full() : return true if there are maxsize elements in the queue. If the queue is initialized with maxsize=0, then full() never returns true.
- get() : remove and return an item from the queue. If the queue is empty, wait until an item is available.
- get_nowait() : Return an item if one is immediately available, else raise QueueEmpty.
- put(element) : Put an item into the queue. If the queue is full, wait until a free slot is available before adding the item.
- put_nowait(element) : Put an item into the queue without blocking. If no free slot is immediately available, raise QueueFull.
- qsize() : Return the number of items in the queue.



```python
from queue import Queue
q = Queue(maxsize = 3)
print(q.qsize())
q.put('a')
q.full()
q.get()
q.put(1)
q.empty()
```

To learn in more detail you can refer: [Queue in Python | GeeksforGeeks]()

In C++

Functions	Description
front()	Access the front element of the queue.
back()	Access the end element of the queue.

empty()	Check whether a queue is empty or not.
size()	Returns the number of elements in the queue.
push()	Adding an element at the back of the queue.
push_range()	Adding multiple elements at the end of queue.
emplace()	Add the constructs element in top of the queue.
pop()	Delete the front element of the queue.
swap()	Swap two queues.

Inserting an element

```
// Pushing elements into the queue
q.push(3);
```

Accessing element

```
// Accessing the front and back elements
q.front();
q.back();
```

Deleting element (dequeue)

```
// Deleting elements from front side of the queue
q.pop();
```

Pseudo traversal

Since only the front and back element can be accessed in a queue, we cannot directly traverse it. We can create a copy of the queue, access the front element, and then delete it.

```
q.push(3);
q.push(4);
q.push(5);

// Create a copy
queue<int> temp(q);

while(!temp.empty()) {
    cout << temp.front() << " ";
    temp.pop();
}
```

To learn in more detail you can refer: [Queue in C++ STL | GeeksforGeeks]

In javascript

Using array

Inserting element

```
enqueue(element) {
  this.items.push(element);
}
```

Removing element

```
dequeue() {
  return this.items.shift();
}
```

Some other operations

```
 peek() {
   return this.items[0];
 }

 isEmpty() {
   return this.items.length === 0;
 }

 size() {
   return this.items.length;
 }
```

Using linked list

```
// Queue class using linked list
class Queue {
  constructor() {
    this.front = null; // Head of the queue
    this.rear = null;  // Tail of the queue
    this.length = 0;
  }
```

```
// Add to the end of the queue
enqueue(value) {
  const newNode = new Node(value);
  if (this.isEmpty()) {
    this.front = this.rear = newNode;
  } else {
    this.rear.next = newNode;
    this.rear = newNode;
  }
  this.length++;
}
```

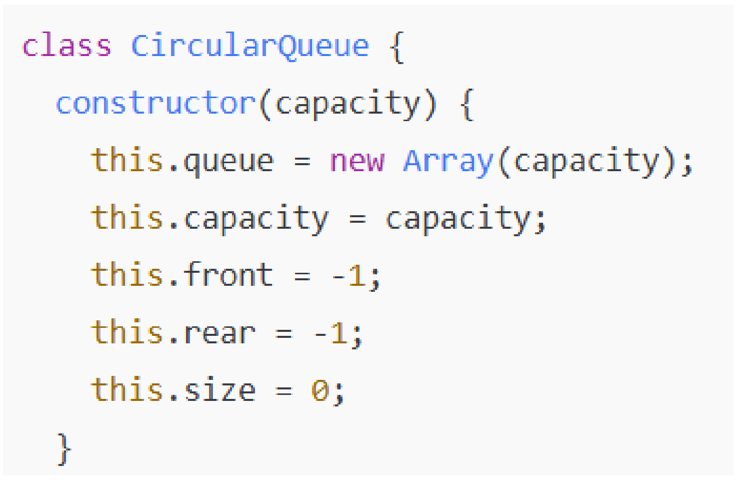
```
// Remove from the front of the queue
dequeue() {
  if (this.isEmpty()) return null;

  const removedValue = this.front.value;
  this.front = this.front.next;
  this.length--;

  if (this.isEmpty()) {
    this.rear = null; // Reset rear if queue becomes empty
  }
```

```
// Peek at the front value
peek() {
  return this.front ? this.front.value : null;
}
```

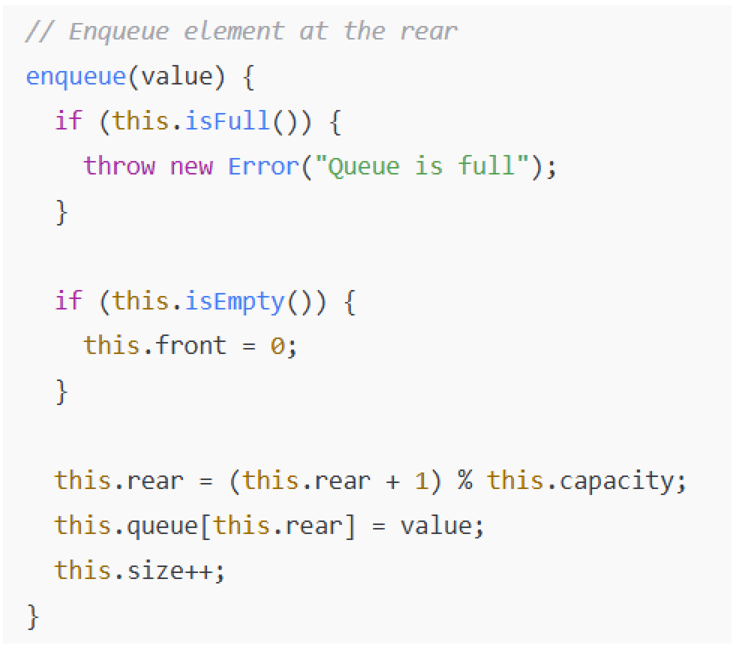
```
isEmpty() {
  return this.length === 0;
}

size() {
  return this.length;
}
```

Using circular array

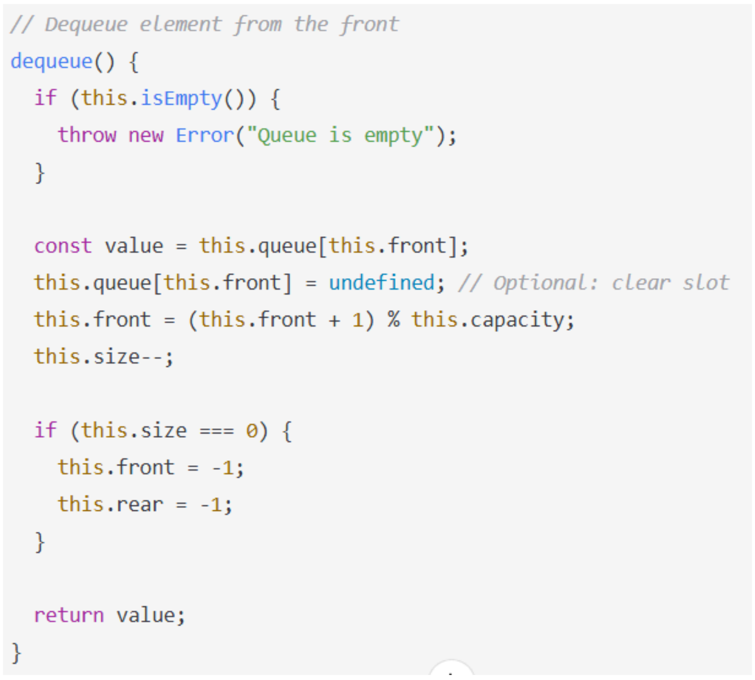
In a Circular Array Queue, we use a fixed-size array to store the elements of the queue. However, the key idea is that the array is circular—meaning the last element connects back to the first element in a loop, forming a continuous, circular structure.



```
class CircularQueue {
  constructor(capacity) {
    this.queue = new Array(capacity);
    this.capacity = capacity;
    this.front = -1;
    this.rear = -1;
    this.size = 0;
  }
```



```
// Enqueue element at the rear
enqueue(value) {
  if (this.isFull()) {
    throw new Error("Queue is full");
  }

  if (this.isEmpty()) {
    this.front = 0;
  }

  this.rear = (this.rear + 1) % this.capacity;
  this.queue[this.rear] = value;
  this.size++;
}
```



```
// Dequeue element from the front
dequeue() {
  if (this.isEmpty()) {
    throw new Error("Queue is empty");
  }

  const value = this.queue[this.front];
  this.queue[this.front] = undefined; // Optional: clear slot
  this.front = (this.front + 1) % this.capacity;
  this.size--;

  if (this.size === 0) {
    this.front = -1;
    this.rear = -1;
  }

  return value;
}
```

```
// Get front value
peek() {
  if (this.isEmpty()) return null;
  return this.queue[this.front];
}
```

```
isEmpty() {
  return this.size === 0;
}

isFull() {
  return this.size === this.capacity;
}
```

To learn in more detail you can refer: Implementation of Queue in Javascript | GeeksforGeeks

Resources

https://www.geeksforgeeks.org/queue-data-structure/
https://www.geeksforgeeks.org/deque-set-1-introduction-applications/

Practice problems

Easy

1. Implement Queue using Stacks - LeetCode | solution
2. https://leetcode.com/problems/first-unique-character-in-a-string/ | solution
3. https://leetcode.com/problems/number-of-recent-calls/ | solution
4. https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/ | solution
5. https://leetcode.com/problems/time-needed-to-buy-tickets/ | solution

Medium

1. https://leetcode.com/problems/flatten-nested-list-iterator/ | solution
2. https://leetcode.com/problems/dota2-senate/ | solution
3. https://leetcode.com/problems/maximum-sum-circular-subarray/ | solution
4. https://leetcode.com/problems/reveal-cards-in-increasing-order/ | solution

5. https://leetcode.com/problems/number-of-people-aware-of-a-secret/ | solution

Hard

1. https://leetcode.com/problems/shortest-subarray-with-sum-at-least-k/ | solution
2. https://leetcode.com/problems/stamping-the-sequence/ | solution
3. https://leetcode.com/problems/delivering-boxes-from-storage-to-ports/ | solution