

LẬP TRÌNH JAVASCRIPT CƠ BẢN

NGĂN XẾP (STACK) VÀ HÀNG ĐỢI (QUEUE)



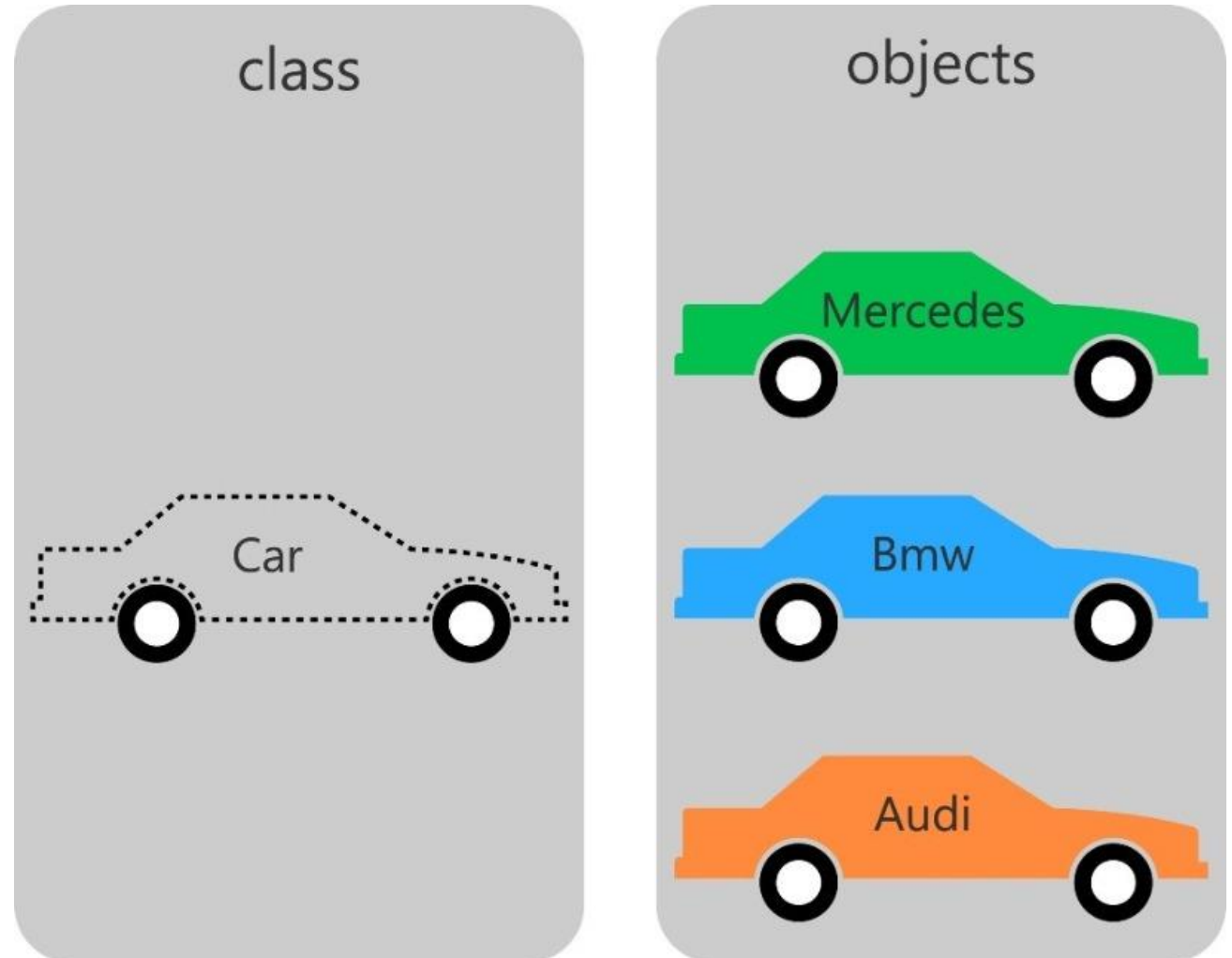
Nội dung:

1. Từ khóa class, constructor trong JavaScript
2. Ngăn xếp (Stack) là gì?
3. Các kỹ thuật để thao tác với ngăn xếp (Stack)
4. Hàng đợi (Queue) là gì?
5. Các kỹ thuật để thao tác với hàng đợi (Queue)
6. So sánh ngăn xếp (Stack) và hàng đợi (Queue)



Từ khóa class, constructor trong JavaScript

- ❑ Trong **ES6**, JavaScript đã giới thiệu từ khóa **class** để làm mẫu cho các đối tượng trong JavaScript.
- ❑ Khi dùng **class**, chúng ta thường **thêm 1 phương thức** có tên **constructor()** trong class để làm **phương thức khởi tạo cho lớp đối tượng**.



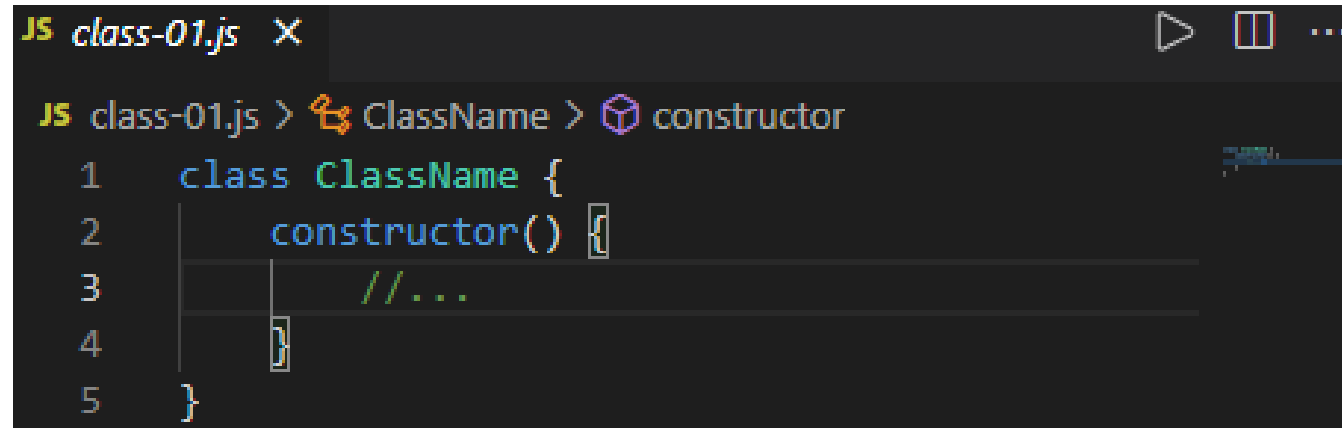


Từ khóa class, constructor trong JavaScript

❑ Một **class** trong JavaScript **không phải là một đối tượng**.

❑ Cú pháp:

```
class ClassName {  
    constructor() { ... }  
}
```



The screenshot shows a code editor with a file named 'class-01.js'. The code defines a class 'ClassName' with a constructor method. The code is as follows:

```
JS class-01.js > ClassName > constructor  
1 class ClassName {  
2     constructor() {  
3         //...  
4     }  
5 }
```

❑ Tên class thường viết hoa, không theo kiểu camel case.

❑ Một class thường **có thể không có** hoặc **sẽ có 1 hoặc nhiều**:

➤ Thuộc tính (properties)

➤ Phương thức (method)

❑ **Lưu ý**: nếu dùng class để khởi tạo đối tượng mà **không xác định một phương thức khởi tạo**, JavaScript sẽ thêm **một phương thức khởi tạo rỗng**.



Từ khóa class, constructor trong JavaScript

- ❑ Phương thức khởi tạo là **một phương thức đặc biệt**:
 - Được dùng với **từ khóa constructor**.
 - Được thực thi tự động khi một đối tượng được tạo.
 - Được dùng để khởi tạo các thuộc tính của đối tượng.

❑ Ví dụ:

```
class Car {  
    constructor(name, year) {  
        this.name = name;  
        this.year = year;  
    }  
}
```

```
JS class-02.js X  
JS class-02.js > ...  
1  class Car {  
2      constructor(name, year) {  
3          this.name = name;  
4          this.year = year;  
5      }  
6  }  
7  
8  let carA = new Car();  
9  console.log(carA);  
10  
11 let carB = new Car("Ford", 2014);  
12 console.log(carB);
```



Từ khóa class, constructor trong JavaScript

- ❑ Các phương thức của class được tạo với cú pháp **giống như** các phương thức của đối tượng.
- ❑ Chúng ta **có thể thêm vào class** một hoặc nhiều phương thức khác nhau.
- ❑ Ví dụ:

```
class ClassName {  
    constructor() { ... }  
    methodA() { ... }  
    methodB() { ... }  
    methodC() { ... }  
}
```

```
JS class-03.js X  
JS class-03.js > ...  
1  class Car {  
2      constructor(name, year) {  
3          this.name = name;  
4          this.year = year;  
5      }  
6      age() {  
7          let date = new Date();  
8          return date.getFullYear() - this.year;  
9      }  
10 }  
11  
12 let myCar = new Car("Ford", 2014);  
13 console.log("Xe của tôi đã " + myCar.age() + " tuổi.");
```



Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Ngăn xếp (Stack) là gì?

- Ngăn xếp (stack) là một cấu trúc dữ liệu trừu tượng hoạt động theo nguyên lý "**vào sau ra trước**". (LIFO: Last In First Out)
- Nguyên lý "**vào sau ra trước**": nghĩa là phần tử **cuối cùng** được chèn vào **ngăn xếp** sẽ là phần tử **đầu tiên** được lấy ra khỏi **ngăn xếp**.

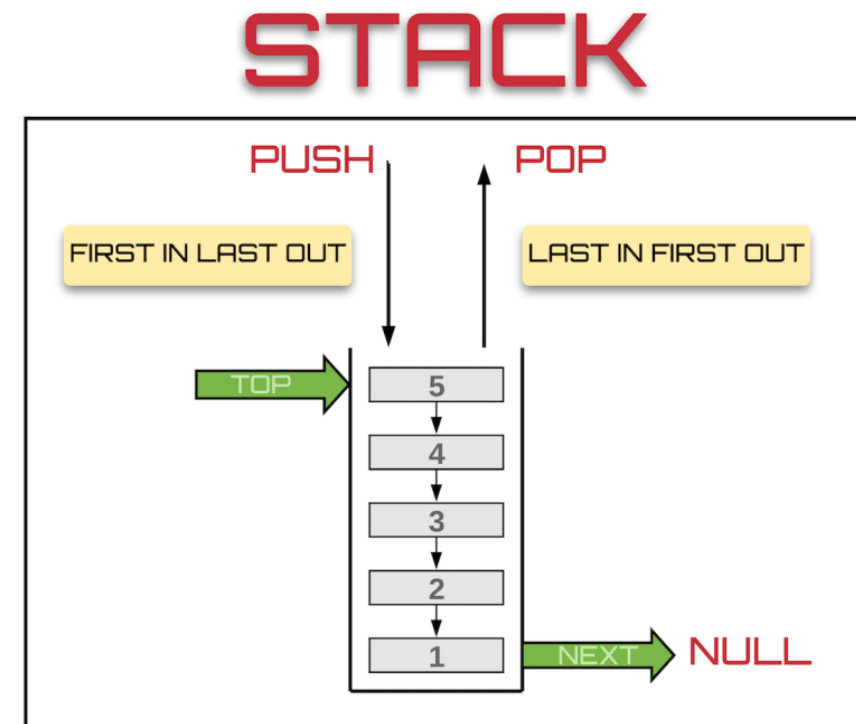
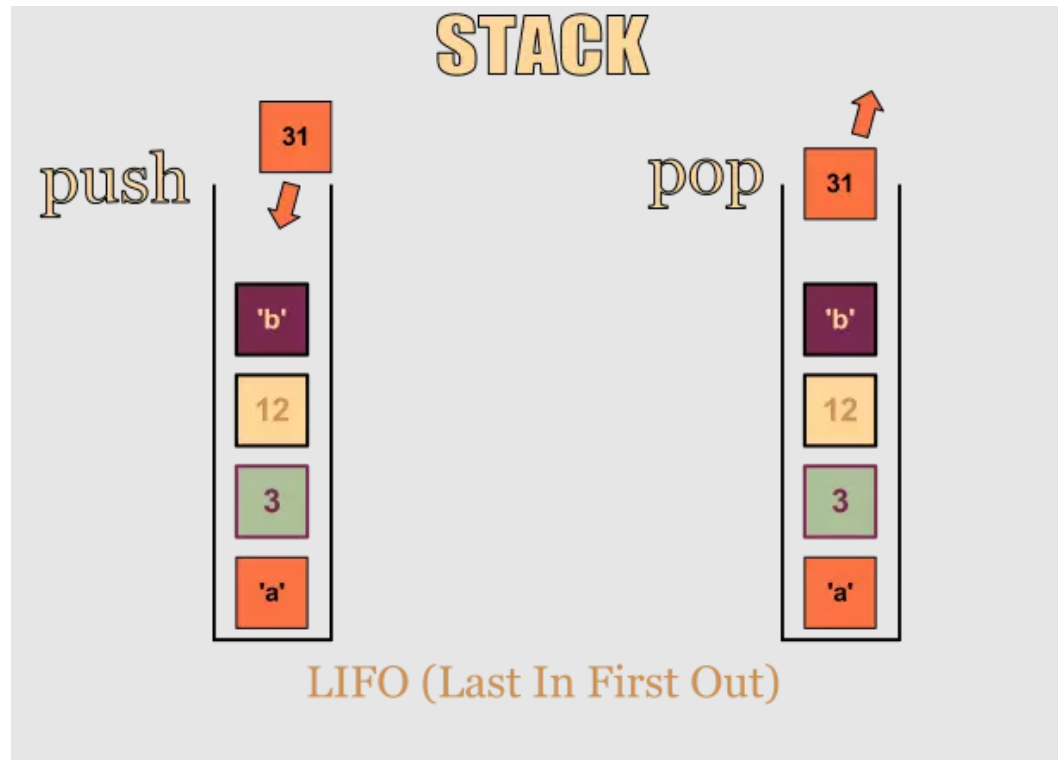




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ **push**: thêm 1 phần tử vào đỉnh của ngăn xếp, số phần tử của ngăn xếp tăng thêm 1.

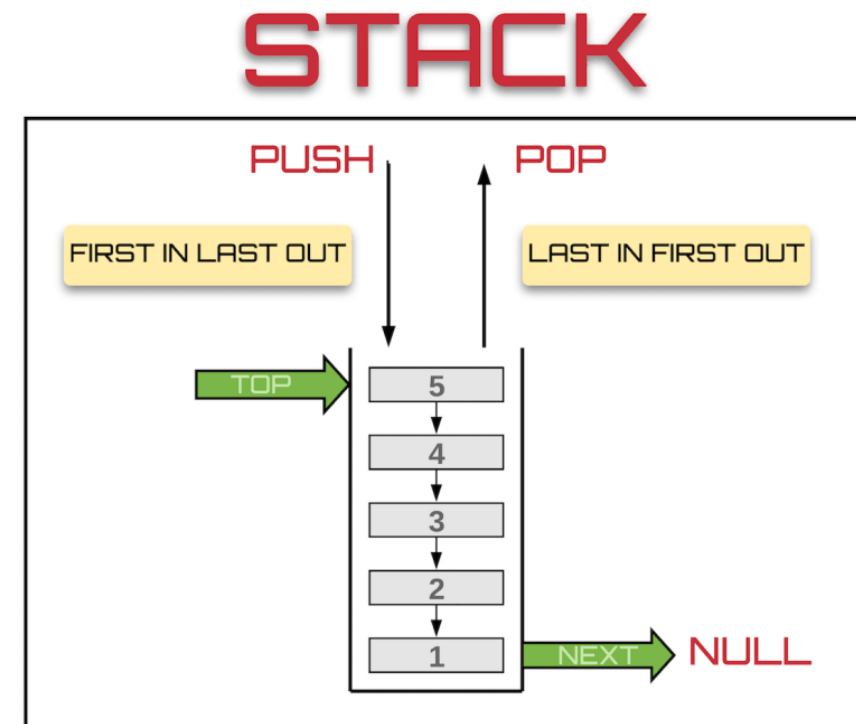
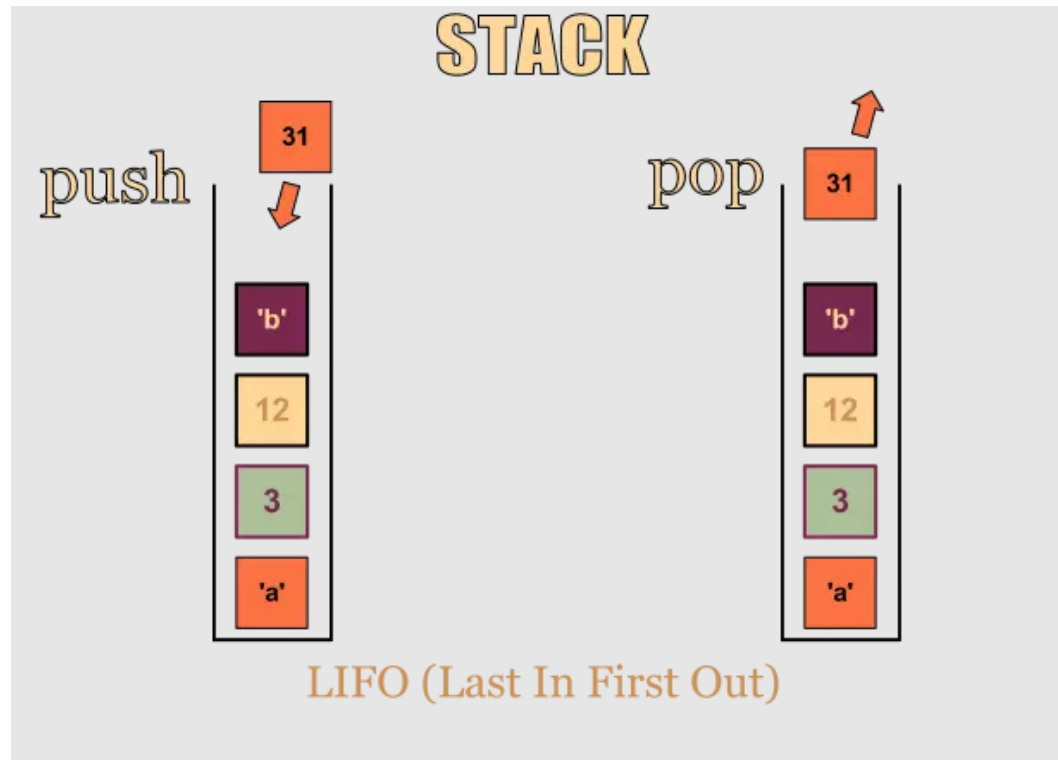




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ **pop**: xoá phần tử đầu tiên ở đỉnh của ngăn xếp, số phần tử của ngăn xếp giảm 1.

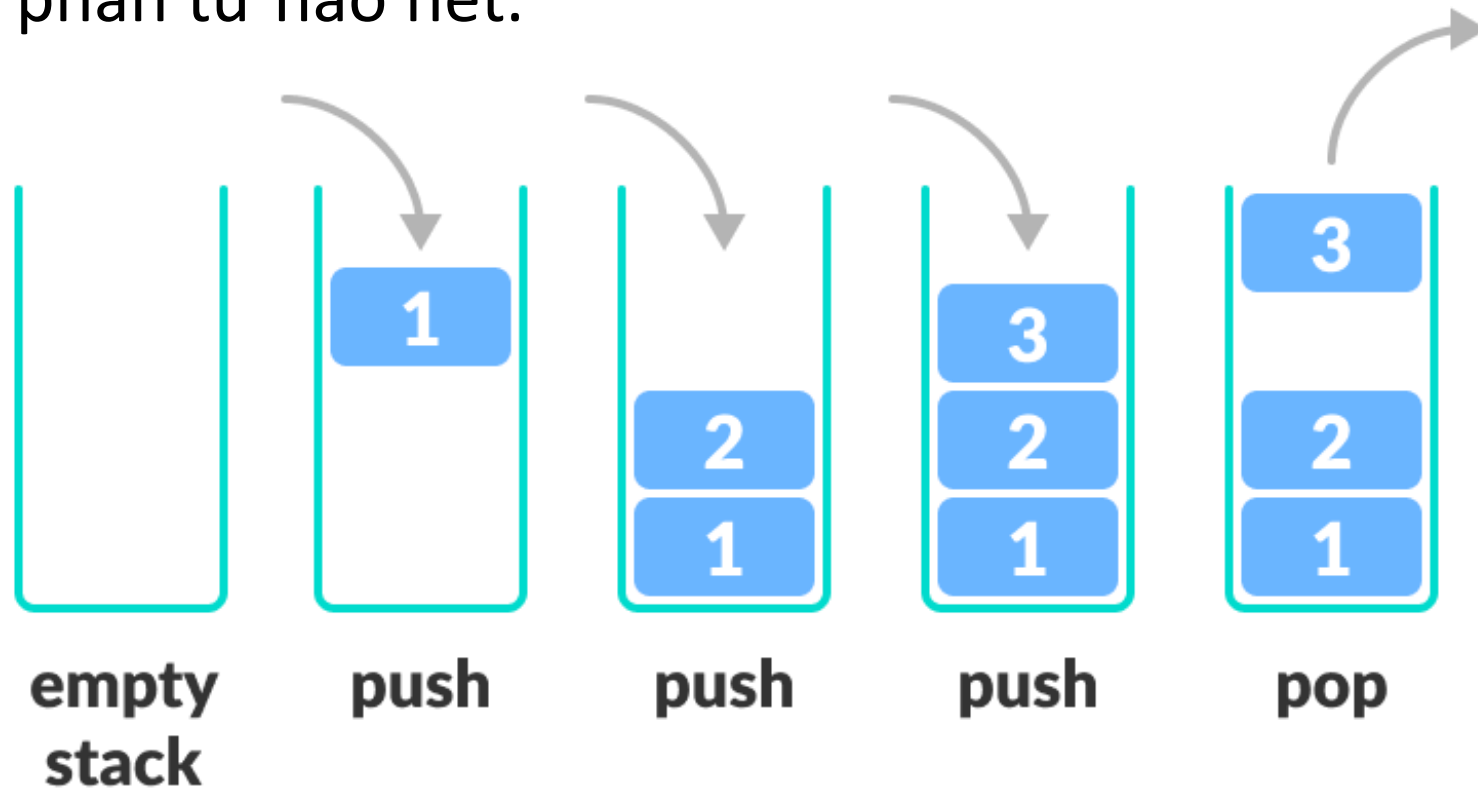




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ **isEmpty**: kiểm tra ngăn xếp có trống hay không, ngăn xếp trống là ngăn xếp không có phần tử nào hết.

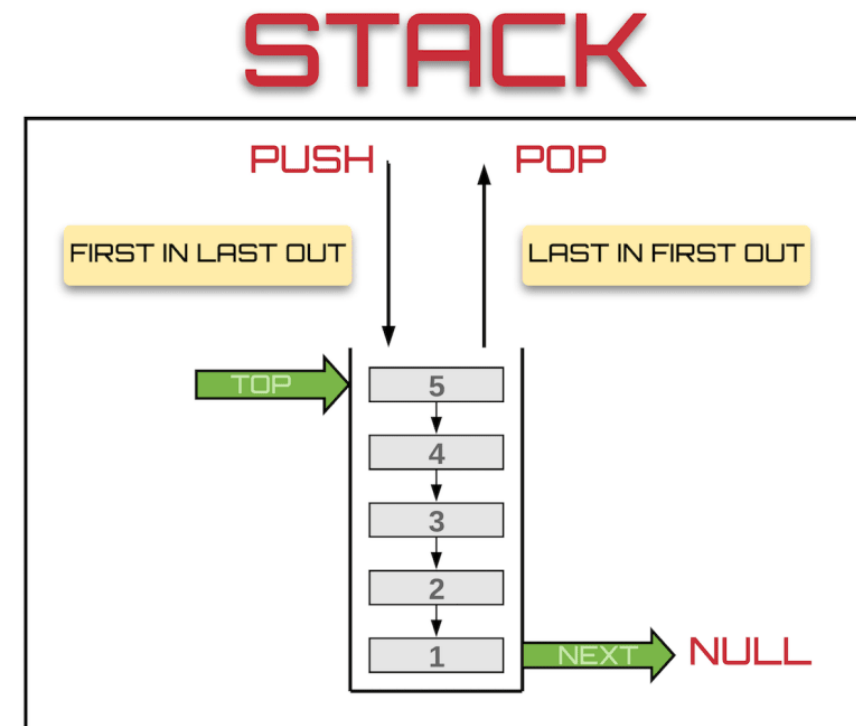
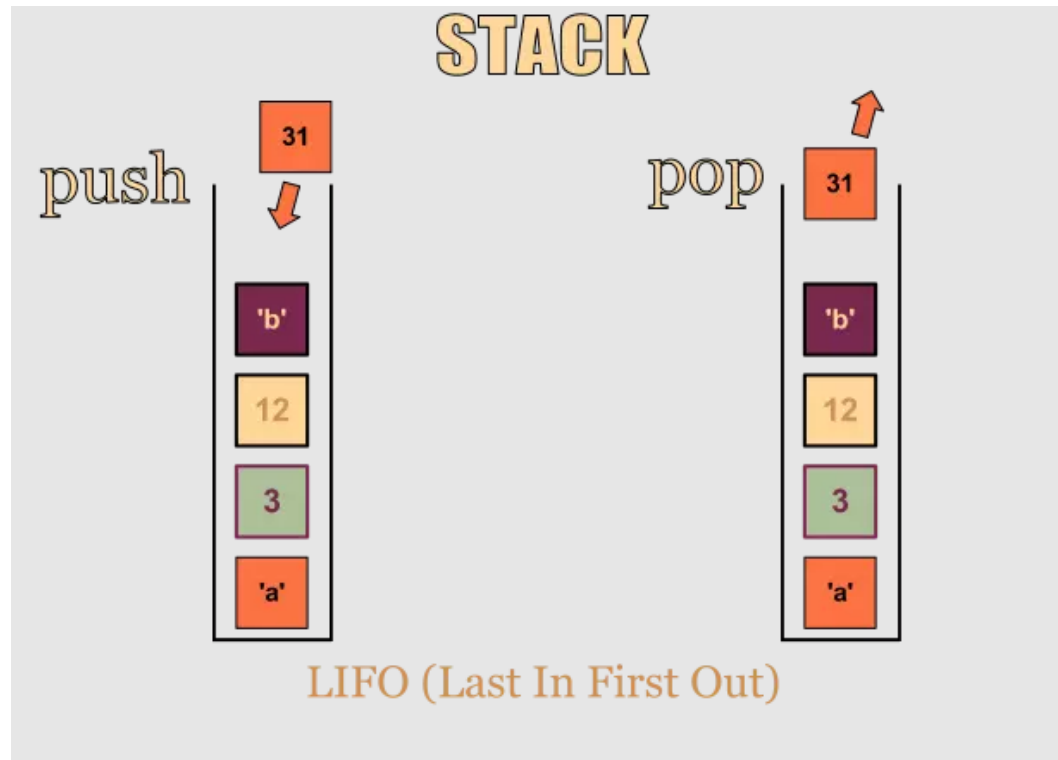




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ **isFull**: kiểm tra ngăn xếp đã đầy hay chưa, ngăn xếp đầy là ngăn xếp đã có số lượng phần tử bằng với giới hạn của mảng (length) đã khởi tạo ban đầu.





Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ Khai báo ngăn xếp:

```
let stack = [];
```

```
var capacity; // Sức chứa của ngăn xếp
```



Stack of books



Stack of Coins



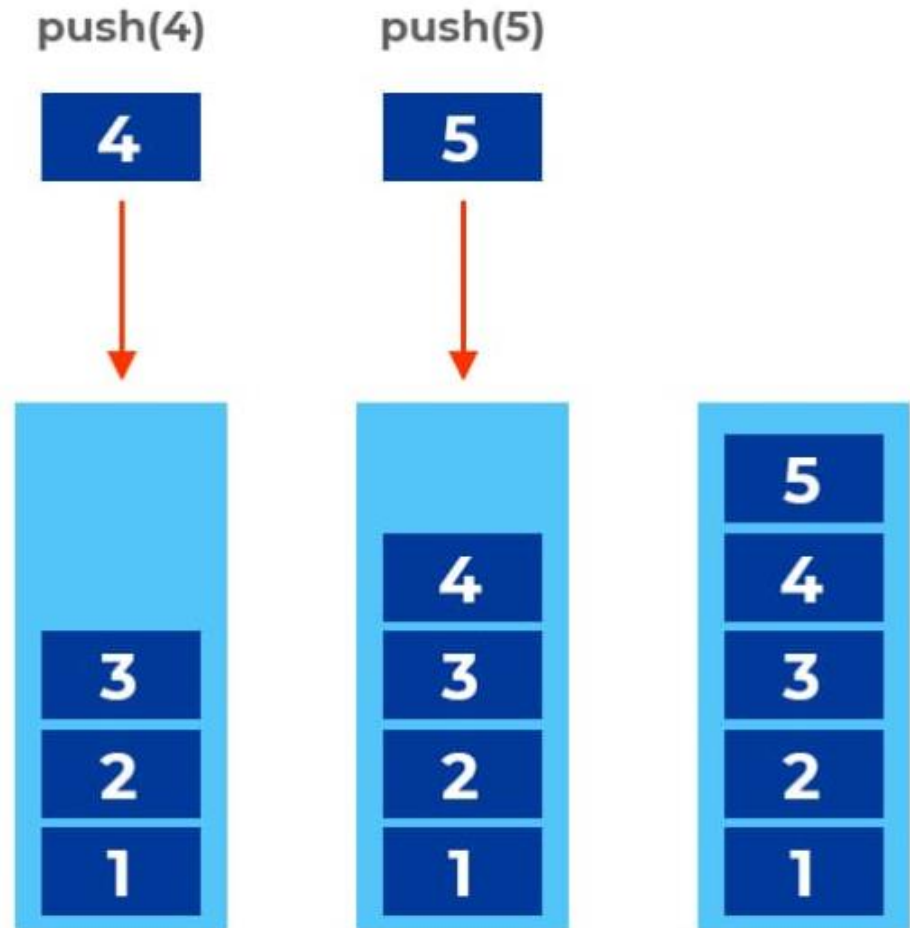


Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ Kiểm tra ngăn xếp **đã đầy**?

```
function isFull() {  
    // số lượng phần tử >= sức chứa  
    if (stack.length == capacity) {  
        return true;  
    } else {  
        return false;  
    }  
}
```



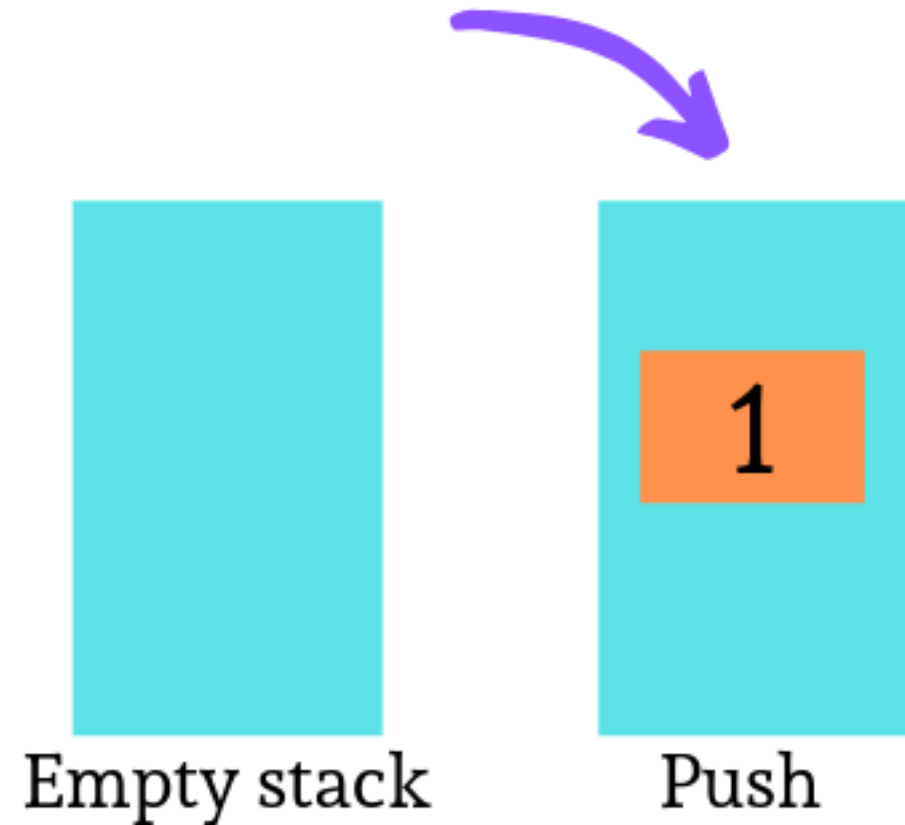


Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ Kiểm tra ngăn xếp **đang rỗng**?

```
function isEmpty() {  
    // số lượng phần tử = 0  
    if (stack.length == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```





Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

- Thêm phần tử vào ngăn xếp (**push**)

```
function push(item) {  
    if (isFull() == true) {  
        console.log("Stack is full");  
    } else {  
        stack.push(item);  
    }  
}
```



Stack of books



Stack of Coins



Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

- Thêm phần tử vào ngăn xếp (**push**)

`stack.push(x);`

- Ví dụ:

```
let stack = [];
```

```
stack.push(1);
```

```
stack.push(2);
```

```
stack.push(3);
```

```
stack.push(4);
```

```
stack.push(5);
```



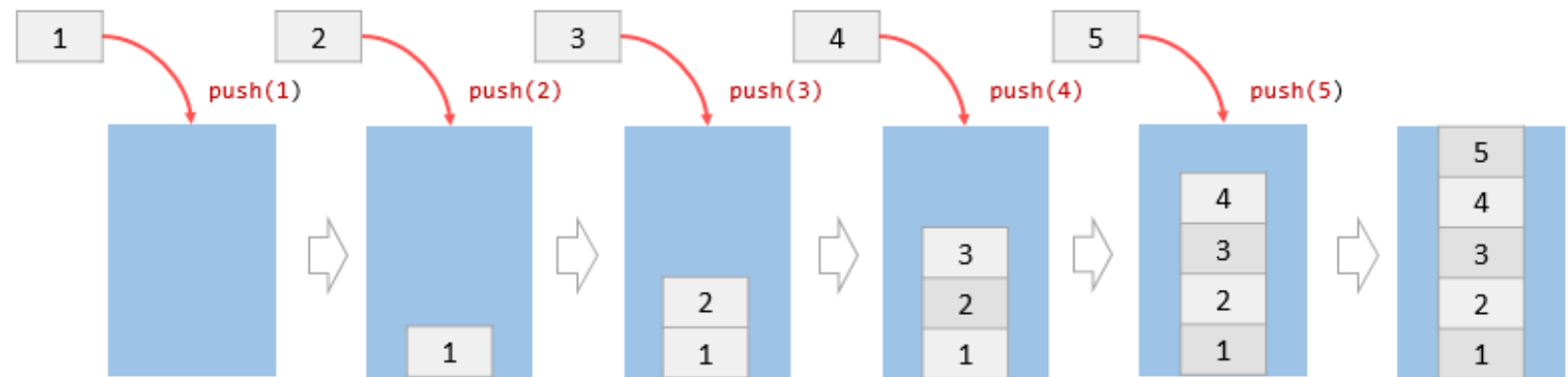
Stack of Books



Stack of Dishes



Stack of Discs





Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

- Xóa phần tử khỏi ngăn xếp (**pop**)

```
function pop() {  
    if (isEmpty() == true) {  
        console.log("Stack is empty");  
    } else {  
        stack.pop();  
    }  
}
```



Stack of books



Stack of Coins



Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

- Xóa phần tử khỏi ngăn xếp (**pop**)

`stack.pop();`

- Ví dụ:

`stack.pop();`

`stack.pop();`

`stack.pop();`

`stack.pop();`

`stack.pop();`



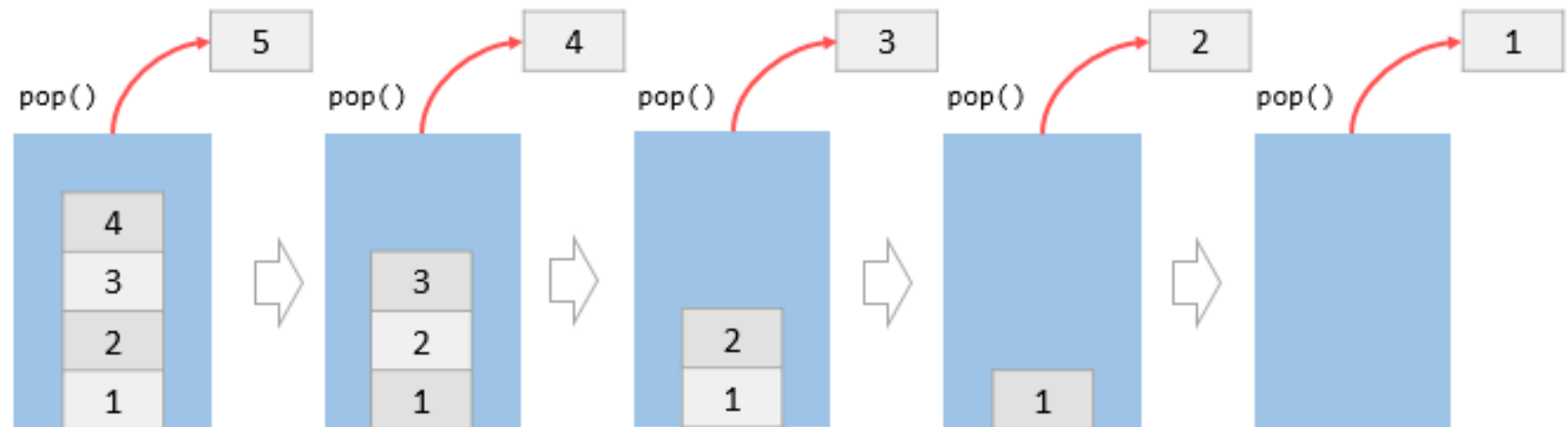
Stack of Books



Stack of Dishes



Stack of Discs



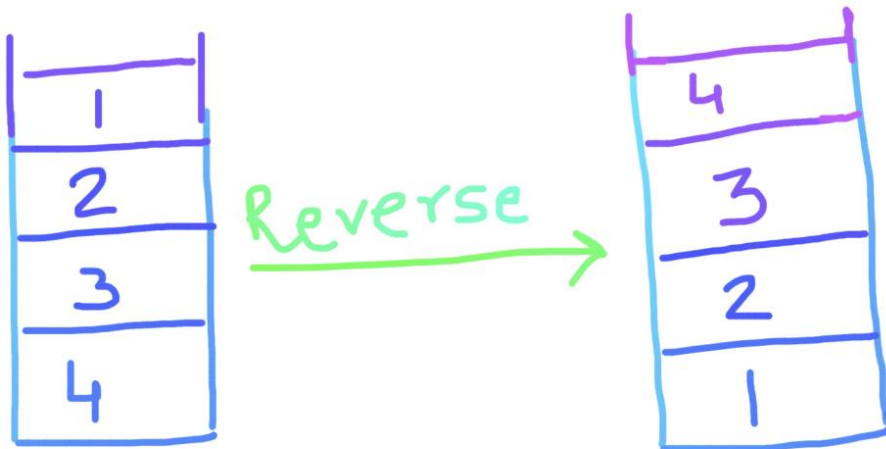


Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với ngăn xếp (Stack):

➤ Nghịch đảo ngăn xếp (**reverse**)

```
function reverse() {  
    return stack.reverse();  
}
```



Stack of books



Stack of Coins



Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Hàng đợi (Queue) là gì?

- Ngăn xếp (stack) là một cấu trúc dữ liệu trừu tượng hoạt động theo nguyên lý "**vào trước ra trước**". (FIFO: First In First Out)
- Nguyên lý "**vào trước ra trước**": nghĩa là phần tử **đầu tiên** được chèn vào **hàng đợi** sẽ là phần tử **đầu tiên** được lấy ra khỏi **hàng đợi**.

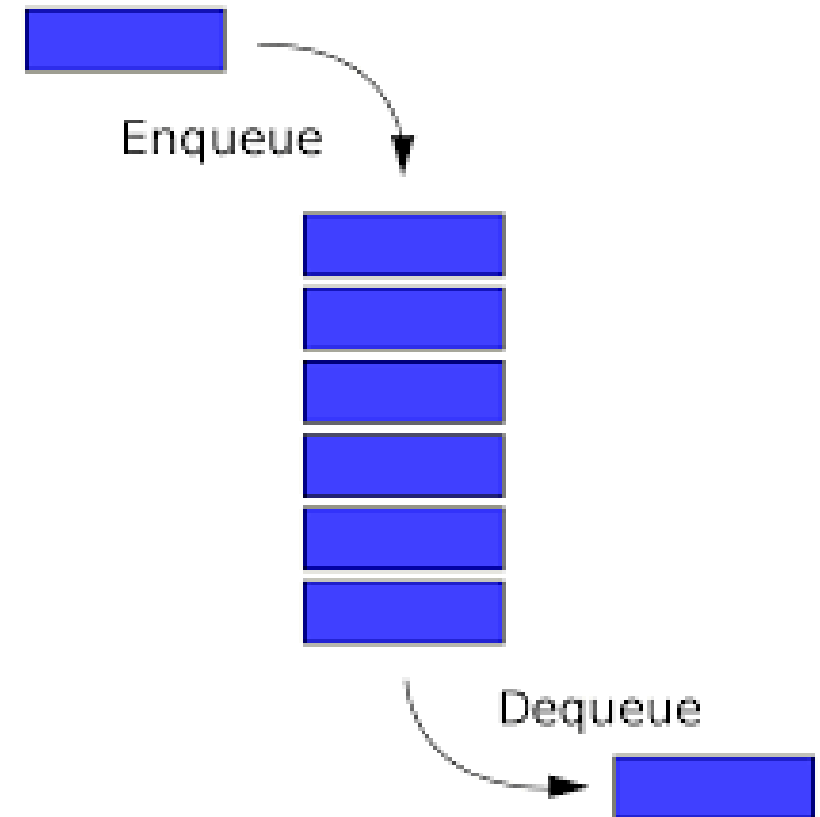
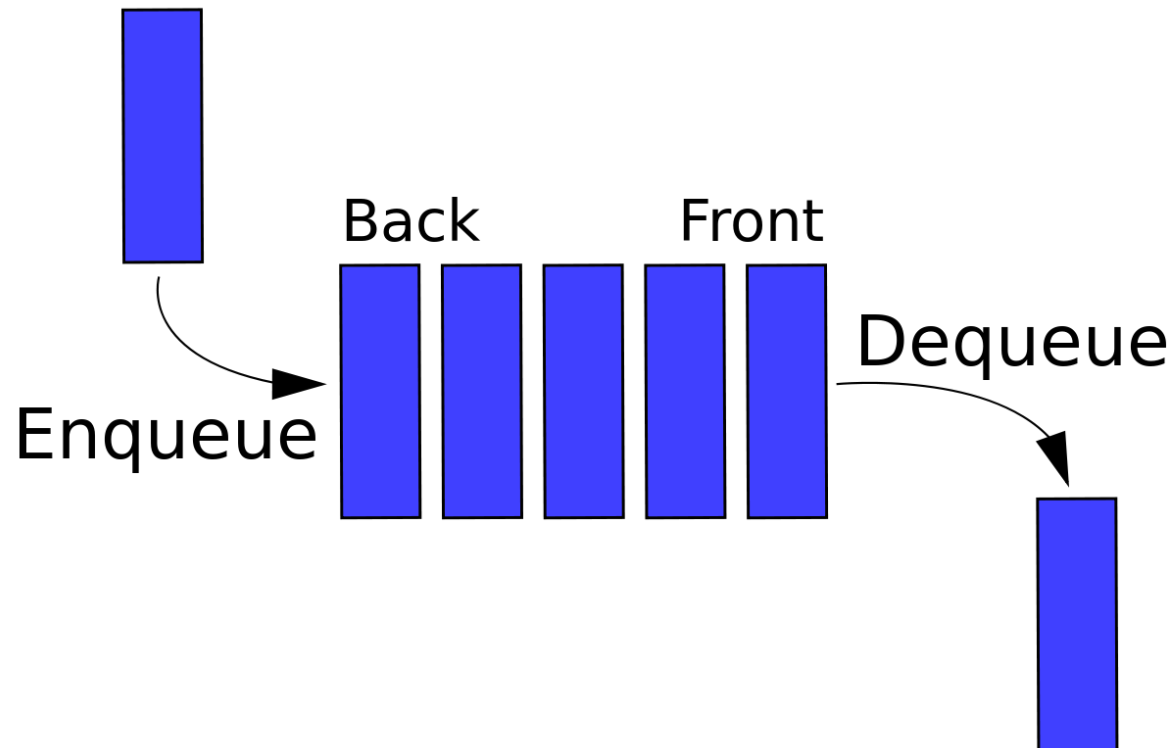




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ **enqueue**: thêm 1 phần tử vào cuối (rear) của hàng đợi, số phần tử của hàng đợi tăng thêm 1.

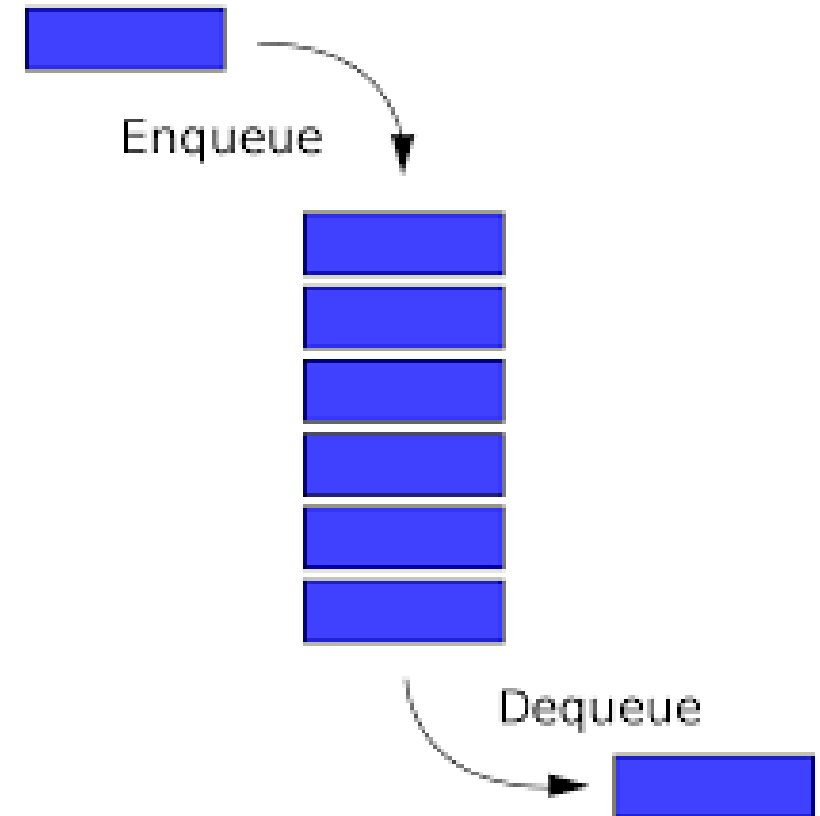
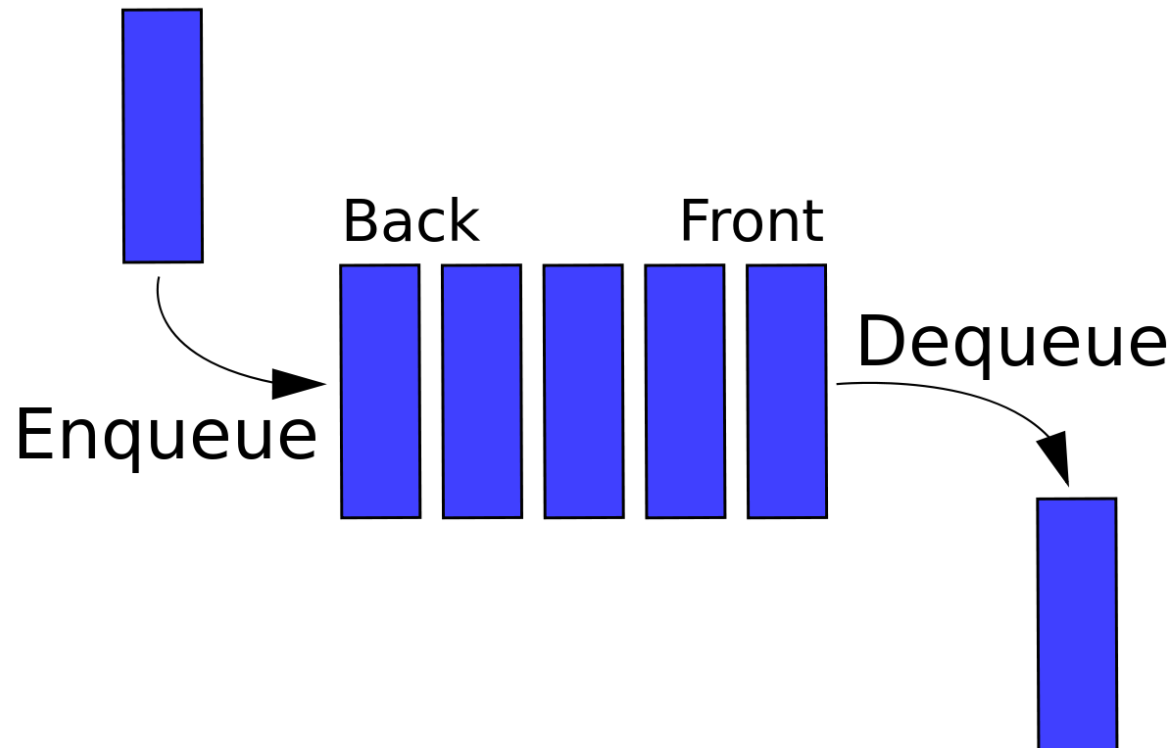




Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ **dequeue**: xoá phần tử khỏi đầu (front) của hàng đợi, số phần tử của ngăn xếp giảm 1.



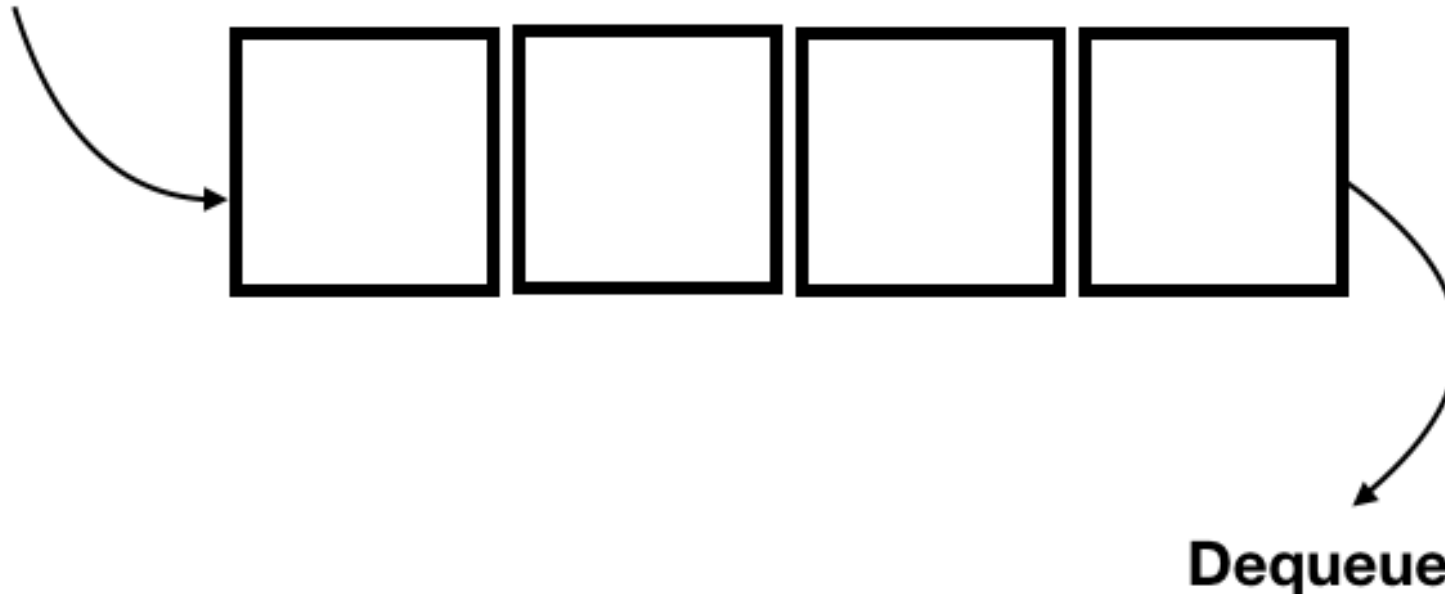


Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ **isEmpty**: kiểm tra hàng đợi có trống hay không, hàng đợi trống là hàng đợi không có phần tử nào hết.

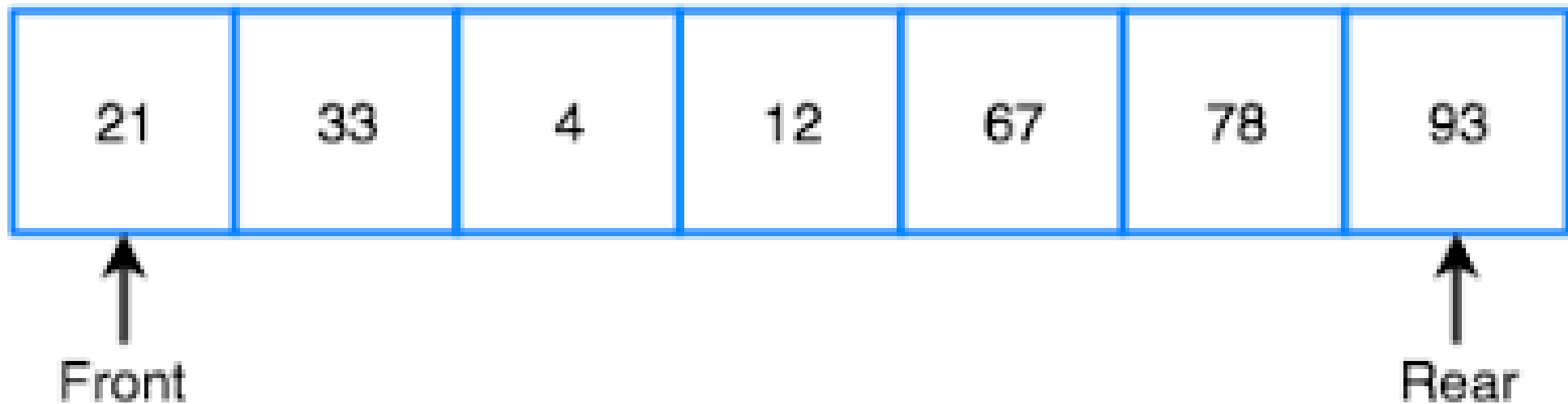
Enqueue





➤ **isFull**: kiểm tra hàng đợi đã đầy hay chưa, hàng đợi đầy là hàng đợi đã có số lượng phần tử bằng với giới hạn của mảng (length) đã khởi tạo ban đầu.

Queue is Full





Ngăn xếp (Stack), Hàng đợi (Queue)

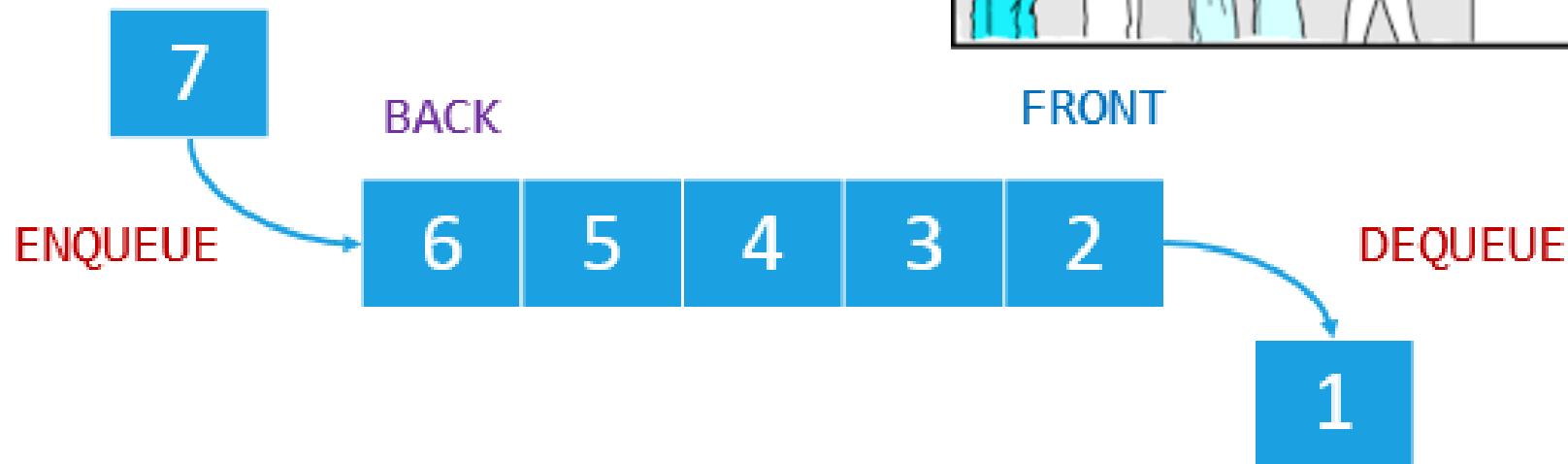
❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ Khai báo hàng đợi:

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

```
var front; // Đầu hàng đợi
```

```
var rear; // Cuối hàng đợi
```





➤ Kiểm tra hàng đợi **đã đầy?**

// số lượng phần tử = kích cỡ Queue is Full

Diagram illustrating a queue implemented as an array. The array contains the values 21, 33, 4, 12, 67, 78, and 93. The 'Front' pointer points to the first element (21), and the 'Rear' pointer points to the last element (93).

```
} else {
```

}

}



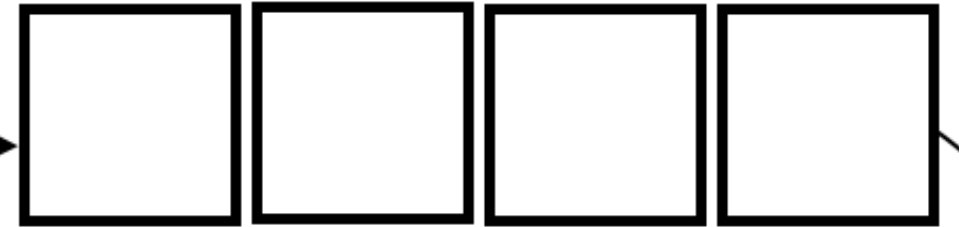
Ngăn xếp (Stack), Hàng đợi (Queue)

❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ Kiểm tra hàng đợi **đang rỗng**?

```
function isEmpty() {  
    // số lượng phần tử = 0  
    if (queue.length == 0) {  
        return true;  
    } else {  
        return false;  
    }  
}
```

Enqueue



Dequeue



Ngăn xếp (Stack), Hàng đợi (Queue)

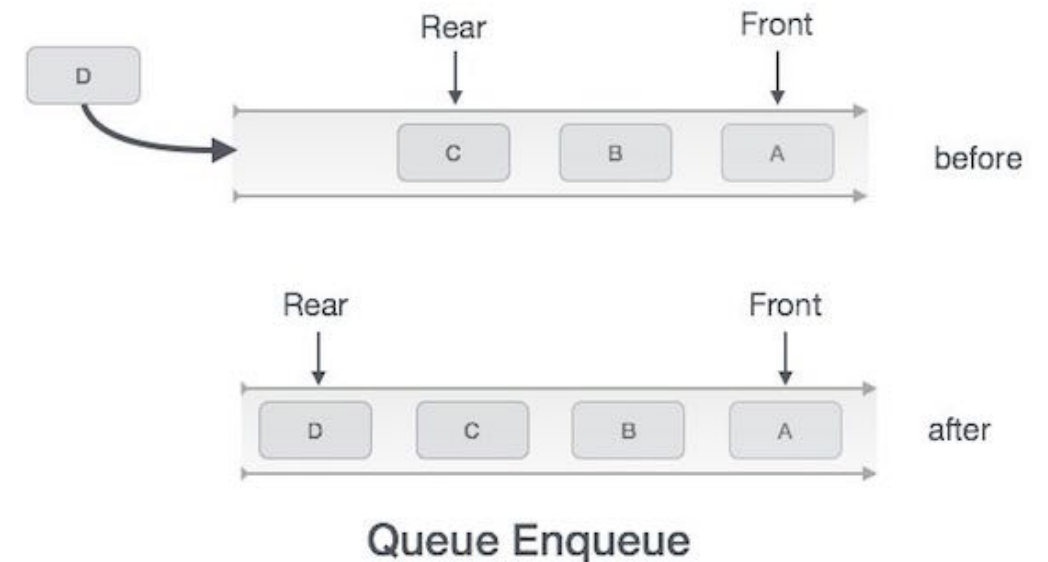
❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ Thêm phần tử vào hàng đợi (**enqueue**):

```
function enqueue(item) {  
    return queue.push(item);  
    // return queue.unshift(item);  
}
```

➤ Lưu ý khi dùng:

- push (enqueue) -> shift (dequeue)
- unshift (enqueue) -> pop (dequeue)





Ngăn xếp (Stack), Hàng đợi (Queue)

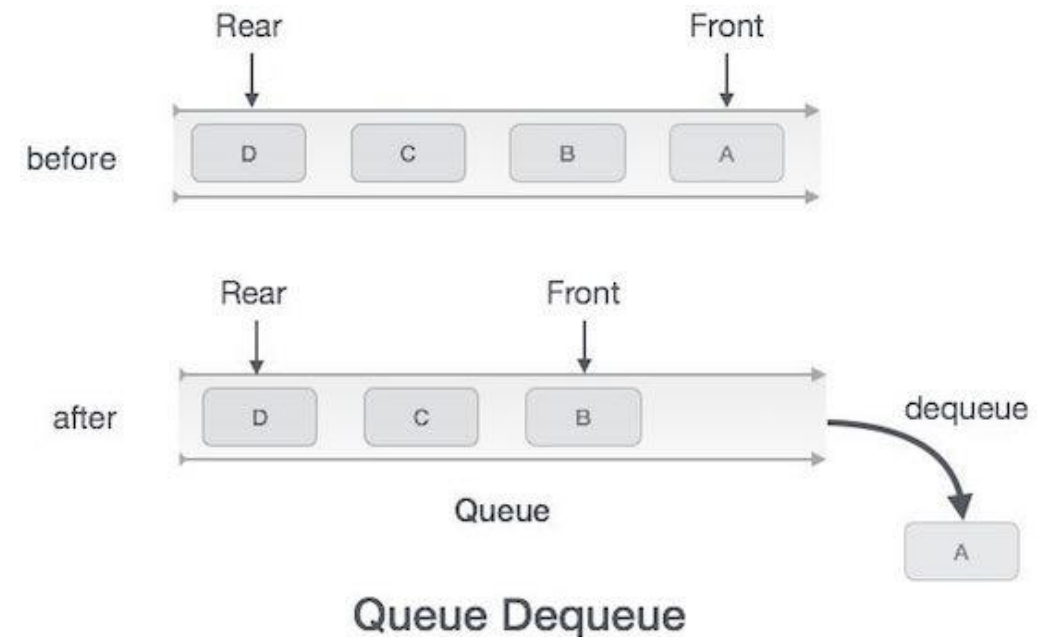
❑ Các kỹ thuật thao tác với hàng đợi (Queue):

➤ Xóa phần tử khỏi hàng đợi (**dequeue**):

```
function dequeue(item) {  
    return queue.shift(item);  
    // return queue.pop(item);  
}
```

➤ Lưu ý khi dùng:

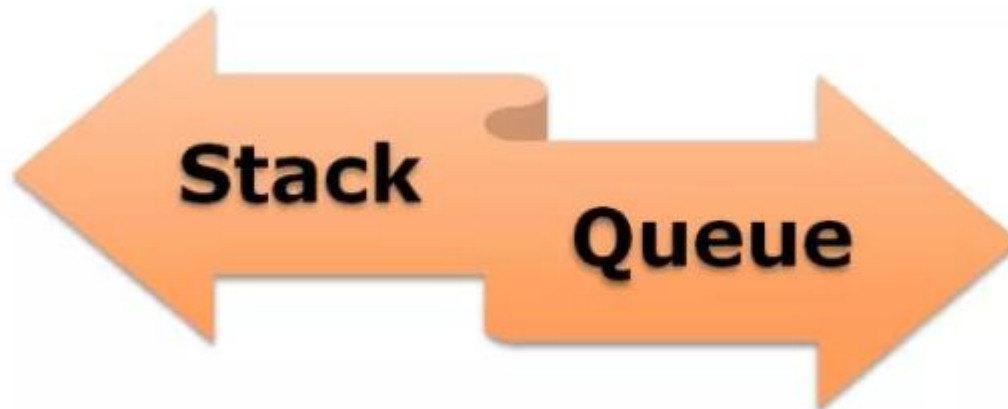
- push (enqueue) -> shift (dequeue)
- unshift (enqueue) -> pop (dequeue)





So sánh ngăn xếp (Stack) và hàng đợi (Queue)

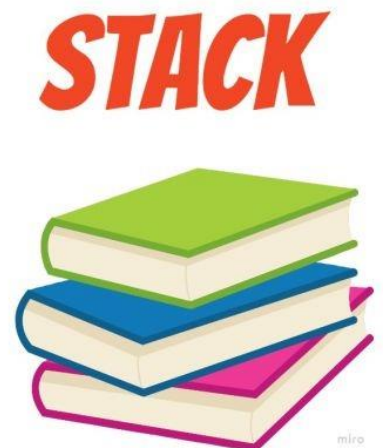
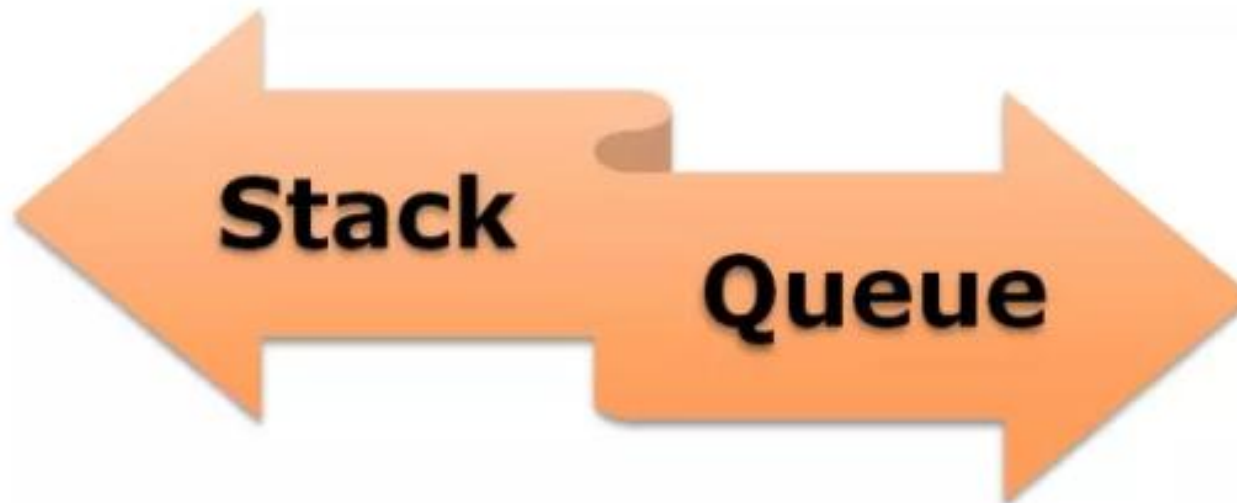
- ❑ **Stack** và **Queue** đều là các cấu trúc dữ liệu không nguyên thủy (non-primitive).
- ❑ Sự khác biệt lớn nhất giữa **Stack** và **Queue** là:
 - **Stack** sử dụng phương thức **LIFO** (Last In First Out) để truy cập và thêm các phần tử dữ liệu
 - **Queue** sử dụng phương thức **FIFO** (First In First Out) để truy cập và thêm các phần tử dữ liệu.





So sánh ngăn xếp (Stack) và hàng đợi (Queue)

- ❑ **Stack** chỉ có một đầu mở để **pushing** và **popping** các phần tử dữ liệu, còn **Queue** có cả hai đầu mở để **enqueueing** và **dequeueing** các phần tử dữ liệu.
- ❑ **Stack** và **Queue** là các cấu trúc dữ liệu được sử dụng để lưu trữ các yếu tố dữ liệu và nó dựa trên một số các ví dụ có thực trong cuộc sống hàng ngày của chúng ta.





So sánh ngăn xếp (Stack) và hàng đợi (Queue)

- ❑ **Stack** chỉ có một đầu mở để **pushing** và **popping** các phần tử dữ liệu, còn **Queue** có cả hai đầu mở để **enqueueing** và **dequeueing** các phần tử dữ liệu.
- ❑ Ví dụ, **Stack** là một chồng đĩa CD, nơi bạn có thể lấy ra và đưa vào đĩa CD thông qua đỉnh của ngăn xếp đĩa CD.



Stack of Books



Stack of Dishes



Stack of Discs



So sánh ngăn xếp (Stack) và hàng đợi (Queue)

- ❑ **Stack** chỉ có một đầu mở để **pushing** và **popping** các phần tử dữ liệu, còn **Queue** có cả hai đầu mở để **enqueueing** và **dequeueing** các phần tử dữ liệu.
- ❑ Tương tự, **Queue** là hàng đợi cho mua vé của Nhà hát nơi người đứng ở vị trí đầu tiên sẽ được phục vụ trước, người đến sau sẽ ở phía sau hàng đợi.





So sánh ngăn xếp (Stack) và hàng đợi (Queue)

❑ Sau khi học tìm hiểu, ta có **bảng so sánh một số sự khác nhau** như sau:

Cơ sở để so sánh	STACK	QUEUE
Nguyên tắc làm việc	LIFO (Last in First out)	FIFO (First in First out)
Structure	Dùng một đầu để chèn và xóa các phần tử dữ liệu	Có 2 đầu để xử lý dữ liệu, một đầu chèn một đầu xóa
Số con trỏ được sử dụng	Một	Hai (Trong trường hợp đơn giản)
Hoạt động được thực hiện	Push và Pop	Enqueue và dequeue
Kiểm tra empty condition	Top == -1	Front == -1
Examination full condition	Top == Max - 1	Rear == Max - 1
Biến thể	Không có biến thể	Nó có các biến thể như hàng đợi tròn, hàng đợi ưu tiên, hàng đợi kết thúc gấp đôi.
Thực hiện	Đơn giản	Tương đối phức tạp



Tổng kết:

- ☐ Tổng quan về ngăn xếp (Stack), hàng đợi (Queue)
- ☐ Ngăn xếp (Stack) là gì?
- ☐ Các kỹ thuật để thao tác với ngăn xếp (Stack)
- ☐ Hàng đợi (Queue) là gì?
- ☐ Các kỹ thuật để thao tác với hàng đợi (Queue)
- ☐ So sánh ngăn xếp (Stack) và hàng đợi (Queue)

Let's
Recap

