

This app works best with JavaScript enabled.



হাতেকলমে জাভাস্ক্রিপ্ট



জাভাস্ক্রিপ্ট অ্যাডভান্সঃ call(), bind() এবং apply() মেথড

গত পর্বে আমি **this** কীওয়ার্ড নিয়ে আলোচনা করেছি। সেখানে **this** কীওয়ার্ডের ভ্যালু বা অন্য কথায় **this** এর কন্টেক্সট(Context) ডিটারমাইন করার জন্যে চারটা রুলস এর কথা বলেছিলামঃ

১। গ্লোবাল রুলস

২। অবজেক্ট রুলস

৩। স্পষ্ট রুলস

তার মধ্যে প্রথম দুইটা আলোচনা করা হয়েছে গত পর্বেই। এই পর্বে তিন নাম্বার রুলস নিয়ে আলোচনা করবো। এখানেই আমি **call()**, **bind()** ও **apply()** এই তিনটা মেথডকে পরিচয় করিয়ে দিয়েছিলাম। জাভাস্ক্রিপ্ট ভালো করে আয়ত্তে আনতে হলে এই তিনটা মেথড সম্পর্কে আপনার পরিষ্কার ধারণা থাকতে হবে। আজকে তাই আমি এগুলো নিয়ে বিস্তারিত লিখবো।

এই তিনটা মেথডই প্রথম আর্গুমেন্ট হিসাবে **this** কীওয়ার্ডের কন্টেক্সট বা ভ্যালু কি হবে সেটা নেয়। তারমধ্যে **call()** আর **bind()** আনলিমিটেড আর্গুমেন্ট নিতে পারে আর অন্যদিকে **apply()** মাত্র দুইটা আর্গুমেন্ট(প্রথমটা সবসময়েই **this** এর ভ্যালু ডিটারমাইন করতে, আর দ্বিতীয়টা একটা অ্যারে) নেয়। এখন বুঝলাম প্রথম আর্গুমেন্ট **this** ডিটারমাইন করার জন্যে দেওয়া হয়, কিন্তু বাকী আর্গুমেন্টগুলো কিসের? হ্যাঁ বাকিগুলো আপনি যে ফাংশনের সাথে এই মেথডগুলো লাগাবেন সেই ফাংশনেরও আর্গুমেন্ট থাকতে পারে, সে আর্গুমেন্ট যতটাই হউক আপনি **call()** আর **bind()** এর ক্ষেত্রে একটার পর একটা দিতে পারবেন। অন্যদিকে **apply()** এর ক্ষেত্রে যে অ্যারেটা দিবেন সেটা হবে সেই ফাংশনের সবগুলো আর্গুমেন্টের অ্যারে। কনফিউজড হয়ে গেলে সমস্যা নাই আমি প্রত্যেকটা উদাহরনসহ নিচে দেখাবো। আশা করি পরিষ্কার ধারণা হয়ে যাবে।

আরেকটা ডিফারেন্স হলো **call()**, **apply()** আর **bind()** এর মধ্যে। যেখানে **call()**, **apply()** যে ফাংশনের সাথে ইউজ করবেন সেটা সাথে সাথে কল হয়ে যাবে। অন্যদিকে **bind()** সাথে সাথে ফাংশনটাকে কল করে না, বরং আপনি সেটা পরে যেকোনো সময় চাইলে নিজের মন মতো করে কল করতে পারবেন।

call() মেথডঃ

আগের পর্বে একটা উদাহরণ দিয়েছিলাম। সেইমটার উপরেই আজকে কাজ করবোঃ

```
var myCustomObj = {
  name: 'Zonayed Ahmed',
```

```

age: 21,
job: 'Student',
anotherObj: {
  name: 'Ahmed Zonayed',
  value: function() {
    console.log('My name is ' + this.name);
  }
}
}

```

এখানে যদি আমরা `value()` ফাংশনটাকে কল করি তাহলে *My name is Ahmed Zonayed* দেখাবে। মানে এখানে `this` এর ভ্যালু হচ্ছে `anotherObj`, আর তাই এভাবে আউটপুট পাবেন।

```
myCustomObj.anotherObj.value();
```

```

> var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
    name: 'Ahmed Zonayed',
    value: function() {
      console.log('My name is ' + this.name);
    }
  }
}
< undefined
> myCustomObj.anotherObj.value();
My name is Ahmed Zonayed

```

কিন্তু এখানে যদি আমরা চাই `value()` এর ভিতরের `this` এর কন্টেক্সট বা ভ্যালু হিসাবে `myCustomObj` সেট করতে তাহলে সেটা আমরা স্পষ্ট করে `call()` দিয়ে সেট করে দিতে পারি

এভাবেঃ

```
myCustomObj.anotherObj.value.call(myCustomObj);
```

```
> var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
    name: 'Ahmed Zonayed',
    value: function() {
      console.log('My name is ' + this.name);
    }
  }
}
< undefined
> myCustomObj.anotherObj.value();
My name is Ahmed Zonayed
< undefined
> myCustomObj.anotherObj.value.call(myCustomObj);
My name is Zonayed Ahmed
< undefined
```

দেখুন এবার প্রিন্ট হয়েছে *My name is Zonayed Ahmed*, মানে `this` এর ভ্যালু এখানে আমরা আমাদের মতো করে চেঞ্জ করতে পেরেছি। আরেকটা জিনিস আমরা চাইলে `call()` টা আমাদের অবজেক্ট এর সাথেও দিতে পারতাম। কিন্তু ঐ যে বললাম `call()` যেখানে ইউজ করা হয় সেটা সাথে সাথে কল হয়ে যায়, তাই আপনি যেরকম আশা করবেন সেরকম রেজাল্ট নাও আসতে পারেঃ

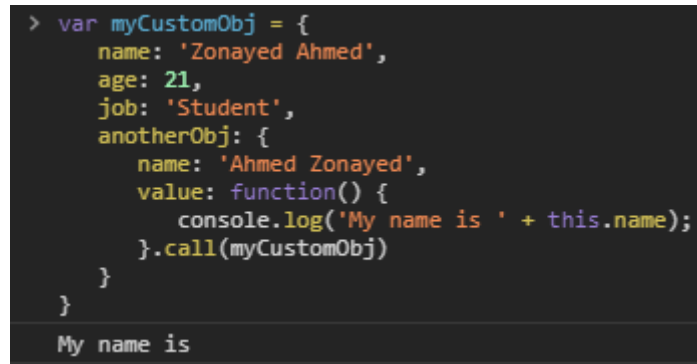
```
var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
```

```

    name: 'Ahmed Zonayed',
    value: function() {
        console.log('My name is ' + this.name);
    }.call(myCustomObj)
}
}

```

এটা এভাবে রান করার পর পরই দেখবেন আপনার ফাংশন কল করা ছাড়াই প্রিন্ট হয়ে গেছেঃ



```

> var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
    name: 'Ahmed Zonayed',
    value: function() {
      console.log('My name is ' + this.name);
    }.call(myCustomObj)
  }
}

```

My name is

সেইমভাবে আপনার দুইটা সম্পূর্ণ পৃথক পৃথক অবজেক্ট এর ক্ষেত্রেও আপনি এই মেথডগুলো কাজে লাগাতে পারবেন। যেমন ধরি আমাদের দুইটা অবজেক্ট আছে এরকমঃ

```

var karim = {
  name: 'Karim Rahman',
  dob: 1996,
  age: function(currentYear) {
    console.log(this.name + ' is ' + (currentYear -
this.dob) + ' years old!');
  }
}

```

```
}
```

আরেকটাঃ

```
var rahim = {  
  name: 'Rahim Abdu',  
  dob: 1986  
}
```

দেখুন প্রথম অবজেক্ট **karim** থেকে আমরা খুব সহজেই **age** ফাংশনটা কল করে **karim** এর বয়স জানতে পারবোঃ

```
karim.age(2018)
```

```
> karim.age(2018)  
Karim Rahman is 22 years old!
```

এখন লক্ষ্য করুন **rahim** এ আমাদের কিন্তু **age** নামক ফাংশনটা নাই, কিন্তু তারপরেও আমরা চাইলে এই **age** ইউজ করে এর সাথে **call()**, **bind()** বা **apply()** দিয়ে এর ভিতরের **this** এর ভ্যালু চেঞ্জ করে সেটা **rahim** এর জন্যেও ইউজ করতে পারিঃ

```
karim.age.call(rahim, 2018);
```

```
> karim.age.call(rahim, 2018)
Rahim Abdu is 32 years old!
```

আরো দেখবেন এখানে দ্বিতীয় আর্গুমেন্ট হিসাবে `age` এর আর্গুমেন্ট হিসাবে ভ্যালু পাস করেছি। সেইম জিনসটা `bind()` আর `apply()` দিয়েও করা যাবে।

apply() মেথডঃ

এটাও সেইম টু সেইম `call()` এর মতোই, জাস্ট এটা দুইটা আর্গুমেন্ট নিবে আর দ্বিতীয় আর্গুমেন্ট টা আপনার ফাংশনের জন্যে যে কয়টা আর্গুমেন্ট থাকবে সেগুলার অ্যারে নিবেঃ

```
var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
    name: 'Ahmed Zonayed',
    value: function() {
      console.log('My name is ' + this.name);
    }
  }
}
```

আগের এই সেইম উদাহরণে `apply()` ইউজ করলেও সেইম রেজাল্ট পাবেনঃ


```
myCustomObj.anotherObj.value.apply(myCustomObj);
```

```
> var myCustomObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  job: 'Student',
  anotherObj: {
    name: 'Ahmed Zonayed',
    value: function() {
      console.log('My name is ' + this.name);
    }
  }
}
< undefined
> myCustomObj.anotherObj.value.apply(myCustomObj);
My name is Zonayed Ahmed
```

`call()` থেকে `apply()` এর পার্থক্যটা দ্বিতীয় আর্গুমেন্ট নেওয়ার ক্ষেত্রে যেখানে `apply()`

আর্গুমেন্ট এর অ্যারে নেয়। আগের `rahim` আর `karim` একটু মডিফাই করেঃ

```
var karim = {
  name: 'Karim Rahman',
  dob: 1996,
  age: function(currentYear, msg) {
    console.log(msg + ' ' + this.name + ' is ' +
      (currentYear - this.dob) + ' years old!');
  }
}
```

এবংঃ

```
var rahim = {  
  name: 'Rahim Abdu',  
  dob: 1986  
}
```

`karim` এর `age` কল করলেঃ

```
karim.age(2018, 'Hello World!');
```

`age` ফাংশনটায় দুইটা আর্গুমেন্ট লাগিয়েছি বুঝার সুবিধার্থে। এটা রান করলে কন্সোলে পাবেনঃ

```
> karim.age(2018, 'Hello World!');  
Hello World! Karim Rahman is 22 years old!
```

এখন এই সেইম `age` ফাংশন `rahim` এ ঠিক আগের মতো করে ইউজ করতে চাচ্ছি। কিন্তু এবার `apply()` দিয়েঃ

```
karim.age.apply(rahim, [2018, 'Hello World!']);
```

লক্ষ্য করুন এখানে দ্বিতীয় আর্গুমেন্ট টা একটা অ্যারে যেটা আপনার `age` ফাংশনের সবগুলো আর্গুমেন্ট নিয়েছেঃ

```
> karim.age.apply(rahim, [2018, 'Hello World!']);  
Hello World! Rahim Abdu is 32 years old!
```

আশা করি এবার `call()` আর `apply()` মধ্যে তফাৎটা ধরতে পেরেছেন। এবার চলি আসুন `bind()` নিয়ে কথা বলি।

bind() মেথডঃ

`bind()` ঠিক `call()` এর মতো হলেও যেখানে `call()` আর `apply()` সাথে সাথে যে ফাংশনের সাথে ইউজ করা হয়েছে সেটাকে কল করে ফেলে, `bind()` সেখানে সে ফাংশনকে কল করে না, বরং এটা সেই ফাংশনের আরেকটা ডেফিনেশন রিটার্ন করে যেটা পরবর্তিতে আপনি যেকোনো জায়গায় কল করতে পারবেন বা ইউজ করতে পারবেন। আগের সেইম উদাহরণেই যদি সেইমভাবে `bind()` ইউজ করি তাহলে পার্থক্যটা ধরতে পারবেনঃ

```
var myCustomObj = {  
  name: 'Zonayed Ahmed',  
  age: 21,  
  job: 'Student',  
  anotherObj: {  
    name: 'Ahmed Zonayed',  
    value: function() {  
      console.log('My name is ' + this.name);  
    }  
  }  
}
```

এবার সেইমভাবে `value` এর `this` এর ভ্যালু `myCustomObj` এ সেট করতে আগের মতোই `bind()` মেথড ইউজ করলেঃ

```
myCustomObj.anotherObj.value.bind(myCustomObj);
```

এটা দেখবেন সরাসরি আপনার কাজিত লেখা প্রিন্ট করছে না, বরং এটা যা রিটার্ন করছে সেটা আরেকটা ফাংশন ডেফিনেশনঃ

```
> myCustomObj.anotherObj.value.bind(myCustomObj);  
↪ f () {  
    console.log('My name is ' + this.name);  
}
```

এখন এই ফাংশনটাকে আপনি আরেকটা ভ্যারিয়েবলে স্টোর করে পরে যেকোনো সময়, যেকোনো জায়গায় ইউজ করতে পারবেনঃ

```
var anotherFunc =  
myCustomObj.anotherObj.value.bind(myCustomObj);
```

এবার এই ফাংশন যেখানে কল করবেন সেখানেই আপনার কাজিত ফলাফল আসবেঃ

```
anotherFunc();
```

```

> var anotherFunc = myCustomObj.anotherObj.value.bind(myCustomObj);
< undefined
> anotherFunc();
  My name is Zonayed Ahmed
< undefined

```

আর এ জন্যেই গত পর্বে উদাহরনটায় আমরা `bind()` ইউজ করেছিলাম `call()` বা `apply()` ইউজ না করে। আর এজন্যেই এই তিনটা মেথডের মধ্যে এই `bind()` সবচেয়ে ইউজফুল।

`bind()` এ আপনি আর্গুমেন্টগুলোও পৃথক পৃথক ভাবে কল করতে পারবেন। ধরেন প্রথমে আপনি আপনার কিছু আর্গুমেন্ট দিলেন, পরে আবার ফাংশন কল করার সময় বাকি আর্গুমেন্টগুলো দিতে পারবেন। যেমন `apply()` তে ইউজ করা `kahim` আর `rahim` এর উদাহরণের ক্ষেত্রেঃ

```

var karim = {
  name: 'Karim Rahman',
  dob: 1996,
  age: function(currentYear, msg) {
    console.log(msg + ' ' + this.name + ' is ' +
      (currentYear - this.dob) + ' years old!');
  }
}

```

আরঃ

```

var rahim = {
  name: 'Rahim Abdu',
  dob: 1986
}

```

```
}
```

এই উদাহরণের ক্ষেত্রে যদি আমরা চাই `msg` এর ভ্যালু পরে সেট করতে তাহলে এটা এভাবেও করা যাবেঃ

```
var rahimAge = karim.age.bind(rahim, 2018);
```

লক্ষ্য করুন আমরা এখানে মাত্র একটা আর্গুমেন্ট দিয়েছি, আরেকটা দেই নাই। যেটা পরবর্তিতে আমরা এই ফাংশনটাকে কল করার সময় দিতে পারবোঃ

```
rahimAge('Hello World!');
```

```
> var rahimAge = karim.age.bind(rahim, 2018);  
← undefined  
> rahimAge('Hello World!');  
Hello World! Rahim Abdu is 32 years old!  
← undefined
```

বা আমরা চাইলে এটাকে আরো এক ধাপ আগায় নিতে পারি এভাবে। প্রথমে জাস্ট `bind()` দিয়ে আরেকটা ফাংশন বানাইলাম যেটার কাজ হবে `rahim` এর `age` ক্যালকুলেট করাঃ

```
var rahimAgeCalculate = karim.age.bind(rahim);
```

এখন এই `rahimAgeCalculate` ফাংশনে বাকি আর্গুমেন্ট গুলো পাস করতে পারবোঃ

```
rahimAgeCalculate(2018, 'Hello Dolly!');
```

```
> var rahimAgeCalculate = karim.age.bind(rahim);  
← undefined  
> rahimAgeCalculate(2018, 'Hello Dolly!');  
Hello Dolly! Rahim Abdu is 32 years old!
```

যতবার যতভাবে ইচ্ছাঃ

```
rahimAgeCalculate(2028, 'Hello Ahmed!');  
rahimAgeCalculate(2028, 'Hello Zonayed!');  
rahimAgeCalculate(2050, 'Hi!');
```

```
> rahimAgeCalculate(2028, 'Hello Ahmed!');
rahimAgeCalculate(2028, 'Hello Zonayed!');
rahimAgeCalculate(2050, 'Hi!');
Hello Ahmed! Rahim Abdu is 42 years old!
Hello Zonayed! Rahim Abdu is 42 years old!
Hi! Rahim Abdu is 64 years old!
```

এখানেই এই `bind()` একটু স্পেশাল আর ইউজ কেসও বেশী এটার।

স্পেশাল উদাহরণঃ

নিচের এই উদাহরণে `this` এর ভ্যালু কি হতে পারে অনুমান করার চেষ্টা করুনঃ

```
var myObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log('My name is ' + this.name);
    }, 1000)
  }
}
```

এখান আমরা `timer` ফাংশনটাকে কল করলেঃ

```
myObj.timer();
```

এটা দ্বিতীয় রুলস(অবজেক্ট রুলস) এ পড়ে যেহেতু এই ফাংশনটা একটা কাস্টমভাবে ডিফাইন করা

অবজেক্ট এর ভিতরে আছে। তাহলে সে হিসাবে এটার ভিতরের `this` আমার সেই অবজেক্ট `myObj` কেই ইন্ডিকেট করার কথা। কিন্তু এটা এক্সিকিউট করলে ১০০০ মিলিসেকেন্ড পরে এমন আউটপুট আসবেঃ

```
> var myObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log('My name is ' + this.name);
    }, 1000)
  }
}
< undefined
> myObj.timer()
< undefined
My name is
```

কোনো কারনে `name` এর ভ্যালু আসছে না, তারমানে এখানে তাহলে কোথাও একটা প্রবলেম হচ্ছে।
আচ্ছা তাহলে ঠিক এখানে `this` এর ভ্যালু কি দেখে নেওয়া যাকঃ

```
var myAnotherObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log(this);
    }, 1000)
  }
}
```

এখানে `timer` ফাংশনটাকে কল করলেঃ

```
myAnotherObj.timer();
```

১০০০ মিলিসেকেন্ড পরে প্রিন্ট করবেঃ

```
> var myAnotherObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log(this);
    }, 1000)
  }
}
< undefined
> myAnotherObj.timer();
< undefined
  ▶ Window {postMessage: f, blur: f, focus: f, close: f, frames: Window, ...}
```

এখানে কোনো কারনে **this** গ্লোবাল অবজেক্ট(ব্রাউজারের ক্ষেত্রে **window** অবজেক্ট) কে ইন্ডিকেট করছে। কেনো? হ্যাঁ আপনি যদি আপনার গ্লোবাল অবজেক্ট ওপেন করে দেখেন দেখবেন এই **setTimeout** আসলে সেই গ্লোবাল অবজেক্ট এর একটা মেথডঃ

```
console.dir(window);
```

এটা ওপেন করলে অনেকগুলো মেথড পাবেন। তারমধ্যে **setTimeout** ও পাবেনঃ

```

scrollX: 0
scrollY: 14213
▶ scrollbars: BarProp {visible: true}
▶ self: Window {postMessage: f, blur: f, focus:
▶ sessionStorage: Storage {lightstep/clock_stat
▶ setInterval: f (b,c)
▶ setTimeout: f (b,c)
▶ speechSynthesis: SpeechSynthesis {pending: fa
status: ""
▶ statusbar: BarProp {visible: true}

```

তো আমাদের কথামতো `setTimeout` এর ভিতরে `this` তাই গ্লোবাল অবজেক্ট কেই ইন্ডিকেট করছে। যদিও `setTimeout` আরেকটা কাস্টমভাবে ডিফাইনকৃত অবজেক্ট এর ভিতরে কিন্তু এটার ভিতরে থাকা `this` এর সবচেয়ে কাছের অবজেক্ট হচ্ছে গ্লোবাল অবজেক্ট যেহেতু `setTimeout` আসলে গ্লোবাল অবজেক্ট এরই একটা মেথড। তাই এভাবে একটার ভিতরে আরেকটা মেথড থাকলেও আপনার `this` এর ভ্যালু চেঞ্জ হয়ে যেতে পারে।

এখন আমরা যেহেতু `call()`, `apply()`, `bind()` মেথডগুলো জানি তাই চলেন আমরা এটা ফিক্স করে ঠিক যেটা প্রিন্ট করাতে চাচ্ছিলাম সেটাই প্রিন্ট করাইঃ

```

var myObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log('My name is ' + this.name);
    }.bind(myObj), 1000)
  }
}

```

এখন `timer` ফাংশনটাকে কল করলেঃ

```
myObj.timer();
```

```
> var myObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log('My name is ' + this.name);
    }.bind(myObj), 1000)
  }
}
< undefined
> myObj.timer();
< undefined
My name is Zonayed Ahmed
```

এখন এখানে `call()` বা `apply()` কেনো ইউজ করলাম না? হ্যাঁ এই দুইটা মেথড ইউজ করা যাবে কিন্তু আমরা যেভাবে আশা করছিলাম সেরকম রেজাল্ট আসবে নাঃ

```
var myObj = {
  name: 'Zonayed Ahmed',
  age: 21,
  timer: function() {
    setTimeout(function() {
      console.log('My name is ' + this.name);
    }.call(myObj), 1000)
  }
}
```

`setTimeout` দিয়ে আমরা চাচ্ছি ১০০০ মিলিসেকেন্ড পরে উক্ত লেখাটা প্রিন্ট হউক, কিন্তু `call()` ইউজ করায় সেটা সাথে সাথেই কল হয়ে যাবেঃ

```
myObj.timer();
```

দেখবেন আপনার কন্সোলে সাথে সাথে প্রিন্ট হয়ে গেছে, কন্সোল ১০০০ মিলিসেকেন্ড ওয়েট করে নাইঃ

```
> var myObj = {  
  name: 'Zonayed Ahmed',  
  age: 21,  
  timer: function() {  
    setTimeout(function() {  
      console.log('My name is ' + this.name);  
    }.call(myObj), 1000)  
  }  
}  
← undefined  
> myObj.timer()  
My name is Zonayed Ahmed
```

সেইম `apply()` ক্ষেত্রেও হবে। তাই ক্ষেত্রবিশেষে এদের বিহেভিয়ার অনুযায়ী আপনাকে যেসময় যেটা দরকার সেটা ইউজ করতে হবে।

তো আজকে এই পর্যন্তই, ভালো থাকবেন আর পাশের মানুষটিকে ভালো রাখবেন।

আপনার মন্তব্যঃ

যদি এই পোস্টে কোন ভুল(যেকোনো ধরনের) পেয়ে থাকেন অথবা কোনো ব্যাপারে সন্দেহ থাকে তাহলে এখানে জানাতে পারবেন।



জাভাস্ক্রিপ্ট ব্যাসিক



জাভাস্ক্রিপ্ট অ্যাডভান্স



জাভাস্ক্রিপ্ট ইএস৬



জাভাস্ক্রিপ্ট ডম ম্যানিপুলেশন



নিত্যদিনের জাভাস্ক্রিপ্ট

জাভাস্ক্রিপ্ট অ্যালগরিদম ও ডাটা স্ট্রাকচার



জাভাস্ক্রিপ্ট সফট স্কিল

সম্পর্কেঃ

প্রোজেক্টটি সম্পূর্ণ সোর্স কোডসহ গিটহাবে রয়েছে। আপনার ভালো লেগে থাকলে স্টার দিয়ে আসবেন। আপনার পরামর্শ, মন্তব্য এবং ভুলত্রুটি গিটহাবে ইস্যু করে দেওয়ার জন্যে অনুরোধ থাকলো

আপনার জন্যঃ

রিঅ্যাক্ট জেএস শিখুন

আমি মিডিয়ামে

আমার ব্লগ

রিসোর্সঃ

এমডিএন ডকুমেন্টেশন

ইউডেমি কোর্স

Eloquent JavaScript

You Don't Know JS



♥ এর সাথে ডেভেলপ করেছে **জুনায়েদ আহমেদ**