

# Java Script Fundamentals

## Print:

document.write():

```
<script>
  document.write("Hello World" + "<br>");
</script>
```

Output:

Hello World

console.log():

Result will be shown on browser's console. Browser's inspect -> console:

```
<script>
  console.log("Hello World");
</script>
```

Output:

Hello World

---

## Variable declaration:

```
<script>
  let a = 10;
  let b = 20;
  var c = 30;
  const d = 40;
</script>
```

---

## Operators:

## Assignment:

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y
**=	x **= y	x = x ** y

## Arithmetic:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

```
<script>
  let a = 10;
  let b = 20;

  document.write("Hello World" + "<br>");
  document.write(a + "<br>");
  document.write(b + "<br>");
  document.write(a + b + "<br>");
  document.write(a - b + "<br>");
  document.write(a * b + "<br>");
  document.write(a / b + "<br>");
  document.write((a % b) + "<br>");
</script>
```

## Comparison:

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

## Logical:

Operator	Description
&&	logical and
	logical or
!	logical not

## Type Operators:

Operator	Description
typeof	Returns the type of a variable
instanceof	Returns true if an object is an instance of an object type

## Bitwise Operators:

Operator	Description	Example	Same as	Result	Decimal
&	AND	5 & 1	0101 & 0001	0001	1
	OR	5   1	0101   0001	0101	5
~	NOT	~ 5	~0101	1010	10
^	XOR	5 ^ 1	0101 ^ 0001	0100	4
<<	left shift	5 << 1	0101 << 1	1010	10
>>	right shift	5 >> 1	0101 >> 1	0010	2
>>>	unsigned right shift	5 >>> 1	0101 >>> 1	0010	2

---

## Comments:

Single Line:

```
// document.write("Hello World" + "<br>");
```

Multiple Line:

```
/*  
document.write(a + "<br>");  
document.write(b + "<br>");  
document.write(a + b + "<br>");  
document.write(a - b + "<br>");  
document.write(a * b + "<br>");  
document.write(a / b + "<br>");  
document.write((a % b) + "<br>");  
*/
```

---

## Data Types:

1. String
2. Number
3. Boolean
4. Null
5. Array (Non Primitive)
6. Float
7. Object (Non Primitive)
8. Undefined
9. RegEx (Regular Expression) (Non Primitive)

```
<script>
```

```
let a = "John Doe"; // String  
let b = 20; // Number  
let c = true; // Boolean  
let d = 10.1; // float  
let e = null; // null  
let f = [1, 2, 3]; // Array  
let g = ["John", "Doe"]; // Array  
let h = {  
  name: "John Doe",
```

```
    city: "New York",
    age: 30,
  }; // Object
  let i; // undefined

  // to show on browser
  document.write(a + "<br>");
  document.write(b + "<br>");
  document.write(c + "<br>");
  document.write(d + "<br>");
  document.write(e + "<br>");
  document.write(f + "<br>");
  document.write(g + "<br>");
  document.write(h + "<br>");
  document.write(i + "<br>");

  // to show on console
  console.log(a);
  console.log(b);
  console.log(c);
  console.log(d);
  console.log(e);
  console.log(f);
  console.log(g);
  console.log(h);
  console.log(h["name"]); // picking specific value
  console.log(h.city); // picking specific value
  console.log(i);

</script>
```

### Output:

on browser:

```
John Doe
20
true
10.1
null
1,2,3
John,Doe
[object Object]
undefined
```

on console:

John Doe

```
03. data-types.html:37 20
03. data-types.html:38 true
03. data-types.html:39 10.1
03. data-types.html:40 null
03. data-types.html:41 (3) [1, 2, 3]
03. data-types.html:42 (2) ['John', 'Doe']
03. data-types.html:43 {name: 'John Doe', city: 'New York', age: 30}
03. data-types.html:44 John Doe
03. data-types.html:45 New York
03. data-types.html:46 undefined
```

---

## If-Else:

```
<script>

    const marks = 80;

    if (marks >= 80 && marks <= 100) {
        console.log("A");
    } else if (marks >= 60 && marks < 80) {
        console.log("B");
    } else if (marks >= 40 && marks < 60) {
        console.log("C");
    } else {
        console.log("D");
    }

</script>
```

---

## Loop:

1. For Loop
2. For In Loop
3. While Loop
4. Do While Loop

### for loop:

```
<script>

    let i;
```

```
    for (i = 0; i < 10; i++) {  
        console.log(i);  
    }  
  
    for (i = 0; i < 10; i++) {  
        document.write("<button>Click</button><br/>");  
    }  
  
</script>
```

## while loop:

```
<script>  
  
    let i = 0;  
  
    while (i < 10) {  
        console.log(i);  
        i++;  
    }  
  
</script>
```

## do while loop:

```
<script>  
  
    let i = 0;  
  
    do {  
        console.log(i);  
        i++;  
    } while (i < 10);  
  
</script>
```

```
<script>  
  
    let i = 0;  
  
    do {  
        document.write("<button>Click</button><br/>");  
        i++;  
    } while (i < 10);
```

</script>

## Function:

# JavaScript ফাংশন (Function) কি?

## সাধারণ ভাষায় বুঝি 🤔

ফাংশন হল একটি "মজিক বাক্স" যা কিছু নির্দিষ্ট কাজ করে। আপনি যখন ফাংশনকে "কল" করেন, সেটি তার ভিতরের কোড গুলো সম্পাদন করে।

## কোডের বিস্তারিত ব্যাখ্যা 🧐

### ফাংশন ডিক্লেয়ারেশন

```
function add() {  
  let a = 10;  
  let b = 20;  
  let c = a + b;  
  console.log(c);  
}
```

### ফাংশনের প্রধান অংশগুলি:

1. `function` কীওয়ার্ড
2. ফাংশনের নাম ( `add` )
3. `()` বন্ধনী
4. `{ }` ব্লক যার মধ্যে কোড থাকে

### ফাংশন কল করা

```
add(); // এটি ফাংশনকে কল করে
```

## ফাংশনের বিভিন্ন ধাপ

### 1. ভ্যারিয়েবল ডিক্লেয়ারেশন

```
let a = 10; // প্রথম সংখ্যা  
let b = 20; // দ্বিতীয় সংখ্যা
```



## 2. গণনা

```
let c = a + b; // a এবং b যোগ করে c-তে সংরক্ষণ
```

## 3. আউটপুট

```
console.log(c); // c-এর মান প্রিন্ট করবে
```

## সহজ উদাহরণ:

মনে করুন, আপনি একটি ফাংশন বানাচ্ছেন যা আপনার নাম প্রিন্ট করে:

```
function sayHello() {  
    console.log("হ্যালো, আমি রাহুল!");  
}  
sayHello(); // এটি কল করলে "হ্যালো, আমি রাহুল!" প্রিন্ট হবে
```

## গুরুত্বপূর্ণ নিয়ম

1. ফাংশন নাম সাধারণত camelCase হয়
2. ফাংশন নাম দিতে হবে অর্থবহ
3. ফাংশনের কাজ সুস্পষ্ট হওয়া উচিত

## সাবধানতা

- ফাংশন কল না করলে তার ভিতরের কোড চলবে না
- নাম ঠিকভাবে লিখতে হবে

## মজার তথ্য

JavaScript-এ ফাংশন হল "first-class citizen" - মানে আপনি ফাংশনকে ভ্যারিয়েবল হিসাবেও ব্যবহার করতে পারেন!

## সাধারণ ফাংশন সিনট্যাক্স

```
function functionName() {  
    // কোড ব্লক  
    // এখানে আপনার কাজগুলো লিখবেন  
}
```

## কোড ব্যাখ্যা

## 1. function কীওয়ার্ড

- ফাংশন তৈরির জন্য ব্যবহৃত হয়
- JavaScript বুঝতে পারে যে এখন একটি ফাংশন শুরু হচ্ছে

## 2. functionName

- আপনার ফাংশনের নাম
- সাধারণত camelCase ব্যবহার করুন
- অর্থবহ নাম দিন যেন বুঝা যায় ফাংশন কী করে

## 3. ()

- খালি বন্ধনী
- পরবর্তীতে এখানে আর্গুমেন্ট পাস করা যাবে

## 4. { }

- কোড ব্লক
- ফাংশন যে কাজগুলো করবে সেগুলো এখানে লিখবেন

## ফাংশন কল করা 📞

```
functionName(); // ফাংশন কল করা হয় এইভাবে
```

## সম্পূর্ণ উদাহরণ

```
function add() {  
  let a = 10;    // প্রথম সংখ্যা  
  let b = 20;    // দ্বিতীয় সংখ্যা  
  let c = a + b;  // যোগফল  
  console.log(c); // ফলাফল দেখান  
}  
  
add(); // ফাংশন কল করা হল
```

## গুরুত্বপূর্ণ বিষয়সমূহ ✅

- ফাংশন নাম ইংরেজিতে লিখতে হয়
- নাম কোনো সংখ্যা দিয়ে শুরু হতে পারবে না

## Arrow Function: মজার মজার ভাবে বুঝি 🚀

# কল্পনা করুন...

মনে করুন আপনি একটি "মহাকাশের যান" পেয়েছেন যা খুব দ্রুত কাজ করতে পারে! এই মহাকাশ যানটিই হচ্ছে Arrow Function! 🚀

## সাধারণ ফাংশন vs Arrow Function 🔍

### পুরানো পদ্ধতি (Normal Function)

```
function add(a, b) {  
  return a + b;  
}
```

### নতুন পদ্ধতি (Arrow Function)

```
const add = (a, b) => a + b;
```

## কেন এটা মজার? 😊

1. দ্রুত কাজ
  - কম লাইন কোড
  - তাৎক্ষণিক ফলাফল
  - খুব সহজ লেখার নিয়ম
2. সোজা বোঝা
  - `=>` চিহ্ন মানে "যাও" বা "কর"
  - খুব সহজ সিনট্যাক্স

## কখন ব্যবহার করবেন? 💡

- ✅ যখন কাজটা খুব সহজ
- ✅ দ্রুত ফলাফল চাই
- ✅ কম কোড লিখতে চাই

## সাবধানে থাকুন ⚠️

- সব জায়গায় ব্যবহার করা যায় না
- জটিল কাজের জন্য সাধারণ ফাংশন ভাল

## মজার তথ্য 🍕

Arrow Function হল JavaScript-এর "স্মার্ট শর্টকাট"। আপনি কম কোডে বেশি কাজ করতে পারবেন!

# Functions with Parameters: সহজ ভাষায় ব্যাখ্যা



## ফাংশন কী? 🤔

ফাংশন হল কাজ করার জন্য তৈরি একটি ছোট মেশিন। এটি প্যারামিটার নামক "ইনপুট" নেয়, কাজ করে এবং কখনো "আউটপুট" দেয়।

এখন আসুন এই উদাহরণগুলো সহজভাবে বুঝি। 😊

## ১. দুইটি সংখ্যার যোগফল বের করা

কোড:

```
function add(a, b) {  
  let c = a + b; // a এবং b যোগ করে c তে জমা করা হচ্ছে।  
  console.log(c); // c-এর মান দেখানো হচ্ছে।  
}  
  
add(10, 20); // ফাংশনটি ১০ এবং ২০ যোগ করবে।
```

কীভাবে কাজ করে?

- `add(10, 20)` ডাক: এখানে `a = 10` এবং `b = 20`।
- গণনা: `c = 10 + 20 = 30`।
- ফলাফল দেখানো: `console.log` ব্যবহার করে ৩০ দেখাবে।

ফলাফল:

30

## ২. একটি সংখ্যার বর্গ (Square) বের করা

```
function square(x) {  
  const result = x * x; // x এর বর্গফল বের করে result-এ রাখছি।  
  return result; // ফলাফল ফেরত দিচ্ছি।  
}  
  
console.log(square(10)); // ১০ এর বর্গফল দেখাবে।
```

কীভাবে কাজ করে?

- `square(10)` ডাক: এখানে `x = 10`।
- গণনা: `result = 10 * 10 = 100`।
- ফলাফল ফেরত দেওয়া: `return` দিয়ে 100 পাঠানো হয়েছে।

ফলাফল:

```
100
```

## বর্গ বের করার আরও সহজ পদ্ধতি (Arrow Function):

```
const square = (x) => x * x; // একই কাজ কম কোডে।  
console.log(square(10)); // ১০ এর বর্গফল দেখাবে।
```

## ৩. একটি বার্তা এবং নাম দেখানো

```
function sayHello(greetings, name) {  
  console.log(greetings + ' ' + name); // সাধারণ যোগ।  
  console.log(`${greetings} ${name}`); // স্ট্রিং লিটারাল দিয়ে।  
}  
sayHello("Hello", "John"); // বার্তা এবং নাম দেখাবে।
```

কীভাবে কাজ করে?

- `sayHello("Hello", "John")` ডাক: এখানে `greetings = "Hello"` এবং `name = "John"`।
- প্রথম লাইন: `Hello` এবং `John` একত্রে দেখাবে: `"Hello John"`।
- দ্বিতীয় লাইন (স্ট্রিং লিটারাল): একই ফলাফল আরও সহজ পদ্ধতিতে দেখাবে।

ফলাফল:

```
Hello John  
Hello John
```


## উপসংহার:

ফাংশন আপনার কোডকে পুনরায় ব্যবহারযোগ্য করে তোলে। আপনি:

1. গণনা করতে পারবেন।
2. বার্তা তৈরি করতে পারবেন।
3. এবং আরও অনেক কাজ সহজে করতে পারবেন! 🚀

---

## ডিফল্ট প্যারামিটার (Default Parameters)

কল্পনা করুন... 

মনে করুন, আপনি একটি যাদুকরী বাক্সে কিছু জিনিস রাখতে চান। এই বাক্সে যদি আপনি কিছু না রাখেন, তবে এটি স্বয়ংক্রিয়ভাবে একটি নির্দিষ্ট জিনিস রাখবে। এই বাক্সটি হচ্ছে "ডিফল্ট প্যারামিটার"! 📦 ✨

## কোডের উদাহরণ 🔍

### ১. স্বাগতম বলা (sayHello)

```
function sayHello(greetings, name = "John") {  
    console.log(`${greetings} ${name}`); // স্ট্রিং লিটারেল 14  
}  
  
sayHello("Hello"); // দ্বিতীয় প্যারামিটারের জন্য কিছু না দেওয়া হলে, এটি স্বয়ংক্রিয়ভাবে  
"John" নেবে
```

#### ব্যাখ্যা:

- এখানে sayHello একটি ফাংশন যা দুটি প্যারামিটার নেয়: greetings এবং name।
- name প্যারামিটারের জন্য একটি ডিফল্ট মান দেওয়া হয়েছে, যা হলো "John"।
- যখন আপনি sayHello("Hello") কল করেন, তখন name প্যারামিটারটি কিছু না দেওয়া হলে, এটি স্বয়ংক্রিয়ভাবে "John" নেবে।

### ২. নাম পরিবর্তন করা

```
function sayHello(greetings, name = "John") {  
    console.log(`${greetings} ${name}`); // স্ট্রিং লিটারেল  
}  
  
sayHello("Hello", "Jane"); // ডিফল্ট প্যারামিটার সেট করা থাকলেও, নতুন নাম "Jane"  
পাস করলে সেটি গ্রহণ করবে
```

#### ব্যাখ্যা:

- এখানে আবার sayHello ফাংশনটি ব্যবহার করা হয়েছে, কিন্তু এবার আমরা name প্যারামিটারে "Jane" পাস করেছি।
- যেহেতু আমরা একটি নতুন নাম পাস করেছি, তাই ডিফল্ট "John" এর পরিবর্তে "Jane" ব্যবহার করা হবে।

## কেন এটি মজার? 😊

- সহজ ব্যবহার: আপনি যদি দ্বিতীয় প্যারামিটার না দেন, তবে এটি স্বয়ংক্রিয়ভাবে একটি মান নেয়।
- নাম পরিবর্তন: আপনি চাইলে ডিফল্ট নাম পরিবর্তন করতে পারেন, এবং এটি খুব সহজ!

## কখন ব্যবহার করবেন? 💡

- ✓ যখন আপনি চান যে কিছু প্যারামিটার স্বয়ংক্রিয়ভাবে একটি মান নিক।
- ✓ যখন আপনি ফাংশনকে আরও নমনীয় করতে চান।

## immediate invoked function (IIF)/Anonymous function:

```
(()=>{  
    let a = 10;  
    let b = 20;  
    let c = a + b;  
  
    console.log(c);  
})();
```

### ব্যাখ্যা:

- এটি একটি বিশেষ ধরনের ফাংশন যা স্বয়ংক্রিয়ভাবে কাজ করে।
- এখানে `a` এবং `b` নামের দুটি সংখ্যা (১০ এবং ২০) তৈরি করা হয়েছে।
- `c` নামের একটি নতুন সংখ্যা তৈরি হয়েছে যা `a` এবং `b` এর যোগফল।
- `console.log(c)` ব্যবহার করে আমরা `c` এর মান (৩০) দেখতে পাচ্ছি।

## ভেরিয়েবল ঘোষণা সহ (অ্যারো সহ):

```
<script>  
  
let add = (a, b) => {  
    let c = a + b;  
  
    console.log(c);  
};  
  
add(10, 20);  
  
</script>
```

### ব্যাখ্যা:

- এখানে একটি ফাংশন `add` তৈরি করা হয়েছে যা দুটি সংখ্যা (যেমন `a` এবং `b`) গ্রহণ করে।
- `c` নামের একটি নতুন সংখ্যা তৈরি হয় যা `a` এবং `b` এর যোগফল।
- `console.log(c)` ব্যবহার করে আমরা `c` এর মান দেখতে পাই।
- যখন আমরা `add(10, 20)` লিখি, তখন এটি ১০ এবং ২০ এর যোগফল (৩০) দেখায়।

## অন্য পদ্ধতি (অ্যারো ছাড়া):

```
const square = function (a) {  
    return a * a;  
};  
  
console.log(square(10));
```

### ব্যাখ্যা:

- এখানে একটি সাধারণ ফাংশন `square` তৈরি করা হয়েছে যা একটি সংখ্যা (যেমন `a`) গ্রহণ করে।
- এটি `a` এর বর্গফল (যেমন  $10 \times 10$ ) প্রদান করে।
- `console.log(square(10))` ব্যবহার করে আমরা `10` এর বর্গফল (`100`) দেখতে পাই।

## Unlimited Parameter Passing: সহজ ভাষায় ব্যাখ্যা ✨

### কীভাবে কাজ করে? 😞

কখনো কখনো আপনি জানেন না কতগুলো ইনপুট প্যারামিটার লাগবে। `...args` (Rest Parameter) ব্যবহার করে আপনি অসীম সংখ্যক প্যারামিটার নিতে পারবেন এবং এগুলোকে একটি অ্যারেতে পেতে পারবেন।

### ১. প্যারামিটারগুলোকে একটি অ্যারেতে দেখানো

```
<script>  
  
    numbers = (...args) => {  
        console.log(args); // args হলো সব প্যারামিটারের একটি অ্যারে।  
    };  
  
    numbers(10, 20, 30, 40, 50, 60, 70, 80);  
    // Output: [10, 20, 30, 40, 50, 60, 70, 80] (একটি অ্যারে)  
  
</script>
```

### কীভাবে কাজ করে?

- `...args`: এটি সকল প্যারামিটারকে একটি অ্যারেতে রূপান্তরিত করে।
- `console.log(args)`: প্যারামিটারগুলোর অ্যারে দেখাবে।

#### ফলাফল:

```
[10, 20, 30, 40, 50, 60, 70, 80]
```



## ২. অসীম সংখ্যক প্যারামিটারের যোগফল বের করা

```
<script>

add = (...args) => {
  let sum = 0; // যোগফলের জন্য প্রাথমিক মান
  for (let i = 0; i < args.length; i++) { // প্রতিটি প্যারামিটার ধরে যোগ
    sum += args[i];
  }

  console.log(sum); // যোগফল দেখানো।
};

add(10, 20, 30, 40, 50); // Output: 150

</script>
```

### কীভাবে কাজ করে?

- `...args` : প্যারামিটারগুলোকে একটি অ্যারেতে রূপান্তরিত করেছে।
- **লুপ**: প্রতিটি মান ধরে `sum`-এ যোগ করেছে।
- **ফলাফল দেখানো**: `console.log(sum)` ব্যবহার করে যোগফল দেখানো হয়েছে।

### ফলাফল:

150

### উপসংহার:

`...args` ব্যবহার করলে আপনি যত ইচ্ছা প্যারামিটার পাঠাতে পারবেন। 🚀

---

## JavaScript অবজেক্ট (Object) কি?

### সাধারণ ভাষায় বুঝি 😞

অবজেক্ট হচ্ছে একটি "বাক্স" যার মধ্যে আমরা বিভিন্ন তথ্য সংরক্ষণ করতে পারি। এটা একটি বিশেষ জিনিস যেখানে আমরা একই সাথে একাধিক তথ্য রাখতে পারি।

### কোডের বিস্তারিত ব্যাখ্যা 🤖

### অবজেক্ট ডিক্লেয়ারেশন

```
const person = {
  firstName: "John",
  lastName: "Doe",
```

```
    age: 50,  
    eyeColor: "blue"  
};
```

এখানে `person` হচ্ছে আমাদের অবজেক্টের নাম। এর মধ্যে আমরা রেখেছি:

- `firstName`: ব্যক্তির প্রথম নাম
- `lastName`: ব্যক্তির শেষ নাম
- `age`: ব্যক্তির বয়স
- `eyeColor`: চোখের রঙ

## অবজেক্ট থেকে তথ্য বের করা

```
person.firstName // "John"  
person.lastName  // "Doe"  
person.age       // 50  
person.eyeColor  // "blue"
```

## টেমপ্লেট লিটারাল ব্যবহার ☞

```
document.write(`This person's name is ${person.firstName} ${person.lastName}, his  
age is ${person.age} and his eye color is ${person.eyeColor}.`);
```

## সহজ উদাহরণ

মনে করুন, আপনি এক বাক্সে (অবজেক্ট) আপনার তথ্য রাখলেন:

```
const student = {  
  name: "রাহুল",  
  age: 12,  
  className: "৬ষ্ঠ শ্রেণী"  
};
```

## গুরুত্বপূর্ণ নিয়ম ✅

1. অবজেক্ট ডিক্লেয়ার করার সময় `const` ব্যবহার করুন
2. `key` এবং `value` এর মাঝে `:` ব্যবহার করুন
3. `key` গুলো হয় `camelCase` বা `snake_case` হয়

## সাবধানতা ⚠️

- `key` এর নাম ইংরেজিতে হবে
- `value` যে কোনো ধরনের হতে পারে (string, number, boolean)

# মজার তথ্য 🎉

JavaScript-এ অবজেক্ট হচ্ছে "key-value pair" এর একটি সংগ্রহ, যেখানে প্রতিটি `key` হল নাম এবং `value` হল সেই নামের মান!

---

## Array Declaration & Usage: সহজ ভাষায় ব্যাখ্যা ✨

### Array কী?

**Array** একটি বিশেষ "ডাটা স্টোরেজ" যেখানে আপনি একাধিক মান (value) রাখতে পারেন। এটি একটি বাক্সের মতো যেখানে প্রতিটি মান একটি নির্দিষ্ট "স্থান" (index) অনুযায়ী রাখা হয়।

### Array ডিক্লারেশন:

```
const variableName = [  
  value1, // প্রথম মান  
  value2, // দ্বিতীয় মান  
  value3  // তৃতীয় মান  
];
```

### উদাহরণ: ফলের তালিকা

```
const fruits = ["apple", "banana", "orange", 10];
```

- `"apple"`: প্রথম মান (index 0)
- `"banana"`: দ্বিতীয় মান (index 1)
- `"orange"`: তৃতীয় মান (index 2)
- `10`: চতুর্থ মান (index 3)

### ডেটা অ্যাক্সেস করা:

```
document.write(`Please bring ${fruits[0]}, ${fruits[1]} and ${fruits[2]}.  
${fruits[3]}kgs each.`);
```

### কীভাবে কাজ করে?

- `fruits[0]`: apple নেয়।
- `fruits[1]`: banana নেয়।
- `fruits[2]`: orange নেয়।
- `fruits[3]`: সংখ্যা 10 নেয়।

### আউটপুট:

Please bring apple, banana and orange. 10kgs each.

## উপসংহার:

Array ব্যবহার করে অনেকগুলো মান সহজেই সংরক্ষণ এবং অ্যাক্সেস করা যায়।

## Looping over array:

```
<script>
  const fruits = ["apple", "banana", "orange"];

  for (const fruit in fruits) {
    document.write(fruits[fruit] + "<br>");
  }
</script>
```

### কোডের ব্যাখ্যা:

এই কোডটি একটি অ্যারে (array) ব্যবহার করে, যেখানে কিছু ফলের নাম রাখা হয়েছে। অ্যারে হলো একটি তালিকা, যেখানে আমরা একাধিক তথ্য সংরক্ষণ করতে পারি। এখানে আমাদের অ্যারের নাম `fruits` এবং এর মধ্যে তিনটি ফলের নাম আছে: "apple", "banana", এবং "orange"।

### লুপিং (Looping) এর ব্যাখ্যা:

লুপিং মানে হলো একটি কাজ বারবার করা। এখানে আমরা একটি `for...in` লুপ ব্যবহার করছি। এটি আমাদেরকে অ্যারেতে থাকা প্রতিটি ফলের নামের উপর কাজ করতে সাহায্য করে।

1. `for (const fruit in fruits)`: এই লাইনটি বলে যে, আমরা `fruits` অ্যারেতে থাকা প্রতিটি ফলের জন্য একটি পরিবর্তনশীল (variable) `fruit` তৈরি করছি। এটি আমাদেরকে অ্যারের প্রতিটি ইনডেক্স (index) এর জন্য কাজ করতে দেয়।
2. `document.write(fruits[fruit] + "<br>")`: এই লাইনটি `fruits` অ্যারেতে `fruit` পরিবর্তনশীলের মাধ্যমে যে ফলের নাম আছে, সেটি লেখে। `<br>` ট্যাগটি নতুন লাইনে লেখার জন্য ব্যবহৃত হয়।

### সহজ ভাষায়:

- **অ্যারেতে ফলের নাম:** এখানে আমরা "apple", "banana", এবং "orange" নামের ফলের একটি তালিকা তৈরি করেছি।
- **লুপিং:** আমরা একটি লুপ ব্যবহার করে এই ফলগুলোর নাম একের পর এক দেখাচ্ছি।
- **কী হচ্ছে:** যখন কোডটি চলে, তখন এটি "apple", "banana", এবং "orange" এই তিনটি ফলের নাম আলাদা আলাদা লাইনে দেখাবে।

এভাবে, আমরা অ্যারেতে থাকা প্রতিটি ফলের নামকে একের পর এক দেখাতে পারি।

## Different built in method for Array operation:

- `concat()` : এই মেথডটি দুটি বা তার বেশি অ্যারেকে একত্রিত করে একটি নতুন অ্যারে তৈরি করে।
- `from()` : এই মেথডটি একটি অ্যারে তৈরি করে একটি নির্দিষ্ট অবজেক্ট বা iterable (যেমন string) থেকে।
- `push()` : এই মেথডটি একটি বা একাধিক উপাদানকে অ্যারের শেষে যোগ করে।
- `pop()` : এই মেথডটি অ্যারের শেষ থেকে একটি উপাদান মুছে ফেলে এবং সেই উপাদানটি ফেরত দেয়।
- `forEach()` : এই মেথডটি অ্যারের প্রতিটি উপাদানের জন্য একটি ফাংশন চালায়।
- `reverse()` : এই মেথডটি অ্যারেটির উপাদানগুলিকে উল্টে দেয়।
- `sort()` : এই মেথডটি অ্যারের উপাদানগুলিকে সাজায়।
- `slice()` : এই মেথডটি একটি নতুন অ্যারে তৈরি করে, যা মূল অ্যারের একটি নির্দিষ্ট অংশকে ধারণ করে।
- `splice()` : এই মেথডটি অ্যারের নির্দিষ্ট স্থানে উপাদান যোগ বা মুছে ফেলার জন্য ব্যবহৃত হয়।

### `concat()`:

```
<script>
  const fruits1 = ["apple", "banana", "orange"];
  const fruits2 = ["mango", "pineapple", "grapes"];

  /**
   concat() method
   concatenation syntax: arrayName1.concat(arrayName2)
  **/

  const allFruits1 = fruits1.concat(fruits2);
  const allFruits2 = fruits2.concat(fruits1);

  document.write(allFruits1 + "<br />");
  document.write(allFruits2 + "<br />");
</script>
```

এই কোডটি দুটি আলাদা ফলের তালিকা (অ্যারে) তৈরি করেছে এবং সেগুলিকে একত্রিত (`concat`) করার জন্য `concat()` মেথড ব্যবহার করেছে।

1. `fruits1` এবং `fruits2`: এখানে দুটি অ্যারে তৈরি করা হয়েছে।

- `fruits1` এর মধ্যে আছে: "apple", "banana", "orange"
- `fruits2` এর মধ্যে আছে: "mango", "pineapple", "grapes"

2. `concat()` মেথড:

- `const allFruits1 = fruits1.concat(fruits2);` এই লাইনটি `fruits1` এবং `fruits2` কে একত্রিত করে একটি নতুন অ্যারে তৈরি করে, যার

নাম `allFruits1`।

- `const allFruits2 = fruits2.concat(fruits1);` এই লাইনটি `fruits2` এবং `fruits1` কে একত্রিত করে একটি নতুন অ্যারে তৈরি করে, যার নাম `allFruits2`।

3. `document.write()`:

- `document.write(allFruits1 + "<br />");` এই লাইনটি `allFruits1` এর উপাদানগুলোকে পৃষ্ঠায় লেখে।
- `document.write(allFruits2 + "<br />");` এই লাইনটি `allFruits2` এর উপাদানগুলোকে পৃষ্ঠায় লেখে।

## সহজ ভাষায়:

- **ফলগুলোর তালিকা:** এখানে দুটি তালিকা তৈরি করা হয়েছে, একটি তালিকায় কিছু ফলের নাম এবং অন্যটিতে কিছু ভিন্ন ফলের নাম।
- **একত্রিত করা:** `concat()` মেথড ব্যবহার করে আমরা দুটি তালিকাকে একত্রিত করেছি।
  - প্রথম তালিকা (`fruits1`) এবং দ্বিতীয় তালিকা (`fruits2`) একত্রিত করে `allFruits1` তৈরি হয়েছে।
  - দ্বিতীয় তালিকা (`fruits2`) এবং প্রথম তালিকা (`fruits1`) একত্রিত করে `allFruits2` তৈরি হয়েছে।

## from():

```
<script>
  const fruits = "John Doe";

  // will convert every single character to an array
  const array = Array.from(fruits);
  document.write(array + "<br />");

  const no2IndexElement = array[2];
  document.write(no2IndexElement);
</script>
```

এই কোডটি একটি স্ট্রিং (string) থেকে একটি অ্যারে (array) তৈরি করতে `Array.from()` মেথড ব্যবহার করছে।

1. `fruits`: এখানে একটি স্ট্রিং তৈরি করা হয়েছে যার মান "John Doe"।
2. `Array.from(fruits)`:
  - এই লাইনটি `fruits` স্ট্রিংটিকে একটি অ্যারেতে রূপান্তরিত করে।
  - প্রতিটি অক্ষর আলাদা আলাদা উপাদান হিসেবে অ্যারেতে যুক্ত হয়।
  - ফলে, `array` হবে: `["J", "o", "h", "n", " ", "D", "o", "e"]`।
3. `document.write(array + "<br />")`:
  - এই লাইনটি `array` এর উপাদানগুলোকে পৃষ্ঠায় লেখে।
4. `const no2IndexElement = array[2];`:

- এখানে `array` এর তৃতীয় উপাদান (যার ইনডেক্স 2) `no2IndexElement` নামক একটি পরিবর্তনশীল (variable) এ সংরক্ষণ করা হয়েছে।
- `array[2]` মানে হলো তৃতীয় উপাদান, যা "h"।

5. `document.write(no2IndexElement);` :

- এই লাইনটি `no2IndexElement` এর মান (যা "h") পৃষ্ঠায় লেখে।

## সহজ ভাষায়:

- **স্ট্রিং থেকে অ্যারে:** এখানে "John Doe" নামের একটি স্ট্রিং আছে। আমরা `Array.from()` ব্যবহার করে এই স্ট্রিংটিকে একটি অ্যারেতে রূপান্তরিত করছি।
- **অক্ষরগুলো আলাদা করা:** `Array.from()` মেথডটি প্রতিটি অক্ষরকে আলাদা করে একটি তালিকায় (অ্যারে) রাখে।
- **দেখানো:** আমরা পুরো অ্যারেটি পৃষ্ঠায় দেখাচ্ছি, এবং তারপর তৃতীয় অক্ষর (যা "h") আলাদাভাবে দেখাচ্ছি।

## আউটপুট:

যখন কোডটি চলে, তখন পৃষ্ঠায় নিচের ফলাফল দেখা যাবে:

```
J,o,h,n, ,D,o,e
2
```



**.filter() :**

কল্পনা করুন, আপনি একটি "ফল বাছাইকারী" যন্ত্র পেয়েছেন যা শুধুমাত্র জোড় সংখ্যা বাছাই করতে পারে! এই যন্ত্রটি হচ্ছে `.filter()` মেথড! 🚀

## কোডের ব্যাখ্যা 🔍

```
<script>
  const array = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];

  /**
   * .filter() method
   * filtering syntax:
   * arrayVariableName.filter((element) => {
   *   statement
   * });
   */

  const filteredResult = array.filter((element) => {
    return element % 2 === 0;
  });

  document.write(filteredResult);
</script>
```

1. `array`: এখানে 1 থেকে 10 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে। এটি আমাদের "ফল"।

2. `.filter()` মেথড:

- `const filteredResult = array.filter((element) => { return element % 2 === 0; });`
- এই লাইনটি `array` এর প্রতিটি উপাদানের জন্য একটি ফাংশন চালায়।
- ফাংশনটি চেক করে যে উপাদানটি (`element`) 2 দিয়ে ভাগ করলে 0 আসে কিনা (অর্থাৎ, এটি একটি জোড় সংখ্যা কিনা)।
- যদি শর্তটি সত্য হয়, তাহলে সেই উপাদানটি নতুন অ্যারেতে অন্তর্ভুক্ত হয়।

3. `document.write()`:

- `document.write(filteredResult);` এই লাইনটি `filteredResult` এর মান (যা জোড় সংখ্যা) পৃষ্ঠায় লেখে।

## সহজ ভাষায়:

- **অ্যারেতে সংখ্যা:** এখানে 1 থেকে 10 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে আছে।
- **জোড় সংখ্যা বের করা:** আমরা `.filter()` মেথড ব্যবহার করে অ্যারেটি থেকে জোড় সংখ্যা বের করছি।
- **দেখানো:** আমরা জোড় সংখ্যা সম্বলিত নতুন অ্যারেটি পৃষ্ঠায় দেখাচ্ছি।

## ★ আউটপুট:

যখন কোডটি চলে, তখন পৃষ্ঠায় নিচের ফলাফল দেখা যাবে:

12,4,6,8,10

## 💡 কখন ব্যবহার করবেন?

- ☒ যখন আপনি একটি অ্যারে থেকে নির্দিষ্ট শর্ত পূরণকারী উপাদানগুলোকে আলাদা করতে চান।
- ☒ যখন আপনার কাজটি সহজ এবং দ্রুত ফলাফল প্রয়োজন।

## ⚠ সাবধানতা:

- সব জায়গায় ব্যবহার করা যায় না; জটিল কাজের জন্য সাধারণ ফাংশন ভাল।

## 🍷 মজার তথ্য:

`.filter()` মেথড হল JavaScript-এর "ফল বাছাইকারী"। এটি আপনাকে একটি অ্যারে থেকে নির্দিষ্ট শর্ত পূরণকারী উপাদানগুলোকে সহজে আলাদা করতে সাহায্য করে!

## 📖 শিক্ষণীয় উপসংহার:

`.filter()` মেথড শুধুমাত্র কোড লেখার একটি সহজ উপায় নয়, এটি আপনার প্রোগ্রামিং যাত্রাকে আরও মজাদার করে তুলবে!





## .find() :

কল্পনা করুন, আপনি একটি "মহাকাশের অনুসন্ধানকারী" পেয়েছেন যা একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানটি খুঁজে বের করতে পারে! এই যন্ত্রটি হচ্ছে .find() মেথড! 🚀

## কোডের ব্যাখ্যা 🔍

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   * .find() method : Will find and return that only one value inside an array,
   * based on the condition (for these conditions (< / <=), that will be the first
   * value in that array, for another condition (>) that will be the exact next value
   * and for another condition (>=) that will return the same value in that array)
   * finding syntax:
   * arrayVariableName.find((element) => {
   *   statement
   * });
   */
  const findElement1 = array.find((element) => {
    return element < 100; // will return only first value 10
  });

  const findElement2 = array.find((element) => {
    return element <= 100; // will return only first value 10
  });

  const findElement3 = array.find((element) => {
    return element > 30; // will return only next value 40
  });

  const findElement4 = array.find((element) => {
    return element >= 30; // will return the same value 30
  });

  document.write(findElement1 + "<br />");
  document.write(findElement2 + "<br />");
  document.write(findElement3 + "<br />");
  document.write(findElement4 + "<br />");
</script>
```

1. **array** : এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে। এটি আমাদের "ফল"।
2. **.find() মেথড**:
  - `const findElement1 = array.find((element) => { return element < 100; });`

- এই লাইনটি প্রথম উপাদানটি খুঁজে বের করে যা 100 এর চেয়ে ছোট। ফলস্বরূপ, এটি 10 ফেরত দেবে।
- `const findElement2 = array.find((element) => { return element <= 100; });`
- এই লাইনটি প্রথম উপাদানটি খুঁজে বের করে যা 100 এর সমান বা ছোট। এটি আবারও 10 ফেরত দেবে।
- `const findElement3 = array.find((element) => { return element > 30; });`
- এই লাইনটি প্রথম উপাদানটি খুঁজে বের করে যা 30 এর চেয়ে বড়। এটি 40 ফেরত দেবে, কারণ এটি 30 এর পরের প্রথম সংখ্যা।
- `const findElement4 = array.find((element) => { return element >= 30; });`
- এই লাইনটি প্রথম উপাদানটি খুঁজে বের করে যা 30 এর সমান বা বড়। এটি 30 ফেরত দেবে।

## সহজ ভাষায়:

- **অ্যারেতে সংখ্যা:** এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে আছে।
- **প্রথম উপাদান খুঁজে বের করা:** আমরা `.find()` মেথড ব্যবহার করে অ্যারেটি থেকে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানগুলো খুঁজে বের করছি।
- **দেখানো:** আমরা খুঁজে পাওয়া উপাদানগুলো পৃষ্ঠায় দেখাচ্ছি।

## 🌟 আউটপুট:

110 210 340 430

## 💡 কখন ব্যবহার করবেন?

- ☒ যখন আপনি একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানটি খুঁজে বের করতে চান।
- ☒ যখন আপনার কাজটি সহজ এবং দ্রুত ফলাফল প্রয়োজন।

## ⚠️ সাবধানতা:

- `.find()` মেথড শুধুমাত্র প্রথম মিল পাওয়া উপাদানটি ফেরত দেয়। যদি কোনো উপাদান না পাওয়া যায়, তাহলে এটি `undefined` ফেরত দেবে।

## 🎉 মজার তথ্য:

`.find()` মেথড হল JavaScript-এর "অনুসন্ধানকারী"। এটি আপনাকে একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানটি সহজে খুঁজে বের করতে সাহায্য করে!

## 🚀 `.findIndex()` :

কল্পনা করুন, আপনি একটি "মহাকাশের নির্দেশক" পেয়েছেন যা একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানের সূচক (index) খুঁজে বের করতে পারে! এই যন্ত্রটি হচ্ছে `.findIndex()` মেথড! 🧐

## কোডের ব্যাখ্যা 🔍

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   .findIndex() method : Like .find() method, will find and grab that only one
   value inside the array, but instead of returning value, it will return the
   value's index, based on the condition (for these conditions (< / <=), that will
   be the first value index in that array, for another condition (>) that will be
   the exact next value index and for another condition (>=) that will return the
   same value index of that array)
   findIndex syntax:
   arrayVariableName.findIndex((element) => {
     statement
   });
  */
  const findElementIndex1 = array.findIndex((element) => {
    return element < 100; // will return only first value 10's index
  });

  const findElementIndex2 = array.findIndex((element) => {
    return element <= 100; // will return only first value 10's index
  });

  const findElementIndex3 = array.findIndex((element) => {
    return element > 30; // will return only next value 40's index
  });

  const findElementIndex4 = array.findIndex((element) => {
    return element >= 30; // will return the same value 30's index
  });

  document.write(findElementIndex1 + "<br />");
  document.write(findElementIndex2 + "<br />");
  document.write(findElementIndex3 + "<br />");
  document.write(findElementIndex4 + "<br />");
</script>
```

1. **array**: এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে। এটি আমাদের "ফল"।
2. **.findIndex() মেথড**:
  - `const findElementIndex1 = array.findIndex((element) => { return element < 100; });`
    - এই লাইনটি প্রথম উপাদানের সূচক খুঁজে বের করে যা 100 এর চেয়ে ছোট। ফলস্বরূপ, এটি 0 ফেরত দেবে (কারণ 10 এর সূচক 0)।

- `const findElementIndex2 = array.findIndex((element) => { return element <= 100; });`  
 • এই লাইনটি প্রথম উপাদানের সূচক খুঁজে বের করে যা 100 এর সমান বা ছোট। এটি আবারও 0 ফেরত দেবে।
- `const findElementIndex3 = array.findIndex((element) => { return element > 30; });`  
 • এই লাইনটি প্রথম উপাদানের সূচক খুঁজে বের করে যা 30 এর চেয়ে বড়। এটি 3 ফেরত দেবে, কারণ 40 এর সূচক 3।
- `const findElementIndex4 = array.findIndex((element) => { return element >= 30; });`  
 • এই লাইনটি প্রথম উপাদানের সূচক খুঁজে বের করে যা 30 এর সমান বা বড়। এটি 2 ফেরত দেবে, কারণ 30 এর সূচক 2।

### 3. `document.write()`:

- প্রতিটি `findElementIndex` এর মান পৃষ্ঠায় লেখে।

## সহজ ভাষায়:

- **অ্যারেতে সংখ্যা:** এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে আছে।
- **প্রথম উপাদানের সূচক খুঁজে বের করা:** আমরা `.findIndex()` মেথড ব্যবহার করে অ্যারেটি থেকে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানের সূচকগুলো খুঁজে বের করছি।

## ★ আউটপুট:

10 20 33 42

## 💡 কখন ব্যবহার করবেন?

- ☒ যখন আপনি একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানের সূচক খুঁজে বের করতে চান।
- ☒ যখন আপনার কাজটি সহজ এবং দ্রুত ফলাফল প্রয়োজন।

## ⚠ সাবধানতা:

- `.findIndex()` মেথড শুধুমাত্র প্রথম মিল পাওয়া সূচকটি ফেরত দেয়। যদি কোনো উপাদান না পাওয়া যায়, তাহলে এটি -1 ফেরত দেবে।

## 🎉 মজার তথ্য:

`.findIndex()` মেথড হল JavaScript-এর "সূচক অনুসন্ধানকারী"। এটি আপনাকে একটি অ্যারেতে নির্দিষ্ট শর্ত পূরণকারী প্রথম উপাদানের সূচকটি সহজে খুঁজে বের করতে সাহায্য করে!

## 🚀 `.forEach()`:

কল্পনা করুন, আপনি একটি "মহাকাশের লুপ" পেয়েছেন যা একটি অ্যারেতে প্রতিটি উপাদানকে একবার করে দেখতে পারে! এই যন্ত্রটি হচ্ছে `.forEach()` মেথড! 🌌

## কোডের ব্যাখ্যা 🔍

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   * .forEach() method : It'll just work like a loop
   * forEach syntax:
   * arrayVariableName.forEach((element) => {
   *   statement
   * });
   */
  const forEachMethod = array.forEach((element) => {
    document.write(element + "<br />");
  });
</script>
```

1. **array**: এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে। এটি আমাদের "ফল"।

2. **.forEach() মেথড**:

- `const forEachMethod = array.forEach((element) => { document.write(element + "<br />"); });`
- এই লাইনটি `array` এর প্রতিটি উপাদানের জন্য একটি ফাংশন চালায়।
- ফাংশনটি প্রতিটি উপাদানকে পৃষ্ঠায় লেখে, এবং `<br />` ট্যাগ ব্যবহার করে প্রতিটি উপাদানের মধ্যে একটি নতুন লাইন তৈরি করে।

## সহজ ভাষায়:

- **অ্যারেতে সংখ্যা**: এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে আছে।
- **প্রতিটি উপাদান দেখানো**: আমরা `.forEach()` মেথড ব্যবহার করে অ্যারের প্রতিটি উপাদানকে পৃষ্ঠায় দেখাচ্ছি।

## 🌟 আউটপুট:

110 220 330 440 550 660 770 880 990 10100

## 💡 কখন ব্যবহার করবেন?

- ☒ যখন আপনি একটি অ্যারেতে প্রতিটি উপাদানকে একবার করে প্রক্রিয়া করতে চান।
- ☒ যখন আপনার কাজটি সহজ এবং দ্রুত ফলাফল প্রয়োজন।

## ⚠️ সাবধানতা:

- `.forEach()` মেথড একটি নতুন অ্যারে তৈরি করে না; এটি শুধুমাত্র প্রতিটি উপাদানের জন্য একটি ফাংশন চালায়।

## মজার তথ্য:

`.forEach()` মেথড হল JavaScript-এর "লুপিং যন্ত্র"। এটি আপনাকে একটি অ্যারেতে প্রতিটি উপাদানকে সহজে প্রক্রিয়া করতে সাহায্য করে!



### `.includes()` :

কল্পনা করুন, আপনি একটি "সন্ধানকারী" পেয়েছেন যা একটি অ্যারেতে নির্দিষ্ট উপাদানটি আছে কিনা তা খুঁজে বের করতে পারে! এই যন্ত্রটি হচ্ছে `.includes()` মেথড! 🚀

## কোডের ব্যাখ্যা 🔍

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
  /**
   * .includes() method : It helps to find, if an element exists in the array or
   not, result will be given with boolean: true/false
   includes syntax:
   arrayVariableName.includes(element);
   */
  const result1 = array.includes(100);
  const result2 = array.includes(110);

  document.write(result1 + "<br />");
  document.write(result2);
</script>
```

1. `array` : এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে। এটি আমাদের "ফল"।
2. `.includes()` মেথড:
  - `const result1 = array.includes(100);`
    - এই লাইনটি চেক করে যে 100 সংখ্যাটি `array` তে আছে কিনা। ফলস্বরূপ, এটি `true` ফেরত দেবে।
  - `const result2 = array.includes(110);`
    - এই লাইনটি চেক করে যে 110 সংখ্যাটি `array` তে আছে কিনা। ফলস্বরূপ, এটি `false` ফেরত দেবে।
3. `document.write()` :
  - `result1` এবং `result2` এর মান পৃষ্ঠায় লেখে।

## সহজ ভাষায়:

- **অ্যারেতে সংখ্যা:** এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে আছে।
- **উপাদান খুঁজে বের করা:** আমরা `.includes()` মেথড ব্যবহার করে চেক করছি যে নির্দিষ্ট সংখ্যা অ্যারেটিতে আছে কিনা।

## ★ আউটপুট:

true

false`

## 💡 কখন ব্যবহার করবেন?

- ✓ যখন আপনি একটি অ্যারেতে নির্দিষ্ট উপাদানটি আছে কিনা তা দ্রুত চেক করতে চান।
- ✓ যখন আপনার কাজটি সহজ এবং দ্রুত ফলাফল প্রয়োজন।

## ⚠ সাবধানতা:

- `.includes()` মেথড কেবলমাত্র প্রথম মিল পাওয়া উপাদানটি চেক করে এবং এটি একটি বুলিয়ান মান (true/false) ফেরত দেয়।

## 🚀 `.indexOf()` :

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
  /**
   * .indexOf() method : It helps to find, if an element exists in the array or
   * not, like .includes() method, it will also check if there exists the passing
   * value in the array, if exists then it will return the index of that value. if it
   * finds then it will return the index of that value, if not then it will return -1
   * indexOf syntax:
   * arrayVariableName.indexOf(element);
   */
  const result1 = array.indexOf(100);
  const result2 = array.indexOf(110);

  document.write(result1 + "<br />");
  document.write(result2);
</script>
```

## কোডের ব্যাখ্যা 🔍

- array** : এখানে 10 থেকে 100 পর্যন্ত সংখ্যা সম্বলিত একটি অ্যারে তৈরি করা হয়েছে।
- `.indexOf()` মেথড**
  - `const result1 = array.indexOf(100);`
    - এই লাইনটি চেক করে যে 100 সংখ্যাটি `array` তে আছে কিনা এবং যদি থাকে, তাহলে এর সূচক (index) ফেরত দেয়। ফলস্বরূপ, এটি 9 ফেরত দেবে (কারণ 100 এর সূচক 9)।
  - `const result2 = array.indexOf(110);`
    - এই লাইনটি চেক করে যে 110 সংখ্যাটি `array` তে আছে কিনা। এটি -1 ফেরত দেবে, কারণ 110 অ্যারেতে নেই।

### 3. document.write():

- result1 এবং result2 এর মান পৃষ্ঠায় লেখে।

## ✴️ আউটপুট:

19 2-1

## 💡 কখন ব্যবহার করবেন?

- ☒ যখন আপনি একটি অ্যারেতে নির্দিষ্ট উপাদানের সূচক খুঁজে বের করতে চান।
- ☒ যখন আপনি জানতে চান যে একটি উপাদান অ্যারেতে আছে কিনা এবং তার সূচক পেতে চান।

## ⚠️ সাবধানতা:

- .indexOf() মেথড কেবলমাত্র প্রথম মিল পাওয়া উপাদানের সূচকটি ফেরত দেয়। যদি কোনো উপাদান না পাওয়া যায়, তাহলে এটি -1 ফেরত দেয়।

## pop():

### উদ্দেশ্য:

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   .pop() method : It will remove the last element from the array
   pop syntax:
   arrayVariableName.pop();
   **/

  const result1 = array.pop();
  const result2 = array.pop();

  document.write(result1 + "<br />"); // the element that deleted
  document.write(result2 + "<br />"); // the element that deleted
  document.write(array); // [10, 20, 30, 40, 50, 60, 70, 80, 90] new array
</script>
```

জাভাস্ক্রিপ্টের pop() মেথড একটি অ্যারের শেষের দিকের উপাদান মুছে ফেলার জন্য ব্যবহৃত হয়।

### কোড বিশ্লেষণ:

#### 1. অ্যারে ঘোষণা:

- const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
- এই লাইনে array নামে একটি ফ্রবক ভেরিয়েবল ঘোষণা করা হয়েছে এবং এতে ১০ থেকে ১০০ পর্যন্ত দশটি সংখ্যার একটি অ্যারে ইনিশিয়ালাইজ করা হয়েছে।



## 2. প্রথম pop() অপারেশন:

- `const result1 = array.pop();`
  - এই লাইনে `array` এর উপর `pop()` মেথড ব্যবহার করা হয়েছে।
  - `pop()` মেথড অ্যারের শেষের দিকের উপাদানটি মুছে ফেলে এবং সেই মুছে ফেলা উপাদানটিকে রিটার্ন করে।
  - `result1` ভেরিয়েবলে শেষের দিকের মুছে ফেলা উপাদান (যা ১০০) সংরক্ষিত হয়।

## 3. দ্বিতীয় pop() অপারেশন:

- `const result2 = array.pop();`
  - আবারও `array` এর উপর `pop()` মেথড ব্যবহার করা হয়েছে।
  - এবার অ্যারের নতুন শেষের দিকের উপাদান (যা ৯০) মুছে ফেলা হয় এবং সেই মুছে ফেলা উপাদানটিকে `result2` ভেরিয়েবলে সংরক্ষিত হয়।

## 4. আউটপুট:

- `document.write(result1 + "<br />");`
  - এই লাইনে প্রথম `pop()` অপারেশনে মুছে ফেলা উপাদান (`result1`) এর মান (১০০) ডকুমেন্টে লেখা হয়।
- `document.write(result2 + "<br />");`
  - এই লাইনে দ্বিতীয় `pop()` অপারেশনে মুছে ফেলা উপাদান (`result2`) এর মান (৯০) ডকুমেন্টে লেখা হয়।
- `document.write(array);`
  - এই লাইনে `pop()` অপারেশনগুলির পরে পরিবর্তিত `array` এর মান (যা এখন [10, 20, 30, 40, 50, 60, 70, 80, 90] হয়ে গেছে) ডকুমেন্টে লেখা হয়।

## মূল ধারণা:

- **অ্যারে:** মানের সংগ্রহ সঞ্চয় করার জন্য ডেটা স্ট্রাকচার।
- **pop() মেথড:** জাভাস্ক্রিপ্টের একটি বিল্ট-ইন মেথড যা অ্যারের শেষের দিকের উপাদানটি মুছে ফেলে এবং সেই মুছে ফেলা উপাদানটিকে রিটার্ন করে।

## নোট:

- `pop()` মেথড অ্যারেটিকে পরিবর্তন করে। অর্থাৎ, মূল অ্যারে থেকে উপাদান মুছে ফেলা হয়।

## push():

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   .push() method : It will add the element at the end of the array
   push syntax:
   arrayVariableName.push(element);
   **/
  const result1 = array.push(110);
  const result2 = array.push(120);
```

```
document.write(result1 + "<br />"); // the element that added
document.write(result2 + "<br />"); // the element that added

document.write(array); // [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110,
120] new array
</script>
```

## উদ্দেশ্য:

জাভাস্ক্রিপ্টের `push()` মেথড একটি অ্যারের শেষে উপাদান যোগ করার জন্য ব্যবহৃত হয়।

## কোড বিশ্লেষণ:

### 1. অ্যারে ঘোষণা:

- `const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];`
- এই লাইনে `array` নামে একটি ফ্লক ভেরিয়েবল ঘোষণা করা হয়েছে এবং এতে ১০ থেকে ১০০ পর্যন্ত দশটি সংখ্যার একটি অ্যারে ইনিশিয়ালাইজ করা হয়েছে।

### 2. প্রথম `push()` অপারেশন:

- `const result1 = array.push(110);`
  - এই লাইনে `array` এর উপর `push()` মেথড ব্যবহার করা হয়েছে এবং এর আর্গুমেন্ট হিসাবে ১১০ দেওয়া হয়েছে।
  - `push()` মেথড অ্যারের শেষে নতুন উপাদান যোগ করে এবং নতুন অ্যারের দৈর্ঘ্য রিটার্ন করে।
  - `result1` ভেরিয়েবলে নতুন অ্যারের দৈর্ঘ্য (যা এখন ১১) সংরক্ষিত হয়।

### 3. দ্বিতীয় `push()` অপারেশন:

- `const result2 = array.push(120);`
  - আবারও `array` এর উপর `push()` মেথড ব্যবহার করা হয়েছে এবং এর আর্গুমেন্ট হিসাবে ১২০ দেওয়া হয়েছে।
  - এবার অ্যারের শেষে ১২০ যোগ করা হয় এবং নতুন অ্যারের দৈর্ঘ্য (যা এখন ১২) `result2` ভেরিয়েবলে সংরক্ষিত হয়।

### 4. আউটপুট:

- `document.write(result1 + "<br />");`
  - এই লাইনে প্রথম `push()` অপারেশনের পরে অ্যারের দৈর্ঘ্য (`result1`) এর মান (১১) ডকুমেন্টে লেখা হয়।
- `document.write(result2 + "<br />");`
  - এই লাইনে দ্বিতীয় `push()` অপারেশনের পরে অ্যারের দৈর্ঘ্য (`result2`) এর মান (১২) ডকুমেন্টে লেখা হয়।
- `document.write(array);`
  - এই লাইনে `push()` অপারেশনগুলির পরে পরিবর্তিত `array` এর মান (যা এখন [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120] হয়ে গেছে) ডকুমেন্টে লেখা হয়।

## মূল ধারণা:

- **অ্যারে:** মানের সংগ্রহ সঞ্চয় করার জন্য ডেটা স্ট্রাকচার।
- **push() মেথড:** জাভাস্ক্রিপ্টের একটি বিল্ট-ইন মেথড যা অ্যারের শেষে একটি বা একাধিক উপাদান যোগ করে এবং নতুন অ্যারের দৈর্ঘ্য রিটার্ন করে।

### নোট:

- push() মেথড অ্যারেটিকে পরিবর্তন করে। অর্থাৎ, মূল অ্যারেতে নতুন উপাদান যোগ করা হয়।

### reverse() :

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   .reverse() method : It will reverse the array
   reverse syntax:
   arrayVariableName.reverse();
   **/

  const result = array.reverse(); // in result variable it will store the
  reversed array

  document.write(result + "<br />"); // [100, 90, 80, 70, 60, 50, 40, 30, 20,
  10]
  document.write(array); // previous array replaced by reversed array
</script>
```

### কীভাবে জাভাস্ক্রিপ্ট একটি অ্যারের উপাদানগুলোকে উল্টো দিকে সাজাবেন?

জাভাস্ক্রিপ্ট reverse() নামক একটি বিশেষ কাজ আছে। এই কাজটি ব্যবহার করে আমরা যেকোনো অ্যারের উপাদানগুলোকে বিপরীত ক্রমে সাজাতে পারি।

### উদাহরণ:

```
const numbers = [1, 2, 3, 4, 5];
```

এখন আমরা এই তালিকার সংখ্যাগুলোকে উল্টো দিকে সাজাতে চাই (অর্থাৎ ৫, ৪, ৩, ২, ১)। আমরা reverse() ব্যবহার করে এটি করতে পারি:

```
numbers.reverse();
```

এখন numbers তালিকায় সংখ্যাগুলো উল্টো দিকে সাজানো হয়ে গেছে।

### কিভাবে এটি কাজ করে?

- numbers হল আমাদের সংখ্যার তালিকা।

- `reverse()` এই তালিকার নামের পরে ব্যবহার করতে হয়।

এইভাবে খুব সহজেই যেকোনো অ্যারের উপাদানগুলোকে বিপরীত ক্রমে সাজানো যায়।

**আশা করি এই ডকুমেন্টেশনটি সহজ এবং বোধগম্য হয়েছে।**

**নোট:**

- `reverse()` কাজটি মূল অ্যারেকেই পরিবর্তন করে দেয়।
- যদি আপনি মূল অ্যারে পরিবর্তন না করে শুধুমাত্র একটি নতুন উল্টো অ্যারে তৈরি করতে চান, তাহলে আগে মূল অ্যারেকে অন্য একটি নতুন অ্যারেতে কপি করে নিন এবং তারপর `reverse()` ব্যবহার করুন।

**sort():**

```
<script>
    const array = ["cherry", "banana", "apple", "mango", "orange"];
    /**
     * .sort() method : The .sort() method by default only works with strings. It
     * converts array elements to strings and sorts them lexicographically (dictionary
     * order) by default. This works fine for strings but can lead to unexpected results
     * with numbers/floats because they are compared as strings. To sort numbers/floats,
     * we must provide a compare function as an argument to the .sort() method. This
     * compare function defines the sorting logic.
     *
     * sort syntax:
     * arrayVariableName.sort();
     */
    const result = array.sort(); // in result variable it will store the sorted
    array

    document.write(result + "<br />"); // ["apple", "banana", "cherry",
    "mango", "orange"]

    // for numbers sorting
    const numbers = [15, 10, 25, 40, 35, 22.5];
    const result1 = numbers.sort((a, b) => {
        return a - b;
    }); // in result variable it will store the sorted array

    // we can also write like this for simple statement/logic
    const result2 = numbers.sort((a, b) => a - b);
    document.write(result1 + "<br />"); // [10, 15, 22.5, 25, 35, 40]
    document.write(result2); // [10, 15, 22.5, 25, 35, 40]

    /**
     * Strings: .sort() works without extra logic (default lexicographical
     * sorting).
     * Numbers/Floats: we need to provide a compare function for accurate
```

```
sorting.  
    **/  
</script>
```

## sort().reverse() : আইটেমগুলোকে উল্টো ক্রমে সাজানো

```
<script>  
    const array = ["cherry", "banana", "apple", "mango", "orange"];  
    /**  
     * .sort().reverse() method : The .sort().reverse() method by default only  
     * works with strings. It converts array elements to strings and sorts them in  
     * descending order lexicographically (dictionary order) by default. This works fine  
     * for strings but can lead to unexpected results with numbers/floats because they  
     * are compared as strings. To sort numbers/floats, we must provide a compare  
     * function as an argument to the .sort() method. This compare function defines the  
     * sorting logic.  
     * sort(descending) syntax:  
     * arrayVariableName.sort().reverse();  
     */  
    const result = array.sort().reverse(); // in result variable it will store  
    the sorted array in descending order  
    document.write(result + "<br />"); // ["orange", "mango", "cherry",  
    "banana", "apple"]  
  
    // for numbers sorting  
    const numbers = [15, 10, 25, 40, 35, 22.5];  
    const result1 = numbers.sort((a, b) => {  
        return a - b;  
    }); // in result variable it will store the sorted array in descending  
    order  
  
    // we can also write like this for simple statement/logic  
    const result2 = numbers.sort((a, b) => a - b).reverse();  
    document.write(result1 + "<br />"); // [40, 35, 25, 22.5, 15, 10]  
    document.write(result2); // [40, 35, 25, 22.5, 15, 10]  
  
    /**  
     * Strings: .sort() works without extra logic (default lexicographical  
     * sorting).  
     * Numbers/Floats: we need to provide a compare function for accurate  
     * sorting.  
     */  
</script>
```

এই কোডটি জাভাস্ক্রিপ্ট একটি তালিকা (অ্যারে) এর আইটেমগুলোকে উল্টো ক্রমে সাজানোর জন্য ব্যবহৃত হয়।

- স্ট্রিং সাজানো:

- যদি তালিকায় শব্দ থাকে (স্ট্রিং), তাহলে `sort()` কাজটি ব্যবহার করে প্রথমে শব্দগুলোকে আগে-পিছে সাজানো হয় (যেমন: "আপেল" আগে, "আম" পরে)।
- তারপর `reverse()` কাজটি ব্যবহার করে এই সাজানো ক্রমকে উল্টো করে দেওয়া হয়।
- **সংখ্যা সাজানো:**
  - যদি তালিকায় সংখ্যা থাকে, তাহলে `sort()` কাজটির সাথে একটি বিশেষ নির্দেশনা দেওয়া হয়। এই নির্দেশনাটি বলে দেয় কোন সংখ্যা আগে আসবে এবং কোন সংখ্যা পরে আসবে।
  - তারপর `reverse()` কাজটি ব্যবহার করে এই সাজানো ক্রমকে উল্টো করে দেওয়া হয়।

## উদাহরণ:

- **শব্দ সাজানো:**
  - ধরো, আমাদের কাছে ফলের নামের একটি তালিকা আছে: ["আপেল", "কলা", "আম", "লেবু"]
  - `sort()` এবং `reverse()` কাজটি ব্যবহার করলে এই তালিকা হয়ে যাবে: ["লেবু", "আম", "কলা", "আপেল"]
- **সংখ্যা সাজানো:**
  - ধরো, আমাদের কাছে সংখ্যার একটি তালিকা আছে: [5, 2, 8, 1, 9]
  - `sort()` এবং `reverse()` কাজটি ব্যবহার করলে এই তালিকা হয়ে যাবে: [9, 8, 5, 2, 1]

## সহজ কথায়:

এই কোডটি শুধু তালিকার আইটেমগুলোকে উল্টো করে দেয়। যেমন, যদি তালিকায় শব্দ থাকে, তাহলে শেষের শব্দ আগে আসবে এবং প্রথমের শব্দ শেষে যাবে।

## নোট:

- এই কাজগুলো করার সময় মূল তালিকাও পরিবর্তন হয়ে যায়।
- যদি আপনি মূল তালিকা পরিবর্তন করতে না চান, তাহলে আগে মূল তালিকা থেকে একটি নতুন তালিকা তৈরি করুন এবং তারপর এই কাজগুলো নতুন তালিকার উপর করুন।

## slice(): জাভাস্ক্রিপ্ট অ্যারের একটি নির্দিষ্ট অংশ নির্বাচন করা

```
<script>
  const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];

  /**
   * .slice() method : It will slice the array from the starting index to the
   ending index, that means it will show the values from the given starting index to
   the given ending index
   slice syntax:
   arrayVariableName.slice(startingIndex, endingIndex);
   */
  const sliceMethod = array.slice(1, 3); // [20, 30]
```

```
document.write(sliceMethod);  
</script>
```

এই কোডটি জাভাস্ক্রিপ্ট একটি তালিকা (অ্যারে) থেকে নির্দিষ্ট কিছু আইটেম নির্বাচন করার জন্য ব্যবহৃত হয়। `slice()` নামক একটি কাজ ব্যবহার করে আমরা তালিকার যেকোনো অংশ নির্বাচন করতে পারি।

- `slice()` কাজটির মধ্যে দুটি সংখ্যা দেওয়া হয়:
  - প্রথম সংখ্যাটি নির্দেশ করে যে কোন অবস্থান থেকে শুরু করতে হবে।
  - দ্বিতীয় সংখ্যাটি নির্দেশ করে কোন অবস্থান পর্যন্ত নির্বাচন করতে হবে।

### উদাহরণ:

- ধরো, আমাদের কাছে সংখ্যার একটি তালিকা আছে: [10, 20, 30, 40, 50]
- `slice(1, 3)` ব্যবহার করলে এটি দ্বিতীয় অবস্থান থেকে শুরু করে তৃতীয় অবস্থান পর্যন্ত (তৃতীয় অবস্থানটি অন্তর্ভুক্ত নয়) সংখ্যাগুলো নির্বাচন করবে।
- ফলে, আমরা পাব: [20, 30]

### সহজ কথায়:

এটি যেন তালিকা থেকে একটি টুকরা কেটে নেওয়ার মতো।

### নোট:

- `slice()` কাজটি মূল তালিকাকে পরিবর্তন করে না। এটি একটি নতুন তালিকা তৈরি করে যেখানে নির্বাচিত আইটেমগুলো থাকে।

## splice():

```
<script>  
    const array = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]  
  
    /**  
     * .splice() method : It will remove the elements from the array from the  
     * given starting index to the number of elements that we want to remove, even we  
     * can add the elements to the array  
     * splice syntax:  
     * arrayVariableName.splice(startingIndexThatWillBeRemoved,  
     * removeTheNumberOfElementsWeWantFromGivenStartingIndex);  
     * splice can take multiple parameters:  
     * arrayVariableName.splice(startingIndexThatWillBeRemoved,  
     * removeTheNumberOfElementsWeWantFromGivenStartingIndex, itemsToBeAddedIfWeWant,  
     * itemsToBeAddedIfWeWant, itemsToBeAddedIfWeWant...);  
     */  
  
    // (2 parameters) let's say we want to remove 60 from the array which is on  
    index 5
```

```

const spliceMethod = array.splice(5, 2); // for starting index we are
giving 5 and for number of elements we are giving 2, that means we are removing 2
elements from index 5 and this variable holding the spliced
document.write(`${spliceMethod} are replaced <br />`); // [60, 70]
document.write(array + "<br />"); // updated array : [10, 20, 30, 40, 50,
80, 90, 100]

// (multiple parameters) for adding elements to the array
const array1 = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100];
const spliceMethod1 = array1.splice(5, 2, 55, 65, 75); // for starting
index we are giving 5 and for number of elements we are giving 2 and for adding
elements we are giving 55, 66 and 77. So one the index 5, and for number of
elements we are giving 2, on the value of 60 and 70, we are adding 55, 66 and 77,
that means 60 will be replaced by 55, 66 and 77
document.write(`${spliceMethod1} are replaced <br />`); // [60, 70]
document.write(array1 + "<br />"); // updated array : [10, 20, 30, 40, 50,
55, 66, 77, 80, 90, 100]

const array2 = [10, 20, 30, 40, 50, 80, 90, 100];
const spliceMethod2 = array2.splice(5, 0, 55, 65, 75); // for starting
index we are giving 5 and for number of elements we are giving 0 and for adding
elements we are giving 55, 66 and 77. So one the index 5, and for number of
elements we are giving 0, before the value of 80, we are adding 55, 66 and 77
document.write(array2 + "<br />"); // updated array : [10, 20, 30, 40, 50,
55, 66, 77, 80, 90, 100]
</script>

```

splice() নামক একটি বিশেষ কাজ ব্যবহার করে আমরা জাভাস্ক্রিপ্টের একটি তালিকা (অ্যারে) থেকে কিছু উপাদান বাদ দিতে পারি এবং নতুন উপাদানও যোগ করতে পারি।

### কিভাবে এটি কাজ করে:

- splice() কাজটির মধ্যে কয়েকটি সংখ্যা দেওয়া হয়:
  - **প্রথম সংখ্যা:** কোন অবস্থান থেকে শুরু করে পরিবর্তন করতে হবে।
  - **দ্বিতীয় সংখ্যা:** কতগুলো উপাদান বাদ দিতে হবে।
  - **(ঐচ্ছিক) পরবর্তী সংখ্যাগুলো:** যদি নতুন উপাদান যোগ করতে হয়, তাহলে সেগুলো এখানে দেওয়া হয়।

### উদাহরণ:

- **উপাদান বাদ দেওয়া:**
  - ধরো, আমাদের কাছে সংখ্যার একটি তালিকা আছে: [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
  - splice(5, 2) ব্যবহার করলে এটি ৫ম অবস্থান থেকে শুরু করে দুটি উপাদান (৬০ এবং ৭০) বাদ দিয়ে দেবে।
- **উপাদান যোগ করা:**



- `splice(5, 2, 55, 65, 75)` ব্যবহার করলে এটি ৫ম অবস্থান থেকে শুরু করে দুটি উপাদান বাদ দেবে এবং সেই জায়গায় ৫৫, ৬৫ এবং ৭৫ এই তিনটি নতুন সংখ্যা যোগ করবে।
- **শুধুমাত্র উপাদান যোগ করা:**
  - `splice(5, 0, 55, 65, 75)` ব্যবহার করলে এটি ৫ম অবস্থানে কোনো উপাদান বাদ দেবে না, শুধুমাত্র ৫৫, ৬৫ এবং ৭৫ এই তিনটি নতুন সংখ্যা যোগ করবে।

### সহজ কথায়:

`splice()` কাজটি ব্যবহার করে আমরা তালিকার যেকোনো অংশ পরিবর্তন করতে পারি। আমরা কিছু উপাদান বাদ দিতে পারি, কিছু উপাদান যোগ করতে পারি, অথবা বাদ দেওয়ার সাথে সাথে নতুন উপাদানও যোগ করতে পারি।

### নোট:

- `splice()` কাজটি মূল তালিকাকে পরিবর্তন করে দেয়।