



6. Advance JS Interview Question

>> ES6 (10 questions)

1. What are arrow functions in ES6, and how are they different from regular functions?

Arrow ফাংশন (\Rightarrow) হল JavaScript-এর ES6 সংস্করণে প্রবর্তিত একটি সংক্ষিপ্ত পদ্ধতি যা ফাংশন লেখাকে আরও সহজ করে তোলে। এটি সাধারণ ফাংশনের চেয়ে কিছু ভিন্নভাবে কাজ করে, বিশেষ করে `this` কীওয়ার্ড ব্যবহারের ক্ষেত্রে।

Arrow ফাংশনের সিনট্যাক্স

Arrow ফাংশনের একটি উদাহরণ:

```
const যোগ = (a, b) => a + b;
```

উপরের উদাহরণটি নিচের সাধারণ ফাংশনের সমতুল্য:

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

Arrow ফাংশন ও সাধারণ ফাংশনের মধ্যে পার্থক্য

1. `this` এর Binding:

- Arrow ফাংশন নিজস্ব `this` তৈরি করে না। এটি বাইরে থেকে `this` গ্রহণ করে (একে বলা হয় "lexical `this`").
- সাধারণ ফাংশন এর নিজস্ব `this` থাকে, যা কল করা অবজেক্টকে নির্দেশ করে।

উদাহরণ:

```
function RegularFunction() {  
    this.value = 10;  
    setTimeout(function() {  
        console.log(this.value); // এখানে `this` অনিদিষ্ট বা global context বোঝায়  
    }, 1000);  
}  
  
function ArrowFunction() {  
    this.value = 10;  
    setTimeout(() => {  
        console.log(this.value); // এখানে `this` ArrowFunction এর context বোঝায়  
    }, 1000);  
}  
  
new RegularFunction(); // Undefined বা global `this`  
new ArrowFunction(); // সঠিকভাবে `10` দেখাবে
```

2. Syntax এর সরলতা:

- Arrow ফাংশন single-line ফাংশনের জন্য বেশ সংক্ষিপ্ত এবং সরল।
- একটি প্যারামিটার থাকলে, বন্ধনী ঐচ্ছিক, এবং এক্সপ্রেশন একটিই হলে `{}` এবং `return` প্রয়োজন হয় না।

```
const square = x => x * x; // এক লাইনে সংক্ষিপ্ত ফর্ম
```

3. Constructor হিসেবে ব্যবহার:

- **Arrow ফাংশন** constructor হিসেবে কাজ করতে পারে না। অর্থাৎ `new` দিয়ে এটি ব্যবহার করা সম্ভব নয়।
- **সাধারণ ফাংশন** constructor হিসেবে কাজ করতে পারে এবং `new` ব্যবহার করে object তৈরি করা যায়।

```
const MyArrow = () => {};
new MyArrow(); // Error: MyArrow একটি constructor নয়
```

4. Arguments অবজেক্ট:

- **Arrow ফাংশন** এর নিজস্ব `arguments` অবজেক্ট নেই। অনেক প্যারামিটার ব্যবহার করতে হলে rest parameters (`...args`) ব্যবহার করতে হয়।
- **সাধারণ ফাংশন** `arguments` অবজেক্টকে সহজেই গ্রহণ করে, যা ফাংশনে পাঠানো সকল প্যারামিটার ধারণ করে।

```
function regularFunction() {
  console.log(arguments); // সকল arguments দেখাবে
}

const arrowFunction = (...args) => {
  console.log(args); // সকল arguments একটি array আকারে
  দেখাবে
};
```

5. Implicit Return:

- Arrow ফাংশনে implicit return সম্ভব। যদি {} ব্যবহার না করা হয়, তাহলে এক্সপ্রেশনটি implicit return হিসেবে গণ্য হবে।
- সাধারণ ফাংশনের জন্য return keyword অবশ্যই ব্যবহার করতে হয়।

```
const multiply = (a, b) => a * b; // এক লাইনে implicit return
```

সারসংক্ষেপ

Arrow ফাংশন সাধারণত সংক্ষিপ্ত ও সহজ ফাংশনের জন্য বা এমন ক্ষেত্রে উপযুক্ত যেখানে বাইরের `this` প্রয়োজন হয়, যেমন: কলব্যাক ফাংশন। `this`, `arguments`, এবং `constructor` এর ব্যবহারে এটি সাধারণ ফাংশন থেকে কিছুটা আলাদা।

2. What is a destructuring assignment in ES6, and how does it work with arrays and objects?

ES6 এর **destructuring assignment** হলো একটি সহজ ও সংক্ষিপ্ত উপায় যাতে array বা object এর ভেতরের মানগুলোকে আলাদা ভেরিয়েবলে সংরক্ষণ করা যায়। এটি array বা object থেকে মান সহজেই বের করে আনার জন্য বেশ কার্যকর।

Array Destructuring

Array destructuring এর মাধ্যমে array এর মানগুলোকে নির্দিষ্ট ভেরিয়েবলে রাখা যায়।

উদাহরণস্বরূপ:

```
const numbers = [10, 20, 30];
const [a, b, c] = numbers;

console.log(a); // 10
console.log(b); // 20
console.log(c); // 30
```

এখানে, `[a, b, c] = numbers;` এর মাধ্যমে `a`, `b`, এবং `c` ভেরিয়েবলগুলোতে `numbers` array এর মানগুলো ধরে নেওয়া হয়েছে।

Skip করা ও Default Value:

- যদি কিছু উপাদানকে এড়িয়ে যেতে চান, তাহলে কেবল কমা ব্যবহার করতে পারেন।
- কোনো মান না থাকলে Default Value ও সেট করা যায়।

```
const numbers = [10, 20];
const [a, , c = 30] = numbers;
```

```
console.log(a); // 10  
console.log(c); // 30 (কারণ c এর মান numbers এ ছিল না)
```

Object Destructuring

Object destructuring ব্যবহার করে object এর নির্দিষ্ট প্রপার্টিগুলো আলাদা করে ভেরিয়েবলে রাখা যায়।

```
const person = { name: 'Rabbani', age: 25 };  
const { name, age } = person;  
  
console.log(name); // 'Rabbani'  
console.log(age); // 25
```

এখানে `{ name, age } = person;` এর মাধ্যমে `name` এবং `age` ভেরিয়েবলগুলোতে `person` object এর মানগুলো সংরক্ষিত হয়েছে।

Custom Variable Names এবং Default Value:

- আলাদা ভেরিয়েবল নামে মান রাখতে চাইলে `:` ব্যবহার করতে পারেন।
- প্রপার্টি না থাকলে Default Value সেট করা যায়।

```
const person = { name: 'Rabbani' };  
const { name, age = 30, country: nationality = 'Bangladesh' } = person;  
  
console.log(name); // 'Rabbani'  
console.log(age); // 30 (কারণ `age` প্রপার্টি ছিল না)  
console.log(nationality); // 'Bangladesh' (কারণ country প্রপার্টি ছিল না)
```

Nested Destructuring

Destructuring nested arrays বা objects থেকে মান বের করতেও কাজ করে।

```

const user = {
  name: 'Rabbani',
  address: { city: 'Dhaka', zip: 1200 }
};

const { name, address: { city, zip } } = user;

console.log(name); // 'Rabbani'
console.log(city); // 'Dhaka'
console.log(zip); // 1200

```

Function Parameter Destructuring

Destructuring সরাসরি ফাংশনের প্যারামিটারেও ব্যবহার করা যায়, বিশেষ করে যখন object প্যারামিটার থাকে।

```

function display({ name, age }) {
  console.log(`Name: ${name}, Age: ${age}`);
}

const person = { name: 'Rabbani', age: 25 };
display(person); // Output: Name: Rabbani, Age: 25

```

সারসংক্ষেপ

Destructuring array এবং **object** থেকে দ্রুত ও সহজে মান বের করে ভেরিয়েবলে রাখার উপায়। এটি কোডকে সংক্ষিপ্ত, পরিষ্কার এবং পড়তে সহজ করে তোলে।

- **Project:** Write a function that takes an object with properties and logs each property individually.

নিচে একটি ফাংশন দেওয়া হলো যা একটি অবজেক্ট প্রাপ্ত করে এবং প্রতিটি প্রপার্টি আলাদাভাবে কনসোল-এ প্রদর্শন করে:

```

function logProperties(obj) {
  for (const key in obj) {

```

```

        if (obj.hasOwnProperty(key)) {
            console.log(` ${key}: ${obj[key]}`);
        }
    }
}

```

ব্যাখ্যা

- for...in লুপ:** `for...in` লুপ ব্যবহার করে প্রতিটি প্রপার্টির key পেতে পারি।
- hasOwnProperty:** এটি নিশ্চিত করে যে শুধুমাত্র অবজেক্টের নিজস্ব প্রপার্টি (inherited প্রপার্টি নয়) লুপে আসছে।
- Template String:** `${key}: ${obj[key]}` ব্যবহার করে key এবং তার মান `obj[key]` প্রদর্শন করা হচ্ছে।

উদাহরণ ব্যবহার

```

const person = { name: 'Rabbani', age: 25, country: 'Bangladesh' };
logProperties(person);

```

আউটপুট:

```

name: Rabbani
age: 25
country: Bangladesh

```

এভাবে, `logProperties` ফাংশনটি প্রতিটি প্রপার্টি আলাদাভাবে প্রদর্শন করবে।

3. Explain `let`, `const`, and `var` keywords. When should you use each?

JavaScript-এ `let`, `const`, এবং `var` তিনটি কিওয়ার্ড ভেরিয়েবল ডিক্লেয়ার করতে ব্যবহৃত হয়। তবে এদের মধ্যে কিছু গুরুত্বপূর্ণ পার্থক্য রয়েছে যা বিভিন্ন পরিস্থিতিতে এদের ব্যবহার নির্ধারণ করে।

1. `var`

`var` হলো JavaScript-এর প্রাচীনতম কিওয়ার্ড যা ES5 এবং তার আগের সংস্করণে ব্যবহৃত হয়।

বৈশিষ্ট্য:

- Function Scope:** `var` শুধুমাত্র ফাংশন স্কোপকে অনুসরণ করে, অর্থাৎ এটি শুধুমাত্র একটি ফাংশনের ভিতরে থাকলে সেখানে সীমাবদ্ধ থাকে। কিন্তু যদি ফাংশনের বাইরে থাকে, তাহলে এটি **global scope** এ চলে যায়।
- Hoisting:** JavaScript কোড রান করার সময় সব `var` ভেরিয়েবল উপরের দিকে চলে যায় (hoist হয়), তবে তাদের মান হয় না। তাই কোডের শুরুতে `var` কে `undefined` হিসেবে পাওয়া যাবে।
- Redeclaration Allowed:** একই স্কোপে `var` দিয়ে বারবার ভেরিয়েবল ডিক্লেয়ার করা যায়, যা কিছু সমস্যার সূষ্টি করতে পারে।

কখন ব্যবহার করবেন:

- নতুন কোড লেখার সময় `var` এডানো উত্তম। এটি সাধারণত পুরনো কোডে বা compatibility নিশ্চিত করার জন্য ব্যবহৃত হয়।

2. `let`

`let` হলো ES6 এ প্রবর্তিত একটি কিওয়ার্ড যা ব্লক-স্কোপড ভেরিয়েবল ডিক্লেয়ার করতে ব্যবহৃত হয়।

বৈশিষ্ট্য:

- Block Scope:** `let` শুধুমাত্র যেখানে ডিক্লেয়ার করা হয়েছে সেই ব্লকের মধ্যেই সীমাবদ্ধ থাকে (যেমন `{}` এর মধ্যে)।
- No Hoisting (Temporal Dead Zone):** `let` এবং `const` এর ক্ষেত্রে hoisting হয়, তবে এটি **Temporal Dead Zone (TDZ)** এ থাকে যতক্ষণ না পর্যন্ত ভেরিয়েবলটি ডিক্লেয়ার করা হয়। অর্থাৎ, আগে এটি ব্যবহার করলে রেফারেন্স এর দেখায়।
- Redeclaration Not Allowed:** একই স্কোপে `let` দিয়ে আবার ডিক্লেয়ার করা যায় না, যা কোডকে আরও নিরাপদ করে।

কখন ব্যবহার করবেন:

- যখন ভেরিয়েবলটির মান পরিবর্তিত হবে, এবং ব্লক-স্কোপিং এর প্রয়োজন, তখন `let` ব্যবহার করা উত্তম।

উদাহরণ:

```
let count = 1;
count = 2; // অনুমোদিত
```

3. `const`

`const` হলো স্লক-স্কোপড ভেরিয়েবল ডিক্লেয়ার করার জন্য ব্যবহৃত যা স্থির বা অপরিবর্তনীয় (immutable) হতে হবে।

বৈশিষ্ট্য:

- Block Scope:** `const` ঠিক `let` এর মতো স্লক-স্কোপ অনুসরণ করে।
- No Hoisting (Temporal Dead Zone):** `const` ডিক্লেয়ার না করা পর্যন্ত এটি ব্যবহার করা যায় না।
- Immutable Reference:** `const` দিয়ে ঘোষিত ভেরিয়েবলের মান পুনরায় অ্যাসাইন করা যায় না। তবে, যদি এটি একটি অবজেক্ট বা array হয়, তার অভ্যন্তরীণ প্রপার্টিগুলো পরিবর্তন করা যেতে পারে।

কখন ব্যবহার করবেন:

- যখন কোনো মান স্থির থাকবে এবং পরিবর্তন করার প্রয়োজন নেই, তখন `const` ব্যবহার করা উচিত। এটি কোডকে নিরাপদ ও স্থিতিশীল করে তোলে।

উদাহরণ:

```
const age = 25;
// age = 30; // এই ক্ষেত্রে এরর হবে কারণ `age` পুনরায় অ্যাসাইন করা যাবে
না।

const person = { name: 'Rabbani' };
person.name = 'Rahman'; // এখানে কোন সমস্যা হবে না কারণ অবজেক্টের প্র
পার্টি পরিবর্তন করা যায়।
```

সংক্ষিপ্ত তুলনা

Keyword	Scope	Hoisting	Reassignment	Redeclaration
<code>var</code>	Function	Yes (undefined)	Yes	Yes
<code>let</code>	Block	Yes (TDZ)	Yes	No

const	Block	Yes (TDZ)	No	No
-------	-------	-----------	----	----

4. What is the `spread` operator, and how can it be used with arrays and objects?

Spread operator (যা `...` দিয়ে চিহ্নিত) হলো ES6 এ প্রবর্তিত একটি অপারেটর যা arrays এবং objects-এর উপাদানগুলোকে আলাদা করতে বা একত্র করতে ব্যবহৃত হয়। এটি কোডকে সংক্ষিপ্ত এবং সহজ করে তোলে।

Spread Operator এর ব্যবহার

1. Arrays এর সাথে Spread Operator

a. Array Merge বা Concatenation:

Spread operator ব্যবহার করে সহজেই দুটি বা তার বেশি array একত্রিত করা যায়।

```
const arr1 = [1, 2, 3];
const arr2 = [4, 5, 6];
const combined = [...arr1, ...arr2];

console.log(combined); // Output: [1, 2, 3, 4, 5, 6]
```

b. Array Cloning:

Spread operator ব্যবহার করে একটি array এর কপি তৈরি করা যায়। এটি মূল array এর কোন পরিবর্তন এড়াতে সাহায্য করে।

```
const original = [1, 2, 3];
const copy = [...original];
copy.push(4);

console.log(original); // Output: [1, 2, 3]
console.log(copy);    // Output: [1, 2, 3, 4]
```

c. Array এর মধ্যে নতুন উপাদান যুক্ত করা:

Spread operator ব্যবহার করে নতুন উপাদান যুক্ত করতে পারেন যেকোনো জায়গায়।

```

const numbers = [1, 2, 3];
const updated = [0, ...numbers, 4];

console.log(updated); // Output: [0, 1, 2, 3, 4]

```

2. Objects এর সাথে Spread Operator

a. Object Cloning:

Spread operator ব্যবহার করে সহজেই একটি object এর shallow copy তৈরি করা যায়।

```

const person = { name: 'Rabbani', age: 25 };
const copyPerson = { ...person };

console.log(copyPerson); // Output: { name: 'Rabbani', age: 25 }

```

b. Object Merging:

একাধিক object একত্রিত করার জন্য spread operator ব্যবহার করা হয়।

```

const details = { country: 'Bangladesh' };
const person = { name: 'Rabbani', age: 25, ...details };

console.log(person); // Output: { name: 'Rabbani', age: 25, cou
ntry: 'Bangladesh' }

```

c. Object এর মধ্যে নতুন বা পরিবর্তিত প্রপার্টি যুক্ত করা:

Object এর মধ্যে নতুন প্রপার্টি যুক্ত করার জন্য spread operator ব্যবহার করতে পারেন।

```

const person = { name: 'Rabbani', age: 25 };
const updatedPerson = { ...person, age: 26, city: 'Dhaka' };

console.log(updatedPerson); // Output: { name: 'Rabbani', age:
26, city: 'Dhaka' }

```

সংক্ষেপে Spread Operator এর সুবিধা

- **Array এবং Object একত্র করা** সহজ হয়।
- **Array এবং Object ক্লোন করা** দ্রুত এবং সরল হয়।
- নতুন উপাদান বা প্রপার্টি যোগ করতে বা পরিবর্তন করতে সুবিধা হয়।

Spread operator কোডের পড়া, লেখা এবং রক্ষণাবেক্ষণকে আরও সহজ করে তোলে।

- **Project: Merge two arrays using the spread operator.**

নীচে একটি প্রজেক্ট উদাহরণ দেওয়া হলো যেখানে দুটি array কে **spread operator** ব্যবহার করে একত্রিত করা হয়েছে।

Project: Merge Two Arrays Using Spread Operator

Step 1: Arrays তৈরি করা

প্রথমে দুটি আলাদা array তৈরি করা হবে, যা আমরা পরবর্তীতে একত্র করব।

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
```

Step 2: Spread Operator ব্যবহার করে Array একত্র করা

Spread operator (`...`) ব্যবহার করে `array1` এবং `array2` কে একত্র করা হবে।

```
const mergedArray = [...array1, ...array2];
```

Step 3: আউটপুট দেখানো

এখন `mergedArray` কনসোল-এ দেখালে দেখা যাবে, দুটি array একত্র হয়ে একটি নতুন array তৈরি হয়েছে।

```
console.log(mergedArray); // Output: [1, 2, 3, 4, 5, 6]
```

পুরো কোড একসাথে

```
const array1 = [1, 2, 3];
const array2 = [4, 5, 6];
```

```
const mergedArray = [...array1, ...array2];  
  
console.log(mergedArray); // Output: [1, 2, 3, 4, 5, 6]
```

ব্যাখ্যা

- `...array1` এবং `...array2` spread operator এর মাধ্যমে array1 এবং array2 এর সব উপাদান `mergedArray` তে যোগ করেছে।
- এই পদ্ধতিতে মূল array গুলো অপরিবর্তিত থাকে, এবং নতুন একটি array তৈরি হয় যা দুটি array-এর উপাদানগুলোকে একত্রিত করে।

এই প্রজেক্টটি arrays একত্র করার একটি সরল এবং কার্যকর উপায়।

5. How does template literals work in ES6, and how does it make string interpolation easier?

Template literals হলো JavaScript ES6 এর একটি ফিচার, যা ` (backticks) দিয়ে string তৈরি করে এবং সহজে string interpolation (string-এর মধ্যে ভেরিয়েবল বা এক্সপ্রেশন যুক্ত করা) করতে সাহায্য করে। এটি সাধারণ string এর তুলনায় আরও কার্যকর এবং পড়তে সহজ করে তোলে।

Template Literals এর সুবিধা

1. **String Interpolation:** `{}` ব্যবহার করে সহজেই ভেরিয়েবল বা এক্সপ্রেশনকে string-এর মধ্যে যুক্ত করা যায়।
2. **Multiline String:** নতুন লাইন তৈরি করার জন্য `\n` ব্যবহার করার প্রয়োজন নেই।
3. **Expressions:** `{}` এর মধ্যে যেকোনো জাভাস্ক্রিপ্ট এক্সপ্রেশন ব্যবহার করা যায়।

উদাহরণ: Template Literals এর মাধ্যমে String Interpolation

ES6 এর আগে, ভেরিয়েবলকে string এর মধ্যে যুক্ত করতে `+` অপারেটর ব্যবহার করতে হতো, যা পড়তে এবং লিখতে কিছুটা জটিল ছিল।

```

let name = 'Rabbani';
let age = 25;

// আগের পদ্ধতি
let message = 'My name is ' + name + ' and I am ' + age + ' years old.';
console.log(message); // Output: My name is Rabbani and I am 25 years old.

```

ES6 template literals দিয়ে এটি আরও সহজ এবং পরিষ্কারভাবে লেখা যায়:

```

let name = 'Rabbani';
let age = 25;

// Template literal ব্যবহার করে
let message = `My name is ${name} and I am ${age} years old.`;
console.log(message); // Output: My name is Rabbani and I am 25 years old.

```

Multiline String তৈরি করা

ES6 এর আগে, নতুন লাইন তৈরির জন্য `\n` বা string কনক্যাটেনেশনের প্রয়োজন হতো।

```

let multilineString = "This is line one.\nThis is line two.";
console.log(multilineString);

```

Template literals দিয়ে এটি সহজে করা যায়:

```

let multilineString = `This is line one.
This is line two.`;
console.log(multilineString);

```

Expressions ব্যবহার

`{}$` এর মধ্যে শুধু ভেরিয়েবল নয়, সরাসরি যেকোনো জাভাস্ক্রিপ্ট এক্সপ্রেশন ব্যবহার করা সম্ভব।

```

let a = 5;
let b = 10;

let result = `The sum of ${a} and ${b} is ${a + b}.`;
console.log(result); // Output: The sum of 5 and 10 is 15.

```

সারসংক্ষেপ

- **Template literals** ব্যবহার করে ভেরিয়েবল এবং এক্সপ্রেশন সহজেই string-এর মধ্যে যুক্ত করা যায়।
- Multiline string লেখা আরও সহজ হয়েছে।
- `{}$` এর মধ্যে যেকোনো জাভাস্ক্রিপ্ট এক্সপ্রেশন ব্যবহার করা যায়, যা কোডের সংক্ষিপ্ততা এবং পড়ার সহজতা বাড়ায়।

Template literals দিয়ে string interpolation অনেক সহজ এবং কার্যকরী হয়েছে, যা কোডের মান বাড়ায় এবং রক্ষণাবেক্ষণ সহজ করে।

6. What are `default parameters` in ES6, and how do they improve function flexibility?

Default parameters হলো ES6 এর একটি ফিচার, যা ফাংশনের প্যারামিটারগুলোকে একটি ডিফল্ট মান দিয়ে সেট করার সুযোগ দেয়। যদি ফাংশন কল করার সময় কোনো মান প্রদান করা না হয়, তাহলে সেই প্যারামিটার তার নির্ধারিত ডিফল্ট মানটি গ্রহণ করে। এটি ফাংশনগুলোকে আরও ফ্লেক্সিবল এবং রক্ষণাবেক্ষণের জন্য সহজ করে তোলে।

Default Parameters এর ব্যবহার

ফাংশন ডিফাইন করার সময় প্রত্যেক প্যারামিটারের পরে `=` চিহ্ন দিয়ে একটি ডিফল্ট মান সেট করা যায়। উদাহরণস্বরূপ:

```

function greet(name = 'Guest') {
    return `Hello, ${name}!`;
}

```

```
console.log(greet('Rabbani')); // Output: Hello, Rabbani!
console.log(greet());           // Output: Hello, Guest!
```

উপরের উদাহরণে, `greet` ফাংশনের `name` প্যারামিটারে `Guest` ডিফল্ট মান সেট করা হয়েছে। যদি ফাংশন কল করার সময় কোনো মান প্রদান করা না হয়, তাহলে `name` এর মান `Guest` হবে।

Default Parameters এর সুবিধা

- Optional Parameters:** Default parameters ব্যবহার করে একটি ফাংশনে প্রয়োজনমতো প্যারামিটারকে ঐচ্ছিকভাবে ব্যবহার করা যায়।
- Code Readability ও Maintenance উন্নত করে:** ডিফল্ট মান সরাসরি ফাংশন সংজ্ঞায় দেখা যায়, যা কোডকে আরও পরিষ্কার এবং পড়তে সহজ করে।
- Fallback Values:** যখন প্যারামিটার নির্দিষ্ট করা হয় না, তখন ফাংশন ক্রটি ছাড়াই কাজ করতে পারে, কারণ এটি ডিফল্ট মান ব্যবহার করবে।

একাধিক প্যারামিটারের জন্য Default Parameters ব্যবহার

একাধিক প্যারামিটারেও ডিফল্ট মান সেট করা যায়, এবং নির্দিষ্ট ক্রমে ডিফল্ট মান সেট করা হয়।

```
function calculatePrice(price, tax = 0.1, discount = 0) {
    return price + (price * tax) - discount;
}

console.log(calculatePrice(100));           // Output: 110
console.log(calculatePrice(100, 0.2));       // Output: 120
console.log(calculatePrice(100, 0.2, 10));   // Output: 110
```

এখানে, `tax` এবং `discount` এর জন্য ডিফল্ট মান সেট করা হয়েছে। যখন ফাংশন কল করার সময় নির্দিষ্ট মান প্রদান করা হয় না, তখন এগুলো তাদের ডিফল্ট মান গ্রহণ করে।

Expressions এবং Function Calls এর মাধ্যমে Default Parameter সেট করা

ডিফল্ট প্যারামিটার সরাসরি কোনো এক্সপ্রেশন বা ফাংশন কল থেকেও সেট করা যায়।

```
function getDefaultDiscount() {
    return 5;
}
```

```

function calculateTotal(price, discount = getDefaultDiscount())
{
    return price - discount;
}

console.log(calculateTotal(50));           // Output: 45
console.log(calculateTotal(50, 10));       // Output: 40

```

এখনে, `discount` প্যারামিটারটি `getDefaultDiscount()` ফাংশন থেকে একটি ডিফল্ট মান নেয়।

সংক্ষেপে

- **Default parameters** ব্যবহার করে ফাংশনের প্যারামিটারগুলোকে ডিফল্ট মান দেওয়া যায়, যা ফাংশনকে আরও ফ্লেক্সিবল এবং রক্ষণাবেক্ষণের জন্য সহজ করে।
- এটি ফাংশন কলের সময় ঐচ্ছিক প্যারামিটার যুক্ত করা সহজ করে এবং কোডকে আরও পরিষ্কার ও সংক্ষিপ্ত রাখে।
- Expressions, ফাংশন কল, বা সরাসরি মান দিয়ে ডিফল্ট প্যারামিটার সেট করা যায়।

Default parameters ফাংশনের ব্যবহারের জটিলতা হ্রাস করে এবং কোডে ক্রিটি এড়াতে সাহায্য করে।

7. Explain what `Map` and `Set` are in ES6 and how they differ from objects and arrays.

Map এবং **Set** ES6-এ নতুন ডেটা স্ট্রাকচার, যা **Objects** এবং **Arrays** এর তুলনায় কিছু গুরুত্বপূর্ণ পার্থক্য এবং সুবিধা প্রদান করে।

1. Map (ES6)

Map হলো একটি **key-value pair** ডেটা স্ট্রাকচার, যেখানে প্রতিটি `key` এর সাথে একটি `value` যুক্ত থাকে। এটি ES6-এ `Map` কিওয়ার্ডের মাধ্যমে তৈরি করা হয়।

বৈশিষ্ট্যসমূহ:

- **Key can be any type:** `Map` এ key হিসেবে primitive types (যেমন string, number) বা objects (যেমন array, function) যেকোনো ধরনের ডাটা ব্যবহার করা যেতে পারে।
- **Insertion order:** `Map` এ key-value pairs সন্নিবেশের (insertion) ক্রম অনুসারে সরাসরি loop করা যায়।
- **Size property:** `Map` এর একটি `size` প্রপার্টি থাকে, যা পুরো map-এর মোট এলেম্বির সংখ্যা বলে।

উদাহরণ:

```
let map = new Map();

map.set('name', 'Rabbani');
map.set('age', 25);
map.set(1, 'one');
map.set(true, 'boolean');

console.log(map.get('name')); // Output: Rabbani
console.log(map.size); // Output: 4
```

2. Set (ES6)

Set হলো একটি ডেটা স্ট্রাকচার যা শুধুমাত্র ইউনিক (unique) মান সংরক্ষণ করে। এটি ডুপ্লিকেট মানগুলো এড়িয়ে চলে এবং শুধুমাত্র একবার প্রতি মান অন্তর্ভুক্ত করে।

বৈশিষ্ট্যসমূহ:

- **Unique values:** `Set` কোনো ডুপ্লিকেট মান রাখে না। একই মান পুনরায় যোগ করার চেষ্টা করলে তা যুক্ত হবে না।
- **Any type of value:** `Set` এ যেকোনো ধরনের মান (primitive types বা objects) থাকতে পারে।
- **Insertion order:** `Set` এর মধ্যে মানগুলো সন্নিবেশের (insertion) ক্রম অনুযায়ী থাকে এবং আপনি সেগুলো লুপের মাধ্যমে অ্যাক্সেস করতে পারেন।

উদাহরণ:

```

let set = new Set();

set.add(1);
set.add(2);
set.add(3);
set.add(2); // Duplicate, will not be added

console.log(set); // Output: Set { 1, 2, 3 }
console.log(set.size); // Output: 3

```

Objects এবং Arrays এর তুলনায় Map এবং Set এর পার্থক্য

Feature	Map	Set	Object	Array
Key Type	Any type (primitive or object)	N/A	Only strings or symbols (as keys)	N/A
Value Type	Any type (primitive or object)	Any type (primitive or object)	Any type (but only keys can be strings)	Any type
Duplicates	Allows duplicates in values	Does not allow duplicates in values	Allows duplicate values (keys are unique)	Allows duplicates
Order	Maintains insertion order	Maintains insertion order	No guaranteed order of keys	Maintains insertion order
Size	<code>size</code> property for length	<code>size</code> property for length	No built-in size property	<code>length</code> property
Iteration	Can iterate over keys, values, or entries	Can iterate over unique values	Can iterate only over keys	Can iterate over indexes or values
Use Case	Storing key-value pairs with flexibility	Storing unique values	Storing key-value pairs with string keys	Storing ordered collection of elements

Map এবং Set এর সুবিধা

- **Map:** যখন আপনাকে একটি ডেটা স্ট্রাকচারে key-value pair রাখতে হয় এবং key হিসেবে যেকোনো ধরনের ডেটা (string, object, etc.) ব্যবহার করতে হয়, তখন **Map** সবচেয়ে কার্যকরী।
- **Set:** যখন আপনাকে ইউনিক ভ্যালুসমূহ সংরক্ষণ করতে হয় এবং ডুপ্লিকেট এড়াতে হয়, তখন **Set** ব্যবহৃত হয়। এটি মূলত একটি সমষ্টি বা **集合** হিসাবে কাজ করে।

সারাংশ

- **Map:** key-value পেয়ার হিসেবে ডেটা সংরক্ষণ করে এবং key হিসেবে যেকোনো ধরনের ডেটা গ্রহণ করতে পারে। এটি ডেটা সংরক্ষণ ও পরিচালনার জন্য আরও উন্নত ফিচার প্রদান করে।
- **Set:** ইউনিক ভ্যালু সংরক্ষণ করে এবং ডুপ্লিকেট মানগুলো এড়িয়ে চলে।
- **Object:** সাধারণত string বা symbol ধরনের key সহ key-value pair সংরক্ষণ করতে ব্যবহৃত হয়।
- **Array:** ordered collection হিসেবে একই ধরনের উপাদান সংরক্ষণ করতে ব্যবহৃত হয়।
- **Project: Create a simple dictionary with Map to store key-value pairs.**

এখানে একটি **Dictionary** তৈরি করার প্রজেক্ট দেওয়া হলো, যেখানে **Map** ব্যবহার করা হয়েছে key-value pair সংরক্ষণ করার জন্য।

Project: Create a Simple Dictionary with Map

Step 1: Map তৈরি করা

প্রথমে একটি নতুন **Map** তৈরি করব, যেখানে **word** (key) এবং **meaning** (value) সংরক্ষণ করা হবে।

Step 2: Key-Value Pair যোগ করা

পরে **set** মেথড ব্যবহার করে dictionary তে বিভিন্ন শব্দ এবং তাদের অর্থ যোগ করব।

Step 3: Value পড়া

কোনো একটি শব্দের মান পড়তে **get** মেথড ব্যবহার করা হবে।

Step 4: Dictionary প্রদর্শন করা

আমরা dictionary এর সব key-value pair প্রদর্শন করব।

কোড:

```
// Step 1: Create a new Map to act as a dictionary
let dictionary = new Map();

// Step 2: Add key-value pairs (word and meaning)
dictionary.set('apple', 'A round fruit with red or green skin and a whitish interior.');
dictionary.set('banana', 'A long, curved fruit with a yellow skin and soft, sweet, white flesh.');
dictionary.set('cat', 'A small domesticated carnivorous mammal with fur, a short snout, and retractable claws.');
dictionary.set('dog', 'A domesticated carnivorous mammal that is typically kept as a pet or for work or field sports.');

// Step 3: Access meanings using the get() method
console.log(dictionary.get('apple')); // Output: A round fruit with red or green skin and a whitish interior.
console.log(dictionary.get('dog')); // Output: A domesticated carnivorous mammal that is typically kept as a pet or for work or field sports.

// Step 4: Display all words and their meanings
console.log('Dictionary Entries:');
for (let [word, meaning] of dictionary) {
    console.log(` ${word}: ${meaning}`);
}
```

Output:

```
A round fruit with red or green skin and a whitish interior.
A domesticated carnivorous mammal that is typically kept as a pet or for work or field sports.
Dictionary Entries:
apple: A round fruit with red or green skin and a whitish interior.
```

```
banana: A long, curved fruit with a yellow skin and soft, sweet, white flesh.  
cat: A small domesticated carnivorous mammal with fur, a short snout, and retractable claws.  
dog: A domesticated carnivorous mammal that is typically kept as a pet or for work or field sports.
```

ব্যাখ্যা:

1. `Map` তৈরি: `dictionary` নামের একটি নতুন `Map` তৈরি করা হয়েছে, যাতে key-value pair (word-meaning) সংরক্ষণ করা হবে।
2. `set` মেথড: প্রতিটি শব্দ এবং তার মান `set` মেথডের মাধ্যমে `dictionary` তে যোগ করা হয়েছে।
3. `get` মেথড: কোনো একটি শব্দের মান বের করার জন্য `get()` মেথড ব্যবহার করা হয়েছে।
4. **Iteration:** `Map` এর সব key-value pair দেখানোর জন্য `for...of` লুপ ব্যবহার করা হয়েছে।

এটি একটি সিম্পল `dictionary` প্রজেক্ট, যেখানে শব্দ এবং তাদের অর্থ সংরক্ষণ এবং পুনরুদ্ধারের জন্য `Map` ডেটা স্ট্রাকচার ব্যবহার করা হয়েছে।

8. What is the purpose of the `for...of` loop, and how is it different from `for...in` ?

`for...of` এবং `for...in` উভয়ই JavaScript এর লুপিং স্ট্রাকচার, তবে এগুলোর উদ্দেশ্য এবং ব্যবহারের মধ্যে কিছু গুরুত্বপূর্ণ পার্থক্য রয়েছে।

1. `for...of` লুপ

`for...of` লুপ ব্যবহার করা হয় iterable object গুলোর (যেমন arrays, strings, maps, sets ইত্যাদি) উপাদানগুলির উপর লুপ চালানোর জন্য।

বৈশিষ্ট্য:

- **Iterable Objects:** এটি iterable objects (যেমন `Array`, `String`, `Map`, `Set` ইত্যাদি) এর উপাদানগুলির উপর কাজ করে।

- **Value Access:** এটি সরাসরি iterable object এর **value** গুলোর উপর লুপ করে, অর্থাৎ প্রতিটি উপাদান একে একে পাওয়া যায়।

উদাহরণ:

```
// Array এর জন্য `for...of` ব্যবহার
let fruits = ['apple', 'banana', 'cherry'];

for (let fruit of fruits) {
    console.log(fruit); // Output: apple, banana, cherry
}
```

এখানে, `fruits` array এর প্রতিটি উপাদান `fruit` ভেরিয়েবলে আসবে এবং সেটা কনসোল-এ প্রিন্ট হবে।

2. `for...in` লুপ

`for...in` লুপ ব্যবহার করা হয় object এর properties বা arrays এর indexes-এর উপর লুপ চালানোর জন্য।

বৈশিষ্ট্য:

- **Objects (Properties):** এটি সাধারণত object এর **properties** বা **keys** এর উপর লুপ চালানোর জন্য ব্যবহৃত হয়।
- **Index or Key Access:** এটি object বা array এর **key/index** গুলোর উপর লুপ করে, না যে তাদের **value** এর উপর।

উদাহরণ:

```
// Object এর জন্য `for...in` ব্যবহার
let person = {
    name: 'Rabbani',
    age: 25,
    country: 'Bangladesh'
};

for (let key in person) {
    console.log(key + ': ' + person[key]); // Output: name: Rab
```

```

    bani, age: 25, country: Bangladesh
}

```

এখনে, `person` object এর প্রতিটি **key** (`name`, `age`, `country`) ধরে নিয়ে তার মান (`person[key]`) কনসোল-এ প্রিন্ট হচ্ছে।

পার্থক্য: `for...of` এবং `for...in`

Feature	<code>for...of</code>	<code>for...in</code>
Usage	Iterates over values of an iterable (array, string, map, set, etc.)	Iterates over keys/indexes of an object or array
Works with	Iterable objects (arrays, strings, sets, maps, etc.)	Objects (properties) or arrays (indexes)
Access	Accesses the values directly	Accesses the keys/indexes
Best for	Iterating over values in collections like arrays and strings	Iterating over properties of an object or array indexes
Example	<code>for (let item of array)</code>	<code>for (let key in object)</code>

উদাহরণে পার্থক্য:

1. `for...of` এবং Array:

```

let arr = [10, 20, 30];

// `for...of` দিয়ে Array এর value গুলো প্রিন্ট
for (let num of arr) {
    console.log(num); // Output: 10, 20, 30
}

```

2. `for...in` এবং Array:

```

let arr = [10, 20, 30];

// `for...in` দিয়ে Array এর index গুলো প্রিন্ট
for (let index in arr) {

```

```
        console.log(index); // Output: 0, 1, 2 (index values)
    }
```

সারসংক্ষেপ:

- `for...of` লুপ **iterable objects** (যেমন array, set, map) এর **values** এর উপর কাজ করে এবং প্রত্যেকটি মান একে একে access করতে দেয়।
- `for...in` লুপ **object** এর **properties/keys** বা **array** এর **indexes** এর উপর কাজ করে।

এভাবে, যখন আপনি একটি **object** বা **array** এর property বা index নিয়ে কাজ করতে চান, তখন `for...in` ব্যবহার করুন। কিন্তু যদি আপনি **array** বা **string** এর উপাদানগুলোর মান নিয়ে কাজ করতে চান, তাহলে `for...of` ব্যবহার করা উচিত।

9. Explain the concept of **Promise** in JavaScript. How do you use `.then()` and `.catch()` ?

Promise হলো একটি **asynchronous** অপারেশন (যেমন API কল বা ডেটা লোড) পরিচালনা করার জন্য JavaScript-এ ব্যবহৃত একটি অবজেক্ট। এটি একটি ভবিষ্যতের মানের প্রতিনিধিত্ব করে, যা আপাতত উপলব্ধ না হলেও পরে আসবে (success বা failure)।

Promise তিনিটি অবস্থায় থাকতে পারে:

1. **Pending**: Promise এখনো সম্পূর্ণ হয়নি (অর্থাৎ অপারেশনটি এখনও চলছে)।
2. **Fulfilled**: Promise সফলভাবে সম্পন্ন হয়েছে এবং একটি মান প্রদান করেছে।
3. **Rejected**: Promise ব্যর্থ হয়েছে এবং একটি ত্রুটি (error) বা কারণ প্রদান করেছে।

Promise-এর তৈরি করা:

```
let promise = new Promise((resolve, reject) => {
    let success = true; // এইটা পরীক্ষার জন্য কৃতিমভাবে দেয়া
```

```

        if (success) {
            resolve('Operation was successful'); // সফল হলে resolve করা করা
        } else {
            reject('Operation failed'); // ব্যর্থ হলে reject করা করা
        }
    });

```

এখানে, `resolve()` ফাংশন Promise সফলভাবে পূর্ণ হলে কল করা হয়, এবং `reject()` ফাংশন ব্যর্থ হলে কল করা হয়।

`.then()` এবং `.catch()` ব্যবহার:

Promise এর উপর `.then()` এবং `.catch()` মেথড ব্যবহার করা হয়।

- `.then()`: এটি Promise সফলভাবে পূর্ণ হলে কল হয় এবং প্রাপ্তি মান (value) দেয়।
- `.catch()`: এটি Promise ব্যর্থ হলে (rejected) কল হয় এবং ত্রুটি (error) হ্যান্ডেল করতে ব্যবহৃত হয়।

উদাহরণ:

```

let promise = new Promise((resolve, reject) => {
    let success = true; // কৃতিমভাবে সফল হওয়ার জন্য

    if (success) {
        resolve('Operation was successful');
    } else {
        reject('Operation failed');
    }
});

promise
    .then((result) => {
        console.log(result); // যদি Promise সফল হয়, তখন 'Operation was successful' প্রিন্ট হবে।
    })
    .catch((error) => {

```

```

        console.log(error); // যদি Promise ব্যর্থ হয়, তখন 'Operation failed' প্রিন্ট হবে।
    });

```

.then() মেথড:

- `.then()` মেথড Promise সফল হলে কল হয়। এটি দুটি আর্গুমেন্ট গ্রহণ করতে পারে: একটি সফল ফলাফল (value) এবং একটি ত্রুটি (error)।
- আপনি `.then()` চেইনও করতে পারেন, অর্থাৎ একাধিক `.then()` ব্যবহার করতে পারেন যাতে প্রতিটি পরবর্তী `.then()` পেছনের Promise এর ফলাফল গ্রহণ করে।

.then() চেইনিং:

```

let promise = new Promise((resolve, reject) => {
    resolve(5);
});

promise
    .then(result => {
        console.log(result); // Output: 5
        return result * 2;
    })
    .then(newResult => {
        console.log(newResult); // Output: 10
        return newResult + 3;
    })
    .then(finalResult => {
        console.log(finalResult); // Output: 13
    });

```

এখানে, প্রতিটি `.then()` পরবর্তী Promise এর `result` নিয়ে কাজ করছে এবং সেই অনুযায়ী নতুন মান ফেরত দিচ্ছে।

.catch() মেথড:

- `.catch()` মেথড Promise ব্যর্থ হলে কল হয়। এটি সাধারণত ত্রুটি হ্যান্ডলিংয়ের জন্য ব্যবহৃত হয়। যদি Promise এর মধ্যে কোনো error ঘটে, তবে `.catch()` সেই error গ্রহণ করবে।

.catch() ব্যবহার:

```
let promise = new Promise((resolve, reject) => {
    reject('Something went wrong!');
});

promise
    .then(result => {
        console.log(result); // এটি কল হবে না, কারণ Promise reject হয়েছে।
    })
    .catch(error => {
        console.log(error); // Output: Something went wrong!
});
```

finally() মেথড:

`finally()` মেথড Promise এর যে কোনো অবস্থা (fulfilled বা rejected) শেষ হওয়ার পর কল হয়। এটি এমন কোনো কোডের জন্য ব্যবহার করা যেতে পারে যা Promise এর পর সব সময় চলবে, যেমন ক্লিনআপ কাজ বা লগিং।

```
let promise = new Promise((resolve, reject) => {
    let success = true;

    if (success) {
        resolve('Success');
    } else {
        reject('Failure');
    }
});

promise
    .then(result => {
        console.log(result); // Output: Success
    })
    .catch(error => {
```

```

        console.log(error); // Output: Failure
    })
    .finally(() => {
        console.log('This runs no matter what'); // Always run
    });
}

```

সারাংশ:

- **Promise** হলো একটি অবজেক্ট যা ভবিষ্যতে একটি মান প্রদান করবে।
- `.then()` Promise সফল হলে কল হয় এবং সেই ফলাফল ব্যবহৃত হয়।
- `.catch()` Promise ব্যর্থ হলে কল হয় এবং ত্রুটি (error) হ্যান্ডেল করে।
- `.finally()` Promise যে অবস্থাই শেষ হোক, এটি চূড়ান্তভাবে চলে এবং সাধারণত ক্লিনআপ কাজের জন্য ব্যবহৃত হয়।

Promise একটি শক্তিশালী টুল যা asynchronous কোডকে সহজ এবং পঠনযোগ্য করে তোলে।

- **Project: Create a promise that resolves after 2 seconds and logs a message.**

এখানে একটি **Promise** তৈরি করা হবে যা 2 সেকেন্ড পর সফলভাবে resolve হবে এবং একটি বার্তা কনসোলে লগ করবে।

Project: Create a Promise that Resolves After 2 Seconds and Logs a Message

কোড:

```

// Step 1: Create a Promise that resolves after 2 seconds
let promise = new Promise((resolve, reject) => {
    setTimeout(() => {
        resolve('Promise resolved after 2 seconds!');
    }, 2000); // 2000 milliseconds = 2 seconds
});

// Step 2: Using .then() to log the message when the Promise resolves

```

```

promise
  .then(result => {
    console.log(result); // Output: Promise resolved after
2 seconds!
  })
  .catch(error => {
    console.log(error); // In case of an error (won't happen here)
  });

```

ব্যাখ্যা:

- Promise তৈরি:** একটি নতুন Promise তৈরি করা হয়েছে যেখানে `setTimeout` ব্যবহার করা হয়েছে 2 সেকেন্ড পর **resolve** করার জন্য।
- .then()** মেথড: Promise সফলভাবে resolve হওয়ার পর `.then()` মেথডে দেওয়া কলব্যাক ফাংশনটি চালু হয় এবং কনসোলে বার্তাটি লগ করে।

আউটপুট:

2 সেকেন্ড পর কনসোলে এই বার্তাটি দেখা যাবে:

```
Promise resolved after 2 seconds!
```

এটি একটি সাধারণ প্রজেক্ট যা Promise ব্যবহারের মাধ্যমে `setTimeout` এর সঙ্গে asynchronous কোড চালানোর উপায় দেখাচ্ছে।

10. What is `async/await`, and how does it work with Promises?

`async/await` হলো JavaScript-এ **Promises**-এর উপর ভিত্তি করে asynchronous কোড লেখার একটি নতুন এবং আরও পরিষ্কার পদ্ধতি। এটি ES2017 (ES8) সংস্করণে চালু হয়েছে এবং এটি asynchronous কোডকে আরও সহজ, পঠনযোগ্য এবং সিঙ্ক্রোনাস কোডের মতো করে তোলে।

1. `async` ফাংশন:

`async` একটি কীওয়ার্ড যা একটি ফাংশনের আগে ব্যবহৃত হয়, যার ফলে ঐ ফাংশনটি একটি **Promise** ফেরত দেয়, যদিও আপনি সোজা মান ফেরত দিতে চাইলে। অর্থাৎ, একটি `async` ফাংশন কখনোই সাধারণ মান (value) ফেরত দেয় না, বরং একটি **Promise** ফেরত দেয় যা `fulfil` (resolve) বা `reject` হতে পারে।

উদাহরণ:

```
async function myFunction() {
    return 'Hello, World!';
}

myFunction().then(result => {
    console.log(result); // Output: Hello, World!
});
```

এখানে `myFunction` একটি `async` ফাংশন, তাই এটি একটি Promise ফেরত দেয়। যেহেতু এটি কোনো ক্রটি ছাড়াই `return` করেছে, তাই এটি একটি resolved Promise ফেরত দেবে, যার মান হবে `'Hello, World!'`।

2. `await` কীওয়ার্ড:

`await` শুধু `async` ফাংশনের ভিতরেই ব্যবহার করা যায় এবং এটি **Promise** এর ফলাফল (resolved value) পাওয়ার জন্য অপেক্ষা করে। `await` Promise এর কাজ শেষ হওয়া পর্যন্ত ফাংশনটি থামিয়ে রাখে এবং তার পরপরই পরবর্তী কোডটি চালায়।

`await` ব্যবহার করলে, asynchronous কোডকে সিঙ্ক্রোনাস কোডের মতো সহজে লেখা যায়।

উদাহরণ:

```
async function myFunction() {
    let result = await Promise.resolve('Hello, World!');
    console.log(result); // Output: Hello, World!
}

myFunction();
```

এখানে `await` Promise এর resolved value `"Hello, World!"` পাওয়ার জন্য অপেক্ষা করে এবং তার পর `console.log(result)` কল হয়।

3. `async/await` ব্যবহার করে Promise-এর সাথে কাজ করা:

`async/await` Promise এর সঙ্গে কাজ করার ক্ষেত্রে অনেক সহজ এবং সুন্দর উপায় প্রদান করে। আপনি `await` ব্যবহার করে Promise এর ফলাফল পেতে পারেন এবং `try...catch` লক দিয়ে ত্রুটি হ্যান্ডলিংও করতে পারেন।

উদাহরণ:

```
// একটি Promise তৈরি করা
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Data fetched successfully!');
        }, 2000);
    });
}

// async function তৈরি করা
async function fetchDataAsync() {
    try {
        // await ব্যবহার করে Promise এর ফলাফল পাওয়া
        let data = await fetchData();
        console.log(data); // Output: Data fetched successfully!
    } catch (error) {
        console.log('Error:', error);
    }
}

fetchDataAsync();
```

এখানে:

- `fetchData` একটি Promise ফেরত দেয়।

- `fetchDataAsync` একটি `async` ফাংশন যেখানে `await` ব্যবহার করে `fetchData()` Promise এর ফলাফল আসার জন্য অপেক্ষা করা হয়।
- `try...catch` রুক ব্যবহার করে ত্রুটির ক্ষেত্রে error handle করা হয়েছে।

তথ্যপূর্ণ কিছু পয়েন্ট:

- `async` ফাংশন কখনোই Promise ফেরত দেয়। আপনি যদি একটি সাধারণ মান return করেন, তবে সেটা স্বয়ংক্রিয়ভাবে একটি Promise এ রূপান্তরিত হয়ে যাবে।
- `await` শুধুমাত্র `async` ফাংশনের ভিতরেই ব্যবহার করা যেতে পারে।
- যদি Promise reject হয়, তবে `await` সেই error (exception) উঠিয়ে আনবে, এবং আপনি `try...catch` রুক দিয়ে সেগুলো হ্যান্ডল করতে পারেন।

`async/await` vs `.then()` / `.catch()`

`async/await` এবং `.then()` / `.catch()` দুইটাই asynchronous কোড পরিচালনার জন্য ব্যবহৃত হলেও, `async/await` কোডকে আরো সোজা এবং পড়তে সহজ করে তোলে।

উদাহরণ: `.then()` / `.catch()`:

```
function fetchData() {
    return new Promise((resolve, reject) => {
        setTimeout(() => {
            resolve('Data fetched successfully!');
        }, 2000);
    });
}

fetchData()
    .then(result => {
        console.log(result); // Output: Data fetched successfully!
    })
    .catch(error => {
        console.log('Error:', error);
    });

```

উদাহরণ: `async/await` :

```
async function fetchDataAsync() {  
    try {  
        let data = await fetchData();  
        console.log(data); // Output: Data fetched successfull  
    }  
    catch (error) {  
        console.log('Error:', error);  
    }  
}  
  
fetchDataAsync();
```

সারাংশ:

- `async/await` হল `Promise` এর উপর ভিত্তি করে লেখা asynchronous কোডের একটি সরলীকৃত এবং পঠনযোগ্য পদ্ধতি।
- `async` একটি ফাংশনকে asynchronous করে তোলে, যা একটি `Promise` ফেরত দেয়।
- `await` `Promise`-এর ফলাফল পেতে ব্যবহৃত হয় এবং এটি `Promise` এর `resolve` হওয়ার জন্য অপেক্ষা করে।
- `async/await` ব্যবহার করে asynchronous কোড লেখা প্রথমে সিঙ্ক্রোনাস কোডের মতো মনে হয়, যা কোডিংকে আরও সহজ ও পরিষ্কার করে।

>> DOM (8 questions)

1. What is the DOM, and how does JavaScript interact with it?

DOM (Document Object Model) একটি ব্রাউজারের একটি API বা ইন্টারফেস, যা একটি HTML বা XML ড্রুমেন্টকে একটি গাছের (tree) আকারে রূপান্তরিত করে। এই গাছের প্রতিটি নোড বা অংশ একটি HTML ট্যাগ বা একটি টেক্সট বা এটির বৈশিষ্ট্যগুলির প্রতিনিধিত্ব করে। অর্থাৎ, DOM মূলত

HTML ডকুমেন্টের একটি কাঠামো যা ব্রাউজারকে এটি বুঝতে এবং ব্যবহারকারীর সাথে ইন্টারঅ্যাক্ষন করতে সাহায্য করে।

JavaScript কীভাবে DOM-এর সাথে কাজ করে?

JavaScript DOM-এর মাধ্যমে HTML এর বিভিন্ন অংশকে নিয়ন্ত্রণ করতে পারে। JavaScript-এর মাধ্যমে আমরা DOM-এর নোডগুলিতে বিভিন্ন ক্রিয়াকলাপ পরিচালনা করতে পারি, যেমন:

1. **নোড সিলেক্ট করা** - আমরা DOM থেকে স্পেসিফিক এলিমেন্টগুলো খুঁজে পেতে পারি, যেমন `document.getElementById()`, `document.querySelector()` ইত্যাদি মেথড ব্যবহার করে।
2. **Content পরিবর্তন** - JavaScript দিয়ে DOM-এর মধ্যে HTML বা টেক্সট পরিবর্তন করতে পারি। যেমন, `innerHTML` বা `textContent` প্রপার্টি ব্যবহার করে।
3. **স্টাইল পরিবর্তন** - `style` প্রপার্টি ব্যবহার করে CSS পরিবর্তন করা যায়, যেমন একটি বাটনের রঙ বা আকার পরিবর্তন করা।
4. **নতুন এলিমেন্ট যোগ** - JavaScript দিয়ে নতুন HTML এলিমেন্ট তৈরি করে DOM-এ যোগ করা যায়, যেমন `createElement()` এবং `appendChild()` মেথডের মাধ্যমে।
5. **ইভেন্ট হ্যান্ডলিং** - ব্যবহারকারীর সাথে ইন্টারঅ্যাক্ষন করার জন্য ইভেন্ট অ্যাড করা যায়, যেমন `click`, `mouseover`, ইত্যাদি। `addEventListener()` মেথড ব্যবহার করে বিভিন্ন ইভেন্ট যোগ করা হয়।

উদাহরণ

একটি বাটনের ক্লিক ইভেন্টে কিছু টেক্সট পরিবর্তন করার উদাহরণ:

```
<!DOCTYPE html>
<html lang="bn">
<head>
    <meta charset="UTF-8">
    <title>DOM Example</title>
</head>
<body>
    <p id="text">এটি একটি টেক্সট।</p>
    <button onclick="changeText()">টেক্সট পরিবর্তন করুন</button>

    <script>
        function changeText() {
            document.getElementById("text").textContent = "টেক্সট পরিবর্তন করা হচ্ছে।"
        }
    </script>
</body>
</html>
```

```

    ন হয়েছে!" ;
}
</script>
</body>
</html>

```

এখনে বাটনে ক্লিক করার মাধ্যমে `changeText()` ফাংশনটি কল হবে এবং DOM-এ থাকা `<p>` ট্যাগের টেক্স্ট পরিবর্তন করবে।

2. Explain how to select elements in the DOM using methods like `getElementById`, `querySelector`, and `querySelectorAll`.

DOM-এ থাকা এলিমেন্টগুলো JavaScript দিয়ে নির্বাচন করার জন্য বিভিন্ন মেথড রয়েছে। এর মধ্যে `getElementById`, `querySelector`, এবং `querySelectorAll` সবচেয়ে বেশি ব্যবহৃত হয়। এগুলোর প্রতিটি মেথডের কাজ এবং ব্যবহার নিচে ব্যাখ্যা করা হলো:

1. `getElementById()`

এই মেথডটি ID-সহ একটি নির্দিষ্ট এলিমেন্ট নির্বাচন করার জন্য ব্যবহৃত হয়। যেহেতু ID একটি ডকুমেন্টে ইউনিক হয়, তাই এটি একটি মাত্র এলিমেন্টকে রিটার্ন করে।

```
let element = document.getElementById("myId");
```

ব্যবহার: এই মেথডটি HTML ডকুমেন্ট থেকে সেই এলিমেন্টটি রিটার্ন করবে যার ID `myId`। উদাহরণ:

```

<p id="myId">Hello, World!</p>

<script>
  let element = document.getElementById("myId");
  console.log(element.textContent); // "Hello, World!"
</script>

```

2. `querySelector()`

`querySelector` CSS সিলেক্টরের মতো একটি সিলেক্টর ব্যবহার করে প্রথম মিলে যাওয়া এলিমেন্টটি নির্বাচন করে। এটি ID, class, tag বা অন্যান্য CSS সিলেক্টরের মাধ্যমে এলিমেন্ট খুঁজে বের করতে পারে।

```
let element = document.querySelector(".myClass");
```

ব্যবহার: এটি `.myClass` নামক প্রথম মিলে যাওয়া এলিমেন্টকে রিটার্ন করবে। উদাহরণ:

```
<p class="myClass">This is the first paragraph.</p>
<p class="myClass">This is the second paragraph.</p>

<script>
  let element = document.querySelector(".myClass");
  console.log(element.textContent); // "This is the first para
graph."
</script>
```

3. `querySelectorAll()`

`querySelectorAll` CSS সিলেক্টর অনুযায়ী সকল মিলে যাওয়া এলিমেন্টকে নির্বাচন করে এবং একটি NodeList রিটার্ন করে, যা একটি সাদৃশ্যপূর্ণ অ্যারে।

```
let elements = document.querySelectorAll(".myClass");
```

ব্যবহার: এটি `.myClass` নামক সকল মিলে যাওয়া এলিমেন্টকে নির্বাচন করবে। উদাহরণ:

```
<p class="myClass">This is the first paragraph.</p>
<p class="myClass">This is the second paragraph.</p>

<script>
  let elements = document.querySelectorAll(".myClass");
  elements.forEach(element => console.log(element.textContent));
  // Output:
  // "This is the first paragraph."
```

```
// "This is the second paragraph."  
</script>
```

সারসংক্ষেপ

- `getElementById` : নির্দিষ্ট ID সহ একটি এলিমেন্টকে নির্বাচন করে।
- `querySelector` : CSS সিলেক্টর দিয়ে প্রথম মিলে যাওয়া এলিমেন্ট নির্বাচন করে।
- `querySelectorAll` : CSS সিলেক্টর দিয়ে সকল মিলে যাওয়া এলিমেন্ট নির্বাচন করে এবং একটি NodeList রিটার্ন করে।

এগুলোর মাধ্যমে JavaScript ব্যবহার করে DOM-এ থাকা এলিমেন্টগুলোকে সহজেই নিয়ন্ত্রণ করা যায়।

Project: Create a webpage with a button that changes the color of a `div` when clicked.

এখানে একটি সাধারণ প্রজেক্ট দেওয়া হলো যেখানে একটি ওয়েবপেজে একটি বোতাম থাকবে যা ক্লিক করলে একটি `div` এলিমেন্টের রং পরিবর্তিত হবে।

HTML এবং JavaScript কোড

এই উদাহরণে, আমরা একটি `div` এলিমেন্ট এবং একটি বোতাম ব্যবহার করেছি। বোতামে ক্লিক করলে JavaScript-এর মাধ্যমে `div`-এর ব্যাকগ্রাউন্ড কালার পরিবর্তন হবে।

```
<!DOCTYPE html>  
<html lang="bn">  
<head>  
  <meta charset="UTF-8">  
  <meta name="viewport" content="width=device-width, initial-scale=1.0">  
  <title>Color Change Project</title>
```

```

<style>
    /* স্টাইলিং */
    #colorBox {
        width: 200px;
        height: 200px;
        background-color: lightblue;
        margin: 20px auto;
        border: 2px solid #000;
    }

    #changeColorButton {
        display: block;
        margin: 20px auto;
        padding: 10px 20px;
        font-size: 16px;
        cursor: pointer;
    }
</style>
</head>
<body>

    <div id="colorBox"></div>
    <button id="changeColorButton">রং পরিবর্তন করুন</button>

    <script>
        // রং পরিবর্তনের ফাংশন
        function changeColor() {
            const colorBox = document.getElementById("colorBox");
            const colors = ["lightblue", "lightgreen", "lightcoral",
            "lightsalmon", "lightpink"];

            // একটি র্যান্ডম রং নির্বাচন
            const randomColor = colors[Math.floor(Math.random() * colors.length)];

            colorBox.style.backgroundColor = randomColor;
        }
    </script>

```

```

    }

    // বোতামে ইভেন্ট অ্যাড করা
    document.getElementById("changeColorButton").addEventListener("click", changeColor);
</script>

</body>
</html>

```

কীভাবে এটি কাজ করে

- HTML অংশ:** এখানে একটি `div` এবং একটি `button` তৈরি করা হয়েছে।
 - `div` আইডি `colorBox` সেট করা হয়েছে, এবং এটি প্রাথমিকভাবে `lightblue` রঙে দেখা যাবে।
 - `button` আইডি `changeColorButton` সেট করা হয়েছে, যা ক্লিক করলে রং পরিবর্তন হবে।
- CSS অংশ:** `colorBox` কে `200px x 200px` আকারে রাখা হয়েছে এবং সেন্টার করা হয়েছে। বোতামটির ডিজাইন সাধারণভাবে সাজানো হয়েছে।
- JavaScript অংশ:**
 - `changeColor()` ফাংশনে একটি রং-এর অ্যারে তৈরি করা হয়েছে এবং `Math.random()` ব্যবহার করে অ্যারের যেকোনো একটি রং এলোমেলোভাবে নির্বাচিত হয়।
 - `colorBox` এর `backgroundColor` প্রপার্টিকে সেট করা হয়েছে এই র্যান্ডম রং দিয়ে।
 - শেষে, `changeColorButton` বোতামে `click` ইভেন্ট লিসেনার যোগ করে `changeColor()` ফাংশন কল করা হয়েছে।

এখন, যখনই আপনি রং পরিবর্তন করুন বোতামে ক্লিক করবেন, `div` এর ব্যাকগ্রাউন্ড রং এলোমেলোভাবে পরিবর্তিত হবে!

3. How do event listeners work in JavaScript? Explain `addEventListener`.

JavaScript-এ ইভেন্ট লিসেনার (Event Listener) ব্যবহার করা হয় ব্যবহারকারীর সাথে ইন্টারঅ্যাকশন যেমন ক্লিক, কী প্রেস, মাউস ওভার ইত্যাদি ইভেন্টগুলোকে ধরার জন্য। এর মাধ্যমে আমরা DOM-এর কোনো এলিমেন্টে নির্দিষ্ট ইভেন্টে কোনো ফাংশন বা কোড কার্যকর করতে পারি।

`addEventListener` কী?

`addEventListener` মেথডটি JavaScript-এ একটি নির্দিষ্ট ইভেন্টের জন্য একটি ফাংশন (বা ইভেন্ট হ্যান্ডলার) অ্যাসাইন করতে ব্যবহার করা হয়। এর মাধ্যমে আপনি DOM-এ থাকা বিভিন্ন এলিমেন্টের ওপর বিভিন্ন ইভেন্ট সেট করতে পারেন, যেমন `click`, `mouseover`, `keyup` ইত্যাদি।

`addEventListener` এর সিনট্যাক্স

```
element.addEventListener(event, function, useCapture);
```

- **element:** যেই এলিমেন্টে ইভেন্ট লিসেনারটি যোগ করতে চান, সেটি।
- **event:** ইভেন্টের নাম (যেমন `click`, `mouseover`, ইত্যাদি) যা আপনি শোনার জন্য চান।
- **function:** ফাংশনটি, যা ইভেন্টটি ঘটলে কার্যকর হবে।
- **useCapture (ঐচ্ছিক):** এটি `true` অথবা `false` হতে পারে। সাধারণত এটি `false` থাকে, যা ইভেন্ট বুবলিং সক্রিয় রাখে। `true` দিলে ইভেন্ট ক্যাপচারিং মোড সক্রিয় হয়।

উদাহরণ

একটি বোতামে ক্লিক করলে কিছু টেক্সট পরিবর্তন করার জন্য `addEventListener` ব্যবহার করার উদাহরণ:

```
<!DOCTYPE html>
<html lang="bn">
<head>
  <meta charset="UTF-8">
  <title>Event Listener Example</title>
</head>
<body>

  <p id="text">এটি একটি প্রাথমিক টেক্সট।</p>
```

```

<button id="myButton">টেক্স্ট পরিবর্তন করুন</button>

<script>
    // ফাংশন যা টেক্স্ট পরিবর্তন করবে
    function changeText() {
        document.getElementById("text").textContent = "টেক্স্ট পরিব
র্তিত হয়েছে!";
    }

    // ইভেন্ট লিসেনার যোগ করা হচ্ছে
    document.getElementById("myButton").addEventListener("clic
k", changeText);
</script>

</body>
</html>

```

কীভাবে কাজ করে

1. ইলিমেন্ট নির্বাচন: `document.getElementById("myButton")` দিয়ে বোতামটি নির্বাচন করা হয়।
2. ইভেন্ট লিসেনার যোগ করা: `addEventListener` মেথড ব্যবহার করে `click` ইভেন্টের জন্য `changeText` ফাংশনটি অ্যাসাইন করা হয়।
3. ইভেন্ট ঘটানো: যখন ব্যবহারকারী বোতামে ক্লিক করবে, তখন `changeText` ফাংশনটি কল হবে এবং `text` আইডির টেক্স্ট পরিবর্তন হবে।

`addEventListener` কেন ব্যবহৃত হয়?

1. একাধিক ইভেন্ট অ্যাড করা: `addEventListener` দিয়ে একই এলিমেন্টে একাধিক ইভেন্ট হ্যান্ডলার যোগ করা সম্ভব।
2. কোডের বিচ্ছিন্নতা: `addEventListener` আলাদা হ্যান্ডলার দিয়ে কোডকে পরিষ্কার ও সংরক্ষিত রাখতে সাহায্য করে।
3. ইভেন্ট রিমুভ করা: `removeEventListener` মেথড দিয়ে অ্যাসাইনকৃত ইভেন্ট সরানো যায়।

একটি উদাহরণ: দুইটি ইভেন্ট হ্যান্ডলার যোগ করা

```

const myButton = document.getElementById("myButton");

myButton.addEventListener("click", () => {
  console.log("Clicked!");
});

myButton.addEventListener("mouseover", () => {
  console.log("Mouse over the button!");
});

```

এখানে, যখন বোতামে ক্লিক করা হবে, তখন **Clicked!** এবং মাউস বোতামের ওপর রাখলে **Mouse over the button!** কনসোলে দেখা যাবে।

`addEventListener` দিয়ে ইন্টারেক্শন ওয়েবপেজ তৈরি করা খুবই সহজ। এটি ব্যবহার করে বিভিন্ন ইভেন্ট অনুযায়ী পেজের কল্টেন্টকে নিয়ন্ত্রণ করা সম্ভব।

4. What is event delegation, and why is it useful?

ইভেন্ট ডেলিগেশন একটি কার্যকর টেকনিক যেখানে একাধিক এলিমেন্টে আলাদা আলাদা ইভেন্ট লিসেনার যোগ করার পরিবর্তে তাদের অভিভাবক (parent) এলিমেন্টে একটি একক ইভেন্ট লিসেনার অ্যাসাইন করা হয়। এভাবে, ইভেন্টটি ঘটলে প্যারেন্ট এলিমেন্টের মাধ্যমে ইভেন্টকে ধরা যায় এবং চাইল্ড এলিমেন্টে প্রয়োজনীয় কার্যক্রম পরিচালনা করা যায়।

ইভেন্ট ডেলিগেশন কীভাবে কাজ করে?

ইভেন্ট বুবলিং-এর মাধ্যমে ইভেন্ট ডেলিগেশন সম্ভব হয়। ইভেন্ট বুবলিং মানে হলো একটি ইভেন্ট প্রথমে তার টার্গেট এলিমেন্টে ঘটে, এবং এরপর এটি তার প্যারেন্ট এলিমেন্টগুলোর দিকে উপরে ওঠে বা বুবল করে। এই গুণের কারণে আমরা প্যারেন্ট এলিমেন্ট ইভেন্ট লিসেনার সেট করতে পারি এবং চাইল্ড এলিমেন্টের ইভেন্ট ধরতে পারি।

ইভেন্ট ডেলিগেশন কেন দরকার?

- পারফরম্যাল বৃদ্ধি:** একাধিক এলিমেন্ট ইভেন্ট লিসেনার যোগ করা অনেক বেশি মেমোরি ব্যবহার করে। প্যারেন্টে একটি ইভেন্ট লিসেনার যোগ করলে কম মেমোরি লাগে।
- ডায়নামিক কন্টেন্ট ম্যানেজমেন্ট:** নতুন চাইল্ড এলিমেন্ট যোগ করলে আলাদা করে ইভেন্ট অ্যাড করার প্রয়োজন নেই। প্যারেন্টে থাকা ইভেন্ট লিসেনার সব নতুন চাইল্ড এলিমেন্টও কাজ করবে।
- কোডের সরলতা:** একাধিক ইভেন্ট লিসেনার যোগ করার বদলে এক জায়গায় ইভেন্ট লজিক লেখা যায়, যা কোড সহজ ও পরিষ্কার রাখে।

ইভেন্ট ডেলিগেশনের উদাহরণ

ধরুন, আমাদের একটি তালিকা রয়েছে যেখানে প্রতিটি আইটেমে ক্লিক করলে একটি মেসেজ দেখাবে। এখানে আমরা প্রতিটি `li` তে আলাদা ইভেন্ট লিসেনার অ্যাড না করে `ul` প্যারেন্টে ইভেন্ট লিসেনার অ্যাড করব:

```
<!DOCTYPE html>
<html lang="bn">
<head>
  <meta charset="UTF-8">
  <title>Event Delegation Example</title>
</head>
<body>

<ul id="itemList">
  <li>আইটেম ১</li>
  <li>আইটেম ২</li>
  <li>আইটেম ৩</li>
  <li>আইটেম ৪</li>
</ul>

<script>
  // ইভেন্ট ডেলিগেশন ব্যবহার করে প্যারেন্টে ইভেন্ট লিসেনার যোগ করা হচ্ছে
  document.getElementById("itemList").addEventListener("click", function(event) {
    // শুধুমাত্র li তে ক্লিক হলে
    if (event.target.tagName === "LI") {
      alert(` ${event.target.textContent} এ ক্লিক করা হয়েছে! `);
    }
  });
</script>
```

```

    });
</script>

</body>
</html>

```

কীভাবে এটি কাজ করে

১. `ul` প্যারেন্ট ইভেন্ট লিসেনার যোগ: `ul` আইডি `itemList` এ একটি `click` ইভেন্ট লিসেনার যোগ করা হয়েছে।
২. ইভেন্ট টার্গেট চেক: ইভেন্টটি ঘটলে `event.target` এর মাধ্যমে আসল ক্লিক করা এলিমেন্টটি যাচাই করা হয়। এখানে চেক করা হয়েছে `tagName` দিয়ে, অর্থাৎ ক্লিককৃত এলিমেন্ট যদি `li` হয়, তবে মেসেজ দেখাবে।

ইভেন্ট ডেলিগেশন ব্যবহার করার সুবিধা

- **নতুন আইটেমে ইভেন্ট কাজ করবে:** ডায়নামিকভাবে `ul` এ নতুন `li` যোগ করলে আলাদা করে ইভেন্ট যোগ করা লাগবে না।
- **কোড কমপ্লেক্সিটি কম:** কোড সহজ এবং পরিচালনা করা সহজ হয়।
- **কম মেমোরি ব্যবহার:** একাধিক ইলিমেন্টের পরিবর্তে একটি প্যারেন্ট ইভেন্ট লিসেনার অ্যাড করা হয়, যা পারফরম্যান্স বৃদ্ধিতে সাহায্য করে।

সারসংক্ষেপ

ইভেন্ট ডেলিগেশন ব্যবহার করে আমরা DOM ম্যানেজমেন্টকে অনেক সহজ করতে পারি। এটি ডায়নামিক ওয়েব কন্টেন্ট এবং জটিল ইন্টারফেসে অনেক সহায়ক।

Project: Create a list where clicking on each list item logs its content. Use event delegation.

নিচে এমন একটি প্রজেক্ট দেওয়া হলো যেখানে একটি তালিকার প্রতিটি আইটেমে ক্লিক করলে সেটির কন্টেন্ট কনসোলে লগ হবে। এখানে ইভেন্ট ডেলিগেশন ব্যবহার করা হয়েছে, অর্থাৎ প্যারেন্ট `ul`-এ একটি ইভেন্ট লিসেনার অ্যাড করা হয়েছে যা প্রতিটি `li` আইটেমের ক্লিক ইভেন্ট ক্যাপচার করবে।

HTML এবং JavaScript কোড

```
<!DOCTYPE html>
<html lang="bn">
<head>
  <meta charset="UTF-8">
  <title>Event Delegation Project</title>
</head>
<body>

<h2>তালিকায় লিঙ্ক করন এবং কনসোলে দেখুন</h2>

<ul id="itemList">
  <li>আইটেম ১</li>
  <li>আইটেম ২</li>
  <li>আইটেম ৩</li>
  <li>আইটেম ৪</li>
</ul>

<script>
  // প্যারেন্ট `ul`-এ ইভেন্ট লিসেনার যোগ করা হচ্ছে
  document.getElementById("itemList").addEventListener("click", function(event) {
    // চেক করা হচ্ছে যে টার্গেট এলিমেন্টটি একটি `li` কিনা
    if (event.target.tagName === "LI") {
      console.log(event.target.textContent); // `li`-এর কন্টেন্ট কনসোলে লগ হবে
    }
  });
</script>

</body>
</html>
```

কীভাবে এটি কাজ করে

1. ইভেন্ট ডেলিগেশন: `ul` প্যারেন্টে একটি `click` ইভেন্ট লিসেনার যোগ করা হয়েছে, যা তালিকার প্রতিটি আইটেমের ক্লিক ইভেন্ট কেপচার করবে।
2. ইভেন্ট টার্গেট চেক: `event.target` এর মাধ্যমে চেক করা হচ্ছে ক্লিক করা এলিমেন্টটি `LI` কিনা। যদি `LI` হয়, তাহলে তার `textContent` কনসোলে লগ হবে।

আউটপুট

এই প্রজেক্টে, যখনই কোনো `li` আইটেমে ক্লিক করবেন, সেই আইটেমের টেক্সট কনসোলে লগ হবে। উদাহরণস্বরূপ, যদি "আইটেম ২" এ ক্লিক করা হয়, কনসোলে "আইটেম ২" দেখা যাবে।

ইভেন্ট ডেলিগেশনের সুবিধা

- ডায়নামিক আইটেম: নতুন `li` আইটেম যোগ করলেও, আলাদা করে ইভেন্ট লিসেনার যোগ করতে হবে না।
- কোড সহজ: একবারে সব `li` তে ইভেন্ট হ্যান্ডলার যোগ করার পরিবর্তে একটিমাত্র ইভেন্ট হ্যান্ডলার ব্যবহার করা হয়েছে।

এই ধরনের ইভেন্ট ডেলিগেশন ডায়নামিক ওয়েব অ্যাপ্লিকেশন তৈরি করতে অনেক কার্যকর।

5. What is the difference between `innerHTML`, `textContent`, and `innerText` ?

`innerHTML`, `textContent`, এবং `innerText` তিনটি প্রপার্টি যা DOM এলিমেন্টের মধ্যে থাকা কন্টেন্টের সাথে কাজ করতে ব্যবহৃত হয়। যদিও এদের কাজ একই রকম মনে হতে পারে, তবে এদের মধ্যে কিছু মৌলিক পার্থক্য রয়েছে।

1. `innerHTML`

`innerHTML` ব্যবহার করে একটি এলিমেন্টের HTML কন্টেন্ট পড়া এবং সেট করা যায়। এটি HTML ট্যাগ ও তার কন্টেন্ট উভয়কে ধারণ করে, অর্থাৎ, HTML স্ট্রাকচার এবং ট্যাগসহ সম্পূর্ণ কন্টেন্ট পাওয়া যায়।

- পড়তে এবং লিখতে পারা যায়: `innerHTML` ব্যবহার করে আপনি HTML সহ সম্পূর্ণ কন্টেন্ট পড়তে বা পরিবর্তন করতে পারেন।

- **HTML ট্যাগগুলো প্রসেস করে:** এটি HTML ট্যাগগুলোকে পার্স করে। অর্থাৎ, ট্যাগগুলো যেভাবে আছে, সেভাবেই দেখায়।

উদাহরণ:

```
<div id="myDiv"><strong>Bold Text</strong> and some regular text.</div>

<script>
  let content = document.getElementById("myDiv").innerHTML;
  console.log(content); // Output: "<strong>Bold Text</strong> and some regular text."
  
  document.getElementById("myDiv").innerHTML = "<em>Italic Text
</em> and updated text.";
</script>
```

2. **textContent**

`textContent` শুধুমাত্র টেক্স্ট কন্টেন্ট পড়ে এবং সেট করতে ব্যবহৃত হয়। এটি এলিমেন্টের ভেতরের টেক্স্ট ফিরিয়ে দেয়, তবে HTML ট্যাগগুলো বাদ দেয়। এটি HTML ট্যাগগুলোকে উপেক্ষা করে এবং শুধুমাত্র টেক্স্ট অংশ প্রদর্শন করে।

- **HTML ট্যাগগুলো উপেক্ষা করে:** `textContent` শুধুমাত্র টেক্স্ট রিটার্ন করে, এবং HTML ট্যাগগুলো সম্পূর্ণ বাদ দেয়।
- **ফাস্টার:** যেহেতু এটি ট্যাগ পার্স করে না, তাই এটি `innerHTML` থেকে দ্রুত কাজ করে।

উদাহরণ:

```
<div id="myDiv"><strong>Bold Text</strong> and some regular text.</div>

<script>
  let content = document.getElementById("myDiv").textContent;
  console.log(content); // Output: "Bold Text and some regular text"
  
  document.getElementById("myDiv").textContent = "New plain tex
```

```
t content.";  
</script>
```

3. `innerText`

`innerText` একটি এলিমেন্টের টেক্সট কন্টেন্টকে সেট বা পড়ার জন্য ব্যবহৃত হয়। তবে `innerText` কিছু অতিরিক্ত বৈশিষ্ট্য সহ কাজ করে যা `textContent`-এর সাথে মিল থাকলেও আলাদা করে তুলে ধরে। এটি কন্টেন্টকে প্রদর্শনকারী স্টাইলের উপর নির্ভরশীল।

- স্টাইল অনুযায়ী টেক্সট প্রদান করে:** এটি স্টাইল অনুযায়ী কাজ করে এবং CSS `display: none` ইত্যাদির ফলে অদৃশ্য টেক্সট দেখায় না।
- লেআউটের উপর নির্ভরশীল:** এটি ডায়নামিকভাবে রি-ফ্লো বা রি-রেন্ডার ট্রিগার করে।

উদাহরণ:

```
<div id="myDiv" style="display: none">Hidden text.</div>  
  
<script>  
  let content = document.getElementById("myDiv").innerText;  
  console.log(content); // Output: ""  
</script>
```

সারসংক্ষেপ:

প্রপার্টি	HTML ট্যাগ ধরে রাখে?	স্টাইলের উপর নির্ভরশীল?	কি কাজ করে
<code>innerHTML</code>	হ্যাঁ	না	HTML এবং টেক্সট
<code>textContent</code>	না	না	শুধুমাত্র টেক্সট
<code>innerText</code>	না	হ্যাঁ	প্রদর্শিত টেক্সট

এই তিনটি প্রপার্টি DOM কন্টেন্ট ম্যানেজমেন্টে বিভিন্ন পরিস্থিতিতে ব্যবহার হয়।

6. Explain how you can manipulate CSS styles of an element using JavaScript.

JavaScript ব্যবহার করে DOM-এর CSS স্টাইল ম্যানিপুলেট করা যায়, যা এলিমেন্টের চেহারা এবং ফাংশনালিটি পরিবর্তন করতে সহায়ক। JavaScript এর `style` প্রপার্টি ব্যবহার করে সরাসরি CSS প্রোপার্টিগুলোকে ম্যানিপুলেট করা যায়।

১. `.style` প্রপার্টি ব্যবহার করে

`style` প্রপার্টি ব্যবহার করে JavaScript-এ CSS প্রপার্টি সেট করতে পারেন। সরাসরি এলিমেন্টের `style` প্রপার্টি অ্যাক্সেস করলে সেটির ইনলাইন স্টাইল পরিবর্তন করা হয়।

সাধারণ উদাহরণ:

```
<div id="myDiv">আমার স্টাইল পরিবর্তন করুন</div>

<script>
  // ইলিমেন্ট সিলেক্ট করা
  const myDiv = document.getElementById("myDiv");

  // CSS স্টাইল পরিবর্তন করা
  myDiv.style.color = "blue";
  myDiv.style.backgroundColor = "lightgray";
  myDiv.style.padding = "20px";
</script>
```

এখানে `myDiv`-এর টেক্সটের রঙ `blue`, ব্যাকগ্রাউন্ড রঙ `lightgray`, এবং `padding` `20px` সেট করা হয়েছে।

২. ক্যামেলকেস স্টাইল ব্যবহার করে CSS প্রপার্টি অ্যাক্সেস করা

JavaScript-এ CSS প্রপার্টিগুলো ক্যামেলকেস ফর্ম্যাটে লেখা হয়। উদাহরণস্বরূপ, `background-color` প্রপার্টি `backgroundColor` হিসাবে লেখা হয়।

উদাহরণ:

```
myDiv.style.fontSize = "18px";           // CSS: font-size
myDiv.style.marginTop = "10px";          // CSS: margin-top
myDiv.style.borderRadius = "5px";         // CSS: border-radius
```

৩. CSS ক্লাস যোগ বা সরাতে `classList` ব্যবহার করে

একাধিক স্টাইল প্রপার্টি যোগ বা সরানোর জন্য `classList` ব্যবহার করা খুবই কার্যকর। এর মাধ্যমে CSS ক্লাস অ্যাড, রিমুভ বা টগল করা যায়।

উদাহরণ:

```
<div id="myDiv" class="default-style">স্টাইল পরিবর্তন করুন</div>

<style>
  .highlight {
    background-color: yellow;
    color: red;
    font-weight: bold;
  }
</style>

<script>
  const myDiv = document.getElementById("myDiv");

  // highlight ক্লাস যোগ করা
  myDiv.classList.add("highlight");

  // highlight ক্লাস সরানো
  myDiv.classList.remove("highlight");

  // highlight ক্লাস টগল করা (থাকলে সরায়, না থাকলে যোগ করে)
  myDiv.classList.toggle("highlight");
</script>
```

৪. `setProperty` ব্যবহার করে স্টাইল সেট করা

JavaScript-এ `setProperty` মেথড ব্যবহার করেও CSS প্রপার্টি সেট করা যায়, যা কাস্টম প্রপার্টি বা CSS ভ্যারিয়েবল ম্যানিপুলেট করতে সহায়ক।

উদাহরণ:

```
myDiv.style.setProperty("background-color", "lightblue");
myDiv.style.setProperty("font-size", "24px");
```

৫. কম্পিউটেড স্টাইল পড়তে `getComputedStyle` ব্যবহার করা

যদি কোনো এলিমেন্টের বর্তমান প্রপার্টির স্টাইল জানতে চান যা ক্যালকুলেটেড বা ইনহেরিটেড হয়েছে, `getComputedStyle` মেথড ব্যবহার করা হয়।

উদাহরণ:

```
const computedStyle = window.getComputedStyle(myDiv);
console.log(computedStyle.backgroundColor); // ব্যাকগ্রাউন্ড কালার  
লগ করবে
```

৬. ইনলাইন স্টাইল অপসারণ করা

একটি ইনলাইন স্টাইল অপসারণ করতে, `style` প্রপার্টি থেকে `removeProperty` মেথড ব্যবহার করা যায়।

উদাহরণ:

```
myDiv.style.removeProperty("background-color");
```

সারসংক্ষেপ

JavaScript দিয়ে CSS স্টাইল ম্যানিপুলেট করা অনেক কার্যকর, বিশেষত ইন্টারঅ্যাক্টিভ এবং ডায়নামিক কন্টেন্টের জন্য। `style` প্রপার্টি, `classList`, `setProperty`, এবং `getComputedStyle` এর মত টুলগুলো স্টাইলিং ম্যানিপুলেশনে সাহায্য করে।

7. How do you traverse the DOM? Explain `parentNode`, `firstChild`, `lastChild`, etc.

DOM ট্র্যাভার্সাল বলতে DOM-এ থাকা বিভিন্ন এলিমেন্ট বা নোডের মধ্যে এক এলিমেন্ট থেকে আরেকটিতে চলাচল বা নেভিগেট করা বোঝায়। JavaScript-এ DOM ট্র্যাভার্সাল করার জন্য কিছু প্রপার্টি আছে, যা দিয়ে এলিমেন্টের প্যারেন্ট, চাইল্ড বা সিবলিং (ভাই-বোন) খুঁজে বের করা যায়। নিচে কিছু গুরুত্বপূর্ণ DOM ট্র্যাভার্সাল প্রপার্টি ও তাদের বর্ণনা দেওয়া হলো।

১. `parentNode`

`parentNode` ব্যবহার করে একটি নোডের প্যারেন্ট বা অভিভাবক নোড পাওয়া যায়। এটি DOM ট্রিতে উপরের দিকে নেভিগেট করতে সহায়তা করে।

```
<div id="parentDiv">
  <p id="childPara">এই একটি প্যারাগ্রাফ।</p>
</div>

<script>
  const childPara = document.getElementById("childPara");
  const parentDiv = childPara.parentNode;
  console.log(parentDiv); // Output: <div id="parentDiv">...</div>
</script>
```

২. `firstChild` এবং `lastChild`

- `firstChild`: এটি একটি নোডের প্রথম চাইল্ড নোড রিটার্ন করে। যদি চাইল্ড হিসেবে শুধুমাত্র টেক্সট বা স্পেস থাকে, তাহলে সেটিও রিটার্ন করবে।
- `lastChild`: এটি শেষ চাইল্ড নোড রিটার্ন করে।

```
<div id="parentDiv">
  <p>প্রথম প্যারাগ্রাফ</p>
  <p>শেষ প্যারাগ্রাফ</p>
</div>

<script>
  const parentDiv = document.getElementById("parentDiv");
```

```

console.log(parentDiv.firstChild); // Output: #text (যদি টেক্স্ট বা স্পেস থাকে)
console.log(parentDiv.lastChild); // Output: <p>শেষ প্যারাগ্রাফ
</p>
</script>

```

৩. `firstElementChild` এবং `lastElementChild`

- `firstElementChild`: এটি শুধুমাত্র প্রথম চাইল্ড এলিমেন্ট নোড প্রদান করে, টেক্স্ট বা কমেন্ট বাদ দিয়ে।
- `lastElementChild`: এটি শুধুমাত্র শেষ চাইল্ড এলিমেন্ট নোড প্রদান করে।

```

<div id="parentDiv">
  <p>প্রথম প্যারাগ্রাফ</p>
  <p>শেষ প্যারাগ্রাফ</p>
</div>

<script>
  const parentDiv = document.getElementById("parentDiv");
  console.log(parentDiv.firstElementChild); // Output: <p>প্রথম প্যারাগ্রাফ</p>
  console.log(parentDiv.lastElementChild); // Output: <p>শেষ প্যারাগ্রাফ</p>
</script>

```

৪. `nextSibling` এবং `previousSibling`

- `nextSibling`: এটি একই প্যারেন্টের পরের সিবলিং (ভাই-বোন) নোড প্রদান করে।
- `previousSibling`: এটি একই প্যারেন্টের আগের সিবলিং নোড প্রদান করে।

```

<div>
  <p id="firstPara">প্রথম প্যারাগ্রাফ</p>
  <p id="secondPara">দ্বিতীয় প্যারাগ্রাফ</p>
</div>

```

```

<script>
  const firstPara = document.getElementById("firstPara");
  console.log(firstPara.nextSibling); // Output: #text (যদি
  টেক্স্ট বা স্পেস থাকে)

  const secondPara = document.getElementById("secondPara");
  console.log(secondPara.previousSibling); // Output: #text (য
  দি টেক্স্ট বা স্পেস থাকে)
</script>

```

৫. `nextElementSibling` এবং `previousElementSibling`

- `nextElementSibling`: এটি শুধুমাত্র পরবর্তী সিবলিং এলিমেন্ট প্রদান করে।
- `previousElementSibling`: এটি শুধুমাত্র পূর্ববর্তী সিবলিং এলিমেন্ট প্রদান করে।

```

<div>
  <p id="firstPara">প্রথম প্যারাগ্রাফ</p>
  <p id="secondPara">দ্বিতীয় প্যারাগ্রাফ</p>
</div>

<script>
  const firstPara = document.getElementById("firstPara");
  console.log(firstPara.nextElementSibling); // Output: <p id
="secondPara">...</p>

  const secondPara = document.getElementById("secondPara");
  console.log(secondPara.previousElementSibling); // Output: <
p id="firstPara">...</p>
</script>

```

৬. `childNodes` এবং `children`

- `childNodes`: এটি সমস্ত চাইল্ড নোডের একটি `NodeList` প্রদান করে, যেখানে টেক্স্ট, কমেন্ট,
এবং অন্যান্য সব ধরণের নোড অন্তর্ভুক্ত থাকে।
- `children`: এটি শুধুমাত্র চাইল্ড এলিমেন্টগুলোকে প্রদান করে, টেক্স্ট বা কমেন্ট বাদ দিয়ে।

```

<div id="parentDiv">
  <p>প্রথম প্যারাগ্রাফ</p>
  <p>দ্বিতীয় প্যারাগ্রাফ</p>
</div>

<script>
  const parentDiv = document.getElementById("parentDiv");
  console.log(parentDiv.childNodes);    // Output: NodeList(প্রথম
  প্যারাগ্রাফ, টেক্সট ইত্যাদি)
  console.log(parentDiv.children);      // Output: HTMLCollectio
n(প্রথম প্যারাগ্রাফ, দ্বিতীয় প্যারাগ্রাফ)
</script>

```

সারসংক্ষেপ টেবিল:

প্রপার্টি	কাজ
<code>parentNode</code>	প্যারেন্ট বা অভিভাবক নোড প্রদান করে
<code>firstChild</code>	প্রথম চাইল্ড নোড প্রদান করে (টেক্সটসহ)
<code>lastChild</code>	শেষ চাইল্ড নোড প্রদান করে (টেক্সটসহ)
<code>firstElementChild</code>	প্রথম চাইল্ড এলিমেন্ট প্রদান করে
<code>lastElementChild</code>	শেষ চাইল্ড এলিমেন্ট প্রদান করে
<code>nextSibling</code>	পরবর্তী সিবলিং নোড প্রদান করে (টেক্সটসহ)
<code>previousSibling</code>	পূর্ববর্তী সিবলিং নোড প্রদান করে (টেক্সটসহ)
<code>nextElementSibling</code>	পরবর্তী সিবলিং এলিমেন্ট প্রদান করে
<code>previousElementSibling</code>	পূর্ববর্তী সিবলিং এলিমেন্ট প্রদান করে
<code>childNodes</code>	সমস্ত চাইল্ড নোডের একটি <code>NodeList</code> প্রদান করে
<code>children</code>	সমস্ত চাইল্ড এলিমেন্টের একটি <code>HTMLCollection</code> প্রদান করে

এই প্রপার্টিগুলি DOM-এ বিভিন্ন এলিমেন্টের মধ্যে চলাচল এবং প্রয়োজনীয় কাজ সম্পন্ন করতে সহায়তা করে।

8. What is the purpose of `preventDefault()` and `stopPropagation()` in event handling?

JavaScript ইভেন্ট হ্যান্ডলিং-এ `preventDefault()` এবং `stopPropagation()` দুটি গুরুত্বপূর্ণ মেথড। এদের মাধ্যমে ইভেন্টের ডিফল্ট আচরণ এবং ইভেন্টের প্রচার বা `propagation` কন্ট্রোল করা যায়। নিচে এদের কাজ ও উদাহরণ আলোচনা করা হলো।

১. `preventDefault()`

`preventDefault()` একটি ইভেন্টের ডিফল্ট আচরণ প্রতিরোধ করতে ব্যবহৃত হয়। কিছু ইভেন্ট যেমন ফর্ম সাবমিট করা, লিংকে ক্লিক করা, বা রাইট-ক্লিক মেনু প্রদর্শনের মতো ইভেন্টগুলোর ডিফল্ট আচরণ আছে। `preventDefault()` ব্যবহার করলে এই ডিফল্ট আচরণটি প্রতিরোধ করা যায়।

উদাহরণ:

ধরুন, একটি লিংকে ক্লিক করলে সাধারণত নতুন পেজে চলে যাওয়ার কথা। যদি আপনি চান ক্লিক করলে অন্য কোনো কাজ হবে, কিন্তু নতুন পেজে যাবে না, তাহলে `preventDefault()` ব্যবহার করতে পারেন।

```
<a href="https://example.com" id="myLink">লিংকে ক্লিক করুন</a>

<script>
  const link = document.getElementById("myLink");

  link.addEventListener("click", function(event) {
    event.preventDefault(); // ডিফল্ট আচরণ প্রতিরোধ করা হচ্ছে
    console.log("লিংকটি কাজ করেছে কিন্তু নতুন পেজে যায়নি");
  });
</script>
```

এখানে, লিংকে ক্লিক করলে নতুন পেজে না গিয়ে কনসোলে মেসেজ প্রদর্শিত হবে, কারণ `preventDefault()` ডিফল্ট আচরণ প্রতিরোধ করেছে।

২. `stopPropagation()`

`stopPropagation()` মেথডটি ইভেন্ট প্রপাগেশন বন্ধ করতে ব্যবহৃত হয়। DOM-এ ইভেন্টগুলো `bubbling` বা `capturing` মডেলে একটি প্যারেন্ট এলিমেন্ট থেকে চাইল্ড এলিমেন্ট বা চাইল্ড থেকে প্যারেন্ট পর্যন্ত ছড়ায়। যখন `stopPropagation()` ব্যবহার করা হয়, তখন ইভেন্টটি আর প্যারেন্ট বা চাইল্ড এলিমেন্টে ছড়ায় না।

উদাহরণ:

ধরুন, একটি `div` এবং তার ভেতরে একটি `button` আছে। উভয়ের উপরেই ক্লিক ইভেন্ট আছে। এখন, `button`-এ ক্লিক করলে আপনি চান `div`-এর ইভেন্ট ট্রিগার না হোক।

```
<div id="parentDiv" style="padding: 20px; background-color: lightgray;">
    <button id="childButton">বাটনে ক্লিক করুন</button>
</div>

<script>
    const parentDiv = document.getElementById("parentDiv");
    const childButton = document.getElementById("childButton");

    parentDiv.addEventListener("click", function() {
        console.log("ডিভে ক্লিক হয়েছে");
    });

    childButton.addEventListener("click", function(event) {
        event.stopPropagation(); // প্রপাগেশন বন্ধ করা হচ্ছে
        console.log("বাটনে ক্লিক হয়েছে");
    });
</script>
```

এখনে, বাটনে ক্লিক করলে শুধুমাত্র "বাটনে ক্লিক হয়েছে" মেসেজ প্রদর্শিত হবে। `stopPropagation()` এর জন্য `div`-এর ইভেন্টটি আর ট্রিগার হবে না এবং "ডিভে ক্লিক হয়েছে" প্রদর্শিত হবে না।

সারসংক্ষেপ

মেথড	কাজ
<code>preventDefault()</code>	ইভেন্টের ডিফল্ট আচরণ প্রতিরোধ করে
<code>stopPropagation()</code>	ইভেন্টের প্রপাগেশন বা ছড়ানো বন্ধ করে

এই মেথডগুলো ইভেন্ট ম্যানেজমেন্ট নিয়ন্ত্রণ আনার জন্য গুরুত্বপূর্ণ, বিশেষ করে জাভাস্ক্রিপ্ট-নির্ভর ইন্টারঅ্যাকটিভ UI তৈরি করতে।

Project: Create a form that prevents submission and logs a message instead.

এখানে একটি সিম্পল HTML ফর্ম তৈরি করা হচ্ছে, যেটি সাবমিট হওয়ার পরিবর্তে একটি মেসেজ কনসোলে লগ করবে। এর জন্য আমরা JavaScript ব্যবহার করব `preventDefault()` মেথডটি ফর্মের ডিফল্ট সাবমিট আচরণ প্রতিরোধ করতে।

প্রজেক্ট: ফর্ম সাবমিট প্রতিরোধ এবং মেসেজ লগ করা

HTML + JavaScript কোড:

```
<!DOCTYPE html>
<html lang="bn">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>ফর্ম সাবমিট প্রতিরোধ</title>
</head>
<body>
    <h2>ফর্ম সাবমিট প্রতিরোধ এবং মেসেজ লগ করা</h2>

    <!-- ফর্ম -->
    <form id="myForm">
        <label for="name">নাম:</label>
        <input type="text" id="name" name="name" required><br><br>

        <label for="email">ইমেইল:</label>
        <input type="email" id="email" name="email" required><br><br>
    </form>
</body>
```

```

<button type="submit">সাবমিট করুন</button>
</form>

<script>
    // ফর্ম সিলেক্ট করা
    const form = document.getElementById("myForm");

    // সাবমিট ইভেন্ট হ্যান্ডলার
    form.addEventListener("submit", function(event) {
        // ডিফল্ট সাবমিট আচরণ প্রতিরোধ করা
        event.preventDefault();

        // কনসোলে মেসেজ লগ করা
        console.log("ফর্মটি সাবমিট হয়নি, কিন্তু মেসেজ লগ হয়েছে!");
    });
</script>
</body>
</html>

```

ব্যাখ্যা:

1. HTML ফর্ম: একটি সিম্পল ফর্ম তৈরি করা হয়েছে যার মধ্যে দুটি ইনপুট ফিল্ড (নাম এবং ইমেইল) এবং একটি সাবমিট বাটন রয়েছে।

2. JavaScript:

- `form.addEventListener("submit", function(event) {...})`: এই ইভেন্ট লিসনারটি ফর্মের `submit` ইভেন্টে হ্যান্ডল করা হচ্ছে। যখন ফর্মটি সাবমিট হয়, তখন এটি কল হবে।
- `event.preventDefault()`: ফর্মের ডিফল্ট সাবমিট আচরণ প্রতিরোধ করে। এর ফলে পেজ রিলোড বা ফর্ম সাবমিট হবে না।
- `console.log()`: কনসোলে একটি মেসেজ প্রদর্শন করা হচ্ছে যা জানায় যে ফর্মটি সাবমিট হয়নি, কিন্তু মেসেজটি লগ হয়েছে।

ফলাফল:

- ফর্ম সাবমিট করার পর পেজ রিলোড হবে না এবং কনসোলে মেসেজ "ফর্মটি সাবমিট হয়নি, কিন্তু মেসেজ লগ হয়েছে!" দেখা যাবে।

>> BOM (8 questions)

1. What is the Browser Object Model (BOM) in JavaScript?

Browser Object Model (BOM) হল JavaScript-এ ওয়েব ব্রাউজারের সাথে ইন্টারঅ্যাকশন করার জন্য ব্যবহৃত একটি অবজেক্ট মডেল। BOM এর মাধ্যমে JavaScript ব্রাউজারের কিছু বৈশিষ্ট্য এবং ফাংশনালিটির অ্যাক্সেস পায়, যেমন উইন্ডো ম্যানিপুলেশন, URL ম্যানিপুলেশন, কুকি ম্যানেজমেন্ট, ব্রাউজারের আউটপুট স্ক্রিনে ইন্টারঅ্যাকশন, ইত্যাদি। BOM মূলত **DOM (Document Object Model)** এর বাইরে ব্রাউজারের বাইরের কিছু বৈশিষ্ট্য নিয়ন্ত্রণ করে।

BOM এর প্রধান বৈশিষ্ট্য:

১. Window Object

`window` অবজেক্ট হল BOM এর মূল অবজেক্ট। এটি ব্রাউজারের উইন্ডো বা ট্যাবের সাথে সম্পর্কিত সমস্ত ফাংশনালিটি এবং বৈশিষ্ট্য ধারণ করে। `window` অবজেক্টের মাধ্যমে আপনি পেজের অবস্থান, সাইজ, টাইমার, কুকি ইত্যাদি নিয়ন্ত্রণ করতে পারেন।

উদাহরণ:

```
// ব্রাউজারের উইন্ডো সাইজ জানা
console.log(window.innerWidth); // উইন্ডোর প্রস্থ
console.log(window.innerHeight); // উইন্ডোর উচ্চতা

// উইন্ডো ক্লোজ করা
window.close();

// একটি নতুন উইন্ডো বা ট্যাব ওপেন করা
window.open("<https://example.com>", "_blank");
```

২. Navigator Object

`navigator` অবজেক্ট ব্রাউজারের তথ্য প্রদান করে, যেমন ব্রাউজারের নাম, সংস্করণ, অপারেটিং সিস্টেম, প্ল্যাটফর্ম ইত্যাদি।

উদাহরণ:

```
console.log(navigator.userAgent); // ব্রাউজারের ইউজার এজেন্ট  
console.log(navigator.platform); // অপারেটিং সিস্টেমের তথ্য
```

৩. Location Object

`location` অবজেক্টের মাধ্যমে আপনি ব্রাউজারের বর্তমান URL ম্যানিপুলেট করতে পারেন, যেমন নতুন URL লোড করা, কাস্টম URL এ রিডাইরেন্ট করা, ইত্যাদি।

উদাহরণ:

```
// বর্তমান URL দেখা  
console.log(location.href);  
  
// নতুন URL লোড করা  
location.href = "<https://example.com>";  
  
// ব্রাউজারের হোমপেজে রিডাইরেন্ট করা  
location.replace("<https://example.com>");  
  
// URL এর অংশবিশেষ অ্যাক্সেস করা  
console.log(location.pathname); // পাথ  
console.log(location.protocol); // প্রোটোকল (যেমন: http://)  
console.log(location.hostname); // হোস্টনেম (যেমন: www.example.com)
```

৪. History Object

`history` অবজেক্ট ব্রাউজারের ইতিহাস (ব্যাক/ফরওয়ার্ড) নিয়ন্ত্রণ করার জন্য ব্যবহৃত হয়। আপনি

`history` অবজেক্ট ব্যবহার করে পূর্ববর্তী বা পরবর্তী পেজে নেভিগেট করতে পারেন।

উদাহরণ:

```
// ব্রাউজার ইতিহাসে এক পেজ পিছনে যাওয়া  
history.back();
```

```
// ব্রাউজার ইতিহাসে এক পেজ সামনে যাওয়া  
history.forward();  
  
// ইতিহাসে একটি নতুন পেজ ঘোগ করা  
history.pushState({}, "", "newPage.html");
```

৫. Screen Object

`screen` অবজেক্টের মাধ্যমে ব্রাউজারের স্ক্রীনের সাইজ এবং রেজোলিউশন সম্পর্কিত তথ্য পাওয়া যায়।

উদাহরণ:

```
console.log(screen.width); // স্ক্রীনের প্রস্থ  
console.log(screen.height); // স্ক্রীনের উচ্চতা
```

৬. Timers (setTimeout and setInterval)

BOM এ ব্রাউজারে সময় নির্ধারণ করার জন্য `setTimeout()` এবং `setInterval()` ফাংশন ব্যবহার করা হয়। এগুলি সময় নির্ধারিত হলে নির্দিষ্ট ফাংশন কল করে।

উদাহরণ:

```
// setTimeout উদাহরণ: ৩ সেকেন্ড পরে ফাংশন কল  
setTimeout(function() {  
    console.log("৩ সেকেন্ড পরে কল হয়েছে!");  
, 3000);  
  
// setInterval উদাহরণ: প্রতি ২ সেকেন্ডে ফাংশন কল  
setInterval(function() {  
    console.log("প্রতি ২ সেকেন্ডে কল হচ্ছে!");  
, 2000);
```

সারসংক্ষেপ:

BOM JavaScript-এ ব্রাউজারের বাইরের উপাদানগুলোকে অ্যাক্সেস করার জন্য ব্যবহৃত একটি অবজেক্ট মডেল। এর মাধ্যমে আপনি:

- ব্রাউজারের উইন্ডো, লোকেশন, হিস্ট্রি, স্ক্রীন ইত্যাদি নিয়ন্ত্রণ করতে পারেন।
- ব্রাউজারের বিভিন্ন বৈশিষ্ট্য যেমন ব্রাউজারের তথ্য (navigator), টাইমার (setTimeout, setInterval), URL ম্যানিপুলেশন (location), এবং উইন্ডো ম্যানিপুলেশন (window) করতে পারেন।

এগুলি DOM থেকে আলাদা হলেও একে অপরের সাথে সম্পর্কিত, কারণ BOM এবং DOM একসাথে কাজ করে ব্রাউজারে ডাইনামিক কন্টেন্ট এবং ইন্টারঅ্যাকটিভিটি যোগ করতে।

2. How does `window` differ from `document` ?

`window` এবং `document` দুটোই JavaScript-এ গুরুত্বপূর্ণ অবজেক্ট, কিন্তু এগুলোর কাজ এবং ব্যবহার ভিন্ন। এখানে সেগুলোর পার্থক্য আলোচনা করা হলো:

১. `window` Object

`window` অবজেক্টটি ব্রাউজারের উইন্ডো বা ট্যাবের সাথে সম্পর্কিত। এটি **BOM (Browser Object Model)** এর একটি অংশ এবং ব্রাউজারের পরিবেশকে প্রতিনিধিত্ব করে। `window` অবজেক্টের মাধ্যমে আপনি পেজের স্ক্রিন সাইজ, টাইমার, ইভেন্ট হ্যান্ডলার, উইন্ডো ম্যানিপুলেশন, ইত্যাদি নিয়ন্ত্রণ করতে পারেন।

প্রধান বৈশিষ্ট্য:

- **গ্লোবাল অবজেক্ট:** `window` হল গ্লোবাল অবজেক্ট, যার মানে এটি JavaScript এর গ্লোবাল স্কোপের সাথে যুক্ত থাকে। অর্থাৎ, আপনি যেকোনো ফাংশন বা ভ্যারিয়েবলকে সরাসরি `window` অবজেক্টের মাধ্যমে অ্যারেস করতে পারেন।
- **ইউজার ইন্টারফেসের নিয়ন্ত্রণ:** উইন্ডো সাইজ, স্ক্রোল, এবং অন্যান্য ইউজার ইন্টারফেস সম্পর্কিত কাজ।
- **টাইটেল, হিস্ট্রি, এবং লোকেশন:** ব্রাউজারের টাইটেল, লোকেশন, হিস্ট্রি নিয়ন্ত্রণ করার জন্য `window` অবজেক্টের বিভিন্ন প্রপার্টি ব্যবহার করা হয়।

উদাহরণ:

```

console.log(window.innerWidth); // উইন্ডোর প্রস্থ
console.log(window.location.href); // বর্তমান URL

// উইন্ডো ক্লোজ করা
window.close();

```

২. `document` Object

`document` অবজেক্টটি **DOM (Document Object Model)** এর একটি অংশ এবং এটি HTML ডকুমেন্টের কন্টেন্টকে প্রতিনিধিত্ব করে। `document` অবজেক্টের মাধ্যমে আপনি HTML পেজের এলিমেন্ট (যেমন `<div>`, `<p>`, `<a>` ইত্যাদি) নির্বাচন এবং ম্যানিপুলেট করতে পারেন।

প্রধান বৈশিষ্ট্য:

- HTML পেজের কন্টেন্ট:** `document` অবজেক্টের মাধ্যমে পেজের DOM এর সব এলিমেন্ট এবং স্ট্রাকচার অ্যাক্সেস করা যায়। উদাহরণস্বরূপ, আপনি `document.getElementById()`, `document.querySelector()` ব্যবহার করে HTML এলিমেন্ট সিলেক্ট করতে পারেন।
- ইন্টারঅ্যাকশন:** ফর্মের ইনপুট ডেটা সংগ্রহ করা, নতুন এলিমেন্ট তৈরি করা বা পুরোনো এলিমেন্ট মুছে ফেলা।

উদাহরণ:

```

// HTML এলিমেন্ট সিলেক্ট করা
const element = document.getElementById("myElement");

// নতুন এলিমেন্ট তৈরি করা
const newDiv = document.createElement("div");
newDiv.innerHTML = "নতুন ডিভ এলিমেন্ট";

// ডকুমেন্টে যোগ করা
document.body.appendChild(newDiv);

```

প্রধান পার্থক্য:

বৈশিষ্ট্য	<code>window</code>	<code>document</code>
অবজেক্টের ধরন	ব্রাউজারের উইন্ডো বা ট্যাবের সাথে	HTML ডকুমেন্টের কন্টেন্টের সাথে সম্পর্কিত

	সম্পর্কিত	
অ্যাক্সেস	স্ক্রিন সাইজ, হিস্ট্রি, লোকেশন, টাইটেল ইত্যাদি	HTML এলিমেন্ট, DOM ম্যানিপুলেশন
গ্লোবাল অবজেক্ট	হ্যাঁ, JavaScript গ্লোবাল স্কোপের অংশ	না, এটি DOM-এর একটি অংশ
ব্যবহার	উইড্ভো নিয়ন্ত্রণ (স্ক্রিন সাইজ, টাইমার, লোকেশন ইত্যাদি)	HTML পেজের এলিমেন্ট এবং কন্টেন্ট নিয়ন্ত্রণ
উদাহরণ	<code>window.innerWidth</code> , <code>window.location</code>	<code>document.getElementById()</code> , <code>document.querySelector()</code>

সারসংক্ষেপ:

- `window`: ব্রাউজারের উইড্ভো বা ট্যাবের সাথে সম্পর্কিত, যা গ্লোবাল স্কোপ হিসাবে কাজ করে এবং ইউজার ইন্টারফেসের উপাদান এবং পেজের বাইরের তথ্য নিয়ন্ত্রণ করে।
- `document`: HTML ডকুমেন্টের কন্টেন্টের সাথে সম্পর্কিত, যা DOM এর মাধ্যমে HTML এলিমেন্টগুলোর সাথে কাজ করতে ব্যবহৃত হয়।

`window` সাধারণত ব্রাউজারের পরিবেশে কাজ করে এবং `document` সাধারণত ব্রাউজারের ভিতরের ডকুমেন্টের কন্টেন্ট ম্যানিপুলেট করতে ব্যবহৃত হয়।

3. Explain how `localStorage`, `sessionStorage`, and `cookies` work and their differences.

localStorage, **sessionStorage**, এবং **cookies** হল তিনটি ভিন্ন প্রযুক্তি যেগুলি ওয়েব ব্রাউজারে ডেটা স্টোর করার জন্য ব্যবহৃত হয়। যদিও এদের কাজ প্রায় একই—ডেটা স্টোর করা—তবে এদের মধ্যে কিছু গুরুত্বপূর্ণ পার্থক্য রয়েছে। নিচে এই তিনটি প্রযুক্তি এবং তাদের পার্থক্য নিয়ে বিস্তারিত আলোচনা করা হলো।

১. `localStorage`

`localStorage` হল একটি ওয়েব স্টোরেজ API যা ফ্লায়েন্ট সাইডে (ব্রাউজারে) ডেটা সেভ করে রাখে। এটি ডেটা সেভ করার জন্য ব্রাউজারের প্যামেনেন্ট স্টোরেজ ব্যবহার করে, অর্থাৎ ডেটা ব্রাউজার বন্ধ বা রিলোড করার পরও টিকে থাকে।

বৈশিষ্ট্য:

- **ডেটার স্থায়িত্ব:** ডেটা টেম্পোরারি নয়, এটি যতক্ষণ না ব্যবহারকারী ডেটা ম্যানুয়ালি মুছে ফেলবে, ততক্ষণ টিকে থাকে। ব্রাউজার বন্ধ বা রিলোড করার পরও ডেটা থাকবে।
- **স্টোরেজ সাইজ:** সাধারণত `localStorage` এ 5MB পর্যন্ত ডেটা সেভ করা যেতে পারে।
- **ডেটা অ্যাক্সেস:** `localStorage` এর মাধ্যমে ডেটা অ্যাক্সেস করা হয় যেকোনো পেজ থেকে, যতক্ষণ না ব্যবহারকারী কুকি বা লোকাল স্টোরেজ মুছে ফেলেন।

উদাহরণ:

```
// localStorage-এ ডেটা সেভ করা
localStorage.setItem('username', 'Rabbani');

// localStorage থেকে ডেটা পড়া
const username = localStorage.getItem('username');
console.log(username); // Rabbani

// localStorage থেকে ডেটা মুছে ফেলা
localStorage.removeItem('username');
```

২. sessionStorage

`sessionStorage`-ও একইভাবে ওয়েব স্টোরেজ API ব্যবহার করে, তবে এর ডেটা শুধুমাত্র বর্তমান সেশন (একটি ব্রাউজার ট্যাব বা উইন্ডো) পর্যন্ত টিকে থাকে। যখন ব্রাউজার ট্যাব বন্ধ করা হয়, তখন `sessionStorage`-এর সব ডেটা মুছে যায়।

বৈশিষ্ট্য:

- **ডেটার স্থায়িত্ব:** `sessionStorage` এ সেভ করা ডেটা শুধুমাত্র একটিভ সেশনে টিকে থাকে। ট্যাব বা উইন্ডো বন্ধ করলে ডেটা মুছে যায়।
- **স্টোরেজ সাইজ:** `sessionStorage` এ সাধারণত 5MB পর্যন্ত ডেটা সেভ করা যায়।
- **ডেটা অ্যাক্সেস:** শুধুমাত্র একই সেশন (ট্যাব বা উইন্ডো) থেকে অ্যাক্সেস করা যায়। অন্য কোনো ট্যাব বা উইন্ডো থেকে এক্সেস করা সম্ভব নয়।

উদাহরণ:

```
// sessionStorage-এ ডেটা সেভ করা
sessionStorage.setItem('sessionData', 'This is session data');
```

```

// sessionStorage থেকে ডেটা পড়া
const sessionData = sessionStorage.getItem('sessionData');
console.log(sessionData); // This is session data

// sessionStorage থেকে ডেটা মুছে ফেলা
sessionStorage.removeItem('sessionData');

```

৩. Cookies

Cookies হল ছোট ডেটা স্লিপেট যা ক্লায়েন্ট (ব্রাউজার) এবং সার্ভারের মধ্যে পাঠানো হয়। কুকিজ মূলত ব্রাউজারের মধ্যে কিছু স্টেট বা ইউজার সেশনের তথ্য রাখতে ব্যবহৃত হয় এবং সাধারণত খুব ছোট (8KB) ডেটা ধারণ করতে পারে।

বৈশিষ্ট্য:

- ডেটার স্থায়িত্ব: কুকিজে সেভ করা ডেটা একটি নির্দিষ্ট সময় পর্যন্ত থাকে, যেটি কুকির `expires` বা `max-age` অ্যাট্‌রিবিউট দ্বারা নির্ধারিত হয়। যদি সময় শেষ হয়ে যায়, কুকি মুছে যায়।
- স্টোরেজ সাইজ: কুকির স্টোরেজ সাইজ 8KB পর্যন্ত সীমাবদ্ধ।
- ডেটা অ্যাক্সেস: কুকি ডেটা শুধুমাত্র সার্ভার এবং ক্লায়েন্টের মধ্যে পাঠানো হয়। এই ডেটা সাধারণত সার্ভারকে পাঠানো হয় যখন ব্যবহারকারী একটি HTTP রিকোয়েস্ট পাঠায়।

উদাহরণ:

```

// কুকি তৈরি করা
document.cookie = "user=Rabbani; expires=Thu, 18 Dec 2024 12:00:00 UTC; path=/";

// কুকি থেকে ডেটা পড়া
console.log(document.cookie); // "user=Rabbani"

// কুকি মুছে ফেলা
document.cookie = "user=; expires=Thu, 01 Jan 1970 00:00:00 UTC; path=/";

```

পার্থক্য:

বৈশিষ্ট্য	localStorage	sessionStorage	Cookies
ডেটার স্থায়িত্ব	ব্রাউজার বন্ধ হলেও টিকে থাকে	শুধুমাত্র এক সেশন (ট্যাব/উইন্ডো) পর্যন্ত টিকে থাকে	নির্ধারিত সময় পর্যন্ত টিকে থাকে, বা মুছে দেওয়া হয়
স্টোরেজ সাইজ	5MB পর্যন্ত	5MB পর্যন্ত	4KB পর্যন্ত
অ্যাক্সেস	সাইটের যেকোনো পেজ থেকে অ্যাক্সেস করা যায়	শুধুমাত্র বর্তমান সেশন থেকে অ্যাক্সেস করা যায়	সার্ভার ও ক্লায়েন্টের মধ্যে পাঠানো হয়
এপিআই	<code>localStorage.setItem()</code> , <code>localStorage.getItem()</code>	<code>sessionStorage.setItem()</code> , <code>sessionStorage.getItem()</code>	<code>document.cookie</code>
ব্যবহার	স্থায়ী ডেটা সংরক্ষণ	সেশনের জন্য ডেটা সংরক্ষণ	সার্ভারের সাথে কমিউনিকেশন এবং সেশন ট্র্যাকিং

সারসংক্ষেপ:

- **localStorage**: ব্রাউজার বন্ধ হলেও ডেটা সেভ থাকে, বড় সাইজে ডেটা সংরক্ষণ করা যায়, কিন্তু শুধুমাত্র ক্লায়েন্ট সাইডে ব্যবহার করা হয়।
- **sessionStorage**: এক সেশনের (ট্যাব/উইন্ডো) মধ্যে টিকে থাকে, এক্সপায়ারেশন নেই, এবং শুধুমাত্র ক্লায়েন্ট সাইডে ব্যবহার করা হয়।
- **Cookies**: সার্ভার এবং ক্লায়েন্টের মধ্যে ডেটা পাঠানো হয়, ছোট সাইজে ডেটা সংরক্ষণ করা যায়, এবং নির্দিষ্ট সময় পর্যন্ত টিকে থাকে।

এই তিনটি প্রযুক্তি একে অপরকে পরিপূরকভাবে কাজ করতে পারে, এবং যেকোনো ওয়েব অ্যাপ্লিকেশনে ডেটা স্টোর করার জন্য বিভিন্ন পরিস্থিতিতে এগুলোর মধ্যে নির্বাচন করা যেতে পারে।

Project: Store and retrieve user preferences (like theme) using `localStorage`.

এখানে একটি প্রকল্প তৈরি করা হলো যেখানে ব্যবহারকারীর থিম প্রেফারেন্স (যেমন: লাইট বা ডার্ক থিম) `localStorage` ব্যবহার করে সংরক্ষণ এবং পুনরুদ্ধার করা হবে।

প্রকল্পের লক্ষ্য:

- ব্যবহারকারী যেই থিম সিলেক্ট করবে (লাইট বা ডার্ক), তা `localStorage` এ সেভ হবে।

- পরবর্তীতে ব্যবহারকারী পেজটি রিফ্রেশ করলেই তাদের সিলেক্ট করা থিম স্বয়ংক্রিয়ভাবে লোড হবে।

কোড:

- HTML ফাইল:** এখানে দুটি বাটন থাকবে (একটি লাইট থিমের জন্য এবং একটি ডার্ক থিমের জন্য)।
- CSS ফাইল:** দুটি থিমের জন্য আলাদা স্টাইল থাকবে।
- JavaScript ফাইল:** থিম পরিবর্তন এবং `localStorage` এ ডেটা সেভ ও রিট্রিভ করার লজিক থাকবে।

১. HTML (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User Preferences - Theme</title>
    <link rel="stylesheet" href="styles.css">
</head>
<body>
    <div class="container">
        <h1>Choose a Theme</h1>
        <button id="light-btn">Light Theme</button>
        <button id="dark-btn">Dark Theme</button>
    </div>

    <script src="script.js"></script>
</body>
</html>
```

২. CSS (styles.css):

```

/* Default (Light Theme) */
body {
    background-color: #f0f0f0;
    color: #333;
    font-family: Arial, sans-serif;
    transition: background-color 0.3s, color 0.3s;
}

.container {
    text-align: center;
    margin-top: 50px;
}

/* Dark Theme */
body.dark-theme {
    background-color: #333;
    color: #f0f0f0;
}

```

9. JavaScript (script.js):

```

// DOM Elements
const lightBtn = document.getElementById("light-btn");
const darkBtn = document.getElementById("dark-btn");

// Function to apply the selected theme
function applyTheme(theme) {
    if (theme === "dark") {
        document.body.classList.add("dark-theme");
        localStorage.setItem("theme", "dark");
    } else {
        document.body.classList.remove("dark-theme");
        localStorage.setItem("theme", "light");
    }
}

```

```

// Event listeners for the buttons
lightBtn.addEventListener("click", () => {
    applyTheme("light");
});

darkBtn.addEventListener("click", () => {
    applyTheme("dark");
});

// On page load, check if there's a stored theme and apply it
window.onload = function() {
    const savedTheme = localStorage.getItem("theme");
    if (savedTheme) {
        applyTheme(savedTheme);
    }
};

```

কী কী হচ্ছে:

- HTML:** দুটি বাটন দেওয়া আছে—একটি লাইট থিমের জন্য এবং একটি ডার্ক থিমের জন্য।
- CSS:** ডিফল্টভাবে লাইট থিম সেট করা হয়েছে, আর ডার্ক থিমের জন্য একটি ক্লাস (`dark-theme`) দেওয়া হয়েছে, যেটি `body` তে যোগ করা হবে।
- JavaScript:**

- `applyTheme` ফাংশনটি থিম পরিবর্তন করার জন্য এবং `localStorage` এ সেভ করার জন্য ব্যবহৃত হয়।
- যখন ব্যবহারকারী একটি থিম নির্বাচন করে, তা `localStorage` এ সেভ হয় এবং পেজ রিফ্রেশ হওয়ার পর সেই থিম স্বয়ংক্রিয়ভাবে লোড হয়।
- `window.onload` এ চেক করা হয় যদি পূর্বে কোনো থিম সেভ করা থাকে, তাহলে সেটি প্রয়োগ করা হয়।

ব্যবহার:

- যখন ব্যবহারকারী **Light Theme** অথবা **Dark Theme** নির্বাচন করবেন, তা `localStorage` এ সেভ হয়ে যাবে এবং পেজ পুনরায় লোড করার পর তাদের পছন্দ অনুযায়ী থিম স্বয়ংক্রিয়ভাবে

লোড হবে।

এভাবে, আপনি ব্যবহারকারীর পছন্দ অনুযায়ী থিম বা অন্যান্য প্রেফারেন্স সেভ এবং রিট্রিভ করতে পারেন।

4. What is the purpose of the `navigator` object, and what properties does it have?

`navigator` অবজেক্টটি **Browser Object Model (BOM)** এর একটি অংশ এবং এটি ব্যবহারকারীর ব্রাউজার এবং সিস্টেমের তথ্য সংগ্রহ করতে ব্যবহৃত হয়। এর মাধ্যমে আপনি ব্রাউজারের ধরণ, সংস্করণ, প্ল্যাটফর্ম, ল্যাঙ্গুয়েজ এবং অন্যান্য ব্রাউজার সম্পর্কিত তথ্য জানতে পারবেন।

`navigator` অবজেক্টের উদ্দেশ্য:

- ব্রাউজার এবং সিস্টেম তথ্য সংগ্রহ: এটি ডিভাইস বা প্ল্যাটফর্মের বিভিন্ন তথ্য প্রদান করে, যেমন ব্রাউজারের নাম, সংস্করণ, অপারেটিং সিস্টেম, এবং কিছু অন্যান্য তথ্য।
- অনলাইন বা অফলাইন অবস্থান নির্ধারণ: `navigator` অবজেক্টের মাধ্যমে আপনি জানতে পারেন ব্যবহারকারী অনলাইনে আছেন নাকি অফলাইনে।
- ল্যাঙ্গুয়েজ ডিটেকশন: ব্যবহারকারীর ব্রাউজারের ডিফল্ট ভাষা কী, সেটি `navigator` অবজেক্টের মাধ্যমে জানা যায়।

কিছু সাধারণ `navigator` প্রপার্টি:

1. `navigator.appName`

- ব্রাউজারের নাম প্রদান করে। উদাহরণস্বরূপ: "Netscape" বা "Microsoft Internet Explorer"।
- এটি পুরোনো ব্রাউজারগুলোর জন্য খুবই দরকারী, তবে আধুনিক ব্রাউজারে বেশিরভাগ সময় এটি শুধুমাত্র "Netscape" রিটার্ন করে।

উদাহরণ:

```
console.log(navigator.appName); // "Netscape"
```

2. `navigator.appVersion`

- ব্রাউজারের সংস্করণ এবং আরো কিছু সিস্টেম ইনফরমেশন প্রদান করে। এটি একটি স্ট্রিং রিটার্ন করে যা ব্রাউজার এবং সিস্টেমের সংস্করণ সম্পর্কিত তথ্য ধারণ করে।

উদাহরণ:

```
console.log(navigator.appVersion);
```

3. `navigator.userAgent`

- ব্রাউজারের ইউজার এজেন্ট স্ট্রিং প্রদান করে, যা ব্রাউজারের নাম, সংস্করণ, এবং ইনস্টল করা প্ল্যাটফর্মের তথ্য ধারণ করে।
- এটি খুবই উপকারী ব্রাউজারের ডিভাইস, সংস্করণ বা প্ল্যাটফর্ম নির্ধারণ করতে।

উদাহরণ:

```
console.log(navigator.userAgent);
```

4. `navigator.platform`

- ব্যবহারকারীর প্ল্যাটফর্মের তথ্য (যেমন, Windows, Mac, Linux) প্রদান করে।

উদাহরণ:

```
console.log(navigator.platform); // "Win32", "MacIntel", "Linux"
```

5. `navigator.language`

- ব্যবহারকারীর ব্রাউজারের ডিফল্ট ভাষা প্রদান করে (যেমন "en-US", "bn-BD")।

উদাহরণ:

```
console.log(navigator.language); // "en-US"
```

6. `navigator.onLine`

- একটি বুলিয়ান মান রিটার্ন করে যা জানায় ব্যবহারকারী বর্তমানে অনলাইনে আছেন কি না।
`true` যদি ব্যবহারকারী অনলাইনে থাকে এবং `false` যদি অফলাইনে থাকে।

উদাহরণ:

```
if (navigator.onLine) {  
    console.log("You are online!");  
} else {  
    console.log("You are offline.");  
}
```

7. `navigator.geolocation`

- এটি ডিভাইসের জিওলোকেশন (ভৌগলিক অবস্থান) চাওয়ার জন্য ব্যবহৃত হয়। এটি ডিভাইসের ল্যাটিচুড এবং লংগিচুডের মাধ্যমে অবস্থান নির্ধারণ করতে সাহায্য করে।

উদাহরণ:

```
if (navigator.geolocation) {  
    navigator.geolocation.getCurrentPosition(function(position) {  
        console.log(position.coords.latitude, position.coords.longitude);  
    });  
} else {  
    console.log("Geolocation is not supported by this browser.");  
}
```

8. `navigator.cookieEnabled`

- এটি একটি বুলিয়ান মান রিটোর্ন করে যা জানায় কুকি বর্তমানে ব্রাউজারে সক্রিয় কিনা।

উদাহরণ:

```
if (navigator.cookieEnabled) {  
    console.log("Cookies are enabled.");  
} else {  
    console.log("Cookies are disabled.");  
}
```

9. `navigator.deviceMemory`

- এটি ডিভাইসের RAM মেমোরির পরিমাণ রিটার্ন করে, একে গিগাবাইট (GB) এর আকারে রিটার্ন করা হয়।

উদাহরণ:

```
console.log(navigator.deviceMemory); // রিটার্ন হতে পারে: 4, 8, 16 (GB)
```

সারসংক্ষেপ:

- `navigator` অবজেক্টটি ব্রাউজার এবং ব্যবহারকারীর সিস্টেম সম্পর্কিত গুরুত্বপূর্ণ তথ্য সংগ্রহ করতে ব্যবহৃত হয়।
- এটি ব্যবহারকারী অনলাইনে আছেন কিনা, ব্রাউজারের সংস্করণ, ভাষা, প্ল্যাটফর্ম, ডিভাইসের জিওলোকেশন এবং আরও অনেক কিছু নির্ধারণে সাহায্য করে।
- এর সাহায্যে আপনি ওয়েব অ্যাপ্লিকেশন বা ওয়েবসাইটে ব্যবহারকারীর অভিজ্ঞতা আরও ব্যক্তিগতকৃত এবং উপযোগী করে তুলতে পারেন।

5. How do `window.open()` and `window.close()` methods work in JavaScript?

`window.open()` এবং `window.close()` হল JavaScript-এর দুটি গুরুত্বপূর্ণ মেথড, যেগুলি নতুন উইন্ডো খোলার এবং বন্ধ করার জন্য ব্যবহৃত হয়। এই দুটি মেথডকে ব্যবহার করে আপনি ওয়েব পেজের সাথে ইন্টারঅ্যাক্ট করতে পারেন, বিশেষ করে পপআপ উইন্ডো বা নতুন ব্রাউজার ট্যাব চালু বা বন্ধ করতে।

১. `window.open()`

`window.open()` মেথডটি একটি নতুন ব্রাউজার উইন্ডো বা ট্যাব খুলতে ব্যবহার করা হয়। এটি তিনটি প্রধান প্যারামিটার গ্রহণ করে:

- URL:** খোলার জন্য পেজের ইউআরএল। যদি আপনি এটিকে খালি রাখেন, তাহলে একটি নতুন খালি উইন্ডো খুলবে।
- name:** উইন্ডোটির নাম। এটি একটি আইডেন্টিফায়ার হিসেবে কাজ করে, যদি আপনি একই উইন্ডোতে বিভিন্ন পেজ খুলতে চান।

- **specs:** উইন্ডোর বৈশিষ্ট্যগুলির সেট, যেমন সাইজ, অবস্থান, ক্লোলবার এবং অন্যান্য কাস্টমাইজেশন।

Syntax:

```
window.open(URL, name, specs);
```

উদাহরণ:

```
// একটি নতুন উইন্ডো খোলা যেখানে Google এর পেজ দেখা যাবে
window.open("<https://www.google.com>", "_blank", "width=800, height=600");
```

এখানে:

- "<https://www.google.com>" হল ইউআরএল যা নতুন উইন্ডোতে লোড হবে।
- "[_blank](#)" হল উইন্ডোর নাম, যা নির্দেশ করে একটি নতুন ট্যাব বা উইন্ডো খোলা হবে।
- "width=800, height=600" হল উইন্ডোর সাইজের বৈশিষ্ট্য।

২. window.close()

`window.close()` মেথডটি বর্তমান ব্রাউজার উইন্ডো বা ট্যাব বন্ধ করার জন্য ব্যবহৃত হয়। এটি শুধুমাত্র সেই উইন্ডো বা ট্যাব বন্ধ করতে পারে যেটি `window.open()` দিয়ে খোলা হয়েছে। ব্যবহারকারী যদি সাধারণভাবে একটি ব্রাউজার উইন্ডো খুলে থাকে, তবে সে উইন্ডোটি `window.close()` দিয়ে বন্ধ করা যাবে না (সুরক্ষা সীমাবদ্ধতা হিসেবে)।

Syntax:

```
window.close();
```

উদাহরণ:

```
// বর্তমান উইন্ডো বন্ধ করার জন্য
window.close();
```

সংকেত (Important Notes):

- Pop-up Blockers:** অনেক ব্রাউজারে পপ-আপ ব্লকার থাকে, যা পপ-আপ উইন্ডোগুলো ব্লক করে। ব্যবহারকারীর অনুমতি ছাড়া ব্রাউজারের পপ-আপ উইন্ডো খোলার চেষ্টা করলে তা বন্ধ হয়ে যেতে পারে।
- Restrictions:** সাধারণ ব্রাউজার উইন্ডো বা ট্যাব খোলার পরে `window.close()` কাজ করবে না। এটি শুধুমাত্র সেই উইন্ডো বন্ধ করতে পারে যা `window.open()` দিয়ে খোলা হয়েছে।
- User Interaction:** কিছু ব্রাউজারে পপ-আপ বা উইন্ডো খোলার জন্য ইউজার ইন্টারঅ্যাকশন প্রয়োজন হতে পারে, যেমন একটি বাটনে ক্লিক করা। `window.open()` সাধারণত কোনও অটো স্ক্রিপ্ট কাজ করবে না, বরং একটি ইউজার ক্লিক ইভেন্টের মাধ্যমে এটি কার্যকর হতে পারে।

সারসংক্ষেপ:

- `window.open()`: একটি নতুন উইন্ডো বা ট্যাব খোলার জন্য ব্যবহৃত হয় এবং এটি URL, উইন্ডো নাম এবং বৈশিষ্ট্য গ্রহণ করে।
- `window.close()`: বর্তমানে খোলা উইন্ডো বা ট্যাব বন্ধ করতে ব্যবহৃত হয়, তবে এটি শুধুমাত্র `window.open()` দিয়ে খোলা উইন্ডো বন্ধ করতে পারে।

6. Explain how to get the viewport width and height of a browser window using JavaScript.

আপনি JavaScript ব্যবহার করে ব্রাউজারের **viewport** (অথবা দৃশ্যমান এলাকা) এর প্রস্থ (width) এবং উচ্চতা (height) পেতে পারেন। viewport হল সেই অংশ, যেখানে একটি ওয়েব পেজ দৃশ্যমান হয়, যা ব্রাউজারের স্ক্রোলিং এর মাধ্যমে দেখা যায়।

`window.innerWidth` এবং `window.innerHeight`

এই দুটি প্রপার্টি ব্যবহার করে আপনি viewport এর প্রস্থ এবং উচ্চতা পেতে পারেন। এগুলি ব্রাউজারের দৃশ্যমান অংশের প্রস্থ এবং উচ্চতা রিটার্ন করে, যা স্ক্রোলবার অন্তর্ভুক্ত।

- `window.innerWidth`: এটি viewport এর প্রস্থ (width) রিটার্ন করে।
- `window.innerHeight`: এটি viewport এর উচ্চতা (height) রিটার্ন করে।

উদাহরণ:

```
// viewport এর প্রস্থ (width)
console.log("Viewport width: " + window.innerWidth);

// viewport এর উচ্চতা (height)
console.log("Viewport height: " + window.innerHeight);
```

document.documentElement.clientWidth এবং document.documentElement.clientHeight

যদি আপনি শুধুমাত্র **viewport** এর প্রস্থ এবং উচ্চতা চান যা স্ক্রোলবার ছাড়া, তাহলে আপনি **document.documentElement** (যা `<html>` ট্যাগ) ব্যবহার করতে পারেন। এটি স্ক্রোলবার ছাড়া **viewport** এর প্রকৃত আকার প্রদান করে।

- **document.documentElement.clientWidth** : এটি স্ক্রোলবার ছাড়া **viewport** এর প্রস্থ (width) রিটার্ন করে।
- **document.documentElement.clientHeight** : এটি স্ক্রোলবার ছাড়া **viewport** এর উচ্চতা (height) রিটার্ন করে।

উদাহরণ:

```
// viewport এর প্রস্থ (width) স্ক্রোলবার ছাড়া
console.log("Viewport width without scrollbar: " + document.documentElement.clientWidth);

// viewport এর উচ্চতা (height) স্ক্রোলবার ছাড়া
console.log("Viewport height without scrollbar: " + document.documentElement.clientHeight);
```

সারসংক্ষেপ:

- **window.innerWidth** এবং **window.innerHeight** : **viewport** এর আকার স্ক্রোলবারসহ প্রদান করে।
- **document.documentElement.clientWidth** এবং **document.documentElement.clientHeight** : স্ক্রোলবার ছাড়া **viewport** এর আকার প্রদান করে।

এই দুটি পদ্ধতিই **viewport** এর আকার জানার জন্য কার্যকর, তবে আপনি কোনটির ব্যবহার করবেন তা নির্ভর করবে আপনার প্রয়োজনের উপর (যেমন, স্ক্রোলবারসহ বা স্ক্রোলবার ছাড়া)।

Project: Create a function that logs the window's size whenever it's resized.

এই প্রকল্পে একটি ফাংশন তৈরি করা হবে যা ব্রাউজারের উইন্ডো সাইজ যখনই পরিবর্তিত হবে, তখন সাইজটি কনসোলে লগ করবে। আমরা `resize` ইভেন্ট ব্যবহার করে এই ফাংশনটি তৈরি করতে পারব।

কোড:

১. HTML (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Window Resize Logger</title>
</head>
<body>
    <h1>Resize the window and check the console for size logs!
    </h1>

    <script src="script.js"></script>
</body>
</html>
```

২. JavaScript (script.js):

```
// ফাংশন যা উইন্ডোর সাইজ কনসোলে লগ করবে
function logWindowSize() {
    const width = window.innerWidth; // উইন্ডোর প্রস্থ
    const height = window.innerHeight; // উইন্ডোর উচ্চতা
```

```

        console.log(`Window size: ${width}x${height}`);
    }

// উইন্ডো সাইজ পরিবর্তন হলে logWindowSize ফাংশনটি কল হবে
window.addEventListener("resize", logWindowSize);

```

ব্যাখ্যা:

1. `logWindowSize` ফাংশন:

- এই ফাংশনটি `window.innerWidth` এবং `window.innerHeight` ব্যবহার করে উইন্ডোর বর্তমান প্রস্থ এবং উচ্চতা বের করে কনসোলে লগ করবে।

2. `resize` ইভেন্ট:

- `window.addEventListener("resize", logWindowSize)`: এই লাইনে, আমরা `resize` ইভেন্টটি ব্যবহার করছি যা উইন্ডো সাইজ পরিবর্তন হলে `logWindowSize` ফাংশনটি কল করবে।

কীভাবে কাজ করবে:

- যখন ব্যবহারকারী উইন্ডোর সাইজ পরিবর্তন করবে, তখন `resize` ইভেন্ট ট্রিগার হবে এবং `logWindowSize` ফাংশনটি কল হয়ে বর্তমান উইন্ডোর প্রস্থ এবং উচ্চতা কনসোলে প্রিন্ট হবে।

ফলস্বরূপ:

- আপনি যখন ব্রাউজারের উইন্ডো সাইজ পরিবর্তন করবেন, তখন **Console**এ সাইজের তথ্য দেখানো হবে, যেমন:

Window size: 1200x800

এটি একটি সহজ এবং কার্যকর উপায় উইন্ডো রিসাইজ ট্র্যাক করার জন্য।

7. What is the purpose of the `setTimeout` and `setInterval` functions?

`setTimeout` এবং `setInterval` হল JavaScript এর দুটি গুরুত্বপূর্ণ ফাংশন যা নির্দিষ্ট সময় পরে বা নির্দিষ্ট সময় অন্তর কোনো কোড রেক্লামে চালাতে ব্যবহৃত হয়।

১. `setTimeout()`

`setTimeout()` ফাংশনটি একটি নির্দিষ্ট সময় পরে একবার একটি ফাংশন বা কোডের অংশ চালানোর জন্য ব্যবহৃত হয়। এটি একটি একক ইভেন্ট চালাতে সাহায্য করে, যা একবারই নির্দিষ্ট সময় পর সম্পন্ন হয়।

Syntax:

```
setTimeout(function, delay);
```

- `function`: এটি সেই ফাংশন বা কোড যা নির্দিষ্ট সময় পর চলবে।
- `delay`: এটি সময়ের পরিমাণ (মিলিসেকেন্ড) যা পরে ফাংশনটি চালানো হবে।

উদাহরণ:

```
setTimeout(function() {
  console.log("This message appears after 2 seconds.");
}, 2000); // 2000 মিলিসেকেন্ড (2 সেকেন্ড) পর কোড চালাবে।
```

এখানে, "This message appears after 2 seconds." ২ সেকেন্ড পর কনসোলে দেখাবে।

২. `setInterval()`

`setInterval()` ফাংশনটি একটি নির্দিষ্ট সময় অন্তর একটি ফাংশন বা কোডের অংশ চালাতে ব্যবহৃত হয়। এটি পুনরাবৃত্তি হিসেবে কাজ করে, অর্থাৎ যতবার সময় পার হবে, ততবার একই কাজ করবে।

Syntax:

```
setInterval(function, interval);
```

- `function`: এটি সেই ফাংশন বা কোড যা নির্দিষ্ট সময় অন্তর চালানো হবে।
- `interval`: এটি সময়ের পরিমাণ (মিলিসেকেন্ড) যা পরপর একে একে ফাংশনটি চলবে।

উদাহরণ:

```
setInterval(function() {
    console.log("This message appears every 3 seconds.");
}, 3000); // প্রতি 3 সেকেন্ড পর কোড চলবে।
```

এখনে, "This message appears every 3 seconds." প্রতি 3 সেকেন্ড পর কনসোলে দেখাবে।

পার্থক্য:

- `setTimeout`: এটি একবার একটি নির্দিষ্ট সময় পর কোড চালায়। একবারেই কাজ শেষ হয়ে যায়।
- `setInterval`: এটি নির্দিষ্ট সময় অন্তর কোড চালাতে থাকে, যতক্ষণ না এটি `clearInterval()` দিয়ে বন্ধ করা হয়।

`clearTimeout` এবং `clearInterval`

যদি আপনি `setTimeout` বা `setInterval` এর কাজ বন্ধ করতে চান, তাহলে আপনি `clearTimeout` বা `clearInterval` ব্যবহার করতে পারেন।

উদাহরণ:

```
// setTimeout এর কাজ বন্ধ করা
const timeoutId = setTimeout(function() {
    console.log("This will not run.");
}, 5000);
clearTimeout(timeoutId); // এটি timeoutId এর কাজ বন্ধ করে দিবে

// setInterval এর কাজ বন্ধ করা
const intervalId = setInterval(function() {
    console.log("This will stop after 5 seconds.");
}, 1000);
setTimeout(function() {
    clearInterval(intervalId); // 5 সেকেন্ড পর interval বন্ধ হবে
}, 5000);
```

সারসংক্ষেপ:

- `setTimeout()`: একটি নির্দিষ্ট সময় পর একবার একটি কোড চালাতে ব্যবহৃত হয়।

- `setInterval()`: নির্দিষ্ট সময় অন্তর কোড পুনরাবৃত্তি করে চালাতে ব্যবহৃত হয়।
- `clearTimeout()` এবং `clearInterval()`: চলমান `setTimeout()` বা `setInterval()` কাজ বন্ধ করার জন্য ব্যবহার করা হয়।

Project: Create a clock that updates every second using `setInterval`.

এই প্রকল্পে আমরা একটি ডিজিটাল ঘড়ি তৈরি করবো যা প্রতি সেকেন্ডে নিজেকে আপডেট করবে।
আমরা `setInterval()` ব্যবহার করে প্রতি সেকেন্ডে সময় আপডেট করব এবং এটি একটি HTML
পৃষ্ঠায় দেখাবো।

কোড:

১. HTML (index.html):

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Digital Clock</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin-top: 50px;
        }
        #clock {
            font-size: 48px;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <div id="clock"></div>
</body>
</html>
```

```

        </style>
</head>
<body>
    <div id="clock"></div>

    <script src="script.js"></script>
</body>
</html>

```

২. JavaScript (script.js):

```

// ফাংশন যা বর্তমান সময় আপডেট করবে
function updateClock() {
    const now = new Date(); // বর্তমান তারিখ ও সময় নিয়ে আসবে
    let hours = now.getHours(); // ঘণ্টা
    let minutes = now.getMinutes(); // মিনিট
    let seconds = now.getSeconds(); // সেকেন্ড

    // যদি ঘণ্টা, মিনিট বা সেকেন্ড একক ডিজিটে হয়, তাহলে তার আগে 0 যোগ
    করা হবে
    hours = (hours < 10) ? '0' + hours : hours;
    minutes = (minutes < 10) ? '0' + minutes : minutes;
    seconds = (seconds < 10) ? '0' + seconds : seconds;

    // সময়ের ফর্ম্যাট: HH:MM:SS
    const timeString = `${hours}:${minutes}:${seconds}`;

    // ঘড়ির div এ সময় দেখানো
    document.getElementById('clock').textContent = timeString;
}

// প্রতি সেকেন্ডে updateClock ফাংশন কল করা হবে
setInterval(updateClock, 1000);

// প্রথমবারের জন্য ঘড়ি আপডেট করা যাতে পেজ লোডের পর প্রথম সময় দেখতে

```

```
পাওয়া যায়  
updateClock();
```

ব্যাখ্যা:

1. HTML:

- একটি `div` ট্যাগ তৈরি করা হয়েছে যার `id` "clock"। এটি ঘড়ির সময় দেখানোর জন্য ব্যবহৃত হবে।
- `script.js` ফাইলটি **JavaScript** কোড রেন্ডার করতে ব্যবহৃত হবে।

2. CSS:

- `#clock` এর ফন্ট সাইজ এবং স্টাইল দেয়া হয়েছে যাতে ঘড়ির সময় বড় ও স্পষ্ট দেখায়।

3. JavaScript:

- `updateClock` ফাংশনটি:
 - `new Date()` দিয়ে বর্তমান সময় নেয়।
 - `getHours()`, `getMinutes()`, এবং `getSeconds()` ব্যবহার করে ঘণ্টা, মিনিট এবং সেকেন্ড বের করা হয়।
 - `setInterval()` দিয়ে প্রতি সেকেন্ডে `updateClock()` ফাংশনটি কল হয়, যাতে সময় প্রতি সেকেন্ডে আপডেট হয়।
- সময়ের ফরম্যাটে যদি ঘণ্টা, মিনিট বা সেকেন্ড একক ডিজিটে হয়, তবে তার আগে `0` যোগ করা হয়, যেমন `"03:09:05"`।
- প্রথমবার `updateClock()` ফাংশনটি কল করা হয়, যাতে পেজ লোড হওয়ার সাথে সাথে প্রথমবারের সময় দেখা যায়।

ফলস্বরূপ:

- একটি ডিজিটাল ঘড়ি দেখা যাবে যা প্রতি সেকেন্ডে নিজেকে আপডেট করবে এবং বর্তমান সময় দেখাবে, যেমন:

```
12:05:34
```

এটি একটি সহজ এবং কার্যকর ডিজিটাল ঘড়ি তৈরি করার উদাহরণ যা `setInterval()` ব্যবহার করে সময় আপডেট করতে পারে।

8. How can you detect if a user is online or offline using the BOM?

JavaScript ব্যবহার করে Browser Object Model (BOM) এর মাধ্যমে আপনি একটি ব্যবহারকারীর **online** বা **offline** অবস্থান শনাক্ত করতে পারেন। এর জন্য `navigator.onLine` প্রপার্টি এবং `online` এবং `offline` ইভেন্টগুলি ব্যবহার করা হয়।

১. `navigator.onLine`

`navigator.onLine` প্রপার্টি ব্রাউজারকে জানায় যে ব্যবহারকারী বর্তমানে ইন্টারনেটে সংযুক্ত (online) কিনা বা সংযুক্ত নয় (offline)। এটি একটি বুলিয়ান মান প্রদান করে:

- `true` : ব্যবহারকারী ইন্টারনেটের সাথে সংযুক্ত।
- `false` : ব্যবহারকারী ইন্টারনেটের সাথে সংযুক্ত নয়।

উদাহরণ:

```
if (navigator.onLine) {  
    console.log("User is online.");  
} else {  
    console.log("User is offline.");  
}
```

২. `online` এবং `offline` ইভেন্ট

আপনি `online` এবং `offline` ইভেন্ট ব্যবহার করে ব্যবহারকারীর নেটওয়ার্ক অবস্থান পরিবর্তন হলে সেটা শনাক্ত করতে পারেন। এই ইভেন্টগুলি `window` বা `document` অবজেক্টের উপর ত্রিগার হয়, যখন ব্যবহারকারী ইন্টারনেটে সংযুক্ত হয় বা সংযোগ বিচ্ছিন্ন হয়।

- `online` ইভেন্টটি তখন ত্রিগার হয়, যখন ব্যবহারকারী ইন্টারনেটের সাথে সংযুক্ত হয়।
- `offline` ইভেন্টটি তখন ত্রিগার হয়, যখন ব্যবহারকারী ইন্টারনেটের সাথে সংযুক্ত থাকে না।

উদাহরণ:

```

// ব্যবহারকারীর online অবস্থা জানাতে
window.addEventListener('online', function() {
    console.log("User is back online.");
});

// ব্যবহারকারীর offline অবস্থা জানাতে
window.addEventListener('offline', function() {
    console.log("User is offline.");
});

```

পূর্ণ উদাহরণ:

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Online/Offline Detection</title>
</head>
<body>
    <h1 id="status"></h1>

    <script>
        // ফাংশন যা ব্যবহারকারীর অবস্থা আপডেট করবে
        function updateStatus() {
            const statusElement = document.getElementById('stat
us');
            if (navigator.onLine) {
                statusElement.textContent = "You are online.";
                statusElement.style.color = "green";
            } else {
                statusElement.textContent = "You are offline.";
                statusElement.style.color = "red";
            }
        }
    </script>

```

```

    }

    // প্রথমে অবস্থান আপডেট করা
    updateStatus();

    // ইভেন্ট লিসেনার যোগ করা
    window.addEventListener('online', updateStatus);
    window.addEventListener('offline', updateStatus);

```

</script>

</body>

</html>

ব্যাখ্যা:

- `updateStatus()` ফাংশনটি ব্যবহারকারী যদি **online** থাকে, তাহলে "You are online." এবং যদি **offline** থাকে, তাহলে "You are offline." দেখাবে।
- `window.addEventListener('online', updateStatus)` এবং `window.addEventListener('offline', updateStatus)` ব্যবহারকারী যখন **online** বা **offline** হবে, তখন `updateStatus()` ফাংশন কল হবে এবং ব্যবহারকারীর অবস্থা পরিবর্তন হবে।

সারসংক্ষেপ:

- `navigator.onLine`: এটি ব্যবহারকারীর নেটওয়ার্ক সংযোগের অবস্থা জানায় (**online** বা **offline**)।
- `online` এবং `offline` ইভেন্ট: এগুলি ব্যবহারকারী ইন্টারনেটে সংযুক্ত বা সংযোগ বিচ্ছিন্ন হলে ঘটানো হয়, এবং আপনি এই ইভেন্টগুলির মাধ্যমে ব্যবহারের অবস্থা ট্র্যাক করতে পারেন।

>> Web API & JSON (10 questions)

1. What is the Fetch API, and how does it work with Promises?

Fetch API হল একটি ব্রাউজার-বেসড API যা JavaScript-এ HTTP রিকোয়েস্ট পাঠানোর জন্য ব্যবহৃত হয়। এটি **Promises** ব্যবহারের মাধ্যমে অ্যাসিনক্রোনাস (asynchronous) রিকোয়েস্ট পরিচালনা করতে সাহায্য করে এবং ডেটা রিটোর্ন করে। এটি `XMLHttpRequest` এর একটি আধুনিক বিকল্প এবং ক্লিনার সিনট্যাক্স প্রদান করে।

How Fetch API works:

`fetch()` একটি **Promise** রিটোর্ন করে, যা HTTP রিকোয়েস্টের সফল বা ব্যর্থ ফলাফলকে প্রতিফলিত করে।

Syntax:

```
fetch(url, options)
  .then(response => {
    // Handle the response here
  })
  .catch(error => {
    // Handle errors here
  });
}
```

- `url`: এটি সেই রিসোর্সের URL যেটি আপনি রিকোয়েস্ট করতে চান।
- `options` (optional): এটি একটি অবজেক্ট যা রিকোয়েস্টের বিস্তারিত যেমন HTTP মেথড (GET, POST), হেডার, বডি ইত্যাদি নির্ধারণ করে।

Promises and `fetch()`

Promises হল অ্যাসিনক্রোনাস অপারেশনগুলির জন্য ব্যবহৃত একটি অবজেক্ট, যা পরবর্তীতে সফল বা ব্যর্থ হওয়া নির্ধারণ করে। `fetch()` ব্যবহার করলে আপনি `.then()` এবং `.catch()` মেথড ব্যবহার করে রিকোয়েস্টের ফলাফল অথবা ত্রুটি (error) পরিচালনা করতে পারবেন।

- `.then()`: এটি একটি **Promise** সফলভাবে রেজলভ হলে কল হয়, এবং রেসপন্স ডেটা প্রক্রিয়া করতে সাহায্য করে।
- `.catch()`: এটি **Promise** যদি ব্যর্থ হয় (যেমন নেটওয়ার্ক ত্রুটি) তবে এটি কল হয় এবং ত্রুটি হ্যান্ডেল করতে সাহায্য করে।

Basic Example:

```

fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => {
    if (!response.ok) { // চেক করুন রেসপ্লি সঠিক কিনা
      throw new Error('Network response was not ok');
    }
    return response.json(); // রেসপ্লি JSON এ রূপান্তরিত করা
  })
  .then(data => {
    console.log(data); // ডেটা লগ করা
  })
  .catch(error => {
    console.log('There was a problem with the fetch operation:', error);
  });

```

Explanation:

1. `fetch()` একটি HTTP GET রিকোয়েস্ট পাঠায়। এখানে `'https://jsonplaceholder.typicode.com/posts'` > হলো API URL।
2. `.then(response => { ... })`: প্রথম `.then()` রেসপ্লি অবজেক্ট গ্রহণ করে। যদি রেসপ্লি `ok` না হয়, তবে `throw new Error` দিয়ে ত্রুটি সৃষ্টি হবে।
3. `.then(data => { ... })`: দ্বিতীয় `.then()` চেইনে `.json()` মেথড ব্যবহার করে JSON ফরম্যাটে রেসপ্লি পার্স করা হয় এবং পরে ডেটা প্রিন্ট করা হয়।
4. `.catch(error => { ... })`: যদি কোনও ত্রুটি ঘটে, যেমন নেটওয়ার্ক ত্রুটি, তা `.catch()` রেকে হ্যান্ডল করা হয়।

Post Request Example:

```

const data = {
  title: 'foo',
  body: 'bar',
  userId: 1
};

fetch('https://jsonplaceholder.typicode.com/posts', {
  ...
})

```

```

method: 'POST', // HTTP method: POST
headers: {
  'Content-Type': 'application/json'
},
body: JSON.stringify(data) // ডেটা JSON আকারে পাঠানো
})
.then(response => response.json()) // রেসপন্স JSON এ পার্স করা
.then(data => console.log(data)) // রিটার্ন করা ডেটা
.catch(error => console.log('Error:', error)); // অটি হ্যান্ডলিং

```

Explanation:

- `fetch()` এ `method: 'POST'` ব্যবহার করা হয়েছে, যা HTTP POST রিকোয়েস্ট পাঠায়।
- `body: JSON.stringify(data)` দিয়ে ডেটা JSON আকারে পাঠানো হয়।
- `.then(response => response.json())`: রেসপন্স JSON ফরম্যাটে পার্স করা হয়।
- `.catch()`: যদি কোনো অটি ঘটে, তবে তা কনসোলে প্রদর্শিত হবে।

Key Points:

- `fetch()` রিটার্ন করে একটি **Promise**, যা সার্ভার রেসপন্স হওয়ার পর সফল বা ব্যর্থ ফলাফল সরবরাহ করে।
- আপনি `then()` চেইন ব্যবহার করে সফল রেসপন্স হ্যান্ডল করতে পারেন এবং `catch()` দিয়ে অটি হ্যান্ডল করতে পারেন।
- `fetch()` এর সাথে `async` / `await` ব্যবহার করে আরও ক্লিন এবং পাঠযোগ্য কোড লেখা সম্ভব।

async/await Example:

```

async function fetchData() {
  try {
    const response = await fetch('<https://jsonplaceholder.typicode.com/posts>');
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    const data = await response.json();
  }
}

```

```

        console.log(data);
    } catch (error) {
        console.log('Error:', error);
    }
}

fetchData();

```

এখানে, `async/await` ব্যবহার করে কোডটি আরো সহজ এবং পাঠ্যোগ্য হয়েছে।

Summary:

- `fetch()` API একটি Promise রিটার্ন করে যা HTTP রিকোয়েস্ট পাঠায় এবং সেই রিকোয়েস্টের সফল বা ব্যর্থ ফলাফল প্রদান করে।
- **Promises** ব্যবহার করে আপনি অ্যাসিনক্রোনাস অপারেশনগুলির জন্য `.then()` এবং `.catch()` মেথড ব্যবহার করে রেসপন্স এবং ত্রুটি হ্যান্ডলিং করতে পারেন।
- `async/await` ব্যবহার করলে Promise-এর কাজ আরও সোজা এবং ক্লিনভাবে করা যায়।

Project: Fetch data from a public API (e.g., JSONPlaceholder) and display it on the page.

এখানে একটি প্রকল্প তৈরি করা হলো যেখানে আমরা **JSONPlaceholder** API থেকে ডেটা নিয়ে সেটি একটি ওয়েবপেজে প্রদর্শন করব।

প্রকল্পের উদ্দেশ্য:

- **JSONPlaceholder** API থেকে ডেটা (যেমন পোস্ট) নিয়ে সেটি HTML পেজে প্রদর্শন করা।

স্টেপস:

1. **HTML** ফাইল তৈরি করা: পেজের কাঠামো এবং একটি জায়গা যেখানে ডেটা দেখানো হবে।
2. **JavaScript** ফাইল তৈরি করা: `fetch()` API ব্যবহার করে ডেটা নিয়ে তা HTML-এ প্রদর্শন করা।

1. HTML ফাইল (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-
scale=1.0">
    <title>Fetch Data from API</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
        }
        .post {
            border: 1px solid #ddd;
            padding: 20px;
            margin-bottom: 15px;
            border-radius: 8px;
        }
        h2 {
            color: #333;
        }
    </style>
</head>
<body>
    <h1>Posts from JSONPlaceholder</h1>
    <div id="posts-container">
        <!-- Posts will be displayed here -->
    </div>

    <script src="script.js"></script>
</body>
</html>
```

2. JavaScript ফাইল (script.js)

```

// ফাংশন যা API থেকে ডেটা ফেচ করে এবং HTML এ প্রদর্শন করবে
async function fetchPosts() {
    try {
        const response = await fetch('<https://jsonplaceholder.typicode.com/posts>');
        if (!response.ok) {
            throw new Error('Failed to fetch data');
        }

        const posts = await response.json();

        // HTML এ পোস্টগুলো প্রিন্ট করা
        const postsContainer = document.getElementById('posts-container');

        posts.forEach(post => {
            // HTML এলিমেন্ট তৈরি
            const postElement = document.createElement('div');
            postElement.classList.add('post');

            // পোস্টের শিরোনাম এবং বড় যোগ করা
            postElement.innerHTML = `
                <h2>${post.title}</h2>
                <p>${post.body}</p>
            `;

            // পোস্টটিকে কন্টেইনারে অ্যাপেন্ড করা
            postsContainer.appendChild(postElement);
        });
    } catch (error) {
        console.log('Error fetching posts:', error);
    }
}

```

```
// ফাংশনটি কল করা  
fetchPosts();
```

ব্যাখ্যা:

1. HTML ফাইল:

- index.html ফাইলের মধ্যে একটি `div` রয়েছে যার আইডি `posts-container`, এখানে সমস্ত পোস্ট প্রদর্শিত হবে।
- আমরা একটি শিরোনাম দিয়ে ওয়েব পেজ সাজিয়েছি এবং CSS দিয়ে কিছু স্টাইল যোগ করেছি।

2. JavaScript ফাইল:

- `fetchPosts()` নামক একটি অ্যাসিনক্রোনাস ফাংশন ব্যবহার করেছি, যা `fetch()` দিয়ে JSONPlaceholder API থেকে পোস্টগুলোর ডেটা ফেচ করে।
- ডেটা ফেচ করার পর, `response.json()` দিয়ে রেসপন্সটি JSON অবজেক্টে রূপান্তর করা হয় এবং তারপর HTML ডকুমেন্টে প্রদর্শন করা হয়।
- প্রতিটি পোস্টের জন্য একটি `div` তৈরি করা হয়, যেখানে পোস্টের শিরোনাম এবং বিবর প্রদর্শিত হয়।

3. Error Handling:

- যদি কোনো ত্রুটি ঘটে, যেমন নেটওয়ার্ক ইস্যু, তাহলে `catch()` লকে ত্রুটি হ্যান্ডলিং করা হয়েছে।

ফলাফল:

যখন আপনি এই কোডটি ব্রাউজারে রান করবেন, তখন `JSONPlaceholder` থেকে পোস্টের তালিকা ফেচ হবে এবং তা পেজে সুন্দরভাবে প্রদর্শিত হবে।

আপনি এই কোডটি কপি করে আপনার প্রোজেক্টে ব্যবহার করতে পারেন এবং পেজে পোস্টের তালিকা দেখতে পারবেন।

2. How does `XMLHttpRequest` differ from the Fetch API?

`XMLHttpRequest` এবং `Fetch API` উভয়ই JavaScript-এ HTTP রিকোয়েস্ট পাঠানোর জন্য ব্যবহৃত হয়, তবে তাদের মধ্যে কিছু গুরুত্বপূর্ণ পার্থক্য রয়েছে। এখানে উভয়ের মধ্যে মূল পার্থক্যগুলি তুলে ধরা হলো:

1. সিনট্যাক্স ও কোড লেখার পদ্ধতি

- `XMLHttpRequest`: এটি একটি পুরনো এবং কমপ্লেক্স API, যেটি কলব্যাক (callback) ফাংশন ব্যবহার করে অ্যাসিনক্রোনাস রিকোয়েস্ট পরিচালনা করতে হয়। এর জন্য কোড লেখা কিছুটা ভারী এবং জটিল হতে পারে।
- `Fetch API`: এটি আধুনিক API, যেটি **Promises** ব্যবহারের মাধ্যমে অ্যাসিনক্রোনাস রিকোয়েস্ট সমর্থন করে, ফলে কোড লেখা অনেক সহজ এবং পাঠ্যোগ্য।

XMLHttpRequest Example:

```
const xhr = new XMLHttpRequest();
xhr.open("GET", "<https://jsonplaceholder.typicode.com/posts>",
true);
xhr.onreadystatechange = function () {
  if (xhr.readyState === 4 && xhr.status === 200) {
    const data = JSON.parse(xhr.responseText);
    console.log(data);
  }
};
xhr.send();
```

Fetch API Example:

```
fetch('<https://jsonplaceholder.typicode.com/posts>')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.log('Error:', error));
```

2. Return Value (রিটার্ন ভ্যালু)

- `XMLHttpRequest`: এটি একটি অবজেক্ট রিটার্ন করে এবং রিকোয়েস্টের ফলাফল পাওয়ার জন্য একাধিক স্টেট চেক করতে হয় (যেমন `onreadystatechange` ইভেন্ট ব্যবহার করে)।

- `Fetch API` : এটি একটি **Promise** রিটার্ন করে, যা সফল হলে রেসপন্স প্রদান করে এবং ব্যর্থ হলে ত্রুটি (error) প্রদান করে। Promise চেইন ব্যবহার করে সহজে রিকোয়েস্ট পরিচালনা করা যায়।

3. Handling JSON (JSON হ্যান্ডলিং)

- `XMLHttpRequest` : JSON ডেটা হ্যান্ডল করার জন্য আপনাকে ম্যানুয়ালি `JSON.parse()` ব্যবহার করতে হয়।
- `Fetch API` : Fetch API-তে JSON ডেটা সহজেই `.json()` মেথডের মাধ্যমে পার্স করা যায়।

4. Error Handling (ত্রুটি হ্যান্ডলিং)

- `XMLHttpRequest` : Error handling করতে হলে, আপনি `onerror` ইভেন্ট ব্যবহার করতে হবে।
- `Fetch API` : Fetch API-তে error handling `.catch()` মেথডের মাধ্যমে খুব সহজেই করা যায়।

5. CORS (Cross-Origin Resource Sharing)

- `XMLHttpRequest` : Cross-origin (CORS) ক্ষেত্রে, `XMLHttpRequest` কিছু সীমাবদ্ধতা এবং কমপ্লেক্সিটি নিয়ে কাজ করে।
- `Fetch API` : `Fetch` CORS সমর্থন করার জন্য আরও সহজ এবং আরও আধুনিক।

6. Request Cancellation (রিকোয়েস্ট ক্যানসেল করা)

- `XMLHttpRequest` : রিকোয়েস্ট ক্যানসেল করার জন্য `abort()` মেথড ব্যবহার করা হয়।
- `Fetch API` : `Fetch` API-তে রিকোয়েস্ট ক্যানসেল করার জন্য `AbortController` ব্যবহার করা হয়।

7. Stream (স্ট্রিমিং ডেটা)

- `XMLHttpRequest` : এটি স্ট্রিমিং ডেটা হ্যান্ডল করতে সমস্যা তৈরি করতে পারে এবং এটি ফাইল আপলোড/ডাউনলোডের জন্য কম্পেল্ক্স।
- `Fetch API` : Fetch API ডেটা স্ট্রিমিং সহজে সমর্থন করে এবং এটি `ReadableStream` ব্যবহার করে।

8. Browser Compatibility (ব্রাউজার কমপ্যাচিবিলিটি)

- `XMLHttpRequest` : এটি ব্রাউজারের পুরনো সংস্করণেও সমর্থিত এবং অধিকাংশ ব্রাউজারে কমপ্যাচিবল।
- `Fetch API` : এটি নতুন API এবং কিছু পুরনো ব্রাউজারে (যেমন Internet Explorer) সমর্থিত নয়। তবে, বর্তমানে এটি আধুনিক ব্রাউজারে ব্যাপকভাবে সমর্থিত।

Summary of Differences:

Feature	XMLHttpRequest	Fetch API
Syntax	Verbose, Callback-based	Simple, Promise-based
Return Value	XMLHttpRequest object	Promise
JSON Handling	Manually parse with <code>JSON.parse()</code>	Automatically with <code>.json()</code>
Error Handling	<code>onerror</code> event	<code>.catch()</code> method
CORS Support	Limited and complex	Native CORS support
Request Cancellation	<code>abort()</code> method	<code>AbortController</code>
Stream Support	Less efficient	Efficient stream handling
Browser Compatibility	Works in most browsers (even older)	Supported in modern browsers, but not older ones (e.g., IE)

Conclusion:

- `XMLHttpRequest` একটি পুরনো এবং কমপ্লেক্স পদ্ধতি, যা এখনও কিছু জায়গায় ব্যবহৃত হয়।
- `Fetch API` একটি আধুনিক, সহজ এবং পরিষ্কার পদ্ধতি যা প্রতিদিনের ওয়েব ডেভেলপমেন্টে ব্যাপকভাবে ব্যবহৃত হচ্ছে, এবং এটি উন্নত ফিচার এবং ব্যাবহারযোগ্যতা সরবরাহ করে।

তবে, যদি ব্রাউজার কমপ্যাচিভিলিটির সমস্যা থাকে (যেমন Internet Explorer), তাহলে

`XMLHttpRequest` ব্যবহার করা যেতে পারে, কিন্তু আধুনিক ডেভেলপমেন্টের জন্য `Fetch API` পছন্দনীয়।

3. Explain what JSON is and how it's used in JavaScript.

JSON (JavaScript Object Notation) হল একটি সাধারণ, পাঠ্যোগ্য ডেটা বিনিময় ফরম্যাট যা মূলত **জাভাস্ক্রিপ্ট** অবজেক্টের মতো দেখতে, কিন্তু এটি ভাষা-নিরপেক্ষ এবং অন্যান্য ভাষাতেও ব্যবহৃত হয়। JSON সাধারণত ডেটা স্টোর বা ট্রালফার করার জন্য ব্যবহৃত হয়, বিশেষত ওয়েব সার্ভার এবং ন্লাইনেট (যেমন ব্রাউজার) এর মধ্যে।

JSON এর গঠন:

JSON একটি টেক্সট ফরম্যাট যা সাধারণত দুটি ডেটা স্ট্রাকচার ব্যবহার করে:

- অবজেক্ট (Object): `{}` দিয়ে বেষ্টিত একটি কীগুলির (keys) এবং মানগুলির (values) জোড়া (pair)।
- অ্যারেগ (Array): `[]` দিয়ে বেষ্টিত মানগুলির একটি তালিকা।

JSON উদাহরণ:

```
{  
    "name": "John",  
    "age": 30,  
    "isStudent": false,  
    "courses": ["Math", "Science", "History"]  
}
```

এখানে:

- `"name"`, `"age"`, `"isStudent"`, `"courses"` হল কীগুলি (keys), এবং তাদের মানগুলো (values) যথাক্রমে `"John"`, `30`, `false`, এবং একটি অ্যারে `["Math", "Science", "History"]`।
- JSON ফরম্যাটে সব কীগুলি ডাবল কোটস `" "` দিয়ে ঘেরা থাকে।

JSON এর সুবিধা:

- পাঠ্যোগ্যতা:** JSON মানুষের জন্যও সহজে পড়া যায়।
- হালকা:** এটি ছোট এবং কমপ্লেক্স নয়, যা দ্রুত নেটওয়ার্কে ট্রান্সফার করা যায়।
- ভাষা-নিরপেক্ষ:** JSON অন্যান্য প্রোগ্রামিং ভাষায় (যেমন Python, PHP, Ruby) ব্যবহৃত হতে পারে, কিন্তু এটি জাভাস্ক্রিপ্টের জন্য বিশেষভাবে উপযুক্ত।
- ডেটা আদান প্রদান:** ওয়েব অ্যাপ্লিকেশনগুলিতে ক্লায়েন্ট এবং সার্ভারের মধ্যে ডেটা আদান প্রদান করার জন্য JSON বেশ জনপ্রিয়।

জাভাস্ক্রিপ্ট JSON ব্যবহার:

JSON-কে জাভাস্ক্রিপ্ট ব্যবহার করার জন্য দুটি মূল মেথড রয়েছে:

- `JSON.parse()`: এটি একটি JSON স্ট্রিংকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করে।
- `JSON.stringify()`: এটি একটি জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তর করে।

1. `JSON.parse()`

`JSON.parse()` মেথড ব্যবহার করে আপনি একটি JSON স্ট্রিংকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করতে পারেন।

```
const jsonString = '{"name": "John", "age": 30}';  
const parsedObject = JSON.parse(jsonString);  
  
console.log(parsedObject); // { name: 'John', age: 30 }  
console.log(parsedObject.name); // 'John'
```

এখানে, `jsonString` একটি JSON ফর্ম্যাটে স্ট্রিং, এবং `JSON.parse()` এটিকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তরিত করেছে।

2. `JSON.stringify()`

`JSON.stringify()` মেথড ব্যবহার করে আপনি একটি জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তর করতে পারেন।

```
const person = {  
    name: "Alice",  
    age: 25,  
    city: "New York"  
};  
  
const jsonString = JSON.stringify(person);  
console.log(jsonString); // '{"name": "Alice", "age": 25, "city": "New York"}'
```

এখানে, `person` একটি জাভাস্ক্রিপ্ট অবজেক্ট, এবং `JSON.stringify()` এটি JSON স্ট্রিংয়ে রূপান্তরিত করেছে।

JSON এর ব্যবহার:

1. **API থেকে ডেটা ফেচ করা:** ওয়েব API থেকে ডেটা ফেচ করার সময় JSON এক্সচেঞ্চ করা হয়। ওয়েব সার্ভার থেকে JSON রেসপন্স পাওয়ার পর, আপনি `JSON.parse()` ব্যবহার করে সেটি জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করতে পারেন।

```

fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json()) // JSON.parse() হয়ে যায় স্ব
য়ত্ক্রিয়ভাবে
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));

```

- LocalStorage এ ডেটা সংরক্ষণ:** JSON.stringify() ব্যবহার করে আপনি জাভাস্ক্রিপ্ট অবজেক্টকে স্টোরেজে সংরক্ষণ করতে পারেন এবং JSON.parse() ব্যবহার করে পরবর্তীতে তা পুনরুদ্ধার করতে পারেন।

```

// Save data to localStorage
const user = { name: "John", age: 30 };
localStorage.setItem('user', JSON.stringify(user));

// Retrieve data from localStorage
const storedUser = JSON.parse(localStorage.getItem('user'));
console.log(storedUser); // { name: 'John', age: 30 }

```

JSON এর সীমাবদ্ধতা:

- JSON শুধু ডেটা বিনিময়ের জন্য ব্যবহৃত হয়, ফাংশন বা মেথড স্টোর করা যায় না।
- JSON স্ট্রিংয়ে শুধুমাত্র স্ট্রিং, নম্বর, বুলিয়ান, অ্যারেগ এবং **অবজেক্ট** থাকতে পারে; ফাংশন বা `undefined` স্টোর করা যায় না।

Summary:

- JSON** হল একটি হালকা, পাঠ্যোগ্য ডেটা ফরম্যাট যা JavaScript অবজেক্টের মতো দেখতে।
- `JSON.parse()` ব্যবহার করে JSON স্ট্রিংকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করা হয়, এবং `JSON.stringify()` দিয়ে জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তর করা হয়।
- JSON ওয়েব অ্যাপ্লিকেশনগুলোতে ডেটা বিনিময়ের একটি গুরুত্বপূর্ণ অংশ হিসেবে ব্যবহৃত হয়।

4. How do you parse JSON data in JavaScript, and how do you stringify JavaScript objects?

JSON ডেটা পার্স (parse) এবং স্ট্রিংফাই (stringify) করার পদ্ধতি জাভাস্ক্রিপ্ট খুবই সহজ এবং দুটি মূল মেথড ব্যবহার করে তা করা হয়: `JSON.parse()` এবং `JSON.stringify()`। এখানে কীভাবে এগুলি কাজ করে এবং আপনি কিভাবে এগুলি ব্যবহার করবেন তা বিস্তারিতভাবে ব্যাখ্যা করা হলো।

1. JSON Data Parsing (JSON ডেটা পার্সিং)

`JSON.parse()` মেথড ব্যবহার করে আপনি JSON স্ট্রিংকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করতে পারেন। এটি সাধারণত API থেকে প্রাপ্ত JSON ডেটা রূপান্তর করতে ব্যবহৃত হয়।

Syntax:

```
JSON.parse(jsonString);
```

- `jsonString`: যে স্ট্রিংটি JSON ফরম্যাটে লেখা রয়েছে, এটি একটি বৈধ JSON স্ট্রিং হতে হবে।

Example:

```
const jsonString = '{"name": "John", "age": 30, "city": "New York"}';
const person = JSON.parse(jsonString);

console.log(person); // { name: 'John', age: 30, city: 'New York' }
console.log(person.name); // 'John'
console.log(person.age); // 30
```

এখানে:

- `jsonString` একটি JSON স্ট্রিং যা পার্স করার মাধ্যমে `person` নামে একটি জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তরিত হয়েছে।
- আপনি JSON স্ট্রিংকে অবজেক্টে রূপান্তর করতে `JSON.parse()` ব্যবহার করেছেন।

2. Stringifying JavaScript Objects (জাভাস্ক্রিপ্ট অবজেক্ট স্ট্রিংফাই করা)

`JSON.stringify()` মেথড ব্যবহার করে আপনি একটি জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তর করতে পারেন। এটি সাধারণত ডেটা স্টোর করার জন্য বা ওয়েব সার্ভারে পাঠানোর জন্য ব্যবহৃত হয়।

Syntax:

```
JSON.stringify(obj);
```

- `obj`: যে জাভাস্ক্রিপ্ট অবজেক্ট বা অ্যারে আপনি JSON স্ট্রিংয়ে রূপান্তর করতে চান।

Example:

```
const person = {
  name: "Alice",
  age: 25,
  city: "London"
};

const jsonString = JSON.stringify(person);

console.log(jsonString); // '{"name": "Alice", "age": 25, "city": "London"}'
```

এখানে:

- `person` একটি জাভাস্ক্রিপ্ট অবজেক্ট, এবং `JSON.stringify()` এটি JSON স্ট্রিংয়ে রূপান্তরিত করেছে।
- `jsonString` হবে একটি স্ট্রিং যা JSON ফরম্যাটে থাকবে।

কীভাবে ব্যবহার করবেন?

1. API থেকে ডেটা ফেচ করা: JSON ডেটা সাধারণত API থেকে আসে এবং তা পার্স করার জন্য `JSON.parse()` ব্যবহার করা হয়।
2. ডেটা স্টোর করা: আপনি জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তর করতে পারেন এবং পরে সেটি স্টোরেজে সংরক্ষণ করতে পারেন (যেমন `localStorage` বা `sessionStorage`)।

Complete Example (API থেকে JSON ফেচ করা এবং স্ট্রিংফাই করা):

```
// Step 1: JSON ফেচ করা (এটি সাধারণত API থেকে আসে)
fetch('<https://jsonplaceholder.typicode.com/posts>')
  .then(response => response.json()) // রেসপ্লানকে JSON অবজেক্টে
```

রূপান্তর

```
.then(data => {
    console.log('Fetched Data:', data);

    // Step 2: JSON অবজেক্টকে স্ট্রিংফাই করা
    const jsonString = JSON.stringify(data);
    console.log('Stringified Data:', jsonString);
})
.catch(error => console.error('Error:', error));
```

এখানে:

- প্রথমে `fetch()` API দিয়ে JSON ডেটা ফেচ করা হয়েছে।
- তারপর `response.json()` দিয়ে JSON ডেটা জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করা হয়েছে।
- অবশেষে `JSON.stringify()` ব্যবহার করে এই অবজেক্টটিকে আবার JSON স্ট্রিংয়ে রূপান্তর করা হয়েছে।

Important Notes:

- `JSON.parse()` শুধুমাত্র বৈধ JSON স্ট্রিং পার্স করতে পারে, এবং যদি স্ট্রিংটি সঠিকভাবে ফরম্যাট করা না থাকে, তবে এটি `SyntaxError` তৈরি করবে।
- `JSON.stringify()` দিয়ে কিছু জাভাস্ক্রিপ্ট ডেটা (যেমন ফাংশন বা `undefined`) স্ট্রিংফাই করা সম্ভব নয়, এবং তা `null` হিসেবে দেখাবে।

সারাংশ:

- `JSON.parse()`: JSON স্ট্রিংকে জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তরিত করে।
- `JSON.stringify()`: জাভাস্ক্রিপ্ট অবজেক্টকে JSON স্ট্রিংয়ে রূপান্তরিত করে।

এগুলি ওয়েব অ্যাপ্লিকেশনগুলিতে ডেটা আদান-প্রদান এবং স্টোরেজ ব্যবহারের ক্ষেত্রে অত্যন্ত গুরুত্বপূর্ণ এবং দরকারী টুল।

Project: Create a function that converts an object to JSON and back to an object.

এখনে একটি ফাংশন তৈরি করা হলো যা একটি অবজেক্টকে JSON-এ রূপান্তর করবে এবং তারপর সেই JSON থেকে আবার অবজেক্টে রূপান্তর করবে।

Step-by-Step Solution:

- ফাংশন তৈরি করুন যা অবজেক্ট গ্রহণ করবে।
- অবজেক্টিকে JSON স্ট্রিংয়ে রূপান্তর করুন `JSON.stringify()` দিয়ে।
- JSON স্ট্রিংটিকে আবার অবজেক্টে রূপান্তর করুন `JSON.parse()` দিয়ে।
- রূপান্তরিত অবজেক্টটি রিটার্ন করুন।

Code Example:

```
// Function to convert object to JSON and back to object
function convertObjectToJsonAndBack(obj) {
    // Convert the object to JSON string
    const jsonString = JSON.stringify(obj);
    console.log("Converted to JSON:", jsonString);

    // Convert the JSON string back to object
    const parsedObject = JSON.parse(jsonString);
    console.log("Converted back to object:", parsedObject);

    return parsedObject;
}

// Example object
const person = {
    name: "John",
    age: 30,
    city: "New York"
};

// Call the function
const newObject = convertObjectToJsonAndBack(person);
```

ব্যাখ্যা:

1. `JSON.stringify()`: প্রথমে অবজেক্টিকে JSON স্ট্রিংয়ে রূপান্তরিত করেছি। এটি একটি টেক্সট ফরম্যাটে পরিণত হয় যা স্টোর বা পাঠানো সহজ হয়।
2. `JSON.parse()`: এরপর JSON স্ট্রিংটিকে আবার জাভাস্ক্রিপ্ট অবজেক্টে রূপান্তর করেছি, যাতে আপনি ডেটাকে কাজে লাগাতে পারেন।

Output:

```
Converted to JSON: {"name": "John", "age": 30, "city": "New York"}  
Converted back to object: { name: 'John', age: 30, city: 'New York' }
```

ব্যবহার:

এই ফাংশনটি যেকোনো অবজেক্টের জন্য কাজ করবে এবং সেই অবজেক্টিকে JSON ফরম্যাটে রূপান্তরিত করে তা আবার অবজেক্টে ফিরে পাবে।

5. What is CORS, and why do we need it when making API requests?

CORS (Cross-Origin Resource Sharing) হল একটি নিরাপত্তা বৈশিষ্ট্য যা ওয়েব ব্রাউজারগুলিকে নিয়ন্ত্রণ করতে সাহায্য করে যে কোন উৎস (origin) থেকে একটি ওয়েব পেজ বা ওয়েব অ্যাপ্লিকেশন ডেটা ফেচ করতে পারে বা সার্ভারে রিকোয়েস্ট পাঠাতে পারে।

CORS কী?

CORS হল একটি ব্রাউজার-মুখী নিরাপত্তা নীতি যা একটি ওয়েব পেজকে অনুমতি দেয় অন্য ডোমেইন (origin) থেকে ডেটা বা রিসোর্স অ্যাক্সেস করার জন্য, যদি সার্ভার সেই ডোমেইনকে অনুমতি দেয়।

একটি "**origin**" বলতে বোঝায়, একটি ওয়েব পেজের প্রটোকল (HTTP/HTTPS), ডোমেইন (যেমন, `example.com`), এবং পোর্ট (যেমন, `8080`) এর সমষ্টি। যদি কোনো ওয়েব পেজের স্ক্রিপ্ট (যেমন, জাভাস্ক্রিপ্ট) অন্য ডোমেইন থেকে ডেটা ফেচ করার চেষ্টা করে, তবে CORS পলিসি সক্রিয় হয়ে যায় এবং এটি সেই ডোমেইনের সার্ভারকে চেক করে।

কেন CORS প্রয়োজন?

CORS প্রয়োজন কারণ:

- নিরাপত্তা:** ওয়েব অ্যাপ্লিকেশনগুলিকে শুধু তাদের নিজস্ব ডোমেইনে থাকা ডেটা অ্যাক্সেস করার অনুমতি দেয়া উচিত। অন্য ডোমেইন থেকে ডেটা অ্যাক্সেস করতে দিলে, এটি সিকিউরিটি ঝুঁকি তৈরি করতে পারে, যেমন CSRF (Cross-Site Request Forgery) আক্রমণ বা ম্যালওয়্যার ইনজেকশন।
- Cross-Origin Requests:** আধুনিক ওয়েব অ্যাপ্লিকেশনগুলো সাধারণত বিভিন্ন উৎস থেকে ডেটা ফেচ করে (যেমন, API থেকে ডেটা নিয়ে আসে)। কিন্তু ওয়েব ব্রাউজার, নিরাপত্তার কারণে, কেবলমাত্র একটি পেইজের নিজস্ব উৎস (origin) থেকে আসা রিকোয়েস্টে অনুমতি দেয়। CORS এই নিয়মকে মেনে চলতে সাহায্য করে এবং API সার্ভারকে নির্দিষ্ট শর্তে ক্রস-অরিজিন রিকোয়েস্টে অনুমতি দেয়।

CORS কিভাবে কাজ করে?

ধরা যাক, আপনি একটি ওয়েব পেজ থেকে `example.com` API-তে ডেটা পাঠানোর চেষ্টা করছেন। ব্রাউজার প্রথমে একটি "preflight" রিকোয়েস্ট পাঠায় যাতে API সার্ভার জানতে পারে এই রিকোয়েস্টটি ক্রস-অরিজিন। যদি সার্ভার সেই উৎসকে অনুমতি দেয়, তখন মূল রিকোয়েস্ট পাঠানো হয়।

Preflight Request:

- ব্রাউজার HTTP `OPTIONS` রিকোয়েস্ট পাঠায় API সার্ভারে, যেখানে সেটি জানায় কোন মেথড এবং হেডার ব্যবহার করা হচ্ছে।
- যদি সার্ভার অনুমতি দেয়, তাহলে **CORS headers** (যেমন `Access-Control-Allow-Origin`) রেসপন্সে পাঠায়।

CORS Headers:

এগুলো সার্ভার থেকে ফিরে আসা হেডার যা ব্রাউজারকে বলে কোন উৎসকে (origin) রিকোয়েস্ট পাঠানোর অনুমতি দেয়া হয়েছে:

- `Access-Control-Allow-Origin`**: এটি সেট করে যে কোন উৎস (origin) রিকোয়েস্ট পাঠাতে পারবে।
- `Access-Control-Allow-Methods`**: এটি নির্ধারণ করে কোন HTTP মেথড (যেমন, `GET`, `POST`, `PUT`, ইত্যাদি) অনুমোদিত।
- `Access-Control-Allow-Headers`**: এটি জানায়, কোন কাস্টম হেডারগুলো অনুমোদিত।

Example:

API Response Headers:

```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET, POST, PUT
Access-Control-Allow-Headers: Content-Type
```

CORS উদাহরণ:

ধরা যাক, আপনি একটি ওয়েব অ্যাপ্লিকেশন চালাচ্ছেন যা <https://example.com> API থেকে ডেটা ফেচ করতে চায়:

```
fetch('<https://example.com/api/data>')
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(error => console.error('Error:', error));
```

এই রিকোয়েস্টটি যদি example.com API সার্ভার থেকে আসে, তবে ব্রাউজার প্রথমে সার্ভারে একটি **preflight** রিকোয়েস্ট পাঠাবে। যদি সার্ভার তার `Access-Control-Allow-Origin` হেডারে আপনার ডোমেইন বা একটি `*` (সব ডোমেইনের জন্য) নির্দেশ করে, তবে ব্রাউজার মূল রিকোয়েস্টটি সম্পর্ক করবে।

CORS এর সমস্যাগুলি:

- **CORS Error:** যদি সার্ভার থেকে CORS হেডার সঠিকভাবে কনফিগার করা না থাকে, তাহলে ব্রাউজার একটি CORS error দেখাবে, যেমন:

```
Access to fetch at '<https://example.com/api/data>' from origin '<https://yourdomain.com>' has been blocked by CORS policy.
```

- **No CORS Support:** কিছু সার্ভার CORS সাপোর্ট করে না, তখন API কল করার চেষ্টা করলে ব্রাউজার এই রিকোয়েস্টটি ব্লক করে দেয়।

CORS কীভাবে সমাধান করা যায়?

CORS সমস্যা সমাধানের জন্য, সার্ভারের পাশে কিছু কনফিগারেশন করা লাগে:

1. **Access-Control-Allow-Origin:** এই হেডার দিয়ে অনুমতি দেয়া হয় যে কোন ডোমেইন রিকোয়েস্ট পাঠাতে পারবে।

2. **Access-Control-Allow-Methods:** কোন HTTP মেথডগুলি অনুমোদিত হবে।

3. **Access-Control-Allow-Headers:** কাস্টম হেডারগুলো অনুমোদন করতে।

সারাংশ:

CORS হল একটি নিরাপত্তা বৈশিষ্ট্য যা ব্রাউজারের মাধ্যমে ওয়েবের পেজগুলোকে অন্য ডোমেইন থেকে ডেটা বা রিসোর্স অ্যাক্সেসের অনুমতি দেয়। এটি নিরাপত্তার জন্য অত্যন্ত গুরুত্বপূর্ণ, কারণ এটি ক্রস-অরিজিন রিকোয়েস্ট সার্ভারের পক্ষ থেকে অনুমতি চায়, যেন সার্ভার জানে কোন উৎস থেকে রিকোয়েস্ট আসছে এবং সেটা গ্রহণযোগ্য কিনা।

6. Explain how the `FormData` API works and when you'd use it.

FormData API হল একটি জাভাস্ক্রিপ্ট API যা HTML ফর্মের ডেটা সংগ্রহ এবং ফর্ম-ভিত্তিক ডেটা POST রিকোয়েস্ট পাঠানোর জন্য ব্যবহৃত হয়। এটি মূলত ফর্মের ইনপুট ফিল্ডের মান সংগ্রহ করতে সহায়তা করে এবং সেগুলোকে **key-value** পেয়ার হিসেবে একটি **FormData** অবজেক্টে রূপান্তর করে। এই অবজেক্টটিকে পরে ফর্ম ডেটা সাবমিট করার জন্য ব্যবহার করা যায়, বিশেষ করে যখন আপনি AJAX বা **fetch** API-এর মাধ্যমে ডেটা পাঠাতে চান।

FormData কীভাবে কাজ করে?

FormData API মূলত ফর্মের ডেটা সংগ্রহ করতে সহায়তা করে, এবং এটি ব্রাউজারকে সুনির্দিষ্ট ফর্ম ডেটা সংগ্রহ, প্রসেস এবং পাঠানোর সুযোগ দেয়। ফর্মের `input`, `select`, `textarea` সহ অন্যান্য ইনপুট এলিমেন্ট থেকে ডেটা নিয়ে একটি **FormData** অবজেক্টে জমা করা হয়।

FormData ব্যবহারের পদ্ধতি:

1. **FormData** অবজেক্ট তৈরি করা:

আপনি একটি নতুন

FormData অবজেক্ট তৈরি করতে পারেন ফর্ম এলিমেন্টের মাধ্যমে বা কেবল ফর্মের ডেটা দিয়ে।

Syntax:

```
let formData = new FormData(formElement);
```

অথবা, আপনি **key-value** পেয়ার ব্যবহার করেও ডেটা যোগ করতে পারেন।

```
let formData = new FormData();
formData.append('key', 'value');
```

2. ফর্ম থেকে ডেটা সংগ্রহ:

যখন আপনি একটি ফর্ম এলিমেন্ট পাস করেন, ফর্মের সব ইনপুট এলিমেন্টের ডেটা স্বয়ংক্রিয়ভাবে FormData অবজেক্টে অন্তর্ভুক্ত হয়ে যায়।

Example:

```
// ফর্ম এলিমেন্ট নির্বাচন
const form = document.querySelector('form');

// FormData অবজেক্ট তৈরি
const formData = new FormData(form);

// FormData অবজেক্টের মধ্যে থাকা ডেটা দেখতে
for (let [key, value] of formData.entries()) {
  console.log(`${key}: ${value}`);
}
```

3. ডেটা সংগ্রহ এবং API কল করা:

আপনি

fetch বা **XMLHttpRequest** ব্যবহার করে এই **FormData** অবজেক্টকে API রিকোয়েস্টে পাঠাতে পারেন।

Example (fetch API ব্যবহার করে):

```
const form = document.querySelector('form');
const formData = new FormData(form);

fetch('<https://example.com/submit>', {
  method: 'POST',
  body: formData
})
.then(response => response.json())
```

```
.then(data => console.log('Success:', data))
.catch(error => console.error('Error:', error));
```

FormData API এর উপকারিতা:

- সহজে ফর্ম ডেটা সংগ্রহ করা: ফর্মের সব ইনপুট (যেমন, `input`, `select`, `textarea`) থেকে ডেটা সংগ্রহ করার জন্য সহজ উপায় প্রদান করে।
- ফাইল আপলোড করা: ফর্মের মধ্যে ফাইল ইনপুট থাকলে **FormData** ফাইল ডেটা এবং অন্যান্য ইনপুট ডেটা একত্রে পাঠাতে সহায়তা করে।
- AJAX রিকোয়েস্টের জন্য উপযোগী: ফর্মের ডেটা AJAX রিকোয়েস্টে সাবমিট করা সহজ করে, বিশেষ করে যখন `fetch` বা `XMLHttpRequest` ব্যবহৃত হয়।

FormData API এর সুবিধা:

- ডেটা সংগ্রহ করা সহজ: ফর্মের ইনপুট ফিল্ডগুলোর মান নিয়ে কাজ করার জন্য আলাদা করে কোড লেখার প্রয়োজন হয় না। শুধুমাত্র ফর্ম এলিমেন্ট পাস করলে সব ডেটা একত্রিত হয়ে যায়।
- ফাইল আপলোড: ফর্মে থাকা ফাইল ইনপুট ডেটা সঠিকভাবে **multipart/form-data** ফর্ম্যাটে পাঠানো সম্ভব।
- ডেটা অটোমেটিকালি সার্ভারে পাঠানো: এটি ইনপুট ডেটা গুলোকে সার্ভারে পাঠানোর জন্য প্রস্তুত করে, এবং আপনি এটির মাধ্যমে ডেটা অ্যাপ্লিকেশন-লেভেলে সহজে প্রোসেস করতে পারেন।

কবে ব্যবহার করবেন?

FormData API ব্যবহার করার কিছু সাধারণ ক্ষেত্র:

- ফর্ম ডেটা AJAX দিয়ে সাবমিট করা: যখন আপনি ফর্ম ডেটা সাবমিট করতে চান কিন্তু পেজ রিলোড করতে চান না।
- ফাইল আপলোড: ফর্মের মাধ্যমে ফাইল আপলোড করার জন্য **FormData** একটি অত্যন্ত গুরুত্বপূর্ণ টুল।
- বহু ইনপুট ডেটা পাঠানো: যখন একাধিক ইনপুট ফিল্ডের ডেটা একসাথে সংগ্রহ এবং পাঠানোর প্রয়োজন হয়।

Example: ফাইল আপলোড সহ ফর্ম ডেটা সাবমিট

```
<form id="myForm">
  <input type="text" name="name" placeholder="Enter your name">
```

```

    required>
    <input type="file" name="file" required>
    <button type="submit">Submit</button>
</form>

<script>
  const form = document.getElementById('myForm');

  form.addEventListener('submit', (event) => {
    event.preventDefault(); // ফর্ম সাবমিট করা থেকে বিরত থাকুন

    const formData = new FormData(form);

    // ফাইল ডেটার জন্য সঠিকভাবে FormData ব্যবহার করা হচ্ছে
    fetch('<https://example.com/upload>', {
      method: 'POST',
      body: formData
    })
    .then(response => response.json())
    .then(data => console.log('File uploaded successfully:', data))
    .catch(error => console.error('Error uploading file:', error));
  });
</script>

```

সারাংশ:

FormData API একটি জাভাস্ক্রিপ্ট API যা ফর্ম ডেটা সংগ্রহ এবং পঠনযোগ্য ফর্ম্যাটে (যেমন JSON বা multipart) ডেটা পাঠানোর জন্য ব্যবহৃত হয়। এটি ফাইল আপলোড, AJAX রিকোয়েস্ট এবং ফর্ম ডেটা সাবমিট করার জন্য অত্যন্ত উপযোগী।

7. What is the purpose of the `History API`, and how do `pushState` and `replaceState` work?

History API একটি ব্রাউজার API যা আপনাকে ওয়েব পেজের ইতিহাস ম্যানিপুলেট করতে দেয়, যেমন: URL পরিবর্তন করা, ব্রাউজারের পিছনে/সামনে নেভিগেট করা এবং নতুন স্টেট যোগ করা, অথচ পেজ পুনরায় লোড না করে। এটি সাধারণত **single-page applications (SPA)**-এ ব্যবহৃত হয়, যেখানে পেজ রিলোড ছাড়া ইউজার ইন্টারফেসের মধ্যে পরিবর্তন করা হয়।

History API এর মূল উদ্দেশ্য:

- URL ম্যানিপুলেশন:** আপনাকে ব্রাউজারের URL পরিবর্তন করতে দেয়, পেজ রিলোড না করেই।
- হিস্ট্রি স্ট্যাক ম্যানেজমেন্ট:** এটি ব্রাউজারের ইতিহাস স্ট্যাক নিয়ন্ত্রণ করতে দেয়, যেমন নতুন স্টেট যোগ করা এবং পুরনো স্টেটগুলোর সাথে ইন্টারঅ্যাক্ট করা।
- নেভিগেশন কন্ট্রোল:** আপনি ব্রাউজারের `back`, `forward`, এবং `go()` মেথড ব্যবহার করে ইউজারের ইতিহাসের মধ্যে নেভিগেট করতে পারেন।

pushState এবং replaceState কীভাবে কাজ করে?

1. pushState()

`pushState()` ব্যবহার করে আপনি একটি নতুন ইতিহাস স্টেট যোগ করতে পারেন, যা পেজের URL পরিবর্তন করবে কিন্তু পেজ রিলোড হবে না।

Syntax:

```
history.pushState(stateObject, title, url);
```

- stateObject:** এটি একটি অবজেক্ট যা স্টেটের সাথে সম্পর্কিত ডেটা ধারণ করে। এটি ইতিহাস স্ট্যাকের সাথে সংযুক্ত থাকে।
- title:** সাধারণত এটি ফাঁকা রাখা হয় কারণ এটি বর্তমান ব্রাউজারগুলিতে প্রভাব ফেলে না।
- url:** এটি ঐতিহ্যগত URL যা ব্রাউজারে সেট করা হবে। আপনি এটি বর্তমান পেজের URL পরিবর্তন করতে ব্যবহার করতে পারেন, এবং এটি একটি নতুন ইতিহাস এন্ট্রি তৈরি করবে।

Example:

```
// New state added to history
history.pushState({ page: 1 }, "page 1", "/page1");
```

এখানে, `stateObject { page: 1 }` হচ্ছে, যা পরে `popstate` ইভেন্টে ব্যবহৃত হবে। `"/page1"` URL পরিবর্তন করবে, কিন্তু পেজ রিলোড হবে না।

2. `replaceState()`

`replaceState()` একইভাবে কাজ করে, তবে এটি বর্তমান ইতিহাস এন্ট্রি প্রতিস্থাপন করে, নতুন স্টেট সহ। এটি একটি নতুন এন্ট্রি তৈরি করে না, বরং বর্তমান এন্ট্রির URL এবং স্টেট পরিবর্তন করে।

Syntax:

```
history.replaceState(stateObject, title, url);
```

- **stateObject**: এই নতুন স্টেটটি বর্তমান ইতিহাস এন্ট্রির সাথে প্রতিস্থাপিত হবে।
- **title**: ব্রাউজারের টাইটেল, তবে সাধারণত এটি উপেক্ষা করা হয়।
- **url**: বর্তমান URL পরিবর্তন করবে, কিন্তু ইতিহাস স্ট্যাকের মধ্যে নতুন এন্ট্রি তৈরি করবে না।

Example:

```
// Replace current state with new state
history.replaceState({ page: 2 }, "page 2", "/page2");
```

এখানে, বর্তমান ইতিহাস এন্ট্রি প্রতিস্থাপন হবে এবং URL `/page2` হয়ে যাবে, কিন্তু কোনো নতুন এন্ট্রি তৈরি হবে না।

`pushState` এবং `replaceState` এর পার্থক্য:

- **pushState**: নতুন স্টেট যোগ করে এবং একটি নতুন ইতিহাস এন্ট্রি তৈরি করে।
- **replaceState**: বর্তমান ইতিহাস এন্ট্রি প্রতিস্থাপন করে এবং নতুন এন্ট্রি তৈরি করে না।

কবে ব্যবহার করবেন?

- **Single-Page Applications (SPA)**: যখন আপনি কোনো SPA ডেভেলপ করছেন এবং আপনি URL পরিবর্তন করতে চান তবে পেজ রিলোড করতে চান না। এই API ব্যবহার করে আপনি বিভিন্ন ডিউ বা পেজের জন্য URL পরিবর্তন করতে পারেন।
- **নেভিগেশন ইতিহাস নিয়ন্ত্রণ**: যদি আপনি কোনও ইন্টারঅ্যাকশন বা ইউজার অ্যাকশন অনুসারে ব্রাউজার ইতিহাস স্ট্যাক ম্যানেজ করতে চান, তবে এই API ব্যবহার করতে পারেন।

উদাহরণ: একটি SPA-তে `pushState` ব্যবহার

ধৰা যাক, আপনি একটি SPA ডেভেলপ করছেন যেখানে ইউজার একটি পেজে ক্লিক করলে কন্টেন্ট পরিবর্তিত হয়, কিন্তু URL পরিবর্তন করা প্রয়োজন।

```
<a href="#" onclick="loadPage('page1')">Page 1</a>
<a href="#" onclick="loadPage('page2')">Page 2</a>

<div id="content"></div>

<script>
    function loadPage(page) {
        // Load the content dynamically (simulating)
        document.getElementById('content').innerHTML = "This is " +
page;

        // Change the URL without reloading the page
        history.pushState({ page: page }, page, "/" + page);
    }

    // Popstate event listener to handle browser back/forward buttons
    window.addEventListener("popstate", function(event) {
        if (event.state) {
            document.getElementById('content').innerHTML = "This is " +
event.state.page;
        }
    });
</script>
```

এখনে, যখন ইউজার "Page 1" বা "Page 2" লিঙ্কে ক্লিক করে, URL পরিবর্তিত হবে কিন্তু পেজ রিলোড হবে না। এবং যখন ইউজার ভ্রাউজারের ব্যাক বাটন টিপবে, কন্টেন্ট সঠিকভাবে পরিবর্তিত হবে।

History API এবং Single-Page Applications (SPA):

SPA-তে History API খুবই গুরুত্বপূর্ণ। এটি আপনাকে পেজ রিলোড ছাড়াই URL পরিবর্তন এবং স্টেট ম্যানেজ করার সুযোগ দেয়, যা ব্যবহারকারীকে আরও স্নিগ্ধ এবং প্রাকৃতিক অভিজ্ঞতা প্রদান করে।

সারাংশ:

- **History API** আপনাকে ওয়েব অ্যাপ্লিকেশনগুলিতে ইতিহাস স্ট্যাক ম্যানিপুলেট করতে সাহায্য করে, যেমন URL পরিবর্তন করা, স্টেট যোগ করা বা প্রতিস্থাপন করা।
- **pushState()** একটি নতুন ইতিহাস এন্ট্রি তৈরি করে।
- **replaceState()** বর্তমান ইতিহাস এন্ট্রি টি প্রতিস্থাপন করে।
- এটি **single-page applications (SPA)** তৈরি করার সময় বিশেষভাবে কার্যকর, যেখানে আপনি পেজ রিলোড ছাড়াই কন্টেন্ট পরিবর্তন করতে পারেন।

Project: Create a simple single-page navigation system with **pushState**.

প্রকল্প: Simple Single-Page Navigation System with pushState

এই প্রকল্পে আমরা একটি সিম্পল single-page navigation তৈরি করব যেখানে ইউজার একটি লিঙ্ক ক্লিক করলে URL পরিবর্তিত হবে এবং কন্টেন্ট পরিবর্তিত হবে, তবে পেজ রিলোড হবে না।

pushState ব্যবহার করে আমরা URL পরিবর্তন করব এবং সঠিক কন্টেন্ট লোড করতে একটি ডাইনামিক নেভিগেশন সিস্টেম তৈরি করব।

কোড উদাহরণ:

HTML (index.html)

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Simple SPA Navigation</title>
  <style>
    nav a {
      margin: 10px;
```

```

        text-decoration: none;
        color: blue;
    }
}

#content {
    margin-top: 20px;
    padding: 20px;
    border: 1px solid #ccc;
}

</style>
</head>
<body>
<nav>
    <a href="#" onclick="navigateTo('home')">Home</a>
    <a href="#" onclick="navigateTo('about')">About</a>
    <a href="#" onclick="navigateTo('contact')">Contact</a>
</nav>
<div id="content">Welcome to the Home page!</div>

<script src="app.js"></script>
</body>
</html>

```

JavaScript (app.js)

```

// Function to load the content based on the page
function loadContent(page) {
    const contentDiv = document.getElementById('content');
    if (page === 'home') {
        contentDiv.innerHTML = '<h1>Home Page</h1><p>Welcome to the Home page!</p>';
    } else if (page === 'about') {
        contentDiv.innerHTML = '<h1>About Page</h1><p>This is the About page.</p>';
    } else if (page === 'contact') {
        contentDiv.innerHTML = '<h1>Contact Page</h1><p>This is the Contact page.</p>';
    }
}

```

```

    }

}

// Function to handle navigation and change URL using pushState
function navigateTo(page) {
    // Change the URL using pushState (without reloading the page)
    history.pushState({ page: page }, page, `/${page}`);
}

// Load the appropriate content for the page
loadContent(page);
}

// Event listener for the popstate event to handle browser back/forward buttons
window.addEventListener('popstate', function(event) {
    if (event.state) {
        loadContent(event.state.page);
    }
});

```

কোড ব্যাখ্যা:

1. HTML ফাইল:

- এখনে আমরা একটি সিম্পল ন্যাভিগেশন মেনু তৈরি করেছি যা তিনটি লিঙ্ক ধারণ করে: Home, About, এবং Contact।
- প্রতিটি লিঙ্কে `onclick` ইভেন্ট রয়েছে যা `navigateTo` ফাংশনকে কল করে এবং সঠিক পেজের কন্টেন্ট লোড করে।

2. JavaScript (app.js):

- loadContent** ফাংশন: এই ফাংশনটি পেজের নামের ওপর ভিত্তি করে সঠিক কন্টেন্ট পরিবর্তন করে।
- navigateTo** ফাংশন: এই ফাংশনটি `pushState` ব্যবহার করে URL পরিবর্তন করে এবং সঠিক পেজের কন্টেন্ট লোড করে। URL পরিবর্তন করার জন্য `history.pushState` ব্যবহার করা হয়েছে, যাতে পেজ রিলোড না হয়।

- **popstate ইভেন্ট:** এটি ব্রাউজারের ব্যাক/ফরওয়ার্ড বাটন ব্যবহারের জন্য ব্যবহৃত হয়। যখন ইউজার ব্রাউজারের ব্যাক বাটন চাপবে, তখন এই ইভেন্ট ট্রিগার হবে এবং এটি সঠিক কন্টেন্ট লোড করবে।

কী হবে যখন আপনি এই কোডটি রান করবেন?

- যখন আপনি "Home", "About", বা "Contact" লিঙ্কে ক্লিক করবেন, কন্টেন্ট পরিবর্তন হবে এবং URL পরিবর্তিত হবে। তবে পেজ রিলোড হবে না, কারণ **pushState** ব্যবহার করা হয়েছে।
- ব্রাউজারের ব্যাক বা ফরওয়ার্ড বাটন ব্যবহার করলে সঠিক কন্টেন্ট রেন্ডার হবে, কারণ আমরা **popstate** ইভেন্ট ব্যবহার করেছি।

অতিরিক্ত বৈশিষ্ট্য:

- আপনি যদি চান, URL অনুসারে ডিফল্ট কন্টেন্ট লোড করতে পারেন যখন পেজ প্রথমবার লোড হয়, যেমন:

```
// Initial page load based on the current URL
const initialPage = window.location.pathname.replace('/', '');
|| 'home';
loadContent(initialPage);
```

এটি পেজ রিফ্রেশ হওয়ার পরও সঠিক পেজের কন্টেন্ট লোড করবে।

8. How can you handle errors in API requests with **try/catch** blocks and the **.catch()** method?

API রিকোয়েস্ট ত্রুটি (error) হ্যান্ডলিং করতে আপনি দুটি পদ্ধতি ব্যবহার করতে পারেন: **try/catch** রূপ এবং **.catch()** মেথড। এই দুটি পদ্ধতি API রিকোয়েস্টের সময় ঘটে যাওয়া ত্রুটিগুলি ক্যাচ করে, যাতে আপনি সেগুলি ঘটায়থভাবে হ্যান্ডল করতে পারেন।

1. **try/catch** রূপ ব্যবহার করা:

try/catch রূপ ব্যবহার করলে আপনি সিনক্রোনাস কোডের মতোই অ্যাসিনক্রোনাস কোডের ত্রুটি ধরতে পারেন, তবে এটি অ্যাসিনক্রোনাস কোডে **async/await** ব্যবহার করতে হবে। এই পদ্ধতিতে, যদি

API রিকোয়েস্টে কোনো সমস্যা হয় (যেমন, নেটওয়ার্ক সমস্যা, 404 বা 500 ত্রুটি), তাহলে সেই ত্রুটি `catch` লকে ধরা হয়।

উদাহরণ:

```
async function fetchData() {
  try {
    // API রিকোয়েস্ট
    const response = await fetch('https://jsonplaceholder.typicode.com/posts');

    // যদি রেসপন্স সঠিক না হয়
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }

    // রেসপন্স JSON আকারে পার্স করা
    const data = await response.json();
    console.log(data); // API থেকে পাওয়া ডেটা কনসোলে লগ করা

  } catch (error) {
    // যদি কোনো ত্রুটি ঘটে
    console.error('There was a problem with the fetch operation:', error);
  }
}

// কল করা
fetchData();
```

ব্যাখ্যা:

- `try` লক: এখানে `fetch()` API কল করা হচ্ছে। যদি কোনো সমস্যা হয়, যেমন নেটওয়ার্ক ইস্যু বা API সার্ভারের সমস্যা, তাহলে তা `catch` লকে চলে যাবে।
- `catch` লক: কোনো ত্রুটি ঘটলে তা এখানে ধরা হয় এবং আপনি ত্রুটির মেসেজ কনসোলে দেখাতে পারেন।

2. .catch() মেথড ব্যবহার করা:

এটি একটি **Promise-based** পদ্ধতি। আপনি যখন `fetch()` বা কোনো অন্য **Promise-returning** ফাংশন ব্যবহার করেন, তখন `.catch()` মেথডটি ব্যবহার করে ক্রটি হ্যান্ডলিং করতে পারেন। এটি ক্রটি ঘটলে পোমিসের সাথে যুক্ত `catch()` মেথডে চলে যাবে।

উদাহরণ:

```
function fetchData() {
  // API রিকোয়েস্ট
  fetch('https://jsonplaceholder.typicode.com/posts')
    .then(response => {
      // যদি রেসপন্স সঠিক না হয়
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json(); // JSON ডেটা রিটার্ন করা
    })
    .then(data => {
      console.log(data); // API থেকে পাওয়া ডেটা কনসোলে লগ করা
    })
    .catch(error => {
      // যদি কোনো ক্রটি ঘটে
      console.error('There was a problem with the fetch operation:', error);
    });
}

// কল করা
fetchData();
```

ব্যাখ্যা:

- `.then()` : এই মেথডে API থেকে সাফল্যজনক রেসপন্স পাওয়ার পর তার JSON ডেটা প্রক্রিয়া করা হয়।
- `.catch()` : যদি কোনো ক্রটি ঘটে (যেমন, নেটওয়ার্ক সমস্যা বা রেসপন্স কোড 404), তাহলে এটি `.catch()` মেথডে চলে আসে এবং আপনি ক্রটির মেসেজ কনসোলে দেখাতে পারেন।

কখন কোনটি ব্যবহার করবেন?

- `async/await` এর সাথে `try/catch` ব্যবহার করা সুপারিশ করা হয় যখন আপনি অ্যাসিনক্রোনাস কোডে ত্রুটি হ্যান্ডলিং করতে চান, কারণ এটি কোডটিকে আরও ক্লিন এবং পাঠ্যোগ্য করে তোলে।
- `.then()` / `.catch()` পদ্ধতিটি একটি Promise-চেইনিং পদ্ধতি, যেখানে আপনি প্রতিটি রেসপ্লস এবং ত্রুটি `.then()` এবং `.catch()` ব্যবহার করে চেইন করতে পারেন।

সারাংশ:

- `try/catch`: `async/await` ব্যবহার করে অ্যাসিনক্রোনাস কোডে ত্রুটি হ্যান্ডলিং করার একটি আধুনিক পদ্ধতি।
- `.catch()`: Promise-based API তে ত্রুটি হ্যান্ডলিং করার পদ্ধতি, যেখানে আপনি `.then()` চেইনিং ব্যবহার করতে পারেন।

9. What are WebSockets, and how do they differ from HTTP requests?

WebSockets:

WebSockets একটি প্রোটোকল যা ওয়েব ব্রাউজার এবং সার্ভারের মধ্যে দুই-মুখী, লং-লাইভ, এবং রিয়েল-টাইম যোগাযোগ স্থাপন করতে ব্যবহৃত হয়। এটি HTTP প্রোটোকলের উপর ভিত্তি করে কাজ করে, তবে একবার সংযোগ স্থাপিত হলে, এটি HTTP সংযোগের চেয়ে অনেক দ্রুত এবং কার্যকর।

WebSockets মূলত রিয়েল-টাইম অ্যাপ্লিকেশনগুলির জন্য ডিজাইন করা, যেমন চ্যাট অ্যাপ, গেম, লাইভ ট্র্যাকিং, বা মার্কেট ডেটা ইত্যাদি যেখানে সার্ভার এবং ক্লায়েন্টের মধ্যে দ্রুত তথ্য আদান-প্রদান প্রয়োজন।

কীভাবে WebSockets কাজ করে:

1. **সংযোগ স্থাপন:** প্রথমে WebSocket সংযোগটি HTTP প্রোটোকলের মাধ্যমে শুরু হয়। ক্লায়েন্ট (যেমন ব্রাউজার) একটি WebSocket handshake পাঠায় সার্ভারে। সার্ভার সেটি গ্রহণ করলে একটি WebSocket connection তৈরি হয়।
2. **দ্বিমুখী যোগাযোগ:** সংযোগ প্রতিষ্ঠিত হওয়ার পর, সার্ভার এবং ক্লায়েন্ট দু'জনেই একে অপরকে ইভেন্ট বা ডেটা পাঠাতে পারে এবং গ্রহণ করতে পারে। এটি একেবারে রিয়েল-টাইম যোগাযোগ

তৈরি করে।

- পাবলিশ/সাবস্ক্রাইব মডেল:** WebSockets সাধারণত **pub/sub (publish/subscribe)** মডেল ব্যবহার করে, যেখানে সার্ভার একাধিক ক্লায়েন্টকে একসাথে তথ্য পাঠাতে পারে।
- পটভূমি:** একবার সংযোগ স্থাপিত হলে, এটি লং-টাইম স্টেবল থাকে, এবং সার্ভার ও ক্লায়েন্ট একে অপরের সাথে তথ্য পাঠানোর জন্য নতুন HTTP রিকোয়েস্টের প্রয়োজন হয় না।

HTTP requests এবং WebSockets এর মধ্যে পার্থক্য:

বৈশিষ্ট্য	HTTP Requests	WebSockets
সংযোগের ধরন	একতরফা (client → server)	দ্বিমুখী (client ↔ server)
সংযোগের জীবনকাল	সংযোগ প্রতিটি রিকোয়েস্টের জন্য নতুন	একবার সংযোগ স্থাপন হলে স্থায়ী
পদ্ধতি	ক্লায়েন্ট সার্ভার থেকে প্রতিটি রিকোয়েস্টের জন্য নতুন সংযোগ তৈরি করে	একবার সংযোগ স্থাপন হলে, সার্ভার এবং ক্লায়েন্ট একে অপরের সাথে তথ্য আদান-প্রদান করতে পারে
ইনফরমেশন আপডেট	সার্ভার থেকে ক্লায়েন্টে শুধুমাত্র নতুন রিকোয়েস্টের মাধ্যমে তথ্য পাঠানো হয়	রিয়েল-টাইম ডেটা আপডেট সরাসরি ওয়েবসকেট সংযোগের মাধ্যমে পাঠানো হয়
ব্যবহার	সাধারণ ওয়েব অ্যাপ্লিকেশন, API রিকোয়েস্ট	রিয়েল-টাইম অ্যাপ্লিকেশন (যেমন: চ্যাট, গেম, স্টক ট্র্যাকিং)
পারফরম্যান্স	প্রতিটি রিকোয়েস্টে নতুন HTTP হেডার পাঠানো হয়, যা বেশি ট্র্যাফিক তৈরি করে	একবার সংযোগ স্থাপন হলে, অতিরিক্ত হেডার ছাড়া দ্রুত তথ্য পাঠানো হয়

WebSockets এর সুবিধা:

- রিয়েল-টাইম যোগাযোগ:** ক্লায়েন্ট এবং সার্ভারের মধ্যে রিয়েল-টাইম, দুই-মুখী যোগাযোগ সম্ভব হয়।
- পারফরম্যান্স:** HTTP রিকোয়েস্টের মত প্রতিবার নতুন সংযোগ তৈরি না করে একটি স্টেবল সংযোগ থেকে ডেটা আদান-প্রদান করা যায়, যা আরও দ্রুত এবং কম ব্যান্ডউইথ ব্যবহার করে।
- দ্বিমুখী কমিউনিকেশন:** সার্ভার ক্লায়েন্টকে তথ্য পাঠাতে পারে, এবং ক্লায়েন্টও সার্ভারে তথ্য পাঠাতে পারে, এর ফলে সরাসরি কমিউনিকেশন হয়।

WebSockets এবং HTTP এর মধ্যে পার্থক্য:

- সংযোগ স্থাপন:** HTTP রিকোয়েস্টের মধ্যে প্রতিবার নতুন সংযোগ স্থাপন করতে হয়। কিন্তু WebSocket একটি দীর্ঘস্থায়ী সংযোগ বজায় রাখে, যার মাধ্যমে দুই পক্ষে (client ↔ server) অবিচ্ছিন্ন যোগাযোগ ঘটে।

2. **ব্যবহার:** HTTP সাধারণত সিঙ্ক্লোনাস কাজের জন্য (যেমন, ওয়েব পেজ লোড, ফর্ম সাবমিট) ব্যবহৃত হয়। আর WebSockets রিয়েল-টাইম অ্যাপ্লিকেশনগুলির জন্য ব্যবহৃত হয়, যেখানে ডেটা দ্রুত এবং অবিরত আনা প্রয়োজন।
3. **দ্রুততা:** HTTP এ প্রতি রিকোয়েস্টে নতুন হেডার এবং কানেকশন তৈরি করতে হয়, যা লেটেন্সি তৈরি করতে পারে। WebSockets শুধুমাত্র একবার সংযোগ স্থাপন করে পরবর্তীতে কোনো অতিরিক্ত হেডার ছাড়া দ্রুত তথ্য পাঠানো সম্ভব।

WebSocket উদাহরণ:

Client Side (JavaScript):

```
// WebSocket connection তৈরি করা
const socket = new WebSocket('ws://example.com/socket');

// Connection open হওয়ার পর কিছু করা
socket.onopen = function(event) {
    console.log('WebSocket is connected');
    socket.send('Hello Server!');
};

// Server থেকে মেসেজ প্রাপ্তি
socket.onmessage = function(event) {
    console.log('Message from server:', event.data);
};

// Connection বন্ধ হওয়ার পর
socket.onclose = function(event) {
    console.log('WebSocket is closed');
};

// Error handling
socket.onerror = function(error) {
    console.log('WebSocket error:', error);
};
```

Server Side (Node.js Example with Library):

```

const WebSocket = require('ws');
const wss = new WebSocket.Server({ port: 8080 });

wss.on('connection', function connection(ws) {
  console.log('A client connected');

  ws.on('message', function incoming(message) {
    console.log('Received: %s', message);
  });
}

ws.send('Hello Client!');
});

```

সারাংশ:

- **WebSockets** একটি প্রোটোকল যা ক্লায়েন্ট এবং সার্ভারের মধ্যে রিয়েল-টাইম, দ্বিমুখী যোগাযোগ সক্ষম করে।
- **HTTP** সিঙ্ক্রোনাস এবং একতরফা যোগাযোগের জন্য ব্যবহৃত হয়, যেখানে প্রতিটি রিকোয়েস্টে নতুন সংযোগ তৈরি করা হয়।
- WebSockets সরাসরি এবং দ্রুত ডেটা আদান-প্রদান করতে সক্ষম, বিশেষত যেখানে রিয়েল-টাইম তথ্য প্রয়োজন (যেমন চ্যাট অ্যাপ, লাইভ ট্র্যাকিং)।

10. Explain the basics of the Service Worker API and its use in creating progressive web apps (PWAs).

Service Worker API এবং PWAs (Progressive Web Apps):

Service Worker হল একটি স্ক্রিপ্ট যা ব্রাউজারে ব্যাকগ্রাউন্ডে রান করে এবং এটি ওয়েব অ্যাপ্লিকেশনগুলিকে অফলাইন কার্যকারিতা, ব্যাকগ্রাউন্ড সিনক্রোনাইজেশন এবং পুশ নোটিফিকেশন প্রদান করতে সক্ষম করে। Service Workers মূলত **Progressive Web Apps (PWAs)** তৈরির

জন্য ব্যবহৃত হয়। PWAs এমন ওয়েব অ্যাপ্লিকেশন যা মোবাইল অ্যাপ্লিকেশনের মতো কার্যকরী এবং ডেস্কটপ বা মোবাইল ব্রাউজারে চলতে সক্ষম।

Service Worker কী?

Service Worker একটি JavaScript ফাইল যা ওয়েব পেজের থেকে আলাদা এবং ব্রাউজারের ব্যাকগ্রাউন্ডে চলে। এটি HTTP রিকোয়েস্ট ইন্টারসেপ্ট করে এবং তাদের ক্যাশিং এবং ব্যাকগ্রাউন্ড প্রসেসিং পরিচালনা করে। সার্ভিস ওয়ার্কার ব্রাউজারের **main thread** থেকে আলাদা এবং অ্যাসিনক্রোনাসভাবে কাজ করে, যাতে এটি পেজ লোডের উপর প্রভাব না ফেলে।

Service Worker এর মূল বৈশিষ্ট্যসমূহ:

- ব্যাকগ্রাউন্ডে রান করা:** Service Worker কখনোই ডকুমেন্টের অংশ হিসেবে চলে না। এটি ব্যাকগ্রাউন্ডে চলতে থাকে, অর্থাৎ এটি UI থেকে সাথে মেশে না।
- রিকোয়েস্ট ইন্টারসেপ্ট করা:** Service Worker HTTP রিকোয়েস্টগুলোকে ইন্টারসেপ্ট করতে পারে এবং কাস্টম রেসপন্স তৈরি করতে পারে, যেমন ক্যাশিং করা রেসপন্স অথবা নেটওয়ার্ক রিকোয়েস্ট।
- ক্যাশিং:** Service Worker অফলাইন মোডে ওয়েব অ্যাপ্লিকেশনকে সঠিকভাবে কাজ করতে সাহায্য করার জন্য ফাইল ক্যাশ করতে পারে। অর্থাৎ, একবার কোনো রিসোর্স ডাউনলোড হয়ে গেলে তা ভবিষ্যতে অফলাইন অবস্থাতেও ব্যবহার করা যেতে পারে।
- পুশ নোটিফিকেশন:** Service Worker পুশ নোটিফিকেশন পাঠাতে সক্ষম, যা ব্যবহারকারীদের অ্যাপ ব্যবহার না করলেও ব্রাউজারে নোটিফিকেশন প্রদর্শন করতে পারে।
- ব্যাকগ্রাউন্ড সিনক্রোনাইজেশন:** Service Worker ব্যাকগ্রাউন্ডে ডেটা সিঙ্ক্রোনাইজেশন পরিচালনা করতে পারে, যেমন ইউজারের যেকোনো কার্যক্রম সার্ভারে পাঠানো।

Service Worker কাজ করার প্রক্রিয়া:

- রেজিস্ট্রেশন:** প্রথমে সার্ভিস ওয়ার্কারকে রেজিস্টার করতে হয়। এটি ব্রাউজারকে জানায় যে ওয়েব অ্যাপ্লিকেশনটি সার্ভিস ওয়ার্কার ব্যবহার করতে চায়।
- ইনস্টলেশন:** যখন সার্ভিস ওয়ার্কার প্রথমবার রেজিস্টার করা হয়, এটি **install** ইভেন্ট ট্রিগার করবে। এখানে সাধারণত প্রয়োজনীয় ক্যাশ করা ফাইলগুলো ইনস্টল করা হয়।
- একটিভেশন:** সার্ভিস ওয়ার্কার ইনস্টল হওয়া শেষে **activate** ইভেন্ট ট্রিগার হবে। এই সময় সার্ভিস ওয়ার্কার পুরনো ক্যাশ সাফ এবং নতুন ক্যাশ তৈরি করতে পারে।
- ফেচ ইভেন্ট:** সার্ভিস ওয়ার্কার প্রতিটি HTTP রিকোয়েস্ট ইন্টারসেপ্ট করতে পারে এবং কাস্টম রেসপন্স তৈরি করতে পারে, যেমন ক্যাশ থেকে ডেটা ফিরিয়ে দেওয়া।

Service Worker ৱেজিস্ট্ৰেশন উদাহৰণ:

```
// Service Worker ৱেজিস্ট্ৰেশন
if ('serviceWorker' in navigator) {
  window.addEventListener('load', () => {
    navigator.serviceWorker.register('/service-worker.js')
      .then((registration) => {
        console.log('Service Worker registered with scope: ', registration.scope);
      })
      .catch((error) => {
        console.log('Service Worker registration failed: ', error);
      });
  });
}
```

এখনে `/service-worker.js` হলো সেই স্ক্রিপ্ট ফাইল যা সার্ভিস ওয়ার্কাৰ হিসেবে কাজ কৰবে।

Service Worker স্ক্রিপ্ট উদাহৰণ (service-worker.js):

```
self.addEventListener('install', (event) => {
  event.waitUntil(
    caches.open('my-cache-v1').then((cache) => {
      return cache.addAll([
        '/',
        '/index.html',
        '/styles.css',
        '/script.js',
        '/offline.html',
      ]);
    })
  );
});

self.addEventListener('activate', (event) => {
  const cacheWhitelist = ['my-cache-v1'];
```

```

event.waitUntil(
  caches.keys().then((cacheNames) => {
    return Promise.all(
      cacheNames.map((cacheName) => {
        if (!cacheWhitelist.includes(cacheName)) {
          return caches.delete(cacheName);
        }
      })
    );
  );
});

self.addEventListener('fetch', (event) => {
  event.respondWith(
    caches.match(event.request).then((cachedResponse) => {
      if (cachedResponse) {
        return cachedResponse; // ক্যাশ থেকে রেসপন্স ফেরত দেওয়া হচ্ছে
      }
      return fetch(event.request); // নেটওয়ার্ক রিকোয়েস্ট পাঠানো হচ্ছে
    })
  );
});

```

কীভাবে Service Worker PWAs-এ সাহায্য করে?

- অফলাইন মোড:** Service Worker ক্যাশিংয়ের মাধ্যমে অ্যাপ্লিকেশনকে অফলাইনেও কাজ করতে সক্ষম করে। যেমন, যদি ইউজার ইন্টারনেট সংযোগ হারায়, তাহলে অ্যাপের পূর্বে ক্যাশ করা কন্টেন্ট দেখানো যেতে পারে।
- পুশ নোটিফিকেশন:** PWAs তে রিয়েল-টাইম পুশ নোটিফিকেশন পাঠানোর জন্য Service Worker ব্যবহার করা হয়। এটি ব্যবহারকারীদের অ্যাপ ব্যবহার না করেও নোটিফিকেশন পাঠানোর সুযোগ দেয়।
- ব্যাকগ্রাউন্ড সিঙ্ক:** Service Worker ব্যাকগ্রাউন্ডে ডেটা সিঙ্ক্রোনাইজ করতে সক্ষম। যেমন, ইউজার যখন অফলাইনে থাকে, তখন তার অ্যাকশন সিঙ্ক হওয়ার পর আবার অনলাইনে ফিরে আসলে এটি সিঙ্ক হয়ে যাবে।

4. **ক্যাশ-ভিত্তিক পারফরম্যান্স:** পেজ এবং রিসোর্সগুলোকে ক্যাশ করা, যাতে অ্যাপ্লিকেশন দ্রুত লোড হয়, বিশেষ করে অফলাইন মোডে।

Service Worker এর সুবিধা:

- রিয়েল-টাইম পুশ নোটিফিকেশন প্রেরণ
- অফলাইন অ্যাক্সেস সহ দ্রুত লোড টাইম
- ব্যাকগ্রাউন্ড ডেটা সিঙ্ক্রোনাইজেশন এবং কার্যকরী পুশ নোটিফিকেশন
- ফাস্ট এবং রেসপন্সিভ অ্যাপ্লিকেশন সৃষ্টির মাধ্যমে ইউজারের অভিজ্ঞতা উন্নত করা

সারাংশ:

- **Service Worker** হল একটি ব্যাকগ্রাউন্ড স্ক্রিপ্ট যা PWAs তে রিয়েল-টাইম কমিউনিকেশন, অফলাইন সমর্থন, এবং দ্রুত পারফরম্যান্স নিশ্চিত করতে ব্যবহৃত হয়।
- এটি **fetch** ইভেন্টে রিকোয়েস্ট ইন্টারসেপ্ট করে এবং ক্যাশিং, নেটওয়ার্ক ডেটা, ব্যাকগ্রাউন্ড সিনক্রোনাইজেশন, এবং পুশ নোটিফিকেশন যেমন ফিচার প্রদান করতে সহায়তা করে।
- **PWAs** এমন অ্যাপ্লিকেশন যা ওয়েব এবং মোবাইল অ্যাপ্লিকেশনের মধ্যে সেতুবন্ধন তৈরি করে, এবং Service Worker এর মাধ্যমে ওয়েব অ্যাপ্লিকেশনগুলিকে আরও গতিশীল এবং কার্যকরী করে তোলে।