জাভাস্ক্রিপ্ট Array এবং Object: একটি সমন্বিত এবং গভীর বিশ্লেষণ

আমরা জাভাস্ক্রিপ্টের দুটি অপরিহার্য ডেটা স্ট্রাকচার, **Array** এবং **Object**, নিয়ে বিস্তারিত আলোচনা করেছি। এই ডেটা স্ট্রাকচারগুলো কীভাবে ডেটা সংরক্ষণ, সংগঠন এবং ম্যানিপুলেট করতে ব্যবহৃত হয়, তা বোঝা আধুনিক জাভাস্ক্রিপ্ট ডেভেলপমেন্টের জন্য অত্যাবশ্যক।

এই পর্যালোচনায়, আমরা নিম্নলিখিত মূল প্রশ্নগুলির উত্তরগুলি একত্রিত করব:

1. **Array এবং Object কী এবং কেন ব্যবহার করা হয়?**
2. **Array এবং Object-এর প্রোপার্টিগুলো কী কী এবং সেগুলোর কার্যকারিতা কী?**
3. **Array এবং Object-এর মেথডগুলো কী কী, কিভাবে কাজ করে ("কি দিয়ে কি করলে কি হয়"), এবং কেন সেগুলো ব্যবহার করা হয় ("কেন করে")?**
4. **আধুনিক জাভাস্ক্রিপ্টে এই প্রোপার্টি ও মেথডগুলোর ব্যবহার কেমন?**
5. **কখন Array এবং কখন Object ব্যবহার করা উচিত?**
6. **ডেটা ম্যানিপুলেশনের সময় কী কী সাধারণ ভুল এড়িয়ে চলা উচিত?**

---

১. Array (অ্যারে): ক্রমানুসারে ডেটার ঝুড়ি

একটি অ্যারে হলো একটি বিশেষ ধরনের অবজেক্ট যা **ক্রমিক (ordered)** ডেটার একটি সংগ্রহ ধারণ করে। প্রতিটি ডেটার একটি সংখ্যাসূচক **সূচক (index)** থাকে, যা `0` থেকে শুরু হয়।

কেন অ্যারে ব্যবহার করবেন?

- যখন আপনার কাছে **একই ধরনের ডেটা**র একটি তালিকা বা সিরিজ থাকে, যেখানে ডেটাগুলির **ক্রম** গুরুত্বপূর্ণ (যেমন, টাস্ক লিস্ট, স্কোরবোর্ড, ব্যবহারকারীর ইনপুট অর্ডার)।
- যখন আপনি ডেটাগুলিকে তাদের **ক্রমিক ইন্ডেক্স** ব্যবহার করে অ্যাক্সেস, যোগ বা সরাতে চান।

অ্যারে তৈরির আধুনিক উপায়

সবচেয়ে সাধারণ এবং আধুনিক পদ্ধতি হলো **অ্যারে লিটারাল** ব্যবহার করা:

JavaScript

```
// একটি স্ট্রিং অ্যারে
const fruits = ["আপেল", "কলা", "কমলা"];

// একটি সংখ্যা অ্যারে
const rollNumbers = [101, 102, 103];

// বিভিন্ন ডেটা টাইপের মিশ্র অ্যারে
const mixedBag = ["হ্যালো", 2024, true, null, { name: "অবজেক্ট" }];
```

অ্যারে প্রোপার্টিসমূহ:

অ্যারের সবচেয়ে গুরুত্বপূর্ণ প্রোপার্টি হলো `length`।

`Array.prototype.length`

- **কী:** `length`
- **এর ধরন:** একটি **সংখ্যা (Number)**।
- **কী করে:** অ্যারের মোট উপাদানের সংখ্যা নির্দেশ করে। এটি `0` (খালি অ্যারের জন্য) থেকে শুরু করে যেকোনো অঋণাত্মক পূর্ণসংখ্যা হতে পারে।
- **কী হয়:**
    - `length` প্রোপার্টি অ্যাক্সেস করলে অ্যারের বর্তমান দৈর্ঘ্য ফেরত দেয়।
    - আপনি `length` প্রোপার্টির মান পরিবর্তন করতে পারেন, যা **মূল অ্যারের আকার** পরিবর্তন করে।
- **কেন করে:**
    - একটি অ্যারেতে কতগুলি উপাদান আছে তা জানতে।
    - লুপ চালানোর জন্য শর্ত হিসেবে (`for (let i = 0; i < arr.length; i++)`)।
    - অ্যারের আকার নিয়ন্ত্রণ করতে, যেমন অ্যারে ছোট করা বা নির্দিষ্ট ইন্ডেক্স পর্যন্ত খালি করা।
- **খুঁটিনাটি:**
    - `length` প্রোপার্টি সবসময় অ্যারের সর্বোচ্চ সংখ্যাসূচক ইন্ডেক্সের চেয়ে এক বেশি হয়।
    - যদি `length` এর মান কমানো হয়, তাহলে অ্যারের শেষ থেকে উপাদানগুলো **স্থায়ীভাবে মুছে যায়**।
    - যদি `length` এর মান বাড়ানো হয়, তাহলে নতুন যোগ হওয়া ইন্ডেক্সগুলো `empty` বা `undefined` থাকে।
- **উদাহরণ:**

JavaScript

```
const items = ["A", "B", "C", "D"];

// বর্তমান দৈর্ঘ্য
console.log("প্রাথমিক দৈর্ঘ্য:", items.length); // আউটপুট: প্রাথমিক দৈর্ঘ্য: 4

// নতুন উপাদান যোগ করে দৈর্ঘ্য পরিবর্তন
items.push("E");
console.log("যোগ করার পর দৈর্ঘ্য:", items.length); // আউটপুট: যোগ করার পর দৈর্ঘ্য: 5
console.log("যোগ করার পর অ্যারে:", items);        // আউটপুট: যোগ করার পর অ্যারে: ['A', 'B', 'C', 'D', 'E']

// length প্রোপার্টির মান কমিয়ে অ্যারে ছোট করা (মুটেশন)
```

```
items.length = 2; // অ্যারের দৈর্ঘ্য 2-এ সেট হলো। শেষের উপাদানগুলো ('C', 'D', 'E') মুছে যাবে।
console.log("দৈর্ঘ্য 2 করার পর অ্যারে:", items); // আউটপুট: দৈর্ঘ্য 2 করার পর অ্যারে: ['A', 'B']

// length প্রোপার্টির মান বাড়িয়ে অ্যারে প্রসারিত করা (খালি স্লট)
items.length = 4;
console.log("দৈর্ঘ্য 4 করার পর অ্যারে:", items); // আউটপুট: দৈর্ঘ্য 4 করার পর অ্যারে: ['A', 'B', <2 empty items>]
console.log("খালি স্লটের মান:", items[3]);       // আউটপুট: খালি স্লটের মান: undefined
```

অ্যারে মেথডসমূহ:

অ্যারে মেথডগুলোকে তাদের কার্যকারিতা অনুসারে দুটি প্রধান ভাগে ভাগ করা যায়: **পরিবর্তনকারী (Mutator) মেথড** যা মূল অ্যারে পরিবর্তন করে, এবং **অ্যাক্সেসর (Accessor) মেথড** যা একটি নতুন অ্যারে তৈরি করে বা ডেটা অ্যাক্সেস করে কিন্তু মূল অ্যারে অপরিবর্তিত রাখে।

A. অ্যারে পরিবর্তনকারী মেথড (Mutator Methods): (এগুলো মূল অ্যারে পরিবর্তন করে)

1. **`push(element1, ..., elementN)`**
    - **কী করে:** অ্যারের **শেষে** এক বা একাধিক উপাদান যোগ করে।
    - **কী হয়:** অ্যারের **নতুন দৈর্ঘ্য** ফেরত দেয়।
    - **কেন করে:** একটি তালিকার শেষে নতুন আইটেম যোগ করার জন্য, যেমন একটি টাস্ক লিস্টে নতুন কাজ যোগ করা।
    - **উদাহরণ:**

      JavaScript

      ```
      const queue = ["User1", "User2"];
      const newLength = queue.push("User3", "User4");
      console.log(queue);      // ['User1', 'User2', 'User3', 'User4']
      console.log(newLength);  // 4
      ```

2. **`pop()`**
    - **কী করে:** অ্যারের **শেষের** উপাদানটি মুছে ফেলে।
    - **কী হয়:** মুছে ফেলা উপাদানটি ফেরত দেয়।
    - **কেন করে:** তালিকার শেষ আইটেমটি সরিয়ে ফেলা এবং সেটি কী ছিল তা জানার জন্য (যেমন স্ট্যাক ডেটা স্ট্রাকচার)।
    - **উদাহরণ:**

      JavaScript

      ```
      const history = ["PageA", "PageB", "PageC"];
      const lastVisited = history.pop();
      console.log(history);      // ['PageA', 'PageB']
      console.log(lastVisited);  // PageC
      ```

3. **`unshift(element1, ..., elementN)`**

- **কী করে:** অ্যারের **শুরুতে** এক বা একাধিক উপাদান যোগ করে।
- **কী হয়:** অ্যারের **নতুন দৈর্ঘ্য** ফেরত দেয়। বিদ্যমান উপাদানগুলোর ইন্ডেক্স পরিবর্তিত হয়।
- **কেন করে:** তালিকার শুরুতে নতুন আইটেম যোগ করার জন্য, যেমন একটি চ্যাট ফিডে নতুন মেসেজ যোগ করা।
- **উদাহরণ:**

  JavaScript

  ```
  const notifications = ["Old Notification"];
  notifications.unshift("New Notification 1", "New Notification 2");
  console.log(notifications); // ['New Notification 1', 'New
  Notification 2', 'Old Notification']
  ```

4. **`shift()`**

- **কী করে:** অ্যারের **প্রথম** উপাদানটি মুছে ফেলে।
- **কী হয়:** মুছে ফেলা উপাদানটি ফেরত দেয়। বিদ্যমান উপাদানগুলোর ইন্ডেক্স পরিবর্তিত হয়।
- **কেন করে:** একটি "কিউ" থেকে প্রথম আইটেমটি সরিয়ে ফেলা এবং সেটি কী ছিল তা জানার জন্য।
- **উদাহরণ:**

  JavaScript

  ```
  const printJobs = ["Job A", "Job B", "Job C"];
  const currentJob = printJobs.shift();
  console.log(printJobs); // ['Job B', 'Job C']
  console.log(currentJob); // Job A
  ```

5. **`splice(startIndex, deleteCount, item1, ..., itemN)`**

- **কী করে:** অ্যারের যেকোনো নির্দিষ্ট ইন্ডেক্স থেকে উপাদান যোগ, মুছে ফেলা বা প্রতিস্থাপন (replace) করে। এটি খুবই শক্তিশালী।
- **কী হয়:** মুছে ফেলা উপাদানগুলোর একটি **নতুন অ্যারে** ফেরত দেয়।
- **কেন করে:** অ্যারের মাঝখান থেকে ডেটা সরাতে, যোগ করতে বা ডেটা প্রতিস্থাপন করতে।
- **উদাহরণ:**

  JavaScript

  ```
  const colors = ["লাল", "সবুজ", "নীল", "হলুদ"];
  // ইন্ডেক্স 2 থেকে 1টি উপাদান মুছে ফেলে ("নীল")
  const removed = colors.splice(2, 1);
  console.log(colors); // ['লাল', 'সবুজ', 'হলুদ']
  console.log(removed); // ['নীল']
  ```

```
// ইন্ডেক্স 1 এ 0টি উপাদান মুছে "বেগুনি" যোগ করা
colors.splice(1, 0, "বেগুনি");
console.log(colors); // ['লাল', 'বেগুনি', 'সবুজ', 'হলুদ']

// ইন্ডেক্স 3 থেকে 1টি উপাদান মুছে "সাদা" দিয়ে প্রতিস্থাপন করা ("হলুদ" বদলে "সাদা" হবে)
colors.splice(3, 1, "সাদা");
console.log(colors); // ['লাল', 'বেগুনি', 'সবুজ', 'সাদা']
```

6. **`sort(compareFn)`**
   - **কী করে:** অ্যারের উপাদানগুলোকে সাজায়। ডিফল্টভাবে, এটি উপাদানগুলোকে **স্ট্রিং হিসেবে** ধরেবর্ণানুক্রমিকভাবেসাজায়(ইউনিকোড অনুযায়ী)। সংখ্যার জন্য একটি কাস্টম তুলনা ফাংশন (`compareFn`) ব্যবহার করা জরুরি।
   - **কী হয়: মূল অ্যারেটিই পরিবর্তিত** হয় এবং সাজানো অ্যারেটি ফেরত দেয়।
   - **কেন করে:** একটি তালিকা থেকে ডেটাগুলোকে বর্ণানুক্রমিক বা সংখ্যাসূচক ক্রমে সাজানোর জন্য।
   - **উদাহরণ:**

     JavaScript

     ```
     const strArr = ["কমলা", "আপেল", "কলা"];
     strArr.sort();
     console.log(strArr); // ['আপেল', 'কমলা', 'কলা']

     const numArr = [10, 2, 8, 1, 100];
     // ছোট থেকে বড় সাজাতে (আরোহী ক্রম)
     numArr.sort((a, b) => a - b);
     console.log(numArr); // [1, 2, 8, 10, 100]

     // বড় থেকে ছোট সাজাতে (অবরোহী ক্রম)
     numArr.sort((a, b) => b - a);
     console.log(numArr); // [100, 10, 8, 2, 1]
     ```

7. **`reverse()`**
   - **কী করে:** অ্যারের উপাদানগুলোর **ক্রম উল্টে দেয়**।
   - **কী হয়: মূল অ্যারেটিই পরিবর্তিত** হয় এবং উল্টানো অ্যারেটি ফেরত দেয়।
   - **কেন করে:** একটি তালিকার ক্রমকে উল্টো দিক থেকে সাজানোর জন্য।
   - **উদাহরণ:**

     JavaScript

     ```
     const numbers = [1, 2, 3, 4, 5];
     numbers.reverse();
     console.log(numbers); // [5, 4, 3, 2, 1]
     ```

B. নতুন অ্যারে তৈরি/অ্যাক্সেসর মেথড (Accessor Methods): (এগুলো মূল অ্যারে পরিবর্তন করে না)

1. **`concat(array1, ..., arrayN)`**

- **কী করে:** দুটি বা ততোধিক অ্যারে বা ভ্যালুকে একত্রিত করে একটি **নতুন অ্যারে** তৈরি করে।
- **কী হয়:** একটি নতুন অ্যারে ফেরত দেয়। মূল অ্যারেগুলো **অপরিবর্তিত থাকে**।
- **কেন করে:** একাধিক অ্যারের উপাদানগুলোকে একটি একক অ্যারেতে মার্জ করার জন্য কিন্তু মূল অ্যারেগুলো অক্ষত রাখার জন্য।
- **আধুনিক জাভাস্ক্রিপ্ট:** স্প্রেড অপারেটর (`...`) প্রায়শই এর বিকল্প হিসেবে ব্যবহৃত হয়।
- **উদাহরণ:**

JavaScript

```
const arr1 = [1, 2];
const arr2 = [3, 4];
const combined = arr1.concat(arr2, [5, 6]);
console.log(combined);     // [1, 2, 3, 4, 5, 6]
console.log(arr1);         // [1, 2] (মূল অ্যারে অপরিবর্তিত)

// স্প্রেড অপারেটরের ব্যবহার
const combinedSpread = [...arr1, ...arr2, 5, 6];
console.log(combinedSpread); // [1, 2, 3, 4, 5, 6]
```

2. **`slice(startIndex, endIndex)`**

- **কী করে:** একটি অ্যারের একটি অংশ (sub-array) কেটে নেয়। `endIndex` পর্যন্ত (তবে `endIndex` সহ নয়)।
- **কী হয়:** একটি **নতুন অ্যারে** ফেরত দেয়। মূল অ্যারে **অপরিবর্তিত থাকে**।
- **কেন করে:** একটি অ্যারের একটি নির্দিষ্ট অংশ নিয়ে কাজ করতে কিন্তু মূল অ্যারে পরিবর্তন করতে না চাইলে।
- **উদাহরণ:**

JavaScript

```
const students = ["আলি", "বাচ্চু", "করিম", "দীপা", "এমা"];
const selected = students.slice(1, 4); // ইন্ডেক্স 1 থেকে 4 (4 বাদ) -> বাচ্চু, করিম, দীপা
console.log(selected);  // ['বাচ্চু', 'করিম', 'দীপা']
console.log(students);  // ['আলি', 'বাচ্চু', 'করিম', 'দীপা', 'এমা'] (মূল অ্যারে অপরিবর্তিত)
```

3. **`join(separator)`**

- **কী করে:** অ্যারের সমস্ত উপাদানকে একটি স্ট্রিংয়ে একত্রিত করে।
- **কী হয়:** একটি **স্ট্রিং** ফেরত দেয়। প্রতিটি উপাদানের মধ্যে একটি নির্দিষ্ট `separator` (ডিফল্ট কমা) ব্যবহার করা যায়।
- **কেন করে:** একটি অ্যারের ডেটাগুলোকে একটি পঠনযোগ্য স্ট্রিং ফরম্যাটে দেখাতে।
- **উদাহরণ:**

JavaScript

```
const words = ["হ্যালো", "বিশ্ব", "!"];
console.log(words.join(" ")); // "হ্যালো বিশ্ব !"
console.log(words.join("-")); // "হ্যালো-বিশ্ব-!"
```

4. **`includes(valueToFind, fromIndex)`**
    - **কী করে:** অ্যারেতে একটি নির্দিষ্ট মান আছে কিনা তা পরীক্ষা করে। (কেস-সেনসিটিভ)
    - **কী হয়:** `true` অথবা `false` ফেরত দেয়।
    - **কেন করে:** অ্যারেতে কোনো একটি নির্দিষ্ট আইটেম আছে কিনা তা দ্রুত জানতে।
    - **উদাহরণ:**

      JavaScript

      ```
      const fruits = ["আপেল", "কলা", "কমলা"];
      console.log(fruits.includes("কলা"));    // true
      console.log(fruits.includes("পেয়ারা")); // false
      ```

5. **`indexOf(valueToFind, fromIndex)`**
    - **কী করে:** অ্যারেতে একটি নির্দিষ্ট মানের **প্রথম ইন্ডেক্স** ফেরত দেয়।
    - **কী হয়:** মানটি খুঁজে পেলে তার ইন্ডেক্স ফেরত দেয়, না পেলে `-1`।
    - **কেন করে:** একটি নির্দিষ্ট আইটেম অ্যারেতে কোন অবস্থানে আছে তা জানার জন্য।
    - **উদাহরণ:**

      JavaScript

      ```
      const numbers = [10, 20, 30, 20];
      console.log(numbers.indexOf(20)); // 1 (প্রথম 20 এর ইন্ডেক্স)
      console.log(numbers.indexOf(50)); // -1
      ```

6. **`lastIndexOf(valueToFind, fromIndex)`**
    - **কী করে:** অ্যারেতে একটি নির্দিষ্ট মানের **শেষ ইন্ডেক্স** ফেরত দেয়।
    - **কী হয়:** মানটি খুঁজে পেলে তার ইন্ডেক্স ফেরত দেয়, না পেলে `-1`।
    - **কেন করে:** একটি নির্দিষ্ট আইটেম অ্যারেতে সর্বশেষ কোন অবস্থানে আছে তা জানার জন্য।
    - **উদাহরণ:**

      JavaScript

      ```
      const numbers = [10, 20, 30, 20];
      console.log(numbers.lastIndexOf(20)); // 3 (শেষ 20 এর ইন্ডেক্স)
      ```

7. **`toString()`**
    - **কী করে:** অ্যারেটিকে একটি স্ট্রিংয়ে রূপান্তর করে, কমা দিয়ে উপাদানগুলো আলাদা করে।

- **কী হয়:** একটি স্ট্রিং ফেরত দেয়।
- **কেন করে:** অ্যারের একটি স্ট্রিং উপস্থাপনা দরকার হলে।
- **উদাহরণ:**

JavaScript

```
const mixed = [1, "test", true];
console.log(mixed.toString()); // "1,test,true"
```

C. ইটারেটর মেথড (Iterator Methods): (অ্যারের ডেটা প্রসেসিং)

1. **`forEach(callbackFn)`**
   - **কী করে:** অ্যারের প্রতিটি উপাদানের জন্য একটি ফাংশন (callback) এক্সিকিউট করে।
   - **কী হয়:** কোনো কিছু ফেরত দেয় না (`undefined` রিটার্ন করে)। এটি মূলত **পাশের প্রভাব (side effects)** তৈরি করার জন্য ব্যবহৃত হয়।
   - **কেন করে:** যখন আপনি প্রতিটি আইটেমের উপর কোনো অপারেশন করতে চান কিন্তু একটি নতুন অ্যারে তৈরি করতে বা একটি একক মান বের করতে চান না।
   - **গুরুত্বপূর্ণ:** `forEach` লুপের মাঝখানে `break` বা `continue` করা যায় না।
   - **উদাহরণ:**

JavaScript

```
const users = ["আরিফ", "রহিমা", "শফিক"];
users.forEach((user, index) => {
  console.log(`${index + 1}. ${user} কে স্বাগত!`);
});
/*
আউটপুট:
1. আরিফ কে স্বাগত!
2. রহিমা কে স্বাগত!
3. শফিক কে স্বাগত!
*/
```

2. **`map(callbackFn)`**
   - **কী করে:** অ্যারের প্রতিটি উপাদানের উপর একটি ফাংশন প্রয়োগ করে এবং সেই ফাংশনের ফলাফল ব্যবহার করে একটি **নতুন অ্যারে** তৈরি করে।
   - **কী হয়:** একটি **নতুন অ্যারে** ফেরত দেয়। মূল অ্যারে **অপরিবর্তিত থাকে**।
   - **কেন করে:** একটি অ্যারের ডেটাকে অন্য ফরম্যাটে রূপান্তর করতে (যেমন সংখ্যাকে দ্বিগুণ করা, স্ট্রিং আপারকেসে রূপান্তর করা)।
   - **উদাহরণ:**

JavaScript

```
const prices = [100, 200, 300];
const discountedPrices = prices.map(price => price * 0.9); // 10% ছাড়
console.log(discountedPrices); // [90, 180, 270]
```

```
console.log(prices);           // [100, 200, 300] (মূল অ্যারে অপরিবর্তিত)
```

3. **`filter(callbackFn)`**

- **কী করে:** একটি নির্দিষ্ট শর্ত পূরণ করে এমন উপাদানগুলো নিয়ে একটি **নতুন অ্যারে** তৈরি করে।
- **কী হয়:** একটি **নতুন অ্যারে** ফেরত দেয়। মূল অ্যারে **অপরিবর্তিত থাকে**।
- **কেন করে:** একটি তালিকা থেকে নির্দিষ্ট শর্ত অনুযায়ী ডেটা বাদ দিতে বা শুধুমাত্র নির্দিষ্ট ডেটা রাখতে।
- **উদাহরণ:**

  JavaScript

```
const products = [
  { name: "Laptop", price: 1200 },
  { name: "Mouse", price: 25 },
  { name: "Keyboard", price: 75 }
];
const expensiveProducts = products.filter(product => product.price > 100);
console.log(expensiveProducts);
// আউটপুট: [{ name: 'Laptop', price: 1200 }]
```

4. **`reduce(callbackFn, initialValue)`**

- **কী করে:** অ্যারের সমস্ত উপাদানকে একটি **একক আউটপুট ভ্যালুতে** (যেমন একটি সংখ্যা, স্ট্রিং, অবজেক্ট) একত্রিত করে।
- **কী হয়:** একক ফলাফল মান ফেরত দেয়।
- **কেন করে:** একটি অ্যারের ডেটা থেকে একটি একক সমষ্টিগত ফলাফল বের করতে, যেমন যোগফল, গড়, বা ডেটা গ্রুপিং করা।
- **উদাহরণ:**

  JavaScript

```
const cartItems = [{ price: 10, qty: 2 }, { price: 20, qty: 1 }];
// কার্টের মোট মূল্য হিসাব করা
const totalPrice = cartItems.reduce((sum, item) => sum + (item.price * item.qty), 0);
console.log(totalPrice); // 40
```

5. **`every(callbackFn)`**

- **কী করে:** পরীক্ষা করে দেখে যে অ্যারের **সবগুলো উপাদান** একটি নির্দিষ্ট শর্ত পূরণ করে কিনা।
- **কী হয়:** `true` যদি সব উপাদান শর্ত পূরণ করে, অন্যথায় `false`।
- **কেন করে:** একটি অ্যারের সমস্ত আইটেম একটি নির্দিষ্ট মানদণ্ড পূরণ করে কিনা তা নিশ্চিত করতে।

- **উদাহরণ:**

  JavaScript

  ```
  const ages = [20, 22, 25];
  const allAdults = ages.every(age => age >= 18);
  console.log(allAdults); // true

  const mixedAges = [16, 20, 22];
  const allAdults2 = mixedAges.every(age => age >= 18);
  console.log(allAdults2); // false (কারণ 16 আছে)
  ```

6. **`some(callbackFn)`**
   - **কী করে:** পরীক্ষা করে দেখে যে অ্যারের **কমপক্ষে একটি উপাদান** একটি নির্দিষ্ট শর্ত পূরণ করে কিনা।
   - **কী হয়:** `true` যদি কমপক্ষে একটি উপাদান শর্ত পূরণ করে, অন্যথায় `false`।
   - **কেন করে:** একটি অ্যারেতে কোনো একটি নির্দিষ্ট শর্ত পূরণ করে এমন একটি আইটেম আছে কিনা তা জানতে।
   - **উদাহরণ:**

     JavaScript

     ```
     const roles = ["viewer", "editor", "admin"];
     const canEdit = roles.some(role => role === "editor");
     console.log(canEdit); // true

     const hasSuperAdmin = roles.some(role => role === "super_admin");
     console.log(hasSuperAdmin); // false
     ```

7. **`find(callbackFn)`**
   - **কী করে:** অ্যারের প্রতিটি উপাদানের উপর একটি ফাংশন চালায় এবং শর্ত পূরণ করে এমন **প্রথম উপাদানটি** ফেরত দেয়।
   - **কী হয়:** যদি শর্ত পূরণ হয় এমন কোনো উপাদান খুঁজে পায়, তবে সেই উপাদানটি ফেরত দেয়। না পেলে `undefined` ফেরত দেয়।
   - **কেন করে:** একটি নির্দিষ্ট শর্ত পূরণ করে এমন **প্রথম আইটেমটি** খোঁজার জন্য।
   - **উদাহরণ:**

     JavaScript

     ```
     const users = [{ id: 1, name: "আহমেদ" }, { id: 2, name: "সীমা" }];
     const foundUser = users.find(user => user.id === 2);
     console.log(foundUser); // { id: 2, name: 'সীমা' }
     const notFoundUser = users.find(user => user.id === 3);
     console.log(notFoundUser); // undefined
     ```

8. **`findIndex(callbackFn)`**
   - **কী করে:** অ্যারের প্রতিটি উপাদানের উপর একটি ফাংশন চালায় এবং শর্ত পূরণ করে এমন **প্রথম উপাদানের ইন্ডেক্স** ফেরত দেয়।

- **কী হয়:** যদি শর্ত পূরণ হয় এমন কোনো উপাদান খুঁজে পায়, তবে তার ইন্ডেক্স ফেরত দেয়। না পেলে `-1` ফেরত দেয়।
- **কেন করে:** একটি নির্দিষ্ট শর্ত পূরণ করে এমন আইটেমটির ইন্ডেক্স জানার জন্য।
- ** উদাহরণ:**

  JavaScript

  ```
  const points = [10, 25, 30, 40];
  const firstOver25Index = points.findIndex(p => p > 25);
  console.log(firstOver25Index); // 2 (কারণ 30 হলো প্রথম সংখ্যা যা 25 এর চেয়ে বড়)
  ```

9. **`entries()`**
   - **কী করে:** একটি অ্যারে **ইটারেটর অবজেক্ট** ফেরত দেয়, যা অ্যারের প্রতিটি `[index, value]` জোড়া ধারণ করে।
   - **কী হয়:** একটি নতুন `Array Iterator` অবজেক্ট ফেরত দেয়।
   - **কেন করে:** `for...of` লুপ ব্যবহার করে একটি অ্যারের উপাদান এবং তার ইন্ডেক্স উভয়ই অ্যাক্সেস করার জন্য।
   - **উদাহরণ:**

     JavaScript

     ```
     const letters = ["A", "B", "C"];
     for (const [idx, char] of letters.entries()) {
       console.log(`Index ${idx}: ${char}`);
     }
     /*
     আউটপুট:
     Index 0: A
     Index 1: B
     Index 2: C
     */
     ```

---

২. Object (অবজেক্ট): বৈশিষ্ট্যসহ ডেটার ব্যাগ

জাভাস্ক্রিপ্টে অবজেক্ট হলো **কী-ভ্যালু জোড়া (key-value pairs)** আকারে ডেটার একটি সংগ্রহ। এটি একটি **আনঅর্ডার্ড (unordered)** ডেটা স্ট্রাকচার (যদিও আধুনিক ইঞ্জিনগুলি সাধারণত ইনসারশন অর্ডার বজায় রাখে)। একটি অবজেক্টকে একটি বাস্তব বিশ্বের বস্তুর ডিজিটাল মডেল হিসেবে ভাবতে পারেন, যেখানে প্রতিটি বৈশিষ্ট্য (`property`) একটি `key` দ্বারা চিহ্নিত এবং এর একটি `value` থাকে।

কেন অবজেক্ট ব্যবহার করবেন?

- যখন আপনার কাছে **একটি নির্দিষ্ট সত্তা (entity)**-এর বিভিন্ন **বৈশিষ্ট্য (properties)** সংরক্ষণ করতে হয় (যেমন, একজন ব্যক্তি, একটি বই, একটি গাড়ির সম্পূর্ণ ডেটা)।

- যখন ডেটাগুলোকে তাদের **বর্ণনামূলক নাম (key)** ব্যবহার করে অ্যাক্সেস করতে চান, ইন্ডেক্স ব্যবহার করে নয়।
- যখন ডেটাগুলোর কোনো **নির্দিষ্ট ক্রম** না হলেও সমস্যা নেই।

অবজেক্ট তৈরির আধুনিক উপায়

সবচেয়ে প্রচলিত এবং আধুনিক পদ্ধতি হলো **অবজেক্ট লিটারাল** ব্যবহার করা:

JavaScript

```
const userProfile = {
  name: "সাকিব",
  age: 35,
  email: "sakib@example.com",
  isActive: true,
  address: {
    street: "১০ নং রোড",
    city: "ঢাকা"
  } // অবজেক্টের মধ্যে অন্য অবজেক্টও থাকতে পারে
};

const emptyConfig = {}; // একটি খালি অবজেক্ট
```

অবজেক্ট প্রোপার্টিসমূহ:

জাভাস্ক্রিপ্টের প্রতিটি অবজেক্ট প্রোপার্টির সাথে কিছু অভ্যন্তরীণ অ্যাট্রিবিউট যুক্ত থাকে যা সেই প্রোপার্টির আচরণ নিয়ন্ত্রণ করে। এগুলোকে **প্রোপার্টি ডিস্ক্রিপ্টর (Property Descriptors)** বলা হয়।

1. প্রোপার্টি ডিস্ক্রিপ্টর এবং তাদের অ্যাট্রিবিউট (`value`, `writable`, `enumerable`, `configurable`, `get`, `set`)

এগুলো সরাসরি অ্যাক্সেস করা যায় না, তবে `Object.getOwnPropertyDescriptor()` মেথড দিয়ে দেখা যায় এবং `Object.defineProperty()` দিয়ে সেট করা যায়।

- **`value`**: প্রোপার্টির প্রকৃত মান। (শুধুমাত্র ডেটা প্রোপার্টির জন্য)।
- **`writable`**: `true` হলে প্রোপার্টির `value` পরিবর্তন করা যাবে। `false` হলে `value` পরিবর্তন করা যাবে না। (শুধুমাত্র ডেটা প্রোপার্টির জন্য)।
- **`enumerable`**: `true` হলে প্রোপার্টি `for...in` লুপ বা `Object.keys()` এর মতো ইটারেটর মেথড দ্বারা দেখা যাবে। `false` হলে যাবেনা।
- **`configurable`**: `true` হলে প্রোপার্টিটি ডিলিট করা যাবে, এবং এর অ্যাট্রিবিউটগুলো (যেমন `writable`, `enumerable`) পরিবর্তন করা যাবে। `false` হলে এগুলো করা যাবে না।
- **`get`**: একটি ফাংশন যা প্রোপার্টির মান অ্যাক্সেস করার সময় কল হয় (একটি "getter" ফাংশন)। (শুধুমাত্র অ্যাক্সেসর প্রোপার্টির জন্য)।
- **`set`**: একটি ফাংশন যা প্রোপার্টির মান সেট করার সময় কল হয় (একটি "setter" ফাংশন)। (শুধুমাত্র অ্যাক্সেসর প্রোপার্টির জন্য)।

কেন জানবেন: এগুলো আপনাকে অবজেক্টের প্রোপার্টিগুলোর উপর **খুব সূক্ষ্ম নিয়ন্ত্রণ** অর্জন করতে সাহায্য করে। যেমন, একটি প্রোপার্টিকে **রিড-ওনলি** করা, লুপ থেকে লুকানো, বা ডিলিট হওয়া থেকে রক্ষা করা।

উদাহরণ:

JavaScript

```
const product = {
  name: "বই",
  price: 150
};

// 'name' প্রোপার্টির ডিস্ক্রিপ্টর দেখা
console.log("Name Descriptor:", Object.getOwnPropertyDescriptor(product,
'name'));
/*
আউটপুট:
Name Descriptor: {
  value: 'বই',
  writable: true,     // ডিফল্টভাবে পরিবর্তনযোগ্য
  enumerable: true,   // ডিফল্টভাবে গণনাযোগ্য
  configurable: true  // ডিফল্টভাবে কনফিগারযোগ্য
}
*/

// 'price' প্রোপার্টিকে রিড-ওনলি করা (writable: false)
Object.defineProperty(product, 'price', {
  writable: false // এখন price পরিবর্তন করা যাবে না
});
product.price = 200; // এই লাইনটি কাজ করবে না (strict mode এ TypeError দেবে)
console.log("পরিবর্তনের চেষ্টা করার পর price:", product.price); // 150 (মান অপরিবর্তিত)

// একটি হিডেন প্রোপার্টি যোগ করা (enumerable: false)
Object.defineProperty(product, 'internalId', {
  value: "SKU12345",
  enumerable: false, // এটি Object.keys() বা for...in লুপে আসবে না
  writable: false,
  configurable: false
});

console.log("Product keys:", Object.keys(product)); // ['name', 'price']
console.log("Internal ID (সরাসরি অ্যাক্সেস):", product.internalId); // SKU12345
```

2. প্রোটোটাইপ চেইন (`__proto__`, `Object.getPrototypeOf()`)

- **কী:** জাভাস্ক্রিপ্টের অবজেক্টগুলো **প্রোটোটাইপাল ইনহেরিটেন্স** মডেল অনুসরণ করে। প্রতিটি অবজেক্টের একটি **প্রোটোটাইপ অবজেক্ট** থাকে, যা থেকে এটি প্রোপার্টি এবং মেথড ইনহেরিট করে।
- **কী করে:** যখন কোনো প্রোপার্টি সরাসরি একটি অবজেক্টে খুঁজে পাওয়া যায় না, তখন জাভাস্ক্রিপ্ট তার প্রোটোটাইপ চেইন ধরে উপরের দিকে সেই প্রোপার্টির জন্য অনুসন্ধান করে।
- **কেন করে:** ইনহেরিটেন্স বাস্তবায়ন করে, যাতে একাধিক অবজেক্ট একই ফাংশন বা প্রোপার্টি শেয়ার করতে পারে মেমরি বাঁচায় এবং কোড পুনরায় ব্যবহারযোগ্য করে তোলে।
- **খুঁটিনাটি:**

    - `__proto__` প্রোপার্টিটি সরাসরি ব্যবহার করাকে নিরুৎসাহিত করা হয় (deprecated)।
    - এর পরিবর্তে `Object.getPrototypeOf()` (প্রোটোটাইপ অ্যাক্সেস করতে) এবং `Object.setPrototypeOf()` (প্রোটোটাইপ সেট করতে) ব্যবহার করা উচিত।
    - অবজেক্ট তৈরির সময় `Object.create()` ব্যবহার করে প্রোটোটাইপ সেট করা সবচেয়ে আধুনিক ও সঠিক উপায়।
- **উদাহরণ:**

JavaScript

```
// একটি প্রোটোটাইপ অবজেক্ট (পিতামহ)
const vehicle = {
  isEngineOn: false,
  startEngine() {
    this.isEngineOn = true;
    console.log("ইঞ্জিন চালু হয়েছে।");
  }
};

// 'car' অবজেক্ট তৈরি করা, যার প্রোটোটাইপ 'vehicle'
const car = Object.create(vehicle);
car.wheels = 4;
car.model = "Toyota";

console.log(car.isEngineOn); // false (vehicle থেকে ইনহেরিট করেছে)
car.startEngine();           // ইঞ্জিন চালু হয়েছে।
console.log(car.isEngineOn); // true (car অবজেক্টের state পরিবর্তন হয়েছে)

// 'car' এর প্রোটোটাইপ 'vehicle' কিনা তা পরীক্ষা করা
console.log(Object.getPrototypeOf(car) === vehicle); // true

// একটি সাধারণ অবজেক্টের ডিফল্ট প্রোটোটাইপ
const plainObj = {};
console.log(Object.getPrototypeOf(plainObj) === Object.prototype); // true
```

3. কনস্ট্রাক্টর প্রোপার্টি (`constructor`)

- **কী:** `constructor`
- **এর ধরন:** একটি **ফাংশন** (অবজেক্টের কনস্ট্রাক্টর ফাংশন)।
- **কী করে:** একটি অবজেক্টকে যে ফাংশন বা ক্লাস ব্যবহার করে তৈরি করা হয়েছে, সেই কনস্ট্রাক্টর ফাংশনটিকে নির্দেশ করে।
- **কেন করে:** একটি অবজেক্ট কোন ব্লুপ্রিন্ট (constructor) থেকে এসেছে তা জানতে। এটি ডেটা টাইপ চেক করার একটি উপায়।
- **খুঁটিনাটি:**
    - এটি প্রোটোটাইপ চেইনের মাধ্যমে অ্যাক্সেস করা হয়।
    - যদি কোনো কাস্টম কনস্ট্রাক্টর না থাকে, তাহলে এটি `Object` কনস্ট্রাক্টরকে নির্দেশ করে।
- **উদাহরণ:**

JavaScript

```
const arr = [];
console.log(arr.constructor === Array); // true

const obj = {};
console.log(obj.constructor === Object); // true

class User {}
const myUser = new User();
console.log(myUser.constructor === User); // true
```

অবজেক্ট মেথডসমূহ:

অবজেক্ট মেথডগুলো অবজেক্টের প্রোপার্টিগুলো নিয়ে কাজ করার জন্য ডিজাইন করা হয়েছে।

A. প্রোপার্টি অ্যাক্সেস ও ম্যানিপুলেশন

1. **ডট নোটেশন (`object.key`)** এবং **ব্র্যাকেট নোটেশন (`object['key']`)**
    - **কী করে:** অবজেক্টের একটি প্রোপার্টির মান অ্যাক্সেস করে, নতুন প্রোপার্টি যোগ করে, বা বিদ্যমান প্রোপার্টির মান পরিবর্তন করে।
    - **কী হয়:** প্রোপার্টির মান ফেরত দেয় (অ্যাক্সেস করলে) বা মান সেট করে (পরিবর্তন করলে)।
    - **কেন করে:** অবজেক্টের ডেটা পড়া বা আপডেট করার সবচেয়ে মৌলিক এবং সরাসরি উপায়।
    - **উদাহরণ:**

      JavaScript

      ```
      const student = { name: "রায়হান", roll: 101 };
      console.log(student.name);     // রায়হান (ডট নোটেশন)
      student.grade = "A+";          // নতুন প্রোপার্টি যোগ
      student['roll'] = 102;         // ব্র্যাকেট নোটেশন দিয়ে মান পরিবর্তন
      console.log(student);          // { name: 'রায়হান', roll: 102, grade:
      'A+' }

      // ব্র্যাকেট নোটেশন যখন প্রোপার্টির নাম ভ্যারিয়েবলে থাকে বা বিশেষ অক্ষর থাকে
      const propName = "name";
      console.log(student[propName]); // রায়হান
      const item = { "item-code": "XYZ" };
      console.log(item["item-code"]); // XYZ
      ```

2. **`delete object.key`** (বা `delete object['key']`)
    - **কী করে:** অবজেক্ট থেকে একটি নির্দিষ্ট প্রোপার্টি এবং তার মান মুছে ফেলে।
    - **কী হয়:** `true` ফেরত দেয় যদি সফলভাবে মোছা হয়, অন্যথায় `false`।
    - **কেন করে:** অবজেক্ট থেকে অপ্রয়োজনীয় বা অস্থায়ী ডেটা সরাতে।
    - **উদাহরণ:**

JavaScript

```
const config = { logging: true, debug: true, port: 8080 };
delete config.debug;
console.log(config); // { logging: true, port: 8080 }
```

B. প্রোপার্টি আহরণ (Extraction) মেথড

1. **`Object.keys(obj)`**
   - **কী করে:** একটি অবজেক্টের **নিজস্ব (own) এবং এনুমারেবল (enumerable) প্রোপার্টি নাম (key)** গুলোর একটি অ্যারে ফেরত দেয়।
   - **কী হয়:** কী-এর একটি **নতুন অ্যারে** ফেরত দেয়।
   - **কেন করে:** একটি অবজেক্টের সমস্ত প্রোপার্টির নাম জানতে এবং সেই নামগুলো ব্যবহার করে অবজেক্টের ভ্যালু অ্যাক্সেস করতে বা লুপ চালাতে।
   - **উদাহরণ:**

     JavaScript

     ```
     const user = { id: 1, name: "মিনা", status: "active" };
     const userKeys = Object.keys(user);
     console.log(userKeys); // ['id', 'name', 'status']

     // for...of লুপ ব্যবহার করে কীগুলো দিয়ে ভ্যালু অ্যাক্সেস
     for (const key of userKeys) {
       console.log(`${key}: ${user[key]}`);
     }
     ```

2. **`Object.values(obj)`**
   - **কী করে:** একটি অবজেক্টের **নিজস্ব (own) এবং এনুমারেবল (enumerable) প্রোপার্টি ভ্যালু (value)** গুলোর একটি অ্যারে ফেরত দেয়।
   - **কী হয়:** ভ্যালুর একটি **নতুন অ্যারে** ফেরত দেয়।
   - **কেন করে:** একটি অবজেক্টের সমস্ত প্রোপার্টির মান জানতে কিন্তু কীগুলো নিয়ে কাজ করার দরকার নেই।
   - **উদাহরণ:**

     JavaScript

     ```
     const productCounts = { apple: 10, banana: 20, orange: 15 };
     const counts = Object.values(productCounts);
     console.log(counts); // [10, 20, 15]

     const totalItems = counts.reduce((sum, count) => sum + count, 0);
     console.log("মোট আইটেম:", totalItems); // 45
     ```

3. **`Object.entries(obj)`**
   - **কী করে:** একটি অবজেক্টের **নিজস্ব (own) এবং এনুমারেবল (enumerable) প্রোপার্টিগুলোর কী-ভ্যালু জোড়া `[key, value]` ফরম্যাটে** একটি অ্যারে ফেরত দেয়।

- **কী হয়:** অ্যারে অফ অ্যারেস (array of arrays) ফেরত দেয়।
- **কেন করে:** একটি অবজেক্টের কী এবং ভ্যালু উভয়ই একসাথে পেতে এবং সেগুলোর উপর লুপ চালাতে বা অ্যারে মেথড প্রয়োগ করে ডেটা পরিবর্তন করতে।
- **উদাহরণ:**

JavaScript

```
const itemInfo = { id: "P123", name: "Laptop", stock: 50 };
const itemEntries = Object.entries(itemInfo);
console.log(itemEntries);
/*
আউটপুট:
[
  ["id", "P123"],
  ["name", "Laptop"],
  ["stock", 50]
]
*/
// for...of লুপ ব্যবহার করে কী এবং ভ্যালু উভয় অ্যাক্সেস
for (const [key, value] of itemEntries) {
  console.log(`${key.toUpperCase()}: ${value}`);
}
```

4. **`Object.fromEntries(iterable)`**
   - **কী করে:** `Object.entries()` এর ঠিক উল্টো কাজ করে। `[key, value]` জোড়ার একটি ইটারেবল (যেমন অ্যারে অফ অ্যারেস বা `Map` অবজেক্ট) থেকে একটি **নতুন অবজেক্ট** তৈরি করে।
   - **কী হয়:** একটি নতুন অবজেক্ট ফেরত দেয়।
   - **কেন করে:** কী-ভ্যালু জোড়ার তালিকা থেকে (যা আপনি অন্য কোথাও থেকে পেতে পারেন বা `Object.entries()` থেকে ম্যানিপুলেট করার পরে) একটি অবজেক্ট তৈরি করার জন্য। এটি ডেটা ট্রান্সফর্মেশনের জন্য একটি শক্তিশালী টুল।
   - **উদাহরণ:**

JavaScript

```
const dataArray = [
  ["productName", "Smartwatch"],
  ["brand", "XYZ"],
  ["price", 199]
];
const newProduct = Object.fromEntries(dataArray);
console.log(newProduct); // { productName: 'Smartwatch', brand:
'XYZ', price: 199 }

// Object.entries() এবং map() ব্যবহার করে অবজেক্টের ভ্যালু পরিবর্তন করে নতুন
অবজেক্ট তৈরি
const oldSettings = { debugMode: true, verbose: false };
const newSettings = Object.fromEntries(
  Object.entries(oldSettings).map(([key, value]) => [key, !
value]) // ভ্যালুগুলো উল্টে দেওয়া
);
console.log(newSettings); // { debugMode: false, verbose: true }
```

C. অবজেক্ট কপি এবং বৈশিষ্ট্য নিয়ন্ত্রণ মেথড

1. **`Object.assign(target, source1, ..., sourceN)`**
    - **কী করে:** একটি বা একাধিক সোর্স অবজেক্ট থেকে সমস্ত এনুমারেবল, নিজস্ব প্রোপার্টিগুলো একটি টার্গেট অবজেক্টে কপি করে।
    - **কী হয়: টার্গেট অবজেক্টটি পরিবর্তিত হয়** এবং সেটিই ফেরত দেয়। এটি একটি **শ্যালো কপি (shallow copy)**।
    - **কেন করে:** একাধিক অবজেক্টের প্রোপার্টিগুলো একটি একক অবজেক্টে মার্জ করতে, অথবা একটি অবজেক্টের শ্যালো কপি তৈরি করতে (প্রথম আর্গুমেন্ট হিসেবে `{}` ব্যবহার করে)।
    - **আধুনিক জাভাস্ক্রিপ্ট:** স্প্রেড অপারেটর (`...`) আধুনিক এবং বেশি ব্যবহৃত বিকল্প।
    - **উদাহরণ:**

      JavaScript

      ```
      const defaults = { color: "blue", size: "M" };
      const userPrefs = { size: "L", theme: "dark" };

      // নতুন অবজেক্টে মার্জ করা (userPrefs ডিফল্টকে ওভাররাইট করবে)
      const finalPrefs = Object.assign({}, defaults, userPrefs);
      console.log(finalPrefs); // { color: 'blue', size: 'L', theme:
      'dark' }

      // স্প্রেড অপারেটরের ব্যবহার (আসল অবজেক্ট অপরিবর্তিত থাকে)
      const finalPrefsSpread = { ...defaults, ...userPrefs };
      console.log(finalPrefsSpread); // { color: 'blue', size: 'L', theme:
      'dark' }
      ```

2. **`Object.create(proto, propertiesObject)`**
    - **কী করে:** একটি নতুন অবজেক্ট তৈরি করে, যা নির্দিষ্ট প্রোটোটাইপ অবজেক্ট এবং (ঐচ্ছিকভাবে) প্রোপার্টিগুলো ধারণ করে।
    - **কী হয়:** একটি **নতুন অবজেক্ট** ফেরত দেয়।
    - **কেন করে:** প্রোটোটাইপাল ইনহেরিটেন্স ব্যবহার করে অবজেক্ট তৈরি করতে, যেখানে আপনি একটি অবজেক্টের প্রোটোটাইপ সরাসরি সেট করতে চান।
    - **উদাহরণ:**

      JavaScript

      ```
      const baseLogger = {
        log(message) {
          console.log(`[LOG]: ${message}`);
        }
      };
      const appLogger = Object.create(baseLogger);
      appLogger.error = function(message) {
        console.error(`[ERROR]: ${message}`);
      };

      appLogger.log("অ্যাপ শুরু হয়েছে।");     // [LOG]: অ্যাপ শুরু হয়েছে।
      ```

```
appLogger.error("একটি সমস্যা হয়েছে!"); // [ERROR]: একটি সমস্যা হয়েছে!
```

3. **`Object.freeze(obj)`**

- **কী করে:** একটি অবজেক্টকে **ফ্রিজ** করে, অর্থাৎ এটি আর **পরিবর্তনযোগ্য থাকে না**। নতুন প্রোপার্টি যোগ করা যাবে না, বিদ্যমান প্রোপার্টি পরিবর্তন করা যাবে না বা মুছে ফেলা যাবে না। এর প্রোটোটাইপও পরিবর্তন করা যাবে না।
- **কী হয়:** ফ্রিজ করা অবজেক্টটি ফেরত দেয়।
- **কেন করে:** যখন আপনি নিশ্চিত করতে চান যে একটি অবজেক্টের ডেটা বা কাঠামো কোনোভাবে পরিবর্তিত হবে না। এটি কনফিগারেশন অবজেক্ট বা ইম্যিউটেবল ডেটা স্ট্রাকচার তৈরির জন্য উপকারী।
- **উদাহরণ:**

  JavaScript

```
const APP_CONSTANTS = { VERSION: "1.0.0", API_URL: "/api/v1" };
Object.freeze(APP_CONSTANTS);
APP_CONSTANTS.VERSION = "1.0.1"; // কাজ করবে না
delete APP_CONSTANTS.API_URL;    // কাজ করবে না
APP_CONSTANTS.NEW_KEY = "value"; // কাজ করবে না
console.log(APP_CONSTANTS);      // { VERSION: '1.0.0', API_URL:
'/api/v1' } (অপরিবর্তিত)
```

4. **`Object.seal(obj)`**

- **কী করে:** একটি অবজেক্টকে **সিল** করে। এর মানে হলো, বিদ্যমান প্রোপার্টিগুলো মুছে ফেলা বা নতুন প্রোপার্টি যোগ করা যাবে না, তবে **বিদ্যমান প্রোপার্টিগুলোর মান পরিবর্তন করা যাবে**।
- **কী হয়:** সিল করা অবজেক্টটি ফেরত দেয়।
- **কেন করে:** যখন আপনি একটি অবজেক্টের কাঠামোকে স্থিতিশীল রাখতে চান কিন্তু এর ভেতরের ডেটা পরিবর্তন করার অনুমতি দিতে চান।
- **উদাহরণ:**

  JavaScript

```
const userSession = { userId: "user123", status: "active" };
Object.seal(userSession);
userSession.status = "inactive";    // মান পরিবর্তন করা যাবে
userSession.lastLogin = new Date(); // নতুন প্রোপার্টি যোগ করা যাবে না
delete userSession.userId;          // প্রোপার্টি মোছা যাবে না
console.log(userSession); // { userId: 'user123', status: 'inactive'
}
```

5. **`Object.isFrozen(obj)`**

- **কী করে:** পরীক্ষা করে যে একটি অবজেক্ট `Object.freeze()` দ্বারা ফ্রিজ করা হয়েছে কিনা।
- **কী হয়:** `true` অথবা `false` ফেরত দেয়।

6. **`Object.isSealed(obj)`**
    - **কী করে:** পরীক্ষা করে যে একটি অবজেক্ট `Object.seal()` দ্বারা সিল করা হয়েছে কিনা।
    - **কী হয়:** `true` অথবা `false` ফেরত দেয়।

কখন Array (অ্যারে) আর কখন Object (অবজেক্ট) ব্যবহার করবেন?

সঠিক ডেটা স্ট্রাকচার নির্বাচন আপনার কোডের কার্যকারিতা, পঠনযোগ্যতা এবং রক্ষণাবেক্ষণে গুরুত্বপূর্ণ প্রভাব ফেলে।

- **Array (অ্যারে) - ক্রমিক তালিকা:**
    - যখন ডেটাগুলোর **ক্রম** বা ধারাবাহিকতা গুরুত্বপূর্ণ (যেমন, একটি ভিডিও প্লেলিস্টের গানের ক্রম)।
    - যখন ডেটাগুলোকে **সংখ্যাসূচক ইন্ডেক্স** দ্বারা অ্যাক্সেস করতে চান।
    - যখন আপনার ডেটাগুলোকে `map()`, `filter()`, `reduce()` এর মতো লিস্টিং অপারেশন করতে হবে।
- **Object (অবজেক্ট) - বৈশিষ্ট্য বা নাম-ভিত্তিক ডেটা:**
    - যখন আপনি একটি **সত্তা (entity)**-এর **বিভিন্ন বৈশিষ্ট্য** বর্ণনা করতে চান (যেমন, একজন ব্যবহারকারীর নাম, ইমেইল, আইডি)।
    - যখন ডেটাগুলোকে তাদের **বর্ণনামূলক নাম (key)** দ্বারা অ্যাক্সেস করতে চান, ইন্ডেক্স দ্বারা নয়।
    - যখন ডেটাগুলোর কোনো নির্দিষ্ট ক্রমের প্রয়োজন নেই।

সাধারণ ভুল এবং সতর্কতা:

- **অ্যারের জন্য `for...in` ব্যবহার:** `for...in` লুপ অবজেক্টের জন্য ডিজাইন করা হয়েছে এবং এটি ইনহেরিট করা প্রোপার্টিও গণনা করতে পারে, যা অ্যারের জন্য অপ্রত্যাশিত ফলাফল দিতে পারে। অ্যারের জন্য `for...of`, `forEach()`, `map()`, `filter()` ব্যবহার করুন।
- **অ্যারেকে `delete` অপারেটর দিয়ে ডেটা মোছা:** `delete` অ্যারের দৈর্ঘ্য পরিবর্তন করে না এবং মাঝখানে খালি স্লট তৈরি করে, যা অপ্রত্যাশিত আচরণ ঘটাতে পারে। অ্যারে থেকে উপাদান সরাতে `pop()`, `shift()`, `splice()` ব্যবহার করুন।
- **অবজেক্টে ক্রমের উপর নির্ভর করা:** যদিও আধুনিক জাভাস্ক্রিপ্ট ইঞ্জিনগুলি অবজেক্টের কী-এর ইনসারশন অর্ডার বজায় রাখে, তবুও এটি স্ট্যান্ডার্ড দ্বারা নিশ্চিত নয়। যদি ডেটার ক্রম গুরুত্বপূর্ণ হয়, তাহলে অ্যারে ব্যবহার করুন।

- **শ্যালোকপি বনাম ডিপ কপি:** `Object.assign()` এবং স্প্রেড অপারেটর (`{...obj}`) দিয়ে অবজেক্ট কপি করলে **শ্যালোকপি** হয়। এর মানে হলো, যদি অবজেক্টের মধ্যে অন্য অবজেক্ট বা অ্যারে থাকে, তবে সেগুলোর রেফারেন্স কপি হয়, মূল ডেটা নয়। ফলে, নেস্টেড অবজেক্টের পরিবর্তন মূল অবজেক্টকেও পরিবর্তন করবে। গভীর কপি করার জন্য JSON এর `JSON.parse(JSON.stringify(obj))` অথবা একটি বিশেষ ডিপ কপি লাইব্রেরি (যেমন Lodash এর `_.cloneDeep()`) ব্যবহার করতে হয়।
- **অপ্রত্যাশিত মিউটেশন:** `push()`, `pop()`, `splice()`, `sort()`, `reverse()` এবং অবজেক্টের সরাসরি প্রোপার্টি অ্যাসাইনমেন্ট (`obj.key = value`) মূল ডেটা স্ট্রাকচারকে পরিবর্তন করে। যদি আপনি মূল ডেটা অপরিবর্তিত রেখে নতুন ডেটা স্ট্রাকচার তৈরি করতে চান, তাহলে `map()`, `filter()`, `slice()`, `concat()` বা স্প্রেড অপারেটর ব্যবহার করুন।