Destructuring Objects and Arrays

destructuring is a powerful syntax that makes it possible to **unpack values from arrays or properties from objects into distinct, individual variables**.

This feature, introduced in ES6, simplifies code by reducing the need for repetitive property access using dot notation or square brackets, leading to cleaner, more readable, and concise code.

How It Works
Destructuring uses patterns that mirror the structure of the data you are accessing, but on the left-hand side of an assignment operator (=) or in function parameter lists. The original array or object is not modified in the process.
It primarily applies to two data structures:

Array Destructuring: Extracts elements based on their position (index) within the array. It uses square bracket syntax ([]).

```
const fruits = ["Bananas", "Oranges", "Apples"];
const [fruit1, fruit2, fruit3] = fruits;
In general terms,
console.log(fruit1); // Output: "Bananas"
```

Object Destructuring: Extracts properties based on their key names. It uses curly brace syntax ({}).

```
const person = { name: "John Doe", age: 30, city: "New York" };
const { name, age, city } = person;

console.log(name); // Output: "John Doe"
```

Key Benefits and Use Cases:

Improved Readability and Conciseness: It simplifies complex data access into a single, declarative line of code.

Easier Value Swapping: You can swap the values of two variables without needing a temporary variable.

```
let a = 10;
let b = 20;
[a, b] = [b, a]; // a is now 20, b is now 10
```

Handling Function Parameters: It's commonly used to extract specific properties from an object passed as a function argument, making the function's expected inputs explicit.

Setting Default Values: You can provide a default value that will be used if the property or array element is missing or undefined.

Nested Data Extraction: It can extract values from deeply nested objects and arrays with a single expression.