JS

JAVASCRIPT ES6 MAGIC

ES5 vs ES6: Unveiling New Features!

Jimmy Ramani
@jimmyramani

Arrow Functions

Arrow functions provide a concise syntax for writing functions.

```
// ES5
function add(a, b) {
  return a + b;
}

// ES6
const add = (a, b) => a + b;
```

Jimmy Ramani
@jimmyramani

Let and Const Declarations

let and const provide block-scoped variable declarations.

```
// ES5
var x = 5;

// ES6
let x = 5;
const y = 10;
```

Jimmy Ramani
@jimmyramani

Template Literals

Template literals allow embedding expressions within string literals using backticks.

```
// ES5
var name = "John";
var message = "Hello, " + name + "!";

// ES6
const name = "John";
const message = `Hello, ${name}!`;
```

Jimmy Ramani
@jimmyramani

Destructuring Assignment with Arrays

Destructuring allows you to extract values from individual variables.

```
// ES5
var numbers = [1, 2, 3];
var a = numbers[0];
var b = numbers[1];
var c = numbers[2];

// ES6
const numbers = [1, 2, 3];
const [a, b, c] = numbers;
```

Jimmy Ramani
@jimmyramani

Destructuring Assignment with Objects

Destructuring allows you to extract values from individual variables.

```
// ES5
var person = { firstName: 'John', lastName: 'Doe' }
var firstName = person.firstName
var lastName = person.lastName


// ES6
const person = { firstName: 'John', lastName: 'Doe' }

const { firstName, lastName } = person
```

Jimmy Ramani
@jimmyramani

Spread Operator with Arrays

The spread operator allows you to split properties.

```
// ES5
var numbers = [1, 2, 3];
var newNumbers = numbers.concat([4, 5]);

// ES6
const numbers = [1, 2, 3];
const newNumbers = [...numbers, 4, 5];
```

Jimmy Ramani
@jimmyramani

Spread Operator with Objects

The spread operator allows you to split properties.

```
// ES5
var person = { firstName: "John", age: 30 };
var newPerson = Object.assign({}, person, { lastName: "Doe" });

// ES6
const person = { firstName: "John", age: 30 };
const newPerson = { ...person, lastName: "Doe" };
```

Jimmy Ramani
@jimmyramani

Classes

Objects with blueprints for code reusability and inheritance.

```
// ES5
function Person(name, age) {
  this.name = name
  this.age = age
}

Person.prototype.greet = function () {
  console.log(`My name is ${this.name} and age ${this.age}.`)
}

const john = new Person('John', 20)
john.greet()
```

```
// ES6
class Person {
  constructor(name, age) {
    this.name = name
    this.age = age
  }

  greet() {
    console.log(`My name is ${this.name} and age ${this.age}.`)
  }
}

const john = new Person('John', 30)
john.greet()
```

Jimmy Ramani
@jimmyramani

Unlocking the Power of ES6: Embrace the Evolution of JavaScript

🚀 These are merely a glimpse of the myriad of innovations that ES6 has bestowed upon JavaScript. But fret not, for the journey of this dynamic language does not end here! JavaScript's relentless quest for improvement has given birth to newer versions, inviting you to a world of boundless possibilities.

🔍 Venture into the realms of ECMAScript 2016 (ES7), ES8, ES9, and beyond, where a plethora of cutting-edge features await your exploration. Witness the transformation, embrace the advancements, and let your code transcend the ordinary.

🌌 Unleash your creativity with arrow functions, let and const, template literals, and enhanced object literals. Feel the power of destructuring assignment, default parameters, and spread and rest operators. Empower your coding prowess with classes and promises. And step into a world of modularity with native modules.

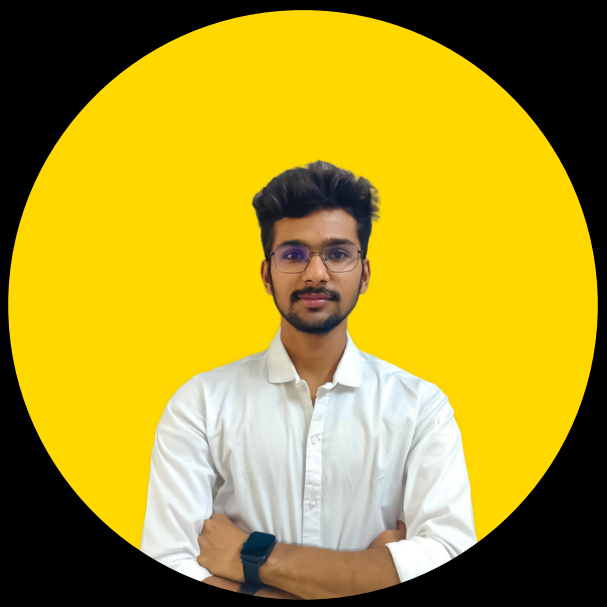
🔮 So, seize the moment! Embrace the evolution of JavaScript, and unlock the gates to a future brimming with innovation. The adventure awaits you—ready to embark on this captivating journey of growth and transformation?

🌠 Dare to explore, for the horizon knows no bounds!🌠

Jimmy Ramani
@jimmyramani

WAS THIS HELPFUL?

Share with a friend who needs it!



Jimmy Ramani

@jimmyramani