ECMAScript 6 (ES6), also known as ECMAScript 2015, introduced several important features and syntax improvements to JavaScript. These features improved readability, maintainability, and performance. Here are some of the key features introduced in ES6:

1. Let and Const

- `let` allows block-scoped variables, which are limited to the block, statement, or expression in which they are used.
- `const` allows block-scoped variables but assigns a constant value that cannot be reassigned after initialization.

```js
let x = 10;
const y = 20;
```

2. Arrow Functions

Arrow functions provide a shorter syntax for writing function expressions and automatically bind `this` to the surrounding context, which simplifies code.

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

3. Template Literals

Template literals use backticks (`` ` ``) to allow string interpolation, multi-line strings, and embedding expressions.

```js
const name = 'Alice';
const greeting = `Hello, ${name}!`;
```

4. Destructuring Assignment

Destructuring allows extracting values from arrays or objects into variables in a concise manner.

```js
// Array Destructuring
const [a, b] = [1, 2];

// Object Destructuring
const { name, age } = { name: 'Alice', age: 30 };
```

5. Default Parameters

Functions can have default parameter values, simplifying function calls by allowing optional arguments.

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

6. Rest and Spread Operators

- **Rest** (`...`) allows collecting remaining elements into an array.
- **Spread** (`...`) allows expanding arrays or objects into individual elements.

```js
// Rest operator
function sum(...numbers) {
  return numbers.reduce((acc, num) => acc + num, 0);
}

// Spread operator
const arr1 = [1, 2];
const arr2 = [...arr1, 3, 4];
```

7. Classes

ES6 introduced class syntax, which is syntactic sugar over JavaScript's prototype-based inheritance. It allows for more readable and object-oriented code.

```js
class Person {
  constructor(name) {
    this.name = name;
  }

  greet() {
    console.log(`Hello, ${this.name}`);
  }
}
```

8. Promises

Promises provide a cleaner way to handle asynchronous operations, replacing older callback-based methods.

```js
const fetchData = () => new Promise((resolve, reject) => {
  setTimeout(() => resolve('Data loaded'), 1000);
});

fetchData().then(result => console.log(result));
```

9. Modules

ES6 introduced support for modules, allowing you to import and export code between files.

```js
// In file1.js
export const name = 'Alice';

// In file2.js
import { name } from './file1';
```

10. Iterators and For...Of Loop

The `for...of` loop allows iterating over arrays, strings, maps, and other iterable objects.

```js
const array = [1, 2, 3];
for (const value of array) {
  console.log(value);
}
```

11. Map and Set

ES6 introduced new data structures:

- **Map**: A collection of key-value pairs.
- **Set**: A collection of unique values.

```js
const map = new Map();
map.set('key', 'value');

const set = new Set([1, 2, 3]);
```

12. Symbols

Symbols are a new primitive data type that can be used to create unique, non-colliding keys for object properties.

```js
const symbol1 = Symbol('description');
```

13. Enhanced Object Literals

Object literals in ES6 provide a shorthand for defining methods and properties, as well as dynamic property names.

```js
const age = 30;
const person = {
  name: 'Alice',
  age, // Shorthand for age: age
  greet() {
    console.log('Hello');
  }
};
```

These ES6 features significantly enhanced JavaScript, making it more powerful and easier to write cleaner, more modular, and maintainable code.