Introduction to ES6:

ECMAScript 6 (ES6) is the sixth major release of the ECMAScript language specification. It was released in 2015 and introduced many new features to the JavaScript programming language. Some of the most notable features include:

- Arrow functions, which provide a shorthand syntax for writing anonymous functions
- Template literals, which allow for easier string interpolation
- Destructuring, which allows for more concise and expressive code when working with arrays and objects
- Modules, which provide a way to organize and reuse code across multiple files
- Classes, which provide a more familiar object-oriented syntax for creating objects
- ES6 is also known as ECMAScript 2015.

ES6 is fully supported by all modern browsers, although some older browsers may not support all of its features. There are also transpilers, like Babel, which can convert ES6 code to be compatible with older browsers.

let and const in JS:

In JavaScript, let and const are used to declare variables. They are both block-scoped, which means that their scope is limited to the block in which they are defined, rather than the entire enclosing function or global scope.

let is used to declare a variable that can be reassigned a new value later in the code. Here's an example:

JavaScript
```javascript
let x = 5;
x = 10; // x is now 10
```

const is used to declare a variable that cannot be reassigned a new value. Once a value is assigned to a variable declared with const, it cannot be changed. Here's an example:

```JavaScript
const y = 5;
y = 10; // this will cause a TypeError
```

It's important to note that when you use const to define an object or array, you can still mutate the object or array by adding or modifying properties, but you can't reassign the variable to point to a new object or array.

```JavaScript
const person = {name: 'John Doe'};
person.name = 'Jane Doe'; // This is allowed
person = {name: 'Jane Doe'}; // This will cause a TypeError
```

In general, it's a good practice to use const by default and only use let if you know that the value of a variable will need to be reassigned later in the code.

Template String:

In JavaScript, template strings are a feature of ES6 (ECMAScript 6) that allow for easier string interpolation. They are defined using backticks (``) instead of single or double quotes. Inside a template string, expressions can be included inside curly braces (${}) which will be replaced with their evaluated values.

Here is an example of a template string:

```JavaScript
let name = "John";
let age = 30;
console.log(`My name is ${name} and I am ${age} years old.`);
// Output: My name is John and I am 30 years old.
```

Template strings can also be used to create multi-line strings, which is not possible with regular strings:

JavaScript
```javascript
console.log(`This is a
multi-line string`);
```

Additionally, you can use template literals to create Tagged template literals, where the first argument will be an array of string literals and the second will be the values of the expressions. It's useful when you want to parse the string and return a new string based on it.

JavaScript
```javascript
function parseString(literals, ...values) {
  let result = "";
  for (let i = 0; i < literals.length; i++) {
    result += literals[i];
    if (i < values.length) {
      result += values[i];
    }
  }
  return result;
}
let name = "John";
let age = 30;
console.log(parseString`My name is ${name} and I am ${age} years
old.`);
```

Template strings are a useful feature that make it easier to work with strings and improve code readability.