JavaScript Overview

High-Level

JavaScript is high-level because it hides complex computer details like memory management and CPU registers. You write human-readable code and JavaScript handles the rest.

```
let x = 5;
let y = 10;
console.log(x + y); // 15
```

Garbage Collection

JavaScript automatically removes data from memory when it’s no longer needed. This means you don’t need to manually free memory.

```
function createUser() {
  return { name: "Alice" };
}

let user = createUser();
user = null; // Garbage collector cleans it
```

Interpreted and JIT (Just-In-Time Compilation)

JavaScript is interpreted line-by-line, but modern engines also use JIT compilation to make frequently used code faster.

```
console.log("Hello JS");
```

Multi-Paradigm

JavaScript supports different styles of programming: Procedural, Object-Oriented, and Functional.

```
// Procedural
let nums = [1, 2, 3];
for (let i = 0; i < nums.length; i++) console.log(nums[i]);

// Object-Oriented
let person = { name: "Bob", greet() { console.log("Hi, I'm " + this.name); } };
person.greet();

// Functional
let doubled = nums.map(n => n * 2);
console.log(doubled); // [2,4,6]
```

Prototype-based Object Oriented

Instead of classes, JavaScript objects inherit from other objects via prototypes.

```
function Animal(name) {
  this.name = name;
}
Animal.prototype.speak = function() {
  console.log(this.name + " makes a noise.");
};

let dog = new Animal("Dog");
dog.speak(); // Dog makes a noise.
```

First-Class Functions

Functions are treated like variables: you can pass them around, store them, and return them.

```
function greet(name) {
  return "Hello " + name;
}

function callWithName(fn, name) {
  console.log(fn(name));
}

callWithName(greet, "Alice"); // Hello Alice
```

Dynamic

Variables in JavaScript don’t have fixed types. A variable can hold different types at different times.

```
let x = 5;      // number
x = "Hello";    // now it's a string
console.log(x); // Hello
```

Single-Threaded

JavaScript executes one thing at a time in a single main thread, making execution predictable.

```
console.log("First");
console.log("Second");
console.log("Third");
// Output: First Second Third
```

Non-Blocking Event Loop

Even though JavaScript is single-threaded, it can handle multiple tasks using the event loop. Tasks like fetching data run in the background and return results asynchronously.

```
console.log("Start");

setTimeout(() => {
  console.log("Done after 2 sec");
}, 2000);

console.log("End");
// Output: Start, End, Done after 2 sec
```