Core JavaScript Mastery Guide

Perfect! Let's break down **Native Prototypes in JavaScript** in a clear, structured way for better understanding:

🔹 JavaScript Native Prototypes

In JavaScript, **every object has a prototype**.
A **prototype** is an object from which other objects inherit properties and methods.

The **native prototype** refers to the default prototype of built-in objects like `Object`, `Array`, `String`, etc.

---

1. Accessing Native Prototype

```
js

Object.prototype
Array.prototype
String.prototype
```

- Each built-in object contains methods and properties in its prototype.
- Example: Checking the prototype of an array:

```
html

<script>
console.log(Array.prototype); // Shows all array methods in the console
</script>
```

> Open the browser console to view the prototype methods.

---

2. Updating Native Prototype

You **can add new methods or properties**, but **modifying existing ones is not recommended**.

✅ **Syntax to update or add a prototype method/property:**

```
js

ObjectName.prototype.methodName = function() { ... };
```

Example: Modifying `toLowerCase()` (Not Recommended)

```
html

<script>
String.prototype.toLowerCase = function () {
  return this.toUpperCase();
}

let str = "Hello World";
document.write(str.toLowerCase()); // HELLO WORLD
</script>
```

---

3. Adding a New Method to Prototype

Instead of modifying built-ins, you can **add new methods** safely:

```
html

<script>
String.prototype.firstCase = function () {
  return this.charAt(0).toUpperCase() + this.slice(1).toLowerCase();
}

let str = "hello world";
document.write(str.firstCase()); // Hello world
</script>
```

---

4. Adding Methods to Constructor Function Prototype

- When you define an object using a constructor, you **cannot add a method via instance**.
- You should add methods to the **constructor's prototype**.

✅ **Example:**

```
html

<script>
function Person(id, name) {
  this.id = id;
  this.name = name;
}

// Add method to prototype
Person.prototype.display = function () {
  document.write(this.id + ", " + this.name + "<br>");
}

const p1 = new Person(1, "James");
const p2 = new Person(2, "Nayan");

p1.display();
p2.display();
</script>
```

📌 **Output:**

```
1, James
2, Nayan
```

> All instances inherit methods from the prototype.

---

5. Prototype Chaining

- If both the object and its prototype have the **same property**, the object's own property **overrides** the prototype property.
- The prototype acts as a **fallback**.

Example:

```
html

<script>
function Person(id, name) {
  this.id = id;
```

```
  this.name = name;
}

Person.prototype.name = "John";
Person.prototype.age = 20;

const p1 = new Person(1, "Adam");

document.write("Id: " + p1.id + ", Name: " + p1.name + ", Age: " + p1.age);
</script>
```

📌 **Output:**

```
yaml

Id: 1, Name: Adam, Age: 20
```

- `name` from object overrides prototype.
- `age` is taken from prototype.

✅ Key Points about Native Prototypes:

1. Every object has a prototype from which it inherits methods.
2. Built-in objects have a **native prototype** with standard methods.
3. Avoid modifying built-in methods. Adding new methods is safe.
4. Prototype methods are **shared across all instances** of the object.
5. Prototype chaining allows inheritance of properties if not present on the object itself.

If you want, I can make a **diagram showing Object → Prototype → Prototype Chain → Native Prototype** so it's easy to visualize inheritance in JavaScript.

Do you want me to create that diagram?