Core JavaScript Mastery Guide

Perfect! Let's summarize **JavaScript Object Constructors** in a clear, structured way, so it's easy for understanding and exam preparation.

🔹 JavaScript Object Constructors

An **object constructor** is a function that is used to **create multiple objects of the same type**.
It acts as a **template** for creating objects and initializing their properties.

---

1. Creating Objects Using a Constructor Function

✅ **Syntax:**

```
js

function ConstructorName(param1, param2, ...) {
  this.prop1 = param1;
  this.prop2 = param2;
}
const obj = new ConstructorName(arg1, arg2);
```

- `this` → refers to the object instance being created.
- `new` → creates a new object and sets its prototype to the constructor's prototype.

---

2. Example: Simple Constructor

```
html

<script>
function Tree() {
  this.name = "palm";
  this.age = 5;
}
const tree1 = new Tree();
```

```
document.write("name = " + tree1.name + ", age = " + tree1.age);
</script>
```

📌 **Output:**

```
name = palm, age = 5
```

---

3. Constructor with Parameters

```
html

<script>
function Bike(name, speed, gear) {
  this.name = name;
  this.speed = speed;
  this.gear = gear;
}

const bike1 = new Bike("Honda", 100, 4);
const bike2 = new Bike("Yamaha", 200, 6);

document.write(JSON.stringify(bike1) + "<br>");
document.write(JSON.stringify(bike2));
</script>
```

📌 **Output:**

```
json

{"name":"Honda","speed":100,"gear":4}
{"name":"Yamaha","speed":200,"gear":6}
```

✅ This shows how constructors **reuse code** for multiple objects.

---

4. Adding Methods in Constructor

```
html

<script>
function HouseAddress(no, street, city) {
  this.houseNo = no;
```

```
  this.street = street;
  this.city = city;
  this.getFullAddress = function () {
    return this.houseNo + ", " + this.street + ", " + this.city;
  };
}

const house = new HouseAddress(20, "Cooper Square", "New York");
document.write(house.getFullAddress());
</script>
```

📌 **Output:**

`20, Cooper Square, New York`

- **Important:** If you add a property to a single object after creation, it won't affect other objects.

---

5. Using Prototype for Methods

- Methods added to **prototype** are shared across all objects created by the constructor.
- Saves memory because the method is not recreated for each object.

✅ **Syntax:**

```
js

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

✅ **Example:**

```
html

<script>
function Bike(name, speed, gear) {
  this.name = name;
  this.speed = speed;
  this.gear = gear;
}

Bike.prototype.BikeDetails = function () {
  return this.name + ", " + this.speed + ", " + this.gear;
};
```

```
const bike1 = new Bike("Honda", 100, 4);
const bike2 = new Bike("Yamaha", 200, 6);

document.write(bike1.BikeDetails() + "<br>");
document.write(bike2.BikeDetails());
</script>
```

📌 **Output:**

```
Honda, 100, 4
Yamaha, 200, 6
```

---

6. Built-in Object Constructors

Constructor	Description	Example
Array	Create an array	`new Array()`
String	Create a string	`new String("Hello")`
Number	Create a number	`new Number(92)`
Boolean	Create a boolean	`new Boolean(true)`
Set	Create a Set	`new Set()`
Map	Create a Map	`new Map()`
Object	Create an Object	`new Object()`
Function	Create a Function	`new Function("x", "return x*x")`
Date	Create a Date object	`new Date()`
RegExp	Create RegExp	`new RegExp("\\d+")`

Constructor	Description	Example
Error	Create Error object	`new Error("message")`
Symbol	Create Symbol	`Symbol("description")`

> ✅ **Tip:** You can also use **literal syntax** (`[]`, `{}`, `""`, `0`) instead of constructors for primitives.

If you want, I can create a **visual diagram of Object Constructors vs Prototype Methods vs Literal Objects** to make it super easy to remember for exams.

Do you want me to do that?