Core JavaScript Mastery Guide

Great! Let's organize **JavaScript ES5 Object Methods** into a clear, structured guide with examples for easy understanding:

🔹 JavaScript ES5 Object Methods

ES5 introduced several static object methods to **manipulate, inspect, and protect objects**.
These methods are categorized into:

1. **Manipulating Objects**
2. **Protecting Objects**

---

1. Methods to Manipulate Objects

a) `Object.create(proto, [propertiesObject])`

Creates a new object with the specified prototype.

```
js

const person = { firstName: "John", lastName: "Doe" };
const student = Object.create(person);
student.age = 18;

console.log(student.firstName); // John
console.log(student.lastName);  // Doe
console.log(student.age);       // 18
```

---

b) `Object.defineProperty(obj, prop, descriptor)`

Defines or updates a single property with detailed attributes.

```
js

const car = { brand: "Tata", model: "Nexon", price: 1000000 };

Object.defineProperty(car, "gears", {
```

```
  value: 6,
  writable: true,
  enumerable: true,
  configurable: true
});

console.log(car);
// { brand: 'Tata', model: 'Nexon', price: 1000000, gears: 6 }
```

---

c) `Object.defineProperties(obj, props)`

Defines or modifies **multiple properties** at once.

```
js

const obj = {};

Object.defineProperties(obj, {
  property1: { value: 42, writable: true },
  property2: { value: "Tutorials Point", writable: false }
});

console.log(obj.property1); // 42
console.log(obj.property2); // Tutorials Point
```

---

d) `Object.getOwnPropertyDescriptor(obj, prop)`

Returns the **descriptor** of a property, including `value`, `writable`, `enumerable`, and `configurable`.

```
js

const obj = { property1: 42 };
const descriptor = Object.getOwnPropertyDescriptor(obj, 'property1');
console.log(descriptor);
// { value: 42, writable: true, enumerable: true, configurable: true }
```

---

e) `Object.getOwnPropertyNames(obj)`

Returns **all property names** of the object (enumerable or not).

```js
const obj = { a: 10, b: 20, c: 30 };
console.log(Object.getOwnPropertyNames(obj)); // ["a", "b", "c"]
```

---

f) `Object.getPrototypeOf(obj)`

Returns the **prototype** of the object.

```js
const proto = { name: "John Doe" };
const obj = Object.create(proto);

console.log(Object.getPrototypeOf(obj)); // { name: "John Doe" }
```

---

g) `Object.keys(obj)`

Returns an **array of enumerable property names**.

```js
const person = { name: "John Doe", age: 20, profession: "Engineer" };
console.log(Object.keys(person)); // ["name", "age", "profession"]
```

---

2. Methods to Protect Objects

a) `Object.freeze(obj)`

Prevents **adding, removing, or modifying properties**.

```js
const obj = { prop: 23 };
Object.freeze(obj);
```

```
obj.prop = 33; // Does nothing in non-strict mode
console.log(obj.prop); // 23
```

---

b) `Object.seal(obj)`

Prevents **adding or deleting properties**, but allows modifying existing ones.

```
js

const obj = { property: 34 };
Object.seal(obj);
obj.property = 33; // Allowed
delete obj.property; // Not allowed
console.log(obj.property); // 33
```

---

c) `Object.isFrozen(obj)`

Checks if the object is frozen.

```
js

const obj = { age: 21 };
console.log(Object.isFrozen(obj)); // false
Object.freeze(obj);
console.log(Object.isFrozen(obj)); // true
```

---

d) `Object.isSealed(obj)`

Checks if the object is sealed.

```
js

const obj = { name: "John Doe" };
console.log(Object.isSealed(obj)); // false
Object.seal(obj);
console.log(Object.isSealed(obj)); // true
```

---

e) `Object.preventExtensions(obj)`

Prevents **adding new properties**.

```js
const obj = {};
Object.preventExtensions(obj);

try {
  Object.defineProperty(obj, 'name', { value: "John Doe" });
} catch(e) {
  console.log(e); // TypeError: Cannot define property name, object is not extensible
}
```

---

f) `Object.isExtensible(obj)`

Checks if an object **can have new properties added**.

```js
const obj = { name: "John Doe" };
console.log(Object.isExtensible(obj)); // true
Object.preventExtensions(obj);
console.log(Object.isExtensible(obj)); // false
```

✅ Key Takeaways:

1. **Manipulation methods** (`create`, `defineProperty`, `defineProperties`, `getOwnPropertyDescriptor`, etc.) allow **dynamic control** over objects.
2. **Protection methods** (`freeze`, `seal`, `preventExtensions`, etc.) enforce **immutability or restricted updates**.
3. ES5 Object methods make **JavaScript objects more powerful, safe, and structured**.

If you want, I can create a **single table with all ES5 object methods, syntax, and examples** for quick reference—so you can memorize or use it for exams.

Do you want me to make that table?