Core JavaScript Mastery Guide

Let's break down **JavaScript Object Properties** with detailed explanations and internal workings. I'll explain it conceptually and with examples so it's easier to visualize.

1️⃣ What is an Object Property?

An **object property** is a **key-value pair**:

```
js

const fruit = {
  name: "Apple",
  price: 100
};
```

- `name` and `price` → **keys** (also called property names)
- `"Apple"` and `100` → **values**
- Keys **must be strings or symbols**; if you use another type (like a number), it gets **converted to string** automatically.
- Values can be **anything**: string, number, array, object, function, etc.

2️⃣ Accessing Object Properties

a) Dot Notation

```
js

fruit.name; // "Apple"
fruit.price; // 100
```

- Simple and most common.
- Works only if the key is a **valid identifier**.

b) Square Bracket Notation

```
js
```

```
fruit["name"]; // "Apple"
fruit["price"]; // 100
```

- Useful when key is **not a valid identifier** (like contains spaces, hyphens, numbers).

```
js

const obj = {
  "fruit-name": "Mango"
};
console.log(obj["fruit-name"]); // Mango
```

c) Using Expressions

```
js

const num = {10: "ten", 20: "twenty"};
const x = 10;
console.log(num[x + 10]); // "twenty"
```

- The expression `x + 10` evaluates to `20`, so `num[20]` is accessed.

3️⃣ Nested Objects

Access **nested properties**:

```
js

const cars = {
  audi: { model: "Q7", price: 10000000 },
  bmw: { model: "S20D", price: 8000000 }
};

console.log(cars.audi.model); // Q7
console.log(cars["bmw"]["price"]); // 8000000
```

4️⃣ Adding or Updating Properties

- **Update existing property**:

```js
fruit.name = "Apple";    // dot notation
fruit["price"] = 200;    // bracket notation
```

- **Add new property:**

```js
fruit.expiry = "5 days";
```

Resulting object:

```js
{ name: "Apple", price: 200, expiry: "5 days" }
```

5️⃣ Deleting Properties

```js
delete fruit.name;
console.log(fruit); // { price: 200, expiry: "5 days" }
```

- Removes the key-value pair from the object.

6️⃣ Enumerating Properties

a) Using `for…in` loop

```js
const table = { color: "brown", shape: "round", price: 10000 };

for (let key in table) {
  console.log(key + ": " + table[key]);
}
```

Output:

```less
```

```
color: brown
shape: round
price: 10000
```

b) Using `Object.keys()` or `Object.entries()`

```js
Object.keys(table); // ["color", "shape", "price"]
Object.values(table); // ["brown", "round", 10000]
Object.entries(table); // [["color","brown"], ["shape","round"], ["price",10000]]
```

7️⃣ Property Attributes (Internal)

Every property has **four attributes**:

Attribute	Description
`value`	The actual value of the property
`writable`	`true` if value can be changed
`enumerable`	`true` if property shows up in loops like `for…in`
`configurable`	`true` if property can be deleted or changed via `defineProperty`

- By default, ** `writable` , `enumerable` , configurable `** are` true` for normal properties.
- To modify attributes (except `value`), use `Object.defineProperty()` :

```js
Object.defineProperty(fruit, "price", {
  writable: false,    // cannot change
  enumerable: false,  // won't show in for...in
```

```
  configurable: false  // cannot delete
});
```

8️⃣ Key Takeaways

1. Objects = collection of key-value pairs.
2. Keys = strings or symbols. Values = anything.
3. Access properties using **dot**, **bracket**, or **expression**.
4. Nested objects follow the same rules.
5. Can **add**, **update**, **delete**, and **enumerate** properties.
6. Each property has **internal attributes** controlling its behavior.

If you want, I can make a **diagram showing objects, properties, and attributes** along with **dot vs bracket access and nested objects**. It will make this very visual and easy to remember.

Do you want me to make that diagram?