**1. Introduction to Objects**

* **Definition**: Objects in JavaScript are collections of key-value pairs, like containers that hold data and methods (functions).
* **Creating an Object**:
  + Using Object Literals:

let person = {

name: "Hamza",

age: 25

};

* + Using the new Object() syntax:

javascript

Copy code

let person = new Object();

person.name = "Hamza";

person.age = 25;

**2. Accessing and Modifying Object Properties**

* **Dot Notation**:

console.log(person.name); // Hamza

person.age = 26; // Modify

* **Bracket Notation**:

console.log(person["name"]); // Hamza

person["age"] = 27; // Modify

* **Adding New Properties**:

person.country = "Nigeria";

**3. Nested Objects and Arrays**

* **Objects within Objects**:

let person = {

name: "Hamza",

address: {

city: "Lagos",

country: "Nigeria"

}

};

console.log(person.address.city); // Lagos

* **Arrays within Objects**:

let person = {

name: "Hamza",

hobbies: ["reading", "coding", "sports"]

};

console.log(person.hobbies[1]); // coding

**4. Object Methods**

* **Adding Methods**:

let person = {

name: "Hamza",

greet: function() {

return "Hello, " + this.name;

}

};

console.log(person.greet()); // Hello, Hamza

**5. this Keyword in Objects**

* this refers to the current object in which a method is defined.
* **Example**:

let person = {

name: "Hamza",

greet() {

console.log("Hello, " + this.name);

}

};

person.greet(); // Hello, Hamza

**6. Object Destructuring (ES6)**

* **Extracting Properties**:

const { name, age } = person;

console.log(name); // Hamza

**7. Object Iteration**

* **Using for...in Loop**:

for (let key in person) {

console.log(key + ": " + person[key]);

}

* **Using Object.keys(), Object.values(), and Object.entries()**:

console.log(Object.keys(person)); // ['name', 'age']

console.log(Object.values(person)); // ['Hamza', 25]

console.log(Object.entries(person)); // [['name', 'Hamza'], ['age', 25]]

**8. Advanced Object Concepts**

* **Object.freeze()**: Makes an object immutable.

Object.freeze(person);

person.name = "Mustapha"; // Error: Cannot modify a frozen object

* **Object.seal()**: Prevents adding or removing properties but allows modifying existing properties.

Object.seal(person);

person.name = "Mustapha"; // Works

person.country = "Nigeria"; // Error: Cannot add new properties

* **Prototype Inheritance**:
  + Objects in JavaScript inherit from a prototype, and you can add properties or methods to prototypes to share them across all instances.

function Person(name) {

this.name = name;

}

Person.prototype.greet = function() {

return "Hello, " + this.name;

};

let hamza = new Person("Hamza");

console.log(hamza.greet()); // Hello, Hamza

**9. Object.assign() and Spread Operator**

* **Merging Objects**:

let person = { name: "Hamza" };

let info = { age: 25, country: "Nigeria" };

let merged = Object.assign({}, person, info);

// OR with spread syntax

let merged2 = { ...person, ...info };

**10. JSON (JavaScript Object Notation)**

* **Convert Object to JSON**:

let jsonString = JSON.stringify(person);

* **Convert JSON to Object**:

let jsonObject = JSON.parse(jsonString);

These lessons will give you a well-rounded understanding of objects in JavaScript, from the basics to more advanced concepts. Practice writing code to see how these concepts work in action!

Object destructuring is a feature introduced in ES6 (ECMAScript 2015) that allows you to extract properties from an object and assign them to variables in a more concise and readable way. It’s particularly helpful for working with objects that have multiple properties, as it allows you to easily "unpack" values from an object into separate variables.

**Basic Syntax**

The syntax for object destructuring is straightforward. Let’s say you have an object and you want to extract some of its properties into variables:

const person = {

name: "Hamza",

age: 25,

country: "Nigeria"

};

// Object destructuring

const { name, age, country } = person;

console.log(name); // "Hamza"

console.log(age); // 25

console.log(country); // "Nigeria"

In this example, { name, age, country } on the left side of the assignment is the destructuring pattern. JavaScript will look for properties with these names in the person object and assign them to variables with the same names.

**Custom Variable Names**

If you want to assign the properties to variables with different names, you can do so like this:

const person = {

name: "Hamza",

age: 25

};

const { name: fullName, age: years } = person;

console.log(fullName); // "Hamza"

console.log(years); // 25

Here, name: fullName tells JavaScript to take the name property from person and assign it to a variable named fullName.

**Default Values**

You can also set default values in case a property doesn’t exist in the object:

const person = {

name: "Hamza"

};

const { name, age = 30 } = person;

console.log(name); // "Hamza"

console.log(age); // 30 (default value used because "age" is undefined in "person")

**Nested Object Destructuring**

Object destructuring also works with nested objects:

const person = {

name: "Hamza",

address: {

city: "Lagos",

country: "Nigeria"

}

};

const { name, address: { city, country } } = person;

console.log(name); // "Hamza"

console.log(city); // "Lagos"

console.log(country); // "Nigeria"

**Destructuring Function Parameters**

Object destructuring is also commonly used in function parameters, especially when working with objects with multiple properties.

const displayInfo = ({ name, age }) => {

console.log(`Name: ${name}, Age: ${age}`);

};

const person = { name: "Hamza", age: 25 };

displayInfo(person); // Name: Hamza, Age: 25

Here, the displayInfo function takes an object as an argument and immediately destructures its name and age properties.

**Rest Operator in Destructuring**

You can use the rest operator (...) to gather the remaining properties into a separate object.

const person = {

name: "Hamza",

age: 25,

country: "Nigeria"

};

const { name, ...rest } = person;

console.log(name); // "Hamza"

console.log(rest); // { age: 25, country: "Nigeria" }

The rest variable will contain the remaining properties not explicitly destructured.

Object destructuring is a powerful, cleaner, and more readable way to work with objects in JavaScript, especially as your code grows and you work with complex objects or large numbers of properties.