JavaScript Objects Review

Object Basics

- **Definition**: An object is a data structure that is made up of properties. A property consists of a key and a value. To access data from an object you can use either dot notation or bracket notation.

Example Code

```
const person = {
  name: "Alice",
  age: 30,
  city: "New York"
};


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

To set a property of an existing object you can use either dot notation or bracket notation together with the assignment operator.

Example Code

```
const person = {
  name: "Alice",
  age: 30
};


person.job = "Engineer"
person["hobby"] = "Knitting"
console.log(person);  // {name: 'Alice', age: 30, job: 'Engineer', hobby: 'Knitting'}
```

Removing Properties From an Object

- **delete Operator**: This operator is used to remove a property from an object.

Example Code

```
const person = {
  name: "Alice",
  age: 30,
  job: "Engineer"
};

delete person.job;

console.log(person.job); // undefined
```

Checking if an Object has a Property

- **hasOwnProperty() Method**: This method returns a boolean indicating whether the object has the specified property as its own property.

Example Code

```
const person = {
  name: "Alice",
  age: 30
};

console.log(person.hasOwnProperty("name")); // true
console.log(person.hasOwnProperty("job")); // false
```

- **in Operator**: This operator will return true if the property exists in the object.

Example Code

```
const person = {
  name: "Bob",
```

```
  age: 25
};

console.log("name" in person);  // true
```

Accessing Properties From Nested Objects

- **Accessing Data**: Accessing properties from nested objects involves using the dot notation or bracket notation, much like accessing properties from simple objects. However, you'll need to chain these accessors to drill down into the nested structure.

Example Code

```
const person = {
  name: "Alice",
  age: 30,
  contact: {
    email: "alice@example.com",
    phone: {
      home: "123-456-7890",
      work: "098-765-4321"
    }
  }
};

console.log(person.contact.phone.work); // "098-765-4321"
```

Primitive and Non Primitive Data Types

- **Primitive Data Types**: These data types include numbers, strings, booleans, null, undefined, and symbols. These types are called "primitive" because they represent single values and are not objects. Primitive values are immutable, which means once they are created, their value cannot be changed.

- **Non Primitive Data Types**: In JavaScript, these are objects, which include regular objects, arrays, and functions. Unlike primitives, non-primitive types can hold multiple values as properties or elements.

Object Methods

- **Definition**: Object methods are functions that are associated with an object. They are defined as properties of an object and can access and manipulate the object's data. The this keyword inside the method refers to the object itself, enabling access to its properties.

Example Code

```
const person = {
 name: "Bob",
 age: 30,
 sayHello: function() {
  return "Hello, my name is " + this.name;
 }
};


console.log(person.sayHello()); // "Hello, my name is Bob"
```

Object Constructor

- **Definition**: In JavaScript, a constructor is a special type of function used to create and initialize objects. It is invoked with the new keyword and can initialize properties and methods on the newly created object. The Object() constructor creates a new empty object.

Example Code

```
new Object()
```

Working with the Optional Chaining Operator (?.)

- **Definition**: This operator lets you safely access object properties or call methods without worrying whether they exist.

Example Code

```
const user = {
  name: "John",
  profile: {
    email: "john@example.com",
    address: {
      street: "123 Main St",
      city: "Somewhere"
    }
  }
};


console.log(user.profile?.address?.street); // "123 Main St"
console.log(user.profile?.phone?.number);   // undefined
```

Object Destructuring

- **Definition**: Object destructuring allows you to extract values from objects and assign them to variables in a more concise and readable way.

Example Code

```
const person = { name: "Alice", age: 30, city: "New York" };


const { name, age } = person;


console.log(name); // Alice
console.log(age);  // 30
```

Working with JSON

- **Definition**: JSON stands for JavaScript Object Notation. It is a lightweight, text-based data format that is commonly used to exchange data between a server and a web application.

Example Code

```
{
  "name": "Alice",
  "age": 30,
  "isStudent": false,
  "list of courses": ["Mathematics", "Physics", "Computer Science"]
}
```

- **JSON.stringify()**: This method is used to convert a JavaScript object into a JSON string. This is useful when you want to store or transmit data in a format that can be easily shared or transferred between systems.

Example Code

```
const user = {
  name: "John",
  age: 30,
  isAdmin: true
};


const jsonString = JSON.stringify(user);
console.log(jsonString); // '{"name":"John","age":30,"isAdmin":true}'
```

- **JSON.parse()**: This method converts a JSON string back into a JavaScript object. This is useful when you retrieve JSON data from a web server or localStorage and you need to manipulate the data in your application.

Example Code

```
const jsonString = '{"name":"John","age":30,"isAdmin":true}';
const userObject = JSON.parse(jsonString);
// result: { name: 'John', age: 30, isAdmin: true }
console.log(userObject);
```