Understanding Objects in JavaScript

In JavaScript, objects are a fundamental part of the language and are essential for building complex applications. They are collections of properties, where each property is a key-value pair. Here’s a detailed overview of JavaScript objects:

Creating Objects

1. Object Literals

The simplest way to create an object is by using an object literal, which is a comma-separated list of key-value pairs enclosed in curly braces {}.

```javascript
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  isEmployed: true
};
```

2. Constructor Functions

You can also create objects using constructor functions. This method allows you to create multiple instances of similar objects.

```javascript
function Person(firstName, lastName, age) {
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
}
const person1 = new Person("John", "Doe", 30);
const person2 = new Person("Jane", "Doe", 25);
```

3. ES6 Classes

ES6 introduced classes, which provide a clearer and more concise syntax to create constructor functions and handle inheritance.

```javascript
class Person {
  constructor(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
  }
  greet() {
    return `Hello, my name is ${this.firstName} ${this.lastName}.`;
  }
}
const person1 = new Person("John", "Doe", 30);
const person2 = new Person("Jane", "Doe", 25);
```

Accessing and Modifying Properties

Dot Notation

Access and modify properties using the dot notation.

```javascript
console.log(person.firstName); // Output: John
person.age = 31;
```

Bracket Notation

Access and modify properties using bracket notation, useful when the property name is dynamic or not a valid identifier.

```javascript
console.log(person["lastName"]); // Output: Doe
person["isEmployed"] = false;
```

Methods

Objects can have methods, which are functions stored as properties.

```javascript
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 30,
  greet() {
    return `Hello, my name is ${this.firstName} ${this.lastName}.`;
  }
};
console.log(person.greet()); // Output: Hello, my name is John Doe.
```

Inheritance

Prototypes

Each object has a prototype from which it inherits properties and methods.

```javascript
```

```
function Person(firstName, lastName) {
  this.firstName = firstName;
  this.lastName = lastName;
}
Person.prototype.greet = function() {
  return `Hello, my name is ${this.firstName} ${this.lastName}.`;
};
const person1 = new Person("John", "Doe");
console.log(person1.greet()); // Output: Hello, my name is John Doe.
```

ES6 Classes and Inheritance

ES6 classes also support inheritance using the `extends` keyword.

```javascript
class Person {
  constructor(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }
  greet() {
    return `Hello, my name is ${this.firstName} ${this.lastName}.`;
  }
}
class Employee extends Person {
  constructor(firstName, lastName, jobTitle) {
    super(firstName, lastName);
    this.jobTitle = jobTitle;
  }
  describe() {
    return `${this.firstName} ${this.lastName} is a ${this.jobTitle}.`;
  }
}
const employee1 = new Employee("Jane", "Doe", "Software Engineer");
console.log(employee1.greet()); // Output: Hello, my name is Jane Doe.
console.log(employee1.describe()); // Output: Jane Doe is a Software Engineer.
```

Built-in Methods

Object.keys()

Returns an array of a given object's property names.

```javascript
console.log(Object.keys(person)); // Output: ["firstName", "lastName", "age", "greet"]
```

Object.values()

Returns an array of a given object's property values.

```javascript
console.log(Object.values(person)); // Output: ["John", "Doe", 30, function greet() { ... }]
```

Object.entries()

Returns an array of a given object's key-value pairs.

```javascript
console.log(Object.entries(person)); // Output: [["firstName", "John"], ["lastName", "Doe"],
["age", 30], ["greet", function greet() { ... }]]
```