Assignment on JavaScript: Entire Object and Methods

Submitted By:
Name: Akuthota Rakesh Kumar - 4742

Introduction

JavaScript is a powerful and dynamic programming language primarily used for enhancing interactivity in web applications. One of the most fundamental concepts in JavaScript is the use of **objects** and their associated **methods**. Objects allow developers to group related data and functionalities, making code more organized and efficient. In this assignment, we will explore what JavaScript objects are, how they are created and manipulated, and how methods function within them.

1. What is an Object in JavaScript?

In JavaScript, an **object** is a standalone entity with properties and a type. It is like real-life objects, such as a car, a person, or a phone — each has specific characteristics and behaviors.

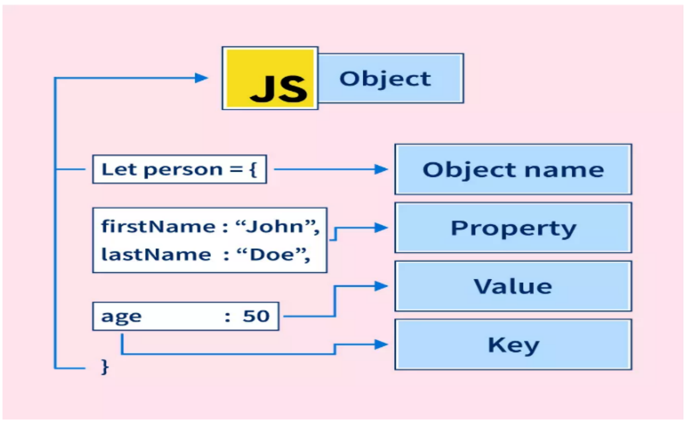
An object is a collection of **key-value pairs**, where the key is always a **string** (also called a property name) and the value can be anything—primitive types, functions, or even other objects.

Example:

```
let person = {
  name: "John",
  age: 30,
  isStudent: false   };
```

Here, a person is an object with three properties: name, age, and isStudent.

Example:



2. Creating Objects in JavaScript

There are multiple ways to create objects:

a. Using Object Literals:

```
let car = {
        brand: "Toyota",
        model: "Camry",
        year: 2020
      };
```

b. Using the new Object() Syntax:

```
let car = new Object();
car.brand = "Toyota";
car.model = "Camry";
car.year = 2020;
```

c. Using Constructor Functions

```
function Person(name, age) {
  this.name = name;
  this.age = age;
}

let student = new Person("Alice", 25);
```

d. Using ES6 Classes

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }
}
```

```
let employee = new Person("David", 40);
```

3. Accessing Object Properties:

a. Dot Notation

```
console.log(person.name); // John
```

b. Bracket Notation

```
console.log(person["age"]); // 30
```

4. Adding and Deleting Properties:

a. Adding New Properties

```
person.city = "New York";
```

b. Deleting Properties

```
delete person.isStudent;
```

5. Object Methods

A **method** is a function stored as a property in an object. Methods are used to define behaviors for objects.

```
let person = {
  name: "Emily",
  age: 22,
  greet: function() {
    console.log("Hello, my name is " + this.name);
  }
};

person.greet(); // Output: Hello, my name is Emily
```

Here, greet is a method of the person object.

6. The this Keyword

In object methods, this refers to the object that is calling the method.

Example:

```
let dog = {
  name: "Buddy",
  speak: function() {
    console.log(this.name + " says Woof!");
  }
};

dog.speak(); // Buddy says Woof!
```

7. Built-in Object Methods

JavaScript provides many built-in object methods:

- **Object.keys(obj)** – returns an array of property names
- **Object.values(obj)** – returns an array of property values
- **Object.entries(obj)** – returns an array of key-value pairs
- **Object.assign(target, source)** – copies properties from source to target
- **Object.freeze(obj)** – makes an object immutable

Example:

```
let user = { name: "Mark", age: 30 };
console.log(Object.keys(user)); // ["name", "age"]
```

8. Nested Objects

Objects can contain other objects.

Example:

```
let student = {
  name: "Sara",
  address: {
    city: "Los Angeles",
    zip: 90001
  }
};

console.log(student.address.city); // Los Angeles
```

9. Looping Through Object Properties

You can use for...in loop to iterate through an object’s properties.

Example:

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

10. Static Methods of Object in JavaScript

Static methods are methods that are called on the **Object constructor itself**, not on instances of objects. These methods are powerful tools for managing and manipulating object properties, prototypes, and metadata.

List of Important Static Methods:

Method	**Description**
Object.assign(target, ...sources)	Copies enumerable properties from one or more source objects to a target object.
Object.create(proto, [propertiesObject])	Creates a new object with the specified prototype.
Object.defineProperty(obj, prop, descriptor)	Adds a property with specific attributes to an object.
Object.defineProperties(obj, props)	Adds multiple properties at once.

Method	**Description**
Object.entries(obj)	Returns an array of [key, value] pairs from the object.
Object.fromEntries(iterable)	Converts key-value pairs into an object.
Object.freeze(obj)	Makes an object immutable (no changes allowed).
Object.getOwnPropertyDescriptor(obj, prop)	Returns the descriptor of a specific property.
Object.getOwnPropertyDescriptors(obj)	Returns all property descriptors.
Object.getOwnPropertyNames(obj)	Returns an array of all property names (including non-enumerable ones).
Object.getOwnPropertySymbols(obj)	Returns an array of symbol keys used in the object.
Object.getPrototypeOf(obj)	Returns the prototype of the object.
Object.hasOwn(obj, prop)	Returns true if the object has the property as its own (modern alternative to hasOwnProperty).
Object.is(value1, value2)	Determines if two values are the same, handling special cases like NaN.
Object.isExtensible(obj)	Checks if new properties can be added to the object.
Object.isFrozen(obj)	Checks if the object is frozen.
Object.isSealed(obj)	Checks if the object is sealed (properties can't be added/removed).
Object.keys(obj)	Returns an array of enumerable property names.
Object.preventExtensions(obj)	Prevents any new properties from being added to the object.

Method	Description
Object.seal(obj)	Prevents adding/removing properties but allows modifying existing ones.
Object.setPrototypeOf(obj, prototype)	Sets the prototype of the object.
Object.values(obj)	Returns an array of enumerable property values.

Examples of Static Methods of Object

Object.assign(target, ...sources)

```
const target = { a: 1 };
const source = { b: 2 };
const result = Object.assign(target, source);
console.log(result); // { a: 1, b: 2 }
```

Object.create(proto, [propertiesObject])

```
const animal = { type: "mammal" };
const dog = Object.create(animal);
dog.name = "Buddy";
console.log(dog.type); // "mammal"
```

Object.defineProperty(obj, prop, descriptor)

```
const person = {};
Object.defineProperty(person, "name", {
  value: "Alice",
  writable: false
});
console.log(person.name); // "Alice"
```

Object.defineProperties(obj, props)

```
const user = {};
Object.defineProperties(user, {
  name: { value: "John", writable: true },
  age: { value: 25, writable: true }
});
console.log(user.name); // "John"
```

Object.entries(obj)

```
const car = { brand: "Toyota", year: 2020 };
console.log(Object.entries(car)); // [ ['brand', 'Toyota'], ['year', 2020] ]
```

Object.fromEntries(iterable)

```
const entries = [['name', 'Bob'], ['age', 35]];
const obj = Object.fromEntries(entries);
console.log(obj); // { name: 'Bob', age: 35 }
```

Object.freeze(obj)

```
const config = { debug: true };
Object.freeze(config);
config.debug = false;
console.log(config.debug); // true
```

Object.getOwnPropertyDescriptor(obj, prop)

```
const item = { price: 100 };
console.log(Object.getOwnPropertyDescriptor(item, "price"));
```

Object.getOwnPropertyDescriptors(obj)

```
const tool = { name: "Hammer", weight: 2 };
console.log(Object.getOwnPropertyDescriptors(tool));
```

Object.getOwnPropertyNames(obj)

```
const gadget = { name: "Phone", model: "X1" };
console.log(Object.getOwnPropertyNames(gadget)); // ['name', 'model']
```

Object.getOwnPropertySymbols(obj)

```
const sym = Symbol("id");
const obj = { [sym]: 123 };
console.log(Object.getOwnPropertySymbols(obj)); // [Symbol(id)]
```

Object.getPrototypeOf(obj)

```
const parent = {};
const child = Object.create(parent);
console.log(Object.getPrototypeOf(child) === parent); // true
```

Object.hasOwn(obj, prop)

```
const student = { name: "Lara" };
console.log(Object.hasOwn(student, "name")); // true
```

Object.is(value1, value2)

```
console.log(Object.is(NaN, NaN)); // true
console.log(Object.is(+0, -0)); // false
```

Object.isExtensible(obj)

```
const product = { name: "Laptop" };
```

```
console.log(Object.isExtensible(product)); // true
```

Object.isFrozen(obj)

```
const frozen = Object.freeze({ a: 1 });
console.log(Object.isFrozen(frozen)); // true
```

Object.isSealed(obj)

```
const sealed = Object.seal({ b: 2 });
console.log(Object.isSealed(sealed)); // true
```

Object.keys(obj)

```
const fruit = { name: "Apple", color: "Red" };
console.log(Object.keys(fruit)); // ['name', 'color']
```

Object.preventExtensions(obj)

```
const dog = { breed: "Labrador" };
Object.preventExtensions(dog);
dog.age = 3;
console.log(dog.age); // undefined
```

Object.seal(obj)

```
const settings = { mode: "dark" };
Object.seal(settings);
settings.newMode = "light";
console.log(settings.newMode); // undefined
```

Object.setPrototypeOf(obj, prototype)

```
const animal = { eats: true };

const cat = { meows: true };

Object.setPrototypeOf(cat, animal);

console.log(cat.eats); // true
```

Object.values(obj)

```
const laptop = { brand: "HP", price: 800 };

console.log(Object.values(laptop)); // ['HP', 800]
```

11. Prototype Methods of Object.prototype

Every object in JavaScript inherits from Object.prototype unless explicitly created with null as prototype. These prototype methods are available to all objects.

Common Prototype Methods:

Method	**Description**
Object.prototype.constructor	**Refers to the function that created the instance's prototype.**
Object.prototype.hasOwnProperty(prop)	**Checks if the object has the specified property as its own.**
Object.prototype.isPrototypeOf(obj)	**Checks if this object exists in another object's prototype chain.**
Object.prototype.propertyIsEnumerable(prop)	**Checks if a property is enumerable.**
Object.prototype.toLocaleString()	**Converts the object to a string localized for the environment.**
Object.prototype.toString()	**Returns a string representation of the object.**

Method	Description
Object.prototype.valueOf()	**Returns the primitive value of the object.**
__proto__ (legacy)	**Getter/setter for the object's prototype (use with caution).**

Examples of Prototype Methods (Object.prototype):

Object.prototype.constructor

```
function Car(model) {
  this.model = model;
}
const myCar = new Car("Tesla");
console.log(myCar.constructor === Car); // true
```

Object.prototype.hasOwnProperty(prop)

```
const book = { title: "JS Guide" };
console.log(book.hasOwnProperty("title")); // true
```

Object.prototype.isPrototypeOf(obj)

```
const animal = { eats: true };
const dog = Object.create(animal);
console.log(animal.isPrototypeOf(dog)); // true
```

Object.prototype.propertyIsEnumerable(prop)

```
const person = { name: "Tom" };
console.log(person.propertyIsEnumerable("name")); // true
```

Object.prototype.toLocaleString()

```
const amount = 123456.78;

console.log(amount.toLocaleString("en-US")); // "123,456.78"
```

Object.prototype.toString()

```
const arr = [1, 2, 3];

console.log(arr.toString()); // "1,2,3"

console.log(Object.prototype.toString.call(arr)); // "[object Array]"
```

Object.prototype.valueOf()

```
const num = 100;

console.log(num.valueOf()); // 100
```

__proto__ (Legacy, not recommended for modern use)

```
const animal = { sound: "roar" };

const lion = {};

lion.__proto__ = animal;

console.log(lion.sound); // "roar"
```

12. Conclusion

The Object in JavaScript is one of the most fundamental and versatile constructs in the language. It serves as the base for nearly all other objects and plays a crucial role in how data is structured and manipulated. Understanding how objects work — including their properties, methods, and behavior in the prototype chain — is essential for writing clean, efficient, and maintainable JavaScript code.

We explored the Object constructor, the powerful static methods (like Object.assign(), Object.freeze(), and Object.create()), and the prototype methods (like hasOwnProperty() and toString()) that are inherited by all object instances. Each method offers specific utilities — from copying and protecting objects to querying and managing property descriptors.

By mastering these tools, developers can confidently work with complex data structures, enhance object behavior, and follow best practices in modern JavaScript programming.

In short, understanding the entire Object system is not just about syntax — it's about unlocking the true power of JavaScript as an object-oriented and prototype-based language.