Programming JS

Margit Tennosaar

Objects

JavaScript objects are fundamental constructs that allow you to store collections of data and more complex entities in a structured way. They group data and functions, offering a flexible mechanism to model real-world entities.

Objects store related data and allow us to structure information meaningfully.

They group properties and methods, making code organized and reusable.

Example:

- A person has a **name**, **age**, and **job**.
- A car has a **brand**, **model**, and **year**.

Objects make it easier to manage complex data.

Creating objects

- Properties are key-value pairs inside {}.
- Keys are always strings (even if quotes are omitted).
- Values can be any data type (string, number, boolean, array, function, etc.).
- The easiest way to create an object is using object literals.

```
const animal = {
  name: 'Fox',
  species: 'Canine',
  age: 4
};
```

Accessing Properties

- Dot notation → The most common and easiest to read.
- Bracket notation → Needed when property names have special characters or are stored in variables.

```
console.log(animal.name); // Outputs: Fox
animal.age = 5; // Updates the age

const property = 'species';
console.log(animal[property]); // Outputs: Canine

animal['age'] = 6;
console.log(animal['age']); // Outputs: 6
```

Modifying and Adding Properties

Objects are dynamic, meaning you can add or remove properties anytime.

```
animal.habitat = 'Forest';
console.log(animal.habitat); // Outputs: Forest

delete animal.age;
console.log(animal.age); // Outputs: undefined
```

Object methods

```
const car = {
  brand: "Toyota",
  start() {
    console.log("The car is starting...");
  }
};

car.start(); // Outputs: The car is starting...
```

- Methods are functions inside objects.
- They allow objects to perform actions.

this Keyword

- **this** refers to the current object.

- It is useful when methods need to access object properties.

```
const animal = {
  name: 'Fox',
  speak: function () {
    console.log(`I am a ${this.name}!`);
  }
};

animal.speak(); // Outputs: I am a Fox!
```

```
const animal = {
  name: 'Fox',
  speak() {
    console.log(`I am a ${this.name}!`);
  }
};

animal.speak(); // Outputs: I am a Fox!
```

- Without **this**, we couldn't dynamically access name.

Object methods vs regular functions

```
const animal = {
  name: "Fox",
  sound() {
    console.log(`${this.name} says: Ring-ding-ding!`);
  }
};

animal.sound(); // "Fox says: Ring-ding-ding!"
```

- **Inside objects,** use this to access properties.
- **Outside objects,** functions work normally.
- If this is missing, it won't reference the correct object.

Looping through an object

- Use for...in to iterate through all properties in an object.

```
const car = { brand: "Tesla", model: "Model 3", year: 2023 };

for (let key in car) {
  console.log(`${key}: ${car[key]}`);
}

// output
brand: Tesla
model: Model 3
year: 2023
```

Objects inside arrays

Objects are often stored inside arrays, especially in real-world apps.

```
const zoo = [
  { name: 'Fox', species: 'Canine' },
  { name: 'Eagle', species: 'Bird' },
  { name: 'Bear', species: 'Mammal' }
];

console.log(zoo[1].name); // Outputs: Eagle
```

Constructor functions

- A constructor function lets you create multiple objects with the same structure.

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

const animal1 = new Animal('Wolf', 'Canine', 5);
```

Using ES6 class syntax

JavaScript introduced classes in ES6 as a more structured way to create objects.

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

const animal2 = new Animal('Deer', 'Herbivore', 2);
```

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

  // Method to get the person's full name
  getFullName() {
    return `${this.firstName} ${this.lastName}`;
  }

  // Method to check if the person is an adult
  isAdult() {
    return this.age >= 18;
  }
}

const person2 = new Person('Bob', 'Smith', 28);

console.log(person2.getFullName()); // Outputs: Bob Smith
console.log(person2.isAdult()); // Outputs: true
```