Programming with Objects

Let's dig out a little more interesting concepts about Objects in JavaScript.

WE'LL COVER THE FOLLOWING

Many books and courses teach object-oriented programming through examples involving animals, cars or bank accounts. Let’s try something cooler and create a mini-role playing game (RPG) using objects. In a role-playing game, each character is defined by many attributes like strength, stamina or intelligence. Here’s the character screen of a very popular online RPG.

In our simpler example, a character will have three attributes:

- her name,
- her health (number of life points),
- her strength.

A Naive Example

Let me introduce you to Aurora, our first RPG character.

```
const aurora = {
  name: "Aurora",
  health: 150,
  strength: 25
};
```

The `aurora` object has three properties: `name`, `health` and `strength`.

> As you can see, you can assign numbers, strings, and even other objects to properties!

Aurora is about to start a series of great adventures, some of which will update her attributes. Check out the following example.

```
const aurora = {
  name: "Aurora",
  health: 150,
  strength: 25
};

console.log(`${aurora.name} has ${aurora.health} health points and ${aurora.strength} as stre

// Aurora is harmed by an arrow
aurora.health -= 20;

// Aurora equips a strength necklace
aurora.strength += 10;

console.log(`${aurora.name} has ${aurora.health} health points and ${aurora.strength} as stre
```

Introducing Methods

In the above code, we had to write lengthy `console.log` statements each time to show our character state. There's a cleaner way to accomplish this.

Adding a Method to an Object

Observe the following example.

```
const aurora = {
  name: "Aurora",
  health: 150,
  strength: 25
};

// Return the character description
function describe(character) {
  return `${character.name} has ${character.health} health points and ${character.strength} a
}

console.log(describe(aurora));
```

The `describe()` function takes an object as a parameter. It accesses that object's properties to create a description string. Below is an alternative approach, using a `describe()` property inside the object.

```
const aurora = {
  name: "Aurora",
  health: 150,
  strength: 25,

  // Return the character description
  describe() {
    return `${this.name} has ${this.health} health points and ${this
      .strength} as strength`;
  }
};

console.log(aurora.describe());
```

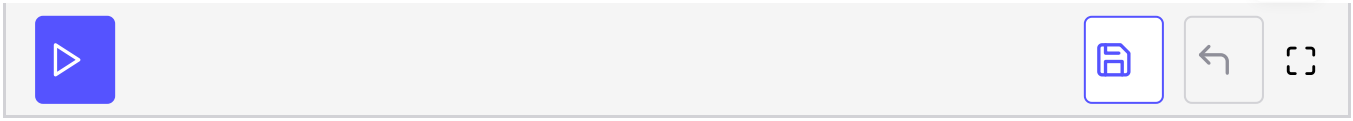
Now our object has a new property available to it: `describe()`. The value of this property is a function that returns a textual description of the object. The execution result is exactly the same as before.

An object property whose value is a function is called a *method*. Methods are used to define *actions* for an object. A method adds some *behavior* to an object.

Calling a Method on an Object

Let's look at the last line of our previous example.

```
console.log(aurora.describe());
```



To show the character description, we use the `aurora.describe()` expression instead of `describe(aurora)`. It makes a crucial difference:

- describe(aurora) calls the `describe()` function with the `aurora` object as an argument. The function is external to the object. This is an example of procedural programming
- `aurora.describe()` calls the `describe()` function on the `aurora` object. The function is one of the object's properties: it is a method. This is an example of object-oriented programming

To call a method named `myMethod()` on an object myObject, the syntax is `myObject.myMethod()`.

> Remember the parentheses, even if empty, when calling a method!

`this` Keyword

Now look closely at the body of the `describe()` method on our object.

```
const aurora = {
  name: "Aurora",
  health: 150,
  strength: 25,

  // Return the character description
  describe() {
    return `${this.name} has ${this.health} health points and ${this
      .strength} as strength`;
  }
};
```

You see a new keyword: `this`. This is automatically set by JavaScript inside a method and represents *the object on which the method was called*.

The `describe()` method doesn't take any parameters. It uses `this` to access the properties of the object on which it is called.