Object Methods, Lookup Tables, State Machines

Software Development Bootcamp

Topic

Object Methods

What Are Object Methods?

Object methods are functions attached to objects as properties

- We can use objects to represent real-life things
- Methods represent actions those things can do
    - Example: `dog.speak()`

You've Already Used Some Methods!

- `console.log("some message")` console is an object, .log is a method
- `string.toUpperCase()` strings have a prototype object, that gives them access all the string methods
- `Math.random()` Math is a JavaScript class that has a static method .random()

Example: Speaking Dog

Let’s set up a simple object, with a method that prints a little message.

- Create an object, and assign it to the variable dog
- Give dog 3 properties, a name, a color, and speak
- When called, speak will print "Bark!" to the console.

Syntax for Passing Arguments to Methods

Because methods are functions, we use them in the same way.

- `objectName.methodName(argument)`

Why Use Object Methods?

1. **Encapsulation**: Methods allow us to bundle data and functionality together, keeping related operations close to the data they work with.
2. **Code Organization**: They help structure our code logically, making it easier to understand and maintain.
3. **Reusability**: Once defined, methods can be reused across multiple instances of an object.
4. **Abstraction**: Methods can hide complex implementation details, providing a simpler interface for other parts of the program.
5. **Behavior Modeling**: They allow us to model real-world behaviors of objects in our code, making our programs more intuitive.
6. **Data Manipulation**: Methods provide controlled ways to access and modify object data, ensuring data integrity.

The `this` Keyword

The `this` keyword refers to the object that is executing the current function. Its value is determined by how a function is called.

- In an object method `this` refers to the object itself
- In arrow functions `this` keeps the same meaning it had in the surrounding code

`this` Example

- Here `this` refers to the rectangle object itself
- When we call `rectangle.area()` JavaScript knows that `this` inside the method should refer to `rectangle`
- Using `this` allows the `area` method to always use the current values of the rectangle it's a part of, even if we change the values later.

```
let rectangle = {
 height: 10,
 width: 8,
 //Creating a method called area
 area: function () {
   //using 'this' to access the height
and width values of the rectangle object
   return this.height * this.width;
 },
};
console.log(rectangle.height); // 10
console.log(rectangle.area()); // 80
```

this Practice

Edit the dog object to use `this`

- Change “Bark!” to say the dog’s name and color
- Use the `this` keyword instead of the object’s variable name

`this` and Binding

- `This` is bound to the context in which it's called
- Dot notation means the context is that object

```
let dog = {
    name: 'Tulip',
    speak: function () {
        console.log(`Woof! my name is
${this.name}`)
    }
}
// Calling the speak method
// dog is the context, so this.name is the same
as dog.name
dog.speak()
// output: `Woof! my name is Tulip
```

`this` and Binding Cont.

- Arrow functions handle this differently than normal functions
- Arrow functions use whatever `this` is outside of them.

```
let dog = {
    name: 'Tulip',
    speak: () => {
        console.log(`Woof! my name is ${this.name}`)
    }
}
// Calling the speak method
// dog is the context, so this.name is the same as dog.name
dog.speak()
// output: `Woof! my name is undefined`
```

Adding to the Object After Creating It

- We can add methods and properties to objects after we've already made them.

```
let rectangle = {
 height: 10,
 width: 8,
};
//Calling the area method before it has been created
rectangle.area(); // TypeError: rectangle.area is not a function
//adding the area method to the rectangle object
rectangle.area = function () {
 return this.height * this.width;
};
rectangle.area(); // 80
```

Topic

Lookup Tables And State Machines

What Are Lookup Tables?

A lookup table is like a dictionary for your code. It's a way to store and quickly find information.

- You have a set of keys (like words in a dictionary)
- Each key has a matching value (like definitions in a dictionary)
- When you need information, you use the key to "lookup" the value.

Example: Translator

- They keys are English fruit names
- The values are Spanish translations
- We can quickly find a translation by using the English name.

```
let fruitTranslator = {
 apple: "manzana",
 banana: "plátano",
 cherry: "cereza",
};

console.log(fruitTranslator.apple);
// Output: "manzana"
console.log(fruitTranslator["banana"]);
// Output: "plátano"
```

Example: Poem

- Inside the function is the lookup table **`poemTitlesByAuthor`** that matches authors with their famous poems
- The function looks up the author's name in the **`poemTitlesByAuthor`** object and returns the title of the poem.

```
// function that matches poet to their poem
function getPoemTitle(authorUserSelected) {
// Lookup table to famous poets and their poem
 let poemTitlesByAuthor = {
   "Robert Frost": "Stopping by Woods on a
Snowy Evening",
   "Shel Silverstein": "Falling Up",
   "Sylvia Plath": "The Bell Jar",
 };
 return
poemTitlesByAuthor[authorUserSelected];
}
console.log(getPoemTitle("Sylvia Plath"));
```

State Machines

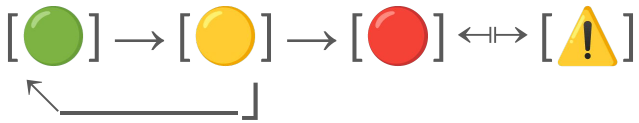
A state machine is like a flowchart for your program. It helps manage complex behavior by breaking it down into simple steps.

- Define a set of “states” your program can be in
- Define rules for how to move between these states
- At any given time, your program is in one specific state
- Based on certain conditions or actions, it can transition to another state.

State Machine Example: Traffic Light

1. How many states? What are their names?
2. Can it be in more than one state at a time?
3. What are the rules for transitioning between states?

State Machine Example: Traffic Light Cont.



- Green can transition to yellow
- Yellow can transition to red
- Red can transition to either a warning to yield or back to green
- The warning to yield can transition back to Red.

Setting up the Allowable Transitions

- The first step when creating a state machine is to set up an object that will hold your allowable transitions.

```
// Traffic light states
let states = {
 green: ["yellow"],
 yellow: ["red"],
 red: ["green", "yield"],
 yield: ["red"],
};
```

Moving Between States

- The `enterState` function knows what the current state is, and accepts the next state as an argument

```
let currentState = "green";
function enterState(newState) {
 // we access the states object using the currentState variable as a key
 let validTransitions = states[currentState];
 // check if the validTransitions array includes the newState parameter
 if (validTransitions.includes(newState)) {
   currentState = newState;
   console.log(`the light is now ${currentState}`);
 } else {
   console.log(
     "Invalid state transition attempted - from " +
       currentState +
       " to " +
       newState
   );
 }
}
```

Using our State Machine

- Now we can run through some state transitions using our state machine.

```
enterState("yellow")
enterState("red")
enterState("yellow")
```

Why Use a State Machine?

State machines are useful for:

- Managing complex workflows
- Controlling the flow of your program
- Making your code more predictable and easier to debug

Transitioning Between Objects

- Often you'll need to transition between complex data structures like objects, not just strings.
- Combining state machines and lookup tables allows for more complex interactions.
- You can track the current state as a string using a state machine, but map it to an object using a lookup table for more detailed operations.

Exercise

Menu Order