JavaScript

WEB
3 - JavaScript

Document summary

The text covers educational objectives including creating objects with the class syntax, DOM manipulation, and drawing on the HTML Canvas with JavaScript. It emphasizes understanding methods, prototypes, and static methods in JavaScript classes, along with private properties and getters/setters. Additionally, it introduces modules for reusable code, DOM manipulation techniques, element selection, and event handling using both traditional and modern approaches.

Table of content

1. Educational objectives

- Create object using the class syntax
- Use the extends keyword to create a class as the child of another class
- Use the super keyword to access and call functions on an object's parent
- Use the static keyword to create static methods and properties
- Can manipulate the DOM using JavaScript
- Can draw in the HTML Canvas using JavaScript
- Can manipulate arrays and objects in JavaScript using the array methods

2. Objects

An object is a mutable unordered collection of properties. A property is a tuple of a key and a value. A property key is either a string or a symbol. A property value can be any ECMAScript language value.

```
let car = {
    make: 'Ford',
    model: 'Mustang',
    year: 1969
}
```

Thoses properties can be accessed using the dot notation or the bracket notation.

2.1. Methods

A method is a function associated with an object. A method is a property of an object that is a function. Methods are defined the same way as regular functions, except that they are assigned as the property of an object.

```
var apple = {
    color: 'red',
    toString: function() {
        return `This fruit is ${this.color}!`;
    }
}
console.log(apple.toString()); // This fruit is red!
```

2.1.1. Prototype

Every JavaScript object has a prototype. The prototype is also an object. All JavaScript objects inherit their properties and methods from their prototype. Objects created using an object literal, or with new Object(), inherit from a prototype called Object.prototype. Objects created with new Date() inherit the Date.prototype. The Object.prototype is on the top of the prototype chain. All JavaScript objects (Date, Array, RegExp, Function, ...) inherit from the Object.prototype.

2.1.1.1. Constructor invocation

Objects inherit the properties of their prototype, hence JavaScript is class-free.

If a function is invoked with the new operator:

- a new object that inherits from the function's prototype is created
- the function is called and this is bound to the created object
- if the function doesn't return something else, this is returned

```
function Fruit(color) {
    this.color = color;
}

Fruit.prototype.toString = function() {
    return `This fruit is ${this.color}!`;
}

var apple = new Fruit("red");
console.log(apple.toString()); // This fruit is red!
```

3. Context

The context of a function is the object to which the function belongs. The context is determined by how a function is invoked. The context of a function can be set using the call, apply, or bind methods.

```
function doTwice(f) {
    f()
    f()
}
let human = {
    age: 32,
    getOlder() {
        this.age++
    }
}
doTwice(human.getOlder)
console.log(human.age)
```

In this case the context of the function getOlder is not shared when calling `doTwice(human.getOlder)`. The context is lost and the age property is not incremented. The program throw an error. To fix this, you can use the bind method to set the context of the function.

```
function doTwice(f) {
    f.call(this); // bind this to the current context
    f.call(this); // bind this to the current context
}
let human = {
    age: 32,
    getOlder() {
        this.age++;
    }
}
doTwice.call(human, human.getOlder); // bind this to human
console.log(human.age); // Output will be 34
```

4. Array object

The Array object is a global object that is used in the construction of arrays, which are high-level, list-like objects.

```
let fruits = ['Apple', 'Banana', 'Pear'];
let fruits = new Array('Apple', 'Banana', 'Pear');
```

Note that Array is a function and the new operator makes it be used as a constructor.

4.1. Indices

Because arrays are objects, getting `myArray[42]` is equivalent to accessing a property named “42” on the object myArray.

That’s why the for … in loop, which iterates over an object’s properties, can be used to iterate over the indices of an array.

```
let fruits = ['Apple', 'Banana', 'Pear'];
for (let index in fruits) {
    console.log(index); // 0, 1, 2
}
```

5. Object-oriented syntax

```
class Fruit {
    constructor(color) {
        this.color = color;
    }
    toString() {
        return `This fruit is ${this.color}!`;
    }
}
class Apple extends Fruit {
    constructor(color, name) {
        super(color);
        this.name = name;
    }
    toString() {
        return super.toString();
    }
}
let apple = new Apple("red", "golden");
console.log(apple.toString()); // This fruit is red!
```

- The extends keyword is used in class declarations or class expressions to create a class as the child of another class.
- The constructor method is a special method for creating and initializing an object described by a class.
- The super keyword is used to access and call functions on an object's parent.

5.1. Private properties, getter and setter

JavaScript offer the opportunity to define private properties or methods using the # prefix.

```
class Fruit {
    #color;
    get color() {
        return this.#color;
    }
    set color(color) {
        this.#color = color;
    }
}
let apple = new Fruit();
apple.color = 'red'; // calls the setter
console.log(apple.color); // red (calls getter)
console.log(apple.#color); // SyntaxError: Private field '#color' must be declared in an
enclosing class
```

5.1.1. Static methods and properties

The static keyword defines a static method or property for a class. Static members (properties and methods) are called without instantiating their class and cannot be called through a class instance.

```
class Point {
    constructor(x, y) {
        this.x = x;
        this.y = y;
    }
    static distance(a, b) {
        const dx = a.x - b.x;
        const dy = a.y - b.y;
        return Math.hypot(dx, dy);
    }
}
let p1 = new Point(5, 5);
let p2 = new Point(10, 10);
console.log(Point.distance(p1, p2)); // 7.0710678118654755
```

6. Modules

Modules are reusable pieces of code that can be exported from one program and imported for use in another program.

```
// Export upon declaring a value
export function sum(a, b) { ... }
export const ANSWER = 42

// Export a declared value
export { val1, val2, ... }
```

The imported script must be loaded as a module with the type="module" attribute.

```
<script type="module" src="module.js"></script>
```

6.1. Import

The import statement must always be at the top of the js file, before any other code.

```
import { export1, export2, ... } from "module-name";  // import specific named values
import { export1 as alias1, ... } from "module-name"; // import value with alias (e.g. to solve
name conflicts)
import * as name from "module_name";  // import all into an object
import name from "module-name";       // import the default export
// equivalent to
import { default as name } from "module-name";
import "module-name"; // imports for side-effects; runs it but imports nothing.
```

7. DOM manipulation

- DOM stands for Document Object Model
- The DOM is a programming interface for HTML and XML
- The DOM represents the structure of a document in memory
- The DOM is an object-oriented representation of the web page
- The DOM lets other programming languages manipulate the document

7.1. Accessing elements

In JavaScript, the Document interface represents any web page loaded in the browser and serves as an entry point into the web page's content, which is the DOM tree.

```
document;
document.location; // Getter returns location object (info about current URL).
document.location = "https://heig-vd.ch/"; // Setter to update displayed URL.
document.designMode = "on"; // Lets user interactively modify page content.
document.referrer; // URL of page that linked to this page.
```

7.2. Selecting elements

```
console.log(document.getElementById("id"));
console.log(document.getElementsByClassName("slide"));
console.log(document.getElementsByTagName("h1"));
```

7.2.1. QuerySelector

The querySelector() method returns the first element that matches a specified CSS selector(s) in the document.

```
document.querySelector("p");
document.querySelector(".slide");
```

7.3. DOM events

DOM Events are sent to notify code of interesting things that have taken place. Each event is represented by an object that inherits from the Event object, and may have additional custom fields and/or functions used to get additional information about what happened.

```
document.onkeydown = function(event) {console.log(event);}
document.addEventListener('keydown', event => console.log(event))
```

You can call a method when action on an element using the `onclick` attribute.

```
<a href="http://www.heig-vd.ch" onclick="event.method();">
```

7.4. Event listeners

Event listeners are a more modern way to handle events. They can be added to any DOM element, not just HTML elements. They are more flexible than using the on-event-attributes.