COMP 2015

Introduction to JavaScript and jQuery

Contents

Functions

A function is a set of statements (i.e. instructions) that can be written once and used repeatedly. Any piece of code that needs to be run multiple times is a good candidate for a function.

JavaScript has many built-in functions, and JavaScript Objects have built-in functions which we refer to as 'methods', e.g. `window.alert()`. We can also define our own functions.

How To Create A Function

Step 1: Start with the `function` keyword, followed by a space and a name that indicates its purpose

```
function welcomeVisitorToTheSite
```

Step 2: Immediately following the name, add parentheses and braces

```
function welcomeVisitorToTheSite() { }
```

Step 3: Add all of the code you want to run when the function is called inside the braces. This can span multiple lines

```
function welcomeVisitorToTheSite() {
    // example welcome code
    var username = prompt('Hello, what is your name?');
    alert('Welcome to the site, ' + username + '!');
}
```

Now we have defined our function we can call (i.e. use) it! The function can be called zero, once, or multiple times.

To call the function we simply write a statement with the name of the function and the parentheses.

example:

```
welcomeVisitorToTheSite();
```

If a function is not called (as shown above), the code inside will never run!

The question is, **when do we want to call our function?**. Because we are trying to welcome visitors to the site, a good time to call the function would be after the page loads. We can accomplish this using the window.onload event we have seen previously.

example:

```
<script>
    function welcomeVisitorToTheSite() {
        var username = prompt('Hello, what is your name?');
        alert('Welcome to the site, ' + username + '!');
    }

    window.onload = welcomeVisitorToTheSite();
</script>
```

Functions With Parameters

If a function requires information in order to do its job, it may receive data when it is called, using parameters. Parameters are variables that functions accept when they are called.

If a function requires a variable, any script that calls the function can also provide the appropriate parameter(s).

example:

```
function alertName(first, last) { // parameters are first and last
    /*
        Note: the variables called 'first' and 'last'
        are only accessible inside the function.

        Trying to access first and last outside the function
        will result in an error
    */
    alert(first + ' ' + last);
}

var firstName = prompt('Please enter your first name');
var lastName = prompt('Please enter your last name');

alertName(firstName, lastName); // calling the function using two 'arguments'
```

Assigning Functions To Variables

We can also assign functions to variables, so that when a variable is used our function code is run. We will see an example of this shortly with *events*, but the syntax is as follows:

```
function add(arg1, arg2) {
    console.log(arg1 + arg2);
}

var printSum = add; // *important* no parentheses! otherwise the code inside
                    // add() is run immediately

printSum(1, 2);
printSum(3, 4);
```

Anonymous Functions

In the example above, we have seen how to assign a named function to a variable. We can also assign a function inline (i.e. in one statement) using unnamed *anonymous* functions.

example:

```
var printSum = function(arg1, arg2) {
    alert(arg1 + arg2);
}
```

Scope

Variables in JavaScript (and most other programming languages) have a 'scope', which defines when and where they can be used.

Variables created **outside** of functions, loops or other code 'blocks' automatically have what is referred to as *global* or *window* scope. This means they can be accessed **anywhere**. For example, we can define a variable inside an external file and use it in the body of our html page!

example:

```
// In external file called variables.js
var x = 5;
```

```
<!-- In HTML file -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="variables.js"></script>
</head>
<body>
    <script>
        alert(x); // accessible because it has GLOBAL scope
    </script>
</body>
</html>
```

Variables created **inside** of functions, loops or other code blocks have 'local' scope. This means that they are only accessible inside the block they are the defined.

example:

```
function scopeTest() {
    var x = 5;
    var y = 10;

    /*
    x and y have 'local' scope, meaning they are only accessible
    inside the function
    */

    alert(x + y);
}

alert(x); // error! x is undefined outside the function
```

We can also use the `let` keyword to make the scope even more strict, which we have seen in for loops.

example:

```
function multiplesOfFive() {
    var x = 5;

    for (let i = 0; i < 10; i++) {
        alert(x * i); // inside the for loop, both x and i are accessible
    }

    alert(i); // error! i is defined using 'let', so it can only be accessed
              // inside the for loop
}

alert(x); // error! x is undefined outside the function
```

Events

JavaScript provides many *events* that we can use to make our apps/pages interactive. We have already seen the 'onload' event, which fires when the page is finished loading, but there are many others that allow us to add dynamic behaviours based on what a user is doing.

Here is a short list of **event handlers** that we will use during the course, but there are many others (see https://developer.mozilla.org/en-US/docs/Web/Events for an exhaustive list)

- click
- dblclick
- mouseover
- mouseout

There are a few different ways that we can use event handlers in our code. The first is by setting the `onclick` property of an object to an anonymous or named function.

examples:

```
// click, using an anonymous function without creating a variable
document.images[0].onclick = function() {
    this.setAttribute('style', 'border: 1px solid red');
}

// dblclick, using a named function and variable
var firstParagraph = document.getElementsByTagName('p')[0];

function changeColor() {
    this.setAttribute('style', 'text-decoration: underline');
}

firstParagraph.ondblclick = changeColor;

// mouseover, using an anonymous function and variable
var firstImage = document.images[0];

firstImage.onmouseover = function() {
    this.setAttribute('style', 'opacity: 0');
}
```

We can also use the addEventListener() method using anonymous or named functions.

examples:

```
// example using anonymous function and variable
var firstImage = document.images[0];

firstImage.addEventListener('click', function() {
    this.setAttribute('style', 'border: 1px solid red');
});

// example using named function without creating a variable
document.images[0].addEventListener('click', function() {
    this.setAttribute('style', 'text-decoration: underline');
});
```

The final (**not recommended**) method is to specify which function should be called directly in your HTML.

example:

```
<p onclick="myFunction(this)">Click me!</p>

<script>
    function myFunction(element) {
        element.setAttribute('style', 'color: blue;');
    }
</script>
```

This method is not advisable, because ideally we want to separate behaviour (JavaScript) and content (HTML) in the same way that we try to separate style (CSS) from content.

‘This’ Keyword

Outside of a function, the ‘this’ keyword refers to the global object (usually the window).

example:

```
<script>
    alert(this);  // returns the window object
</script>
```

Inside of a function, ‘this’ refers to the object that called the function.

example:

```
<script>
    document.images[0].onclick = function() {
        alert(this);  // returns the HTML element that called the function,
                      // i.e. the first <img> on the page
    }
</script>
```

The ‘this’ keyword is useful because it gives us a way to change an object’s properties (or use it’s methods) without having to specify the object by name.

For example, imagine that we want to attach a click event to every image on the page. When an image is clicked, we want it to rotate 180 degrees.

Let’s begin with the basic code structure:

```
var images = document.getElementsByTagName('img');

for (let i = 0; i < images.length; i++) {
    images[i].onclick = rotateImage; // for each image clicked, call the
                                     // rotateImage() function
}

function rotateImage() {

}
```

The problem with the above is, how do we specify inside the rotateImage() function which image we are changing? We can't, for example, pass in a parameter to the rotateImage() function because it will be called immediately, rather than waiting for the user to click the image.

```
var images = document.getElementsByTagName('img');

for (let i = 0; i < images.length; i++) {
    images[i].onclick = rotateImage(i); // DOES NOT WORK! Because of the parentheses,
    // the function is called immediately instead of waiting for the image to be clicked
}

function rotateImage(index) {
    images[index].setAttribute(
        'style',
        '-ms-transform: rotate(180deg); \
        -webkit-transform: rotate(180deg); \
        transform: rotate(180deg);'
    );
}
```

How can we fix this? By using **this**!

```
var images = document.getElementsByTagName('img');

for (let i = 0; i < images.length; i++) {
    images[i].onclick = rotateImage;
}

function rotateImage(index) {
    this.setAttribute(
        'style',
        '-ms-transform: rotate(180deg); \
        -webkit-transform: rotate(180deg); \
        transform: rotate(180deg);'
    );
}
```

jQuery

jQuery is a language written in JavaScript that is understood by all browsers and makes our lives a lot easier.

We can use jQuery by first importing the library from the jQuery CDN, at https://code.jquery.com

Choose a version of jQuery (for our course we'll use minified 3.x), click the link and copy the script tag into the head of your HTML page.

example:

```
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script
        src="https://code.jquery.com/jquery-3.2.1.min.js"
        integrity="sha256-hwg4gsxgFZhOsEEamdOYGBf13FyQuiTwlAQgxVSNgt4="
        crossorigin="anonymous">
    </script>
</head>
```

Now we can change `window.onload = function() { }` to `$(document).ready(function(){ });` and use jQuery's simplified syntax to accomplish all the same tasks we have learned in JavaScript.

Here's an example of how to use a click event to change the text color of each paragraph on a page using JavaScript:

```
var paragraphs = document.getElementsByTagName('p');

for (let i = 0; i < paragraphs.length; i++) {
    paragraphs[i].onclick = changeToRed;
}

function changeToRed() {
    this.setAttribute('style', 'color: red;');
}
```

And the equivalent in jQuery:

```
$('p').click(function() {
    $(this).css('color', 'red');
});
```

Preventing Default Browser Behaviour

Occasionally events will happen in the browser that we want to prevent, e.g. stopping a form from being submitted if a field is empty.

We can accomplish this in two ways: using `return false` or `event.preventDefault()`.

examples:

```
document.images[0].onclick = function() {
    alert('This link cannot be clicked');
    event.preventDefault();

    /*
        Note that we can use the 'event' object reference here despite not
        having explicitly created it.

        This is because event is a global property of the window object,
        accessible anywhere. It is equivalent to calling
        window.event.preventDefault(), in the same way that we shorten alert()
        to window.alert()
    */
}

document.images[1].onclick = function() {
    return false;
}

document.images[0].onclick = function() {
    return confirm('Are you sure you want to click this link?');
}
```

The difference between the two methods is explained clearly in this article by Chris Coyier: https://css-tricks.com/return-false-and-prevent-default/ – for our course either method will work well.

Form Validation

Using the `onsubmit` event handler and `return false` or `event.preventDefault()`, we can now use JavaScript to handle client-side form validation for us.

example:

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Form Validation</title>
</head>
<body>
    <form id="myform" action="">
        <p>
            <label for="email">Email: </label>
            <input type="text" class="required" id="email" name="email">
        </p>

        <p>
            <label for="password">Password: </label>
            <input type="text" class="required" id="password" name="email">
        </p>

        <p><input type="submit"></p>
    </form>

    <script>
        window.onload = function() {
            document.forms[0].onsubmit = function() {
                var inputs = this.elements;
                var labels = document.getElementsByTagName('label');
                var error = false;

                for (let i = 0; i < inputs.length; i++) {
                    var theClass = inputs[i].className;

                    if (theClass.indexOf("required") >= 0 &&
                            inputs[i].value.length < 1) {
                        alert(inputs[i].id + ' must not be empty');
                        labels[i].setAttribute('style', 'color: red');
                        error = true;
                    }
                }

                if (error) {    // equivalent to if (error == true)
                    return false;
                }
            }
        }
    </script>
</body>
</html>
```

We can also use the jQuery validation plugin (https://jqueryvalidation.org) to accomplish the same task – with much less time and effort!

Import the validation by including a script tag in your head - **below the link to jQuery**! – that references one of the sources listed on https://cdnjs.com/libraries/jquery-validate in the `src` attribute

example usage, using the same form HTML as the previous example:

```
$(document).ready(function() {
    $('#myform').validate({
        rules: {
            email: { required: true, email: true },
            password: 'required'
        },
        messages: {
            email: 'valid email is required',
            password: 'password is required'
        },
        submitHandler: function(form) {
                form.submit();
        }
    });
});
```

Next Week

Lab 3 and Assignment 1 due before the beginning of Lesson 4.

Download it from our course section of COMP2015 on D2L http://learn.bcit.ca (COMP2015 > Content > Lesson 3)

Quiz 3 at the beginning of Lesson 4.

Quiz topics can also be found on D2L under COMP2015 > Content > Lesson 3