JavaScript Functions

COEN 161

What is a function?

- A JavaScript function is a block of code designed to perform a specific task
- Functions execute when “something” invokes it (or calls it)

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Defined by the *function* keyword
- The *name* follows the same rules as variables
- The parentheses includes a comma separated list of parameters

Function Parameters

```
function name(parameter1, parameter2, parameter3) {}
```

- **Parameters** are the names in between the parentheses
- **Arguments** are the actual values passed to the function when it’s called
- Parameter Rules:
    1. JS functions **do not** specify the data types for parameters
    2. JS functions **do not** perform type checking on passed arguments
    3. JS functions **do not** check the number of arguments received

Parameter Defaults

- When a function is called with **missing arguments** (less than the number of parameters declared) , the missing values are set to **undefined**
  - Sometimes that’s ok, but sometimes we want to set a default value for the missing arguments

```
function myFunction(x, y) {
    if (y === undefined) {
          y = 0;
    }
}
```

```
function myFunction(x, y) {
    y = y || 0; // not a boolean
}
```

* `expr1 && expr2` returns `expr1` if it evaluates to **false**, else it returns `expr2`

* `expr1 || expr2` returns `expr1` if it evaluates to **true**, else it returns `expr2`

Parameter Defaults

- When a function has too many arguments (more than declared), you can access them using the **arguments** array*

```
x = findMax(1, 123, 500, 115, 44, 88);
function findMax() {
    var i;
    var max = -Infinity;
    for (i = 0; i < arguments.length; i++) {
        if (arguments[i] > max) {
            max = arguments[i];
        }
    }
    return max;
}
```

Function Invocation

- The code in a JS function doesn’t run until “something” invokes it
- We often refer to this as “calling a function”
- To invoke a function we use the `()` operator

```
function myFunction(a, b) {

    return a * b;

}

myFunction(10, 2);           // Will return 20
```

Function Invocation

- If you don’t use the `()` operator it returns the function ***definition*** instead

```
function myFunction(a, b) {

    return a * b;

}

myFunction(10, 2);              /* Will return
                                    ƒ myFunction(a, b) {
                                        return a * b;
                                    }
                                */
```

Functions as Variables

- Functions can be used any place you use variables
- Before:

```
var x = toCelsius(77);

var text = "The temperature is " + x + " Celsius";
```

- After:

```
var text = "The temperature is " + toCelsius(77) + " Celsius";
```

Function Expressions

- JavaScript functions can also be defined in an expression
- That expression can then be stored in a variable to use later

```
var x = function (a, b) {return a * b};

var z = x(4, 3);
```

- This type of function is also referred to as an ***anonymous function*** (a function without a name)
- Since they don't have names, they need to be stored in variables to invoke later

First Class Citizens

- In programming languages, a first-class citizen is any type that can be passed, returned, or assigned.
- In JavaScript, we’ve seen how to pass, return, and assign
    - Numbers
    - Strings
    - Arrays
    - Objects
- But there is one more first-class citizen in JS...functions!

Higher Order Functions

- In programming languages, higher order functions, are functions that can accept and/or return a function

```
var add = function (a) {
    return function(b) {
        return a + b;
    };
};

add(1)(2); // returns 3
```

Function Scope

- Scope defines where variables are *visible* and *accessible*
- JavaScript has ***function scope***
- Variables declared in a function can be used anywhere in that function but they can't be used outside the function - these are called ***local variables***
- Any variables declared outside *any* function are considered ***global variables***

```
// code here can not use carName

function myFunction() {
    var carName = "Volvo";

    // code here can use carName

}
```

```
var carName = " Volvo";

// code here can use carName

function myFunction() {

    // code here can use carName

}
```

Automatically Global

- If you do not declare a variable, it is automatically global

```
myFunction();

// code here can use carName

function myFunction() {
    carName = "Volvo";
}
```

- In "Strict Mode" automatic global creation is not allowed

Global Variables in HTML

- When JavaScript is loaded in an HTML page, the global scope is the entire page, any other JavaScript code on the page can reference it
- The global scope is referenced by the `window` object (more on this later)

```
<script>

    var carName = "Volvo";

    // code here can use window.carName

</script>
```

Variable Hoisting

- In JavaScript, a variable can be declared *after* it is used

```
x = 5; // Assign 5 to x

console.log(x)      // prints out 5

var x; // Declare x
```

```
var x; // Declare x
x = 5; // Assign 5 to x

console.log(x) // prints out 5
```

- How…?

Variable Hoisting

- The word “hoisting” refers to the way JavaScript moves declarations to the top of the current scope
- This can be the current script or the current function

```
x = 5; // Assign 5 to x

console.log(x)      // prints out 5

var x; // Declare x
```

```
var x; // Declare x
x = 5; // Assign 5 to x

console.log(x) // prints out 5
```

Initializations and Hoisting

- JavaScript initializations are **NOT** hoisted

```
var x = 5; // Initialize x

var y = 7; // Initialize y

//Prints "5 7"
console.log(x + " " + y);
```

```
var x = 5; // Initialize x

//Prints "5 undefined"
console.log(x + " " + y);

var y = 7; // Initialize y
```

Long Story Short...

- ..declare all your variables at the top of your scope :)

Lifetime of JavaScript Variables

- A JS variables begins life after its declared;
- Local variables get deleted when the function returns
- In the browser, global variables are deleted when you close your browser (or tab) **but remain available to any other page that is loaded in the same window :(**

Resources

https://www.w3schools.com/js/js_functions.asp

https://www.w3schools.com/js/js_function_parameters.asp

http://ryanchristiani.com/functions-as-first-class-citizens-in-javascript/