Foundation Coding - Week 4: Function Patterns, Object Pattern, Scope, addClass/removeClass

Function Patterns

There are different forms of a function.

We will focus on function declarations and immediately invoked function expressions.

```
// Immediately invoked function expression
(function() {

  'use strict';

  // Create function declaration
  function logToConsole () {
    console.log('working');
  }

  // Call the function
  logToConsole();

}());
// iife ENDS
```

Function Declarations

- There are 3 main types of function pattern: declarations, expressions and methods.
- The function to the right is a function declaration. It is the simplest form of writing a function.

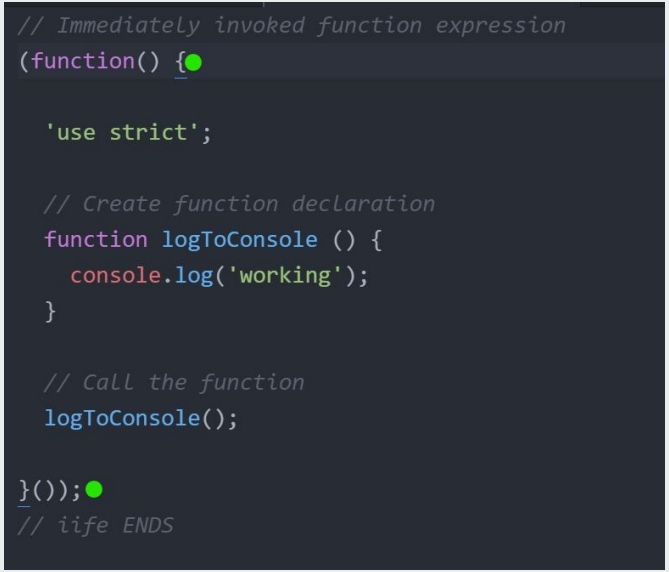
```
// Create function declaration
function logToConsole () {
  console.log('working');
}

// Call the function
logToConsole();
```

iife

- An iife is used to wrap code within an anonymous function wrapper. We do this because:

  - It protects our code from naming conflicts
  - It removes other issues that rise from coding in the “global scope”
  - The global scope is ‘window’. When you code outside of functions you are operating in the window/global scope



```
// Immediately invoked function expression
(function() {

  'use strict';

  // Create function declaration
  function logToConsole () {
    console.log('working');
  }

  // Call the function
  logToConsole();

}());
// iife ENDS
```

Variable Scope

- If a variable is inside of a function, it will be hidden from other references to its name.
- There are ways of accessing the variable, but scope is a key concept in .js coding.
- If something is within a function, it will be hidden from other functions and the global scope.

```
// Create function declaration, this closes
// over the variable we can see on line 9
function checkVariable () {
  var aVariable = 'hello world';

  // The code below will work and can see the variable
  // because we are in the same scope, within the function
  console.log(aVariable);
}

checkVariable();
// Call the function
// We cannot see this variable, because it is within
// the function above --> hidden or another scope
console.log(aVariable);
```

Objects & Data

- We can use storage items like an object, array or even another variable to bypass scope issues.
- In the image on the left we use a cloud object to store values that may be needed later or in other functions/scopes.

```
// Immediately invoked function expression
(function() {

  // An object in a higher scope that can be used to store values
  cloudObject = {};

  // Create function declaration, this closes
  // over the variable we can see on line 9
  function checkVariable () {
    var aVariable = 'hello world';

    // Assign variable value to cloudObject
    cloudObject.aVariable = aVariable;

    // The code below will work and can see the variable
    // because we are in the same scope, within the function
    console.log(aVariable);
  }

  checkVariable();
  // Call the function
  // We can now access the private scoped value though
  // it being re-assigned to an object on line 13
  console.log(cloudObject.aVariable);

}());
// iife ENDS
```

Object Pattern

- We can create programs/scripts with an object pattern.
- We simply use an object and store relevant properties and methods within it.
- The main difference is we must use the object name when referencing its internal properties and methods.

```
// Create an object
var app = {
  firstWord: 'hello',
  secondWord: ' world...',
  getButton: document.getElementById('update'),
  firstMethod: function () {
    console.log(app.firstWord + app.secondWord);
    console.log(app.getButton);
  },
  init: function () {
    // init function scope
    console.log('init working......');
    // onclick event within a method
    app.getButton.onclick = function () {
      console.log('working.....');
      // // Call functions
      app.firstMethod();
    };
  }
};
// app object ENDS
// Calling an object method, remember to reference the object name
// app.firstMethod();
app.init();
```

classList - add/remove

- The classList property allows for some useful DOM manipulation methods.
- This includes adding and removing classes as well as toggle and checking if a class is found.

```
function startAnimation () {
  // Adding a class with vanilla .js
  getDiv.classList.add('animation');
}

function endAnimation () {
  // Removing a class with vanilla .js
  getDiv.classList.remove('animation');
}
```

References:

https://www.javascripttutorial.net/javascript-dom/javascript-classlist/

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

https://gomakethings.com/function-expressions-vs-function-declarations/

Images referenced from Duckett:

---

http://javascriptbook.com/