PURE JAVASCRIPT

ESSENTIALS

FAHIM CHOWDHURY

Pure JavaScript: Essentials
by Fahim Chowdhury

Preface

This book is to help developers understand the basics of JavaScript and the essential knowledge to build advanced JavaScript projects.

This book is for programmers from all levels who wished to build web projects with pure JavaScript to an industry standard.

Most people who develop using JavaScript are familiar with using toolkits/frameworks/plugins such as JQuery and cannot code without these libraries. It is important to know pure JavaScript to be able to modify or customise code to suit your needs.

In this book is examples on how to code and once you are ready you have take the online quiz.

Scopes

The scope refers to the current location. When calling/using the variable ‘this’ it refers to the current scope (current location). Also if you create a local variable this will be associated the current scope. Variable cannot be accessed out of its scope.

When you create a variable on the root of your script, the variable becomes apart of the global scope know as ‘window’. So you can either call a variable by its name or by ‘window.myvar’.

You can also create local variables, when you create a function or an object the scope of the variable is the parent function or object. This means you cannot access it outside the scope.

What is the ‘window’?
The window object represents an open window in a browser. This is the global scope.

Important:

‘this’ does not always refer to the function or object you are in but if the function/object was called by another scope then ‘this’ will refer to the caller not the scope you are in.

There are workarounds.

Global Example:

```
var globalVariable = "This belongs to the window";

console.log(globalVariable,window.globalVariable);
```

Outcome:

As you can see, you can call the variable either way.

```
This belongs to the window This belongs to the window
```

Local Example:

```
function local()
{
      var localVariable ="I am a local variable";
      console.log(localVariable);
}
local();
console.log(localVariable);
```

Outcome:

When we call the variable within the function it works but outside the function the variable does not exist.

```
   I am a local variable main.js:7
 1.  Uncaught ReferenceError: localVariable is not defined
```

How to Encapsulating Code in a Scope

For best practise you should encapsulate code within a scope.
This has many benefits such as it improves performance because the code does not sit in the 'window' scope which means you cannot garage collect any redundant data.

When to encapsulate?
You should encapsulate code which you do not need globally.

Example of Encapsulating Code:

```
(function(window) {
  function Main() {
   console.log("hello world");
}
Main();
}
)(window);
```

Executing Code once all the content has loaded (like JQuery 'ready' method)

It is important to execute your code once the 'window' has loaded. This ensures all scripts and elements have fully loaded so your code can call upon required libraries and objects.

Below is an example of how to implement this method and the example works across all browsers.

Example of 'window' onLoad method:

```
(function(window) {
  function Main() {
    if(window.addEventListener) {
        window.addEventListener("load", onLoad);
    } else {
        window.attachEvent("onload", onLoad);
    }

}
  function onLoad() {
    // the body has loaded.
    // start coding here!
  }

Main();
}
)(window);
```

Creating DOM Elements through javascript

Javascript has a native way to create any 'tag' element via the 'document.createElement' method.

The method takes a string name of the tag you wish to create. Use uppercase for the tag name to ensure cross browser compatibility.

```
var myDiv = document.createElement("DIV");
```

Adding it to HTML

All DOM objects have 'appendChild' to add elements and 'removeChild' to remove elements.

You can add it to the body with the follow method:

```
var myDiv = document.createElement("DIV");
 document.body.appendChild(myDiv);
```

Attaching Attributes to a DOM object

Sometimes you need to add information to a DOM object to retrieve at a later date. You can use the ‘setAttribute’ method on any DOM element to attach data and then ‘getAttribute’ to retrieve it.

Example:

```
myDiv.setAttribute("index",123456);
console.log( myDiv.getAttribute("index"));
```

Outcome:

```
123456
```

Cross Browser Event Listener

To create a cross browser event listener is simple. You need to consider both methods of adding a listener, the IE ‘attachEvent’ method and the modern ‘addEventListener’.

Below is an example of implementing a method to do this. The first parameter is the DOM element that you wish to add the event to, the second is the event name as a string (without the ‘on’) and finally the last parameter is the callback function.

```
addListener(myDiv,"click",onClick);

      function addListener(obj, event, callback) {
            if (obj.attachEvent) {
                  obj.attachEvent("on" + event, callback);
            } else {
                  obj.addEventListener(event, callback);
            }
```

```
}

function onClick(event)
{
	console.log(event);
}
```

Adding Class Names to DOM elements

To add a class name to an element you call the ‘className’ method in any DOM element.

Example:

```
myDiv.className = "box";
console.log(myDiv.className);
```

Adding Multiple

To add multiple class names you need to increment the string with a space to separate the classes.

```
myDiv.className+=" hide";
console.log(myDiv.className);
```

Removing a Class Name

To Remove a class name you need to replace the class name from the string.

```
myDiv.className=myDiv.className.replace("hide","");
console.log(myDiv.className);
```

Styling DOM Elements

To style a DOM element you use the ‘style’ method in the element and call any css attribute. Remember, where there is a ‘-’ you use camelcase.

Example:

```
myDiv.style.position="absolute";
myDiv.style.width=100+"px";
myDiv.style.height=100+"px";
myDiv.style.backgroundColor="#000";
```

Garbage Collection

In Javascript garbage collection happens periodically. anything that is in the temporary scope gets removed such as local variables. But you can purge objects manually.

You can use the ‘delete’ method to remove/purge objects.

```
test="hello world";
console.log(test);
delete test;
console.log(test);
```

Outcome:

```
   hello world main.js:31
 1.  Uncaught ReferenceError: test is not defined main.js:33
```

If you create an object on the ‘window’ without a ‘var’ then this can be removed by using the ‘delete’ method but if you add a var you cannot remove it. It comes stuck in the global scope.

Example:

```
var test="hello world";
console.log(test);
delete test;
console.log(test);
```

Outcome:

```
hello world main.js:31
hello world
```

So it is important that you only create global ‘var’ when necessary.

OOP in JavaScript

Creating a Singleton

This is one way to create a singleton. Create a variable which is an object.

```
var Singleton =
{
_text:””,
text:function(value)
{
     if(value!=undefined)this._text = value;
     return this._text;
}
}

Singleton.text("new text");
console.log(Singleton.text("new text"));
```

Creating a Class Instance

Creating instance of a class allows you to create multiple instances of the same class.

Example:

```
var Car = function(){};

var car1 = new Car();
var car2 = new Car();
```

Adding Properties to a Class Instance

To add properties/methods to the class you need to use ‘this.’ This will allow it to be a public method.

```
var Car = function() {
      this._text = '';
      this.text = function(value) {
            if (value != undefined)
                  this._text = value;
            return this._text;
      }
};

var car1 = new Car();
var car2 = new Car();
console.log(car1.text("this is car 1"));
console.log(car2.text("this is car 2"));
```

Outcome:

```
   this is car 1 main.js:42
   this is car 2
```

Getter and Setters

To create a getter and setter compatible with many libraries such as TweenLite.
This is how you would do it. Check if the parameter exists and if it does store it otherwise return the current value.

```
this.text = function(value) {
            if (value != undefined)
                  this._text = value;
            return this._text;
      }
```

Class Inheritance

Inheriting classes in Javascript is possible but complex. You need to use the ‘prototype’ attribute of an object to create the methods and variables you want to be inheritable.

Here are the steps to create a inheritable class:

In this example I will create a ‘Shape’ class which will extended to create a rectangle and a circle class.

Shape Class class will contain:

- A ‘init’ method which will create the DOM object.
- A ‘build’ method which will be blank to allow extended classes to override and add some custom code.
- A ‘style’ method to set the CSS attribtues.
- Variable to store width, height, colour, x, y and the DOM element.

1. create an empty function.

```
var Shape = function() {};
```

2. Add the methods and variables to the prototype method.

```
Shape.prototype.color = "#f00";
Shape.prototype.width = 100;
Shape.prototype.height = 100;
Shape.prototype.x = 0;
Shape.prototype.y= 0;
Shape.prototype.element = null;
// creates a DOM object
Shape.prototype.init = function() {
      this.element = document.createElement("DIV");
      document.body.appendChild(this.element);
}
// build the type of shape
Shape.prototype.build = function() {
};
// add color to the DOM object
Shape.prototype.style = function() {
      this.element.style.backgroundColor = this.color;
      this.element.style.width = this.width + "px";
      this.element.style.height = this.height + "px";
      this.element.style.position = "absolute";
      this.element.style.top = this.y+"px";
      this.element.style.left = this.x+"px";
}
```

Lets Test it:

```
var myShape = new Shape();
myShape.init();
myShape.build();
myShape.style();
```

Outcome:

A red box should appear in the HTML body.

3. Creating a ‘Rectangle’ class which extends the ‘Shape’ class and overrides the ‘build’ method.

In the ‘build’ method I will change the width to create the rectangle shape and update the y and colour variables.

```
// Rectangle
var Rectangle = function() {
      this.build = function() {
            this.width = 200;
            this.y=150;
            this.color  ="#0f0";
      }
};
```

You create the class as an instance class.

To inherit another class you need to set the prototype method on the class to an instance of super class.

```
//apply inheritance
Rectangle.prototype = new Shape();
Rectangle.prototype.baseConstructor= new Shape();
```

Lets test this:

```
var rectangle = new Rectangle();
rectangle.init();
```

```
rectangle.build();
rectangle.style();
```

Outcome:

You should see a green rectangle.

4. Create a ‘Cirlce’ class like the ‘Rectangle’ class but apply different values to the variables and create the circle shape.

```
//Circle
var Circle = function() {
      this.build = function() {
            this.y=270;
            this.color  ="#00f";
            // no IE8  support in the example
            this.element.style.borderRadius = (this.width *0.5)+"px";
      }

};
//apply inheritance
Circle.prototype = new Shape();
Circle.prototype.baseConstructor= new Shape();
```

Lets test this.

```
var circle = new Circle();
circle.init();
circle.build();
circle.style();
```

Outcome:

You should see a blue circle (does not work in IE8 and below. Requires a workaround).

Parsing JSON

To parse a string You can use the JSON.parse method in Javascript but it will not work in IE 8 and below.

Here is how you parse JSON in modern browsers:

```
var jsonString = '{"text":"hello world"}';
var json = JSON.parse(jsonString);
console.log(json);
```

To make it work for IE you need to use the ‘eval’ method. Be care of using ‘eval’ as you can execute any JavaScript code through it.

```
var jsonString = '{"text":"hello world"}';

            var json =null;
            if(window.JSON && window.JSON.parse)
            {
                  json = JSON.parse(jsonString);
            }else{
                  json = eval('(' + jsonString + ')');
            }
            console.log(json);
```

Creating a Video Player

To create a video player you first build a video element and append it to your holding element such as the body.

Then you create, append each video source and add the source location and the type for each source tag.

Once this is done you can call methods on the video object such as play and stop.

```
var video = document.createElement('video');
            document.body.appendChild(video);
```

```
            var source = document.createElement('source');

            source.src = 'resource/video.mp4';
            source.type = "video/mp4";

            video.appendChild(source);
            video.play();
```

Hiding and Showing Views/Content

To create a hide and show method you can use the ‘display none’ in the style attribute.

Here is an example:

```
function hide()
      {
            myDiv.style.display="none";
      }
      function show()
      {
            myDiv.style.display="block";
      }
```

MVC

What is MVC?

Model–view–controller (MVC) is a design pattern which separate the logic from the user interaction. There are three main part to it:

Model: stores the data and business logic.
Controller: updates the model and controls the view.
View: This will be the user interface. This represents the data.

Creating an MVC pattern

The Model

The model need to set and retrieve data so I will create a class which stores data, retrieves it and removes it. Also there need to be a method which I've called 'construct' to act as a constructor to setup the class.

The model will take in an object parameter which will allow you to pass in multiple data objects at one time.

model:

```
var Model = function(data) {
	this.set = function(name, value) {
		this[name] = value;
	}
	this.get = function(name) {
		if (this[name] != undefined) {
			return this[name];
		}
		return null;
	}
	this.remove = function(name) {
		if (this[name])
			delete this[name];
	}
	this.construct = function(data) {
		if (data) {
			for (var n in data) {
				this.set(n, data[n]);
			}
		}
	};
	if (data)
		this.construct(data);
};
```

How to Initiate the Model

```
var model = new Model();
```

Adding Data to the Model

```
model.set("color","#0f0");
```

Getting Data from the Model

```
model.get("color");
```

Adding Multiple Default Data Vaules

```
var model = new Model({
                  color:"#f00",
                  width:"100px",
                  name:"Test"
            });
```

The View

I'm creating a view which will display a box and will change colour when a user clicks on a button. Most view have the same foundations such as show, hide,destroy and construct.
The 'construct' method will build the Dom element and apply the styles.

The 'show' method will place this view on the body and the 'hide' will remove the view from the body.

Finally I have a 'changeColor' method which will change the colour of the box.

```
var View = function() {
      var _display = null;
      this.button = null;
      this.construct = function() {
            _display = document.createElement("DIV");
            _display.style.width = "100px";
            _display.style.height = "100px";
            _display.style.backgroundColor = "#f00";

            this.button = document.createElement("BUTTON");
            this.button.innerHTML = "Change Colour";

      }
      this.show = function() {
            document.body.appendChild(_display);
            document.body.appendChild(this.button);
      }
      this.hide = function() {
            document.body.removeChild(_display);
            document.body.removeChild(this.button);
      }
```

```
	this.changeColor = function(color) {
		_display.style.backgroundColor = color;
	}
	this.construct();
}
```

How to Initiate the View and Call ‘show’

```
var view= new View();
view.show();
```

The Controller

I’m going to create a controller that listeners to the button click and it updates the view and the model. The box’s new colour will be store in the model and I will call ‘changeColor’ on the view. The controller will have a function called ‘getColor’ to get a random colour.

```
var Controller = function(model, view) {
	var _model = null;
	var _view = null;
	this.construct = function(model, view) {
		_model = model;
		_view = view;
	}
	this.updateColor = function() {
		_model.set("color", getColor());
		_view.changeColor(_model.get("color"));
	}
	function getColor() {
		var letters = '0123456789ABCDEF'.split('');
		var color = '#';
		for (var i = 0; i < 6; i++) {
			color += letters[Math.round(Math.random() * 15)];
		}
		return color;
	}


	this.construct(model, view);
}
```

How to Initiate the Controller

The model and the view needs to be initialised first before the controller as I will be passing the two variables into the controller.

```
var controller = new Controller(model,view);
```

Make the Controller listen to the Button Click

To make the controller act when the button is clicked, I created a function to handle the button click which will work for cross-browser and keep the correct scope.

Here is the Listener Function

```
function addListener(obj,eventName,callback,scope)
      {
            var root = scope;
            if (obj.addEventListener) {
                  obj.addEventListener(eventName,
function(event){root[callback](event)});
            } else {
                  obj.attachEvent("on"+eventName,
function(event){root[callback](event)});
            }
      }
```

Listening to the Button Click

```
addListener(view.button,"click","updateColor",controller);
```

Coding Essentials

Contents

Creating DOM Elements

About this Section

This section will teach you how to create different DOM Elements such as DIVs and images. In most cases you use the 'document.createElement' method. Then supply the tag name such as 'div', 'a', 'script' etc..

Creating a DIV

```
var div = document.createElement('div');
```

Creating a Button

```
var button = document.createElement('button');
```

Creating a Image Tag (option 1)

```
var img = document.createElement('img');
```

Creating an Image Tag (option 2)

Here is another way to create an image tag.

```
var img = new Image();
```

Creating an A Tag

```
var a = document.createElement('a');
```

Usage Example

```
var img = document.createElement('img');
img.src = "pathToMyImage.jpg";
```

```
document.body.appendChild(img);
```

Getting a DOM Element

About this Section

This section will teach you how to get DOM Elements.

Getting a DOM Element By ID

1. Create an element and give it an ID.

2. In a JS file use document.getElementById to obtain the DOM element.
3. //html
```
<div id="test"></div>
```
4. //javascript
```
var test = document.getElementById("test");
```

Getting Elements By the Tag Name

Using this technique you can get all the element with a specific tag name.

```
var divs = document.getElementsByTagName('div');
```

Get A Child Element

This example will show you how to get a child element by its ID using 'childNodes'.
ChildNodes is an Array of elements within a parent DOM element or XML.

```
var children = document.getElementById('test').childNodes;
for(var i = 0; i < children.length; i++ ){
 if( children[ i ].id == "test" ){
 //found it!
 }
}
```

Styling a DOM Element

About this Section

This section will teach you how to style DOM Elements.

Styling a DOM Element (option 1)

1. Get the Element.
2. Use the 'style' method to style your objects. Remember to provide a string value. Use camel case for attributes that are hyphenated.

```
var test = document.getElementById("test");
test.style.display ="block";
test.style.backgroundColor ="#f00";
```

Styling a DOM Element (option 2)

The style property in a DOM element is an Array so you can update the style as shown below. This technique is required in somecases rather than 'option 1'. You put the exact css property reference in the brackets. No need for camel case.

```
var test = document.getElementById("test");
test.style['background-color'] ="#f00";
```

Utils

About this Section

This section will go through some useful methods.

Rounding a Number

```
var rounded = Math.round(25.2)
```

Turning a Number into an Integer

```
var intVal = parseInt(10.33);
```

Check the type of an Object/Element

```
var str ="My String"; alert(typeof str); //alerts string
```

Generate a Random Number Between a Range

```
//set the min and max range
 var min=5; var max=50;
//get the random number
```

```
var random = Math.floor(Math.random() * (max - min + 1)) + min;
```

Checking the Type of an Object

If you want to know the type of an object the 'typeof' operator will return a string representation of the type.

```
var name = "Adam";  console.log(typeof name);
```

To check if it is a type of something you need to compare it to a lowercase string of the name of the other object. This returns a Boolean value.

```
var name = "Adam"; console.log(typeof(name)=="string"));
```

Check the instance(the Constructor) of an Object

Using 'instanceof' operator allows you to find out the constructor of an object. You compare your object with another object and it will return a Boolean value.

```
console.log(myView instanceof View);
```

Execute a String as a Function

If you need to convert a string into a function you can do this using the 'eval' method.

```
var funcstring = "function test(){alert("it works");}test();";
eval(funcstring);
```

Quiz: Test Your New Knowledge

Go to the link below and do the quiz. Test your knowledge gained from this book.

http://fahimchowdhury.com/javascript/quiz1/

Resources

Examples:

http://fahimchowdhury.com/javascript/purejs_fahimchowdhury_examples.zip