ADVANCED JAVASCRIPT FOR WEB SITES AND WEB APPLICATIONS

Session 3

1. Arrays
2. Searching in arrays
3. Objects
4. Iterating over objects

1. ARRAYS

We have seen previously that arrays hold a set of values, and are defined the following way:

```
var foo = ["a", 123, true, 12, "string"];
```

They are used through the language to manipulate data, and have a set of properties and methods that can help us do this.

length

```
array.length // returns the number of items in the array
```

```
var myArray = ["a", "b", "c"];

myArray.length; // returns 3
```

pop

```
array.pop() // removes and returns the last item in the array
```

```
var myArray = ["a", "b", "c"];

myArray.pop(); // "c"

console.log(myArray) // returns ["a", "b"]
```

push

```
array.push(item) // adds a new item to the array as the last element
```

```
var myArray = ["a", "b", "c"];

myArray.pop(); // "c"

console.log(myArray) // returns ["a", "b"]
```

shift

```
array.shift() // removes and returns the first item in the array
```

```
var myArray = ["a", "b", "c"];

myArray.shift(); // "a"

console.log(myArray) // returns ["b", "c"]
```

slice

```
array.slice(start, end) // returns a section of an array
```

```
var myArray = ["a", "b", "c", "d"];

var newArray = myArray.slice(1,3);

console.log(myArray); // has not changed

console.log(newArray); // ["b", "c"]
```

join

```
array.join(seperator) // converts the array to a string, each value delimit
```

```
var myArray = ["a", "b", "c"];

myArray.join(""); // returns "abc"

myArray.join(","); // returns "a,b,c"
```

When dealing with arrays, here are a few optimisations you can use.

When you need to copy an array, use `slice`:

```
var myArray = ["123", "456", "789"];

var newArray = myArray.slice();
```

When building a long string, use an array and join:

```
var myArray = [
  "this needs to be a long string",
  "but it makes it difficult to build",
  "and type in one go",
  "and some browsers have issues with assigning strings",
  "with long values in a simple var assigment"
];

var myString = myArray.join(" ");
```

This also works well when trying to dynamically build a string, and / or for formatting purposes.

```
var htmlTemplateArray = [
  "<div class='my-class'>",
    "<p>A paragraph of text</p>",
    "<p>A second paragraph of text</p>",
  "</div>"
];

var htmlTemplate = htmlTemplateArray.join("\n");
```

2. SEARCHING IN ARRAYS

You sometimes need to check if a value exists in a array.

When you need to do this, you can use `indexOf`.

```
arrayElement.indexOf(searchElement);
```

This method returns -1, if the element isn't found in the array, or the index of the first occurence of the term.

```
myArray = ["abc", 123, "foo", "bar"];

myArray.indexOf("abc"); // returns 0
myArray.indexOf("foo"); // returns 2
```

3. OBJECTS

Objects are a collection of properties, defined as pair/value. The property of an object can be a function, in which case it is known as a method.

```
var myObject = {

  colour: "red",              // property

  state: true,                // property

  action: function () {       //
    //do something            // method
  }                           //

};
```

You can access the properties and method of an object using the dot notation:

```
myObject.colour; // returns "red"

myObject.action(); // runs the function defined in the object
```

If you look back at the revealing module pattern we saw previously, you'll see the pattern returns an object, where we can define the properties and methods we need. We can then access these properties and methods using the dot notation.

```
var APP = (function () {

  var privateVar,

    publicVar,

    privateFunction = function () {},

    publicFunction = function () {};

  // returns a public object
  return {
    returnValue: publicVar,
    returnAction: publicFunction
  };

}());

APP.returnValue;
APP.returnAction();
```

Using this pattern again and an "init" function, you could use an object to store global configration parameters for this app. This can be easier than declaring multiple variables at the start of the pattern:

```
var APP = (function () {

  var
    config = {
      config1: 10,
      config2: true,
      config3: "string"
    },

    publicFunction = function () {
      // use the config object here
    },

    init = function () {
      publicFunction();
    };

  return {
    init: init
  };

}());
APP.init;
```

4. ITERATING OVER OBJECTS

If you need to iterate over objects, you can use a `for...in` loop:

```
var obj = {a:1, b:2, c:3};

for (var prop in obj) {
  console.log("obj." + prop + " = " + obj[prop]);
}

// Output:
// "obj.a = 1"
// "obj.b = 2"
// "obj.c = 3"
```

Exercise 1

Instead of passing multiple arguments to a function, you can pass a single object with multiple properties. This can improve the organisation of the code.

In the followng snippet of code, replace the arguments with an object:

```
function appendText(element, text) {
 element.textContent =  text;
}
```

```
var config = {
  el: element,
  text: text
}

function appendText(config) {
 config.el.textContent =  config.text;
}
```

Exercise 2: Objects as configuration for an application

In the exercise2.js there is an existing script, that uses the revealing module pattern.

What does it do?

Also have a look at the HTML in the exercise2.html file.

This application has one disatvantage, every day, you need to change the code to reflect the day's currency rates.

You can make this easier by creating a new object that holds the different rates and can be used in the application.

Add an object that holds the different rates:

```
rates = {
  GBPtoEUR: 1.26,
  GBPtoUSD: 1.62,
  GBPtoYEN: 172.88
}
```

Use this object to do the conversion:

```
if(currency === "EUR") {
  convertedCurrency = originalAmountValue * rates.GBPtoEUR;
}
else if (currency === "USD") {
  convertedCurrency = originalAmountValue * rates.GBPtoUSD;
}
else if (currency === "YEN") {
  convertedCurrency = originalAmountValue * rates.GBPtoYEN;
}
```

Exercise 3

Build a simple application to manage an online shopping basket.

The basket should be an array.

Yon can add items to the basket. Each item added is an object of the following format:

```
item = {
  name: "name of the product",
  price: 3.45
}
```

You can retrieve the number of items in the basket.

You'll need to expose the methods and properties. You won't need an init function here.

```
var SHOPPINGBASKET = (function () {

  var basket = [],

    addItem = function (newItem) {
      basket.push(newItem);
    },
    countItems = function () {
      return basket.length;
    };

  // returns a public object
  return {
    addItem: addItem,
    countItems: countItems
  };

}());

SHOPPINGBASKET.addItem({name: "product1", price: 34.45});
SHOPPINGBASKET.addItem({name: "product2", price: 1.99});
SHOPPINGBASKET.countItems(); // returns 2
```

Exercise 4

Using a `for...in` loop, build a simple function that takes an object as an argument and counts the number of properties and methods in this object.

Remember that properties are really functions (`typeof` will return the string "function"); and properties are everything else.

You can use this object as a test:

```
var basket = {
  items: 0,
  totalPrice: 0.50,
  addItem: function () {},
  removeItem: function () {},
  howManyItems: function () {}
};
```

```
function countMethodsProperties (obj) {
  var methodTotal = 0,
    propertyTotal = 0;

  for (var prop in obj) {
    if ((typeof obj[prop]) === "function") {
      methodTotal++;
    }
    else {
      propertyTotal++;
    }
  }
  console.log("object " + obj + " has " + methodTotal + " methods and " + p
}

var basket = {
  items: 0,
  totalPrice: 0.50,
  addItem: function () {},
  removeItem: function () {},
  howManyItems: function () {}
};

countMethodsProperties(basket);
```

Exercise 5

Here is some HTML. Using an array, create one string that concatenates all the HTML and add it to the DOM.

You can add the HTML to the workshop3.html page after the h1, using `insertAdjacentHTML`.

```
<ul>
  <li>This is a first item</li>
  <li>This is a second item</li>
  <li>And another one!</li>
  <li>...and a last one</li>
</ul>
```

```
var htmlArray = [
  "<ul>",
      "<li>This is a first item</li>",
      "<li>This is a second item</li>",
      "<li>And another one!</li>",
      "<li>...and a last one</li>",
  "</ul>"
];

var htmlString = htmlArray.join("\n");

var header = document.querySelectorAll(".header")[0];

header.insertAdjacentHTML("afterend", htmlString);
```

Exercise 6

Here is an object holding information about various versions of Windows 8.

Create a `ul` list that will show the name of the products, linking to their pages and show their price.

You can add the HTML to the workshop3.html page after the h1, using `insertAdjacentHTML`.

Your final HTML should look like:

```
<ul>
  <li><a href="url1">product 1</a> - <span>Price: price1</span></li>
  <li><a href="url2">product 2</a> - <span>Price: price 2</span></li>
</ul>
```

```
var basket = {
  item1: {
    value: 99.99,
    name: "Windows 8",
    rating: 3.5,
    url: "http://www.microsoftstore.com/store/msusa/en_US/pdp/Windows-8.1/p
    uid: "288532600"
  },
  item2: {
    value: 189.99,
    name: "Windows 8.1 Pro",
    rating: 5,
    url: "http://www.microsoftstore.com/store/msusa/en_US/pdp/Windows-8.1-P
    uid: "288532900"
  },
  item3: {
    value: 49.99, name: "Windows 8.1 Pro Student", rating: 4, url: "http://

  },
  item4: {
    value: 99.99, name: "Windows 8.1 Pro Pack", rating: 0, url: "http://www
  }
};
```

```
var basket = {item1: {value: 99.99,name: "Windows 8",rating: 3.5,url: "http
  currentItem,
  productHtml,
  endHtml,
  header,
  allProducts = [];

allProducts.push("<ul>");

for(var prop in basket) {
  currentItem = basket[prop];
  productHtml = "<li><a href='" + currentItem.url + "'>" + currentItem.name
              + "</a> - <span>Price: " + currentItem.value + "</span>";
  allProducts.push(productHtml);
}

allProducts.push("</ul>");

endHtml = allProducts.join("\n");

header = document.querySelectorAll(".header")[0];

header.insertAdjacentHTML("afterend", endHtml);
```