ADVANCED JAVASCRIPT FOR WEB SITES AND WEB APPLICATIONS

Session 2

1. Arrays
2. Objects
3. Working with JSON
4. A (very) quick word on prototype

1. ARRAYS

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.

⚠ Although it's often considered harmfull or bad practice, you can also define non numerical, associative arrays.

```
var foo = new Array();
foo["size"] = "small";
foo["colour"] = "red";
foor["make"] = "mustang";
```

Objects are prefered instead of non-numerical arrays. Although it's handy to remember that they also exist, to work with some libraries or plugins.

For the rest of the course, we'll be working with numerical arrays only.

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

```
myArray[0]; // returns a

myArray[2]; // returns c

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

Array manipulations

push

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

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

myArray.push("d");

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

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 delimited by the seperator parameter
```

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

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

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

A few recommendations when working with arrays

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

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

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

When building strings, 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 applies when building dynamic strings (using variables).

This also works well 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. OBJECTS

Objects are a collection of properties, defined as pair/value.

If the property of an object is a function, it is known as a method.

New empty object syntax:

```
var myObject = {};
```

You can create properties and method of an object when you define it:

```
var myObject = {

  colour: "red",

  state: true,

  action: function () {
    //do something
  }

};
```

...and add new properties and methods at any time after this:

```
myObject.newPropMake = "mustang";

myObject.newMethod = function () {
  // do something here
};
```

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
```

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"
```

Rewrite the previous example to use an array and join instead of string concatenation when building each result.

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

for (var prop in obj) {

  var result = [];

  result.push("obj.");
  result.push(prop);
  result.push(" = ");
  result.push(obj[prop]);

  console.log(obj.join());
}
```

Objects as function arguments

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 following code, replace the arguments with an object:

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

appendText(DOMelement, "text to set");
```

```
var config = {
  el: DOMelement,
  text: "text to set"
};

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

Exercise: Objects as configuration for an application

In `converter.js` there is an existing script.

What does it do?

Also have a look at the HTML in `converter.html`.

This application has one problem, the rates are mixed with the code and hard to manage globally.

You can improve this by creating a new object that holds the different rates and reference the object when the rates are needed.

Add an object that holds the different rates:

```
rates = {
  eur: 1.26,
  usd: 1.62,
  yen: 172.88
}
```

Use this object to do the conversion:

```
if(currency === "eur") {
  convertedCurrency = originalAmountValue * rates.eur;
}
else if (currency === "usd") {
  convertedCurrency = originalAmountValue * rates.usd;
}
else if (currency === "yen") {
  convertedCurrency = originalAmountValue * rates.yen;
}
```

You can further improve this by using the object property names directly:

```
convertedCurrency = originalAmountValue * rates.currency;
```

Exercise 1

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 " +
}

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

countMethodsProperties(basket);
```

Exercise 2

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 workshop2.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 3

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 workshop2.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
    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
    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://w
  }
};
```

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

allProducts.push("<ul>");

for(var prop in basket) {
  currentItem = basket[prop];
  productHtml = "<li><a href='" + currentItem.url + "'>" + currentItem.na
              + "</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);
```

3. WORKING WITH JSON

`JSON` is a data storage format, using the JavaScript object notation syntax.

It is:

- lightweight
- transferable as text
- easy to use with JavaScript as it uses the same syntax.

Example:

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25
}
```

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": null
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": false
}
```

⚠ To be valid, all the property names need to be wrapped in double quotes.

In JavaScript, you can parse a JSON string to an object using the native `JSON.parse()`.

On older browsers, you may need a library that provides the equivalent functions.

```
JSON.parse("JSON string");
```

This returns a JavaScript object that you can then interact with.

4. A (very) quick word on prototype

Every instance in JavaScript is an object. This means that everything inherits properties and methods from the global object type.

Every type of object (array, function, strings, numbers etc.) will inherit the properties and methods of the type `object` and will add new ones to this.

This is the `prototype`.

You can add to the prototype of `Arrays` for example, by doing:

```
var myArray = [1, 3];

myArray.newMethod(); // error
```

```
Array.prototype.newMethod = function () {
  return this
}
```

```
myArray.newMethod(); // runs function defined above
```

Avoid adding properties and method to native objects and types!

You risk breaking existing functionalities and code.

You can however define your own object types, use as constructors to build other objects. These new objects will inherit the properties and methods of your constructors. And you can change the prototype for these.

You can check the `constructor` of an object by doing:

```
var myArray = [];

myArray.constructor // returns Array()

var myObject = {};

myObject.constructor // returns Object()

var myString = "";

myString.constructor // returns String()
```

Using the `new` operator, you can create a new instance of a constructor:

```
function Car(name) {
  this.name = name;
}

var myCar = new Car("Mustang");

typeof myCar // returns object
myCar.constructor // returns Car(name)
```

And then add to the prototype of the initial constructor function.

```
Car.prototype = {
  returnName: function(){return this.name;}
};
```