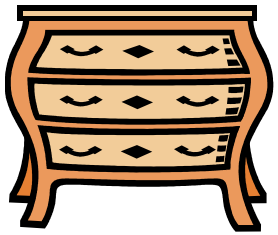
The University Of Sheffield.

COM1004: Web and Internet Technology

Lecture 9: JavaScript: Part 2

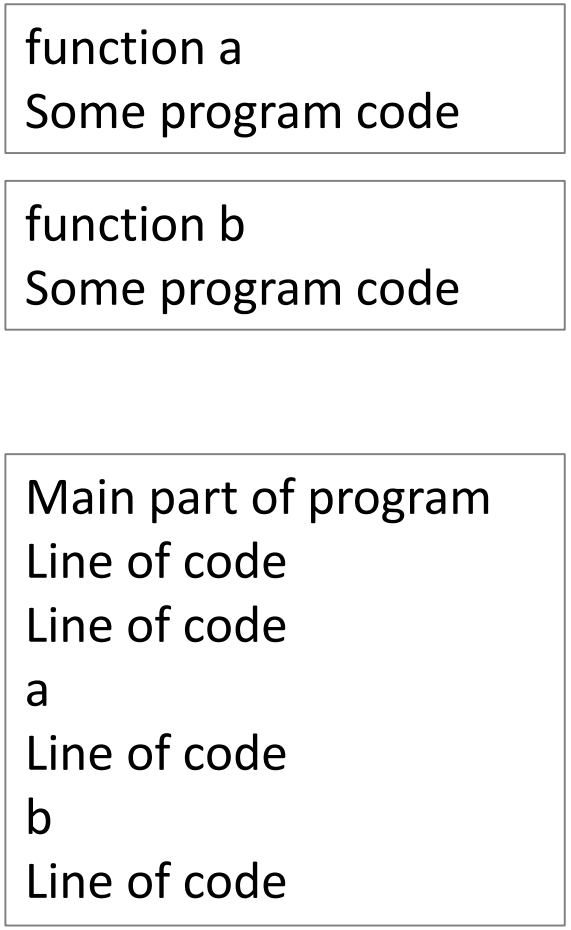


Dr. Steve Maddock
s.maddock@sheffield.ac.uk

1. Introduction

For a Web site:

- Structure using HTML
- Appearance using CSS
- Behaviour using **JavaScript**
  - Although, see recent CSS3 features, e.g. animation
- Last week: The basics of JavaScript
- This lecture:
  - Functions and built-in objects



2. Functions

- A chunk of program code that performs a specific task which is independent from surrounding code
  - Example: Calculate the area of a rectangle
- One of the mainstays of structured programming is to decompose a large problem into simpler, manageable pieces

- We'll look at three examples:
  - Function without parameters
  - Function with parameters
  - Function that returns a value

2.1 Functions without parameters

- We can turn useful program code into reusable commands
- Instead of:

```
document.write("<p>***********<\/p>");
// some other program code
document.write("<p>***********<\/p>");
// some other program code
document.write("<p>***********<\/p>");
```

- We can use:

```
function printStars() {
  document.write("<p>***********<\/p>");
}

printStars();
// some other program code
printStars();
// some other program code
printStars();
```

example

2.2 Functions with parameters

- We can make the process more flexible by using parameters
- The function is declared with one or more *formal parameters*
- *Actual parameters* are matched to formal parameters
- The formal parameter name is used in the function body

Formal parameter

```
function print(message) {
  document.write("<p>"+message+"<\/p>");
}

print("Hello world");
print("******");
print("Another message which can be as long as we like");
```

Actual parameter

example

2.2 Functions with parameters

- In this example, the actual parameter is a number:

```
function printStars(n) {
  for (var i=0; i<n; ++i)
    document.write("*");
}

printStars(1);
document.write("<br />");
printStars(2);
document.write("<br />");
printStars(3);
document.write("<br />");
printStars(4);
document.write("<br />");
```

```
*
**
***
****
```

example

2.2 Functions with parameters

- In this example, the actual parameter is a number:

```
function printStars(n) {
  for (var i=0; i<n; ++i)
    document.write("*");
}

for (var i=1; i<=4; i++) {
  printStars(i);
  document.write("<br />");
}
```

```
*
**
***
****
```

example

2.2 Functions with parameters

- In this example, two parameters are used, separated by commas:

```
function printNChars(n, ch) {
   for (var i=0; i<n; i++)
     document.write(ch);
 }

 document.write("<p>");
 printNChars(Math.random()*10, "a");
 document.write("<br />");
 printNChars(Math.random()*10, "b");
 document.write("</p>");
```

example

2.2 Functions with parameters

- Simple types are passed ***by value***
  - The **value** of the actual parameter is **copied** into the formal parameter
  - Any changes to the value stored in the formal parameter do not affect the actual parameter

```
function printStars(n) {
  for (var i=0; i<n; i++){
    document.write("*");
  }
  n = 10000; // pointless
}

document.write("<p>");
for (var i=1; i<=4; i++) {
  printStars(i);
  document.write("<br />");
}
document.write("</p>");
```

- In contrast, objects are passed *by reference* – see later
  - A modification of the contents of the object referred to by the formal parameter will also affect the contents of the object referred to actual parameter

2.3 return value for a function

```
const VAT_RATE = 0.2;

function getCost() {
  return parseFloat(prompt("Cost before VAT in GBP? "));
}

function calculateVat(x) {
  return x*VAT_RATE;
}

var cost = getCost();
document.write("<p>Cost before VAT: "+cost+"<\/p>");
var vat = calculateVat(cost);
var costWithVat = cost + vat;
document.write("<p>Cost with VAT added:"
               + costWithVat.toFixed(2) + "<\/p>");
```

`const` is part of the latest specification. Typically, use upper-case for constants

Functions can return a value

Returned result must be assigned to a variable or the function must be part of an expression

example

2.3 return value for a function

```
const VAT_RATE = 0.2;
console.log("vat:"+VAT_RATE);

function getCost() {
  return parseInt(prompt("Cost before VAT in GBP? "));
}

function inputCost() {
  var cost = getCost();
  document.write(cost);
  while (isNaN(cost)) {
    document.write("<p>Not a number, try again<\/p>");
    cost = getCost();
  }
  return cost;
}

function calculateVat(x) {
  return x*VAT_RATE;
}

var cost = inputCost();
// and so on
```

Example of one function calling another function

Use while loop to continue reading input until users enters a number

example

2.4 Scope

JavaScript has function-level scoping, *not* Java's block-level scoping

```
var i = 1;                                  //  Global scope
var j = 2;                                  //  Global scope

function a() {
  var i = 0;                                //  Use of var, therefore local scope for this i,
                                            //   i.e. just this function
  document.write(i+"<br \/>");              //  Display 0
  j = 3;                                    //  Changing value of global variable,
}                                           //  but NOT considered good practice

document.write(i + ", " + j);               //  display 1, 2
document.write("<br \/>");
a();
document.write(i + ", " + j);               //  display 1, 3
```

JavaScript has function-level scoping, *not* Java's block-level scoping

2.4 Scope (advanced)

```
var i = 1;                                  //  Global scope
j = 2;                                      //  Global scope, but BAD style
if (i>j) {
  var k = 3;                                //  Global scope  anywhere in program after this line
}

function a() {
  var i = 0;                                //  Use of var, therefore local scope for this i,
                                            //   i.e. just this function
  document.write(i+"<br \/>");              //  Display 0
  m = 4;                                    //  Global scope, once the function is called,
                                            //   but BAD style
  if (m > i)
    var n = 4;                              //  Local scope anywhere in the function after this line
  j = 3;                                    //  Changing value of global variable,
}                                           //  but NOT considered good practice

document.write(i+"<br \/>");                //  display 1
a();
document.write(i + ", " + m);               //  display 1, 4
```

example

2.4 Scope

- Be careful when attaching multiple scripts to the same web page
- Perhaps both scripts are trying to define and make use of the same global variable
- Perhaps script A has left the var off the front of the variable definition
  - Thus script A will be using the variable declared with var in script B
  - Always use var to define a variable

2.5 Functions – advantages and disadvantages

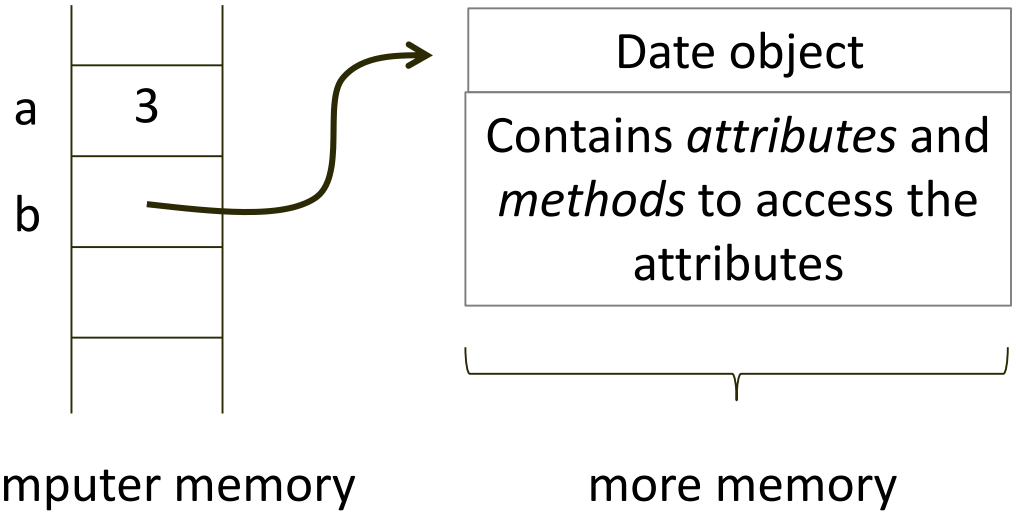
- Advantages:
  - Divides a long program into simpler, manageable pieces
  - Reduces code duplication
  - Promotes code reuse
  - Hides implementation detail
  - Improves traceability which helps debugging
- Disadvantages:
  - Some computational overhead to invoke a function

3. Objects in JavaScript

- JavaScript is an 'object oriented programming language'
  - Unlike Java, it does not use classes, but does have objects
  - Although see ECMAScript 6 (2015) which includes 'class'
- Rather than view a program as a list of tasks, we view it as a collection of interacting objects
- An object consists of attributes (data fields) and methods (ways to access the data fields)

```
var a = 3;

var b = new Date();
```



3.1 Built-in objects in JavaScript

- We'll look at how to create objects in a later lecture

Today…

- Built-in Objects
  - E.g., Array, Boolean, Date, Error, Function, Math, Number, Object, RegExp, String
- *Example*: The Date object
  - Methods: getDate, getDay, getHours, …, getTime, …, setDate, …
- *Example*: String – this is a special object
  - Methods: charAt, substring, toLowerCase, …
- *Example*: Math – lots of maths functions

- There are also objects that relate to the DOM hierarchy and to the browser, e.g. document – see later lecture

3.2 The Date object

- Time is milliseconds since midnight on January 1, 1970

```
<h1>Date example</h1>
<script>
  var now = new Date();
  document.write("Date:" + now + "<br \/>");
  var year = now.getFullYear();
  document.write("Year: " + year + "<br \/>");
  var time = now.getTime();
  document.write("Time: " + time + "<br \/>");
</script>
```

example

3.3 Strings are special

- Strings are immutable – there are no features in JavaScript to alter part of a string once it has been created
- Don't need to use the keyword new to create a String object

```
                                // Note: length is a property, toUpperCase is a method
var str="Hello World!";
document.write(str.length);                    // 12
document.write(str.toUpperCase());             // HELLO WORLD!
```


3.4 The Math object

- Properties and methods for mathematical constants and functions
- It is a global object, so just use Math.*property* or Math.*method*
- Example property is the constant: Math.PI
- Example methods: Math.round(), Math.sqrt(n), Math.ceil(), Math.floor(), Math.random()
- Example:
- Math.sin

```
var radians = 45 *Math.PI/180;
var x = Math.sin(radians);
document.write(x);        // 0.7071067811865475
```

3.5 Advanced: Functions are objects

- Functions may be stored in variables, passed to other functions or stored in objects, where they become methods.

```
function average(a,b) {
  return (a+b)/2;
}

var f = average;
f(124,68);    // answer is 96
```


example

4. Output to the Web page

- So far output has been sent to the webpage using document.write()
- In practice, this is rarely used and generally frowned on
- Instead, we get and set the content of the webpage, i.e. the elements and their content
- For now we'll look at:
  - getElementById()
  - textContent
  - innerHTML

- More on the Document Object Model (DOM) in later lectures

4.1 getElementById

- Select an individual element on the webpage, given the value of its id attribute
- *Example*: calculate room area: **roomarea2.html**

```
<!-- rest of html file here -->
<body>
<h1>Example</h1>

<p id="answer1">Answer here</p>
<p id="answer2">Answer here too</p>
<p>Some more text</p>

<!-- include script at end of file just before end of body tag -->
<script src="./js/roomarea2.js"></script>

</body>
</html>
```

example

4.1 *document*.getElementById

- Select an individual element on the webpage, given the value of its id attribute – a DOM method
- *Example*: calculate room area: **roomarea2.js**

```
var length = parseFloat(prompt("Rectangle length in cm?"));
console.log("Length=" + length);
var width = parseFloat(prompt("Rectangle width in cm?"));
console.log("Width=" + width);

var area = length * width;

var element1 = document.getElementById('answer1');
element1.textContent = "The area is: " + area;

var element2 = document.getElementById('answer2');
element2.innerHTML = "<em>The area is: " + area + "</em>";
```


example

4.1 *document*.getElementById

roomarea2.html snippet:

```
<p id="answer1">Answer here</p>
<p id="answer2">Answer here too</p>
```

```
var length = parseFloat(prompt("Rectangle length in cm?"));
console.log("Length=" + length);
var width = parseFloat(prompt("Rectangle width in cm?"));
console.log("Width=" + width);

var area = length * width;

var element1 = document.getElementById('answer1');
element1.textContent = "The area is: " + area;

var element2 = document.getElementById('answer2');
element2.innerHTML = "<em>The area is: " + area + "</em>";
```

example

4.2 *element*.textContent

- A property of an element on the webpage
- Can be used to get or set the content of an element
- (In IE8 or earlier, textContent does not work)

```
var length = parseFloat(prompt("Rectangle length in cm?"));
console.log("Length=" + length);
var width = parseFloat(prompt("Rectangle width in cm?"));
console.log("Width=" + width);

var area = length * width;

var element1 = document.getElementById('answer1');
element1.textContent = "The area is: " + area;

var element2 = document.getElementById('answer2');
element2.innerHTML = "<em>The area is: " + area + "</em>";
```

example

4.3 *element*.innerHTML

- Can be used to retrieve and replace content of a DOM tree, i.e. the hierarchy of elements on a webpage
- Can retrieve and replace HTML code
- But there are security issues

```
var length = parseFloat(prompt("Rectangle length in cm?"));
console.log("Length=" + length);
var width = parseFloat(prompt("Rectangle width in cm?"));
console.log("Width=" + width);

var area = length * width;

var element1 = document.getElementById('answer1');
element1.textContent = "The area is: " + area;

var element2 = document.getElementById('answer2');
element2.innerHTML = "<em>The area is: " + area + "</em>";
```

example

4.3.1 *element*.innerHTML versus DOM manipulation (Duckett, 2014)

- innerHTML advantages
  - Can be used to add lots of markup to a webpage using minimal code
  - Quicker than using DOM manipulation approaches
  - Simple way to remove lots of nested content
- innerHTML disadvantages
  - It should not be used to add **content that has come from the user**, as this can be a significant security risk – cross-site scripting (XXS) attacks
  - More difficult to isolate single elements in the DOM to update
  - Some issues with event handlers on elements
- More on DOM manipulation in a later lecture

Duckett, J. JavaScript and jQuery: interactive front-end web development, Wiley, 2014

5. Summary

- Functions turn useful code into reusable commands
  - Parameters and the ability to return values make them very powerful
  - Multiple values can be returned using arrays – see next lecture
- JavaScript is an 'object oriented programming language'
  - Lots of built-in objects that are useful, e.g. Date, String, Math
  - We'll look at objects and *prototyping* in more detail in a later lecture
- Do not use document.write() to write content to a webpage
  - Instead, use DOM manipulation
- Do not use of element.innerHTML to add content to a webpage when untrusted content is used, e.g. input by a user
- *Next lecture*: Arrays

Appendix: Another example

```
<body>
  <h1>Example: Date</h1>
  <p id="outputArea">
  Greeting here
  </p>
  <p>Some more text</p>
  <!-- include script at end of file
    just before end of body tag -->
  <script
  src="./js/date2.js">
  </script>
</body>
```

```
var today = new Date();
console.log("Date=" + today);
var hour = today.getHours();
console.log("The hour is: " + hour);
var greeting = "";
if (hour < 12) {
  greeting = "Good morning"
}
else if (hour < 18) {
  greeting = "Good afternoon"
}
else {
  greeting = "Good evening"
}
element = document.getElementById('outputArea');
element.textContent = greeting;
```