

Web Technologies Week 3 Javascript

Dr V. Lanfranchi
Prof Fabio Ciravegna
The University of Sheffield
f.ciravegna@shef.ac.uk



The
University
Of
Sheffield.

2

Javascript

Introduction

- For a Web site:
 - The structure is indicated using HTML
 - The appearance is controlled using CSS
 - The behaviour is controlled using JavaScript
 - Although see HTML5 forms
 - Although see CSS3 features, e.g. animation
- JavaScript can be used to:
 - Interact with the user
 - Control the web browser
 - Alter the document content
 - Examples: Gmail, Twitter, Firefox
- However, later in the module we will see other uses of JS
 - Server side (to create a node.js server)
 - To implement functionality such as video conferences



JavaScript is not Java

- From wikipedia:

Java	JavaScript
Static typing	Dynamic typing – a variable can hold an object of any type
Loaded from compiled bytecode	Loaded as human-readable text; Interpreted programming language

- However:
 - Both have a structured C-like syntax (e.g. if, while, switch)
 - Both are case-sensitive
 - JavaScript copies many names and naming conventions from Java

Javascript - client side

- Our focus will be - for now - on client-side JavaScript
 - Scripts are run by the client computer, not the Web server
 - Allows to manage client-side behaviour
 - We will discuss server side in lecture 6
- Javascript is
 - Executed when the user performs an action
 - Implicitly
 - e.g. when finishing writing an email address in a form
 - Explicitly
 - e.g. when clicking a button

Why Javascript?

- Less server interaction
- Immediate user feedback
- Increased interactivity
- Rich interfaces
- https://www.w3schools.com/html/html5_draganddrop.asp

How to add Javascript to an HTML file

- Internal Javascript
 - In <head> section
 - In <body> section
 - In both sections
 - It is interpreted in the order it appears in the page

```
<script language="javascript" type="text/javascript">  
    JavaScript code  
</script>
```

- External javascript
 - In <head section>

```
<script src="validation.js"></script>
```

Javascript syntax

- Javascript is case-sensitive
- Statements are followed by a semicolon
 - You can omit semicolons if using a new line
 - But NOT good coding practice

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10;

    var2 = 20;

  //-->
</script>
```

```
<script language="javascript" type="text/javascript">
  <!--
    var1 = 10
    var2 = 20

  //-->
</script>
```


Javascript datatypes

- Javascript has five primitive data types
 - Numbers
 - Strings
 - Boolean
 - Null
 - Undefined
- Javascript also support the datatype Object
- **IMPORTANT:**
 - Javascript does not make a distinction between integer values and floating-point values.
 - All numbers in JavaScript are represented as floating-point values.

Javascript variables

- In Javascript you declare a variable with the keyword var

```
<script language="javascript" type="text/javascript">
```

```
var1 = 10;
```

```
var2 = 20;
```

```
</script>
```

- Var is used for declaration or initialisation
- You cannot re-declare same variable
- IMPORTANT
 - Variables are untyped
 - The value type of a variable can change during the execution of a program
 - Gives flexibility
 - Can introduce errors

Javascript variables

- Variables can be
 - Global
 - Local
 - To a function
- The local variable takes priority over the global

```
<html>
  <body onload = checkscope();>
    <script type = "text/javascript">
      <!--
        var myVar = "global"; // Declare a global variable
        function checkscope( ) {
          var myVar = "local"; // Declare a local variable
          document.write(myVar);
        }
      //-->
    </script>
  </body>
</html>
```

Variable Scope

Always declare a variable using the `var` keyword to avoid ambiguity of scope

- **Variable Scope**

In JavaScript, variable scope can be global or local. Scope is determined by where a variable is declared, and to some extent whether the `var` keyword is used. Compared to programming languages like C or Java, this is a very simplistic approach.

Global

A variable that is declared outside any functions is *global*. This means it can be referenced anywhere in the current document.

Declared outside any functions, **with or without** the **var** keyword.

Declared inside a function **without** using the **var** keyword, but only once the function is called.

Local

A variable that is declared inside a function is *local*. This means it can only be referenced within the function it is declared.

Declared in a function **with** the **var** keyword.

Other notes

If a variable is declared inside a conditional statement, it is still available anywhere following the declaration in the containing function (or globally if the conditional is not in a function). However, it will equal *undefined* if the condition evaluates to false, unless the variable is later assigned a value.

http://www.mredkj.com/tutorials/reference_js_intro.html#scope

Scope

- Be careful when attaching multiple scripts to the same web page.
- Possibility of a scope conflict
- Perhaps both scripts are trying to define the same global variable
- Perhaps one script has left the var off the front of the statement to define a variable
 - Always use var to define a variable

Variables

- Rules for identifiers are same as Java
 - First character must be a letter, an underscore (_) or a dollar sign (\$)
 - Can't be keywords
- Further details:
 - http://en.wikipedia.org/wiki/JavaScript_syntax

JavaScript keywords	Reserved for future use	Reserved for browser
break	abstract	alert
case	boolean	blur
catch	byte	closed
continue	char	document
default	class	focus
delete	const	frames
do	debugger	history
else	double	innerHeight
finally	enum	innerWidth
for	export	length
function	extends	location
if	final	navigator
in	float	open
instanceof	goto	outerHeight
new	implements	outerWidth
return	import	parent
switch	int	screen
this	interface	screenX
throw	long	screenY
try	native	statusbar
typeof	package	window
var	private	
void	protected	
while	public	
with	short	
	static	
	super	
	synchronized	
	throws	
	transient	
	volatile	

McFarland, D.S,
JavaScript: the missing
manual, 2nd edition,
O'Reilly, 2009

Comments

```
<script>
  var length = prompt("Rectangle length in cm?");
  var width = prompt("Rectangle width in cm?");
  document.write("Area = "+length*width);
  alert("Area = "+length*width);

  // single line comment
  /* multiple-line comment
     multiple-line comment */
</script>
```

Comments similar to other
programming languages

- The most common ones...
- Given $y=5$, then:

Operator	Description	Example	Result
+	Addition	$x=y+2$	$x=7$
-	Subtraction	$x=y-2$	$x=3$
*	Multiplication	$x=y*2$	$x=10$
/	Division	$x=y/2$	$x=2.5$
%	Modulus (division)	$x=y\%2$	$x=1$

- The most common ones...
- Given $x=10$ and $y=5$, then:

Operator	Example	Same As	Result
=	$x=y$		$x=5$
+=	$x+=y$	$x=x+y$	$x=15$
-=	$x-=y$	$x=x-y$	$x=5$
=	$x=y$	$x=x*y$	$x=50$
/=	$x/=y$	$x=x/y$	$x=2$
%=	$x\%=y$	$x=x\%y$	$x=0$

https://developer.mozilla.org/en/JavaScript/Reference/Operators/Arithmetic_Operators

5.1 The '+' operator

- “The '+' operator is overloaded. It is used...
 - string concatenation...
 - arithmetic addition...
 - to convert numbers to strings.
 - It also has special meaning when used in a regular expression.”

http://en.wikipedia.org/wiki/JavaScript_syntax

```
var vatRate = 0.15;  
var costWithoutVat = 3;  
var vat = costWithoutVat*vatRate;  
var costWithVat = costWithoutVat + vat;  
var part1 = "Hello";  
var part2 = "world!!";  
var message = part1 + " " + part2;
```

The '+' operator

- Automatic type conversion

```
var score = 8;  
var message = "Score out of 10: "+ score;
```

- Can cause problems (https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Core_Language_Features#Values)
- If value1 is just letters rather than numbers, the result is NaN.

The '+' operator

- Can cause problems (https://developer.mozilla.org/en/Core_JavaScript_1.5_Guide/Core_Language_Features#Values)

```
var value1 = "37";  
var value2 = 7;  
var unexpectedResult = value1 + value2; // returns "377"
```

- If value1 is just letters rather than numbers, the result is NaN.

```
var result = +value1 + value2; // returns 44  
var result = Number(value1) + value2; // returns 44  
var value3 = "hello";  
var result3 = +value3 + value1; // returns NaN37
```

Comparison operators

Sr.No	Operator and Description
1	<p>== (Equal)</p> <p>Checks if the value of two operands are equal or not, if yes, then the condition becomes true.</p> <p>Ex: (A == B) is not true.</p>
2	<p>!= (Not Equal)</p> <p>Checks if the value of two operands are equal or not, if the values are not equal, then the condition becomes true.</p> <p>Ex: (A != B) is true.</p>
3	<p>> (Greater than)</p> <p>Checks if the value of the left operand is greater than the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A > B) is not true.</p>
4	<p>< (Less than)</p> <p>Checks if the value of the left operand is less than the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A < B) is true.</p>
5	<p>>= (Greater than or Equal to)</p> <p>Checks if the value of the left operand is greater than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A >= B) is not true.</p>
6	<p><= (Less than or Equal to)</p> <p>Checks if the value of the left operand is less than or equal to the value of the right operand, if yes, then the condition becomes true.</p> <p>Ex: (A <= B) is true.</p>

Logical operators

Sr.No	Operator and Description
1	&& (Logical AND) If both the operands are non-zero, then the condition becomes true. Ex: (A && B) is true.
2	 (Logical OR) If any of the two operands are non-zero, then the condition becomes true. Ex: (A B) is true.
3	! (Logical NOT) Reverses the logical state of its operand. If a condition is true, then the Logical NOT operator will make it false. Ex: ! (A && B) is false.

Assignment operators

Sr.No	Operator and Description
1	<p>= (Simple Assignment)</p> <p>Assigns values from the right side operand to the left side operand</p> <p>Ex: $C = A + B$ will assign the value of $A + B$ into C</p>
2	<p>+= (Add and Assignment)</p> <p>It adds the right operand to the left operand and assigns the result to the left operand.</p> <p>Ex: $C += A$ is equivalent to $C = C + A$</p>
3	<p>-= (Subtract and Assignment)</p> <p>It subtracts the right operand from the left operand and assigns the result to the left operand.</p> <p>Ex: $C -= A$ is equivalent to $C = C - A$</p>
4	<p>*= (Multiply and Assignment)</p> <p>It multiplies the right operand with the left operand and assigns the result to the left operand.</p> <p>Ex: $C *= A$ is equivalent to $C = C * A$</p>
5	<p>/= (Divide and Assignment)</p> <p>It divides the left operand with the right operand and assigns the result to the left operand.</p> <p>Ex: $C /= A$ is equivalent to $C = C / A$</p>
6	<p>%= (Modules and Assignment)</p> <p>It takes modulus using two operands and assigns the result to the left operand.</p> <p>Ex: $C \% = A$ is equivalent to $C = C \% A$</p>

typeof operators

Type	String Returned by typeof
Number	"number"
String	"string"
Boolean	"boolean"
Object	"object"
Function	"function"
Undefined	"undefined"
Null	"object"

Precedence rules

- Same as Java
- Use brackets to clarify meaning of complicated expressions

a += b * 2 + ++c * 6

++ performed 1st (group 1)

* performed 2nd (group 2)

+ performed 3rd (group 3)

+= performed last (group 4)

Group	Operators
1	++ --
2	* / %
3	+ -
4	= += -= *= /=

a += ((b * 2) + ((++c) * 6))

- As with Java, there is:
 - if..else
 - switch
 - for
 - while
 - do..while

The if statement

```
var x = prompt('x?');  
if (isNaN(x)) {  
    document.write('this is not a number');  
}  
else if (x<0) {  
    document.write('x is negative');  
}  
else if (x>0) {  
    document.write('x is positive');  
}  
else {  
    document.write('x is zero');  
}
```

Complex conditions

- Similar to Java

```
var x = prompt('x?');  
  
if (x>1 && x<10) {  
    // x is between 1 and 10  
    if (x>5) {  
        // x is between 5 and 10  
        alert("your number is bigger than 5");  
    }  
    else  
        alert("your number is smaller than 5")  
}  
  
else if (isNaN(x)) {  
    alert('this is not a number');  
}  
  
else {  
    alert('your number is either negative or bigger than 10');  
}
```

for loop

```
for (i = 1; i <= 6; i++) {  
    document.write("<h" + i + ">This is heading " + i);  
    document.write("</h" + i + ">");  
}
```

while loop

- While loop: repeat get a number until !isNaN

```
var x = prompt('x?');  
while (isNaN(x)) {  
    alert('not a number, try again');  
    x = prompt('x?');  
}  
document.write('<p>Number is '+x+'</p>');
```

Arrays

- Allow to store multiple values in a single variable

```
var days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'];  
var playList = [];  
var playList2 = new Array();  
var colours = new Array('red', 'green', 'blue');
```

- An array can also contain unrelated items

```
var preferences = [1, 42, 'www.dcs.shef.ac.uk', true];
```

- Arrays can be nested

```
var questions = [  
    ['How many moons does Earth have?', 1],  
    ['How many moons does Saturn have?', 61]  
];
```

McFarland, D.S, JavaScript: the missing manual, 2nd edition, O'Reilly, 2009

Accessing items in an array

- Arrays are zero-indexed:

```
var days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'];  
alert(days[0]); // Mon
```

- Change the value in an array:

```
days[0] = 'Monday';
```

- days.length returns the length of the array:

```
var days = ['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'];  
var i = days.length-1;  
alert(days[i]); // Sun
```

Adding and removing items from an array

- push – add items on the end

```
var properties = ['red', '14px', 'Arial'];  
properties[properties.length] = 'bold';  
                                // red, 14px, Arial, bold  
properties.push('italic', 'underlined');  
                                // red, 14px, Arial, bold, italic,  
underlined
```

- unshift – add items to beginning

```
var properties = ['red', '14px', 'Arial'];  
properties.unshift('bold');    // bold, red, 14px, Arial
```

- push and unshift return number of items in resulting array

```
var p = [0, 1, 2, 3];  
var numItems = p.push(4, 5);    // numItems = 6
```


Adding and removing items from an array

- `pop()` removes the last item from the array
- `shift()` removes the first item from the array

```
var p = [0,1,2,3];  
var removedItem = p.pop();    // removedItem = 3  
                               // p = [0,1,2];
```

- `splice()` can be used to both add and delete anywhere in an array

```
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
fruits.splice(2,0,"Lemon","Kiwi");
```

Position
to operate at

How many items to remove.
0 means do not remove anything

Elements to be
added (optional)

8. Functions

- Turn useful code into reusable commands

```
<script type="text/javascript">

    function printStars() {
        document.write('<p>*****</p>');
    }

    printStars();

</script>
```

8.1 Functions and parameters

Parameters do not have a declared type!

```
<script type="text/javascript">
  function print(message) {
    document.write('<p>'+message+'</p>');
  }
  print('Hello world');
</script>
```

```
<script type="text/javascript">
  function printStars(n) {
    document.write('<p>');
    for (var i=0; i<n; i++)
      document.write('*');
    document.write('</p>');
  }
  for (var i=1; i<=10; i++)
    printStars(i);
</script>
```

- Javascript can output data in different ways
 - Writing into an HTML element, using **innerHTML**.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Home page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script language="javascript" type="text/javascript">
  document.getElementById("demo").innerHTML = 5+6;
</script>

</body>
</html>
```

- Javascript can output data in different ways
 - Writing into the HTML output using **document.write()**.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Home page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script language="javascript" type="text/javascript">

  document.write("<p> This is a new paragraph</p>")
</script>

</body>
</html>
```

- Javascript can output data in different ways
 - Writing into an alert box, using **window.alert()**.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Home page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script language="javascript" type="text/javascript">

  alert("Hello!")
</script>

</body>
</html>
```

- Javascript can output data in different ways
 - Writing into the browser console, using **console.log()**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Home page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script language="javascript" type="text/javascript">

  var x =1;
  var y = 3;

  console.log("debugging " + (x+y))
</script>

</body>
</html>
```

innerHTML

- Not an official part of the DOM, but available in all modern browsers
- Sets or returns the HTML content (inner HTML) of an element

```
<p id="demo"></p>
```

```
<script language="javascript" type="text/javascript">  
    document.getElementById("demo").innerHTML = 5+6;  
</script>
```


Javascript Objects and Event Handling

Prof Fabio Ciravegna
University of Sheffield

Slides by Dr Steve Maddock

3. Objects in Java

16

- **Java** is an object-oriented language
- Uses classes and objects
- To refer to methods of the object, use `objectName.methodName`
- For *public* properties of the object, use `objectName.propertyName`
- For *private* properties of the object,
 - Define a getter/setter method
 - use `objectName.methodName`

```
Meal curry = new Meal();           // curry is an instance of the class Meal
curry.setPrice(3.99);               // setPrice is a method of Meal
int cost = curry.getPrice();        // Get the price of the Meal curry and
                                     // store it in the variable subtotal

Person p = new Person();            // p is an instance of the class Person
p.setAge(18);                       // setAge is a method of Person
int age = p.getAge();               // Get the age of the Person p and store it
                                     // in the variable age
```

Objects in JavaScript

- JavaScript is an 'object oriented programming language'
 - does not use classes, but does have objects
- dot notation is used to refer to methods *and* properties
 - object.property and object.method()
 - can also use object['property'] and object['method']()

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

Creating objects

```
<script language="javascript" type="text/javascript">

function increaseAge() {
return this.age+=1;
}

var personObj = new Object();
personObj.firstname = "John";           // add a property
personObj.lastname = "Smith";
personObj.age = 42;
personObj.increaseAge = increaseAge; // add a method

//document.write(JSON.stringify(personObj));
document.write("<br \\/>");
document.write(personObj.age);
document.write("<br \\/>");
personObj.increaseAge();
document.write(personObj.age);

</script>
```

Objects can be created
and methods and
properties can be added

Note there are no () for
increaseAge

A method is a function
attached to an object

An object literal

```
personObj = {firstname:"John",lastname:"Smith",age:
43,increaseAge:increaseAge};
```

Creating objects

```
function computearea() {  
    var area = this.width*this.height;  
    return area;  
}  
  
function Rectangle(w, h) {  
    this.width = w;  
    this.height = h;  
    this.area = computearea;  
}  
  
var rect1 = new Rectangle(2,3);  
document.write("<p>rect1 area = " + rect1.area() + "<\n/p>");
```

Creating objects

```
function computearea() {  
    var area = this.width*this.height;  
    return area;  
}  
  
function Rectangle(w, h) {  
    this.width = w;  
    this.height = h;  
    this.area = computearea;  
}  
  
var rect1 = new Rectangle(2,3);  
var rect2 = new Rectangle(2,4);  
document.write("<p>rect1 area = " + rect1.area() + "</p>");  
document.write("<p>rect2 area = " + rect2.area() + "</p>");
```

Creating objects

```
function computeArea() {  
    var area = this.width*this.height;    return area;  
}  
  
function getPerimeter() { return (this.width+this.height)*2; }  
  
function Rectangle(w, h) {  
    this.width = w;    this.height = h;    this.area = computeArea;  
}  
  
var rect1 = new Rectangle(2,3);  
var rect2 = new Rectangle(2,4);  
document.write("<p>rect1 area = " + rect1.area() + "</p>");  
document.write("<p>rect2 area = " + rect2.area() + "</p>");
```

```
rect1.perimeter = getPerimeter;
```

this will create a property perimeter in rect1 only!

```
document.write("<p>rect1 perimeter = " + rect1.getPerimeter() + "</p>");  
document.write("<p>rect2 perimeter = " + rect2.getPerimeter() + "</p>");
```

No output for last line
rect2.perimeter does
not exist

Creating objects – prototype

- All JavaScript objects inherit properties and methods from a prototype
- Prototypes allow you to add
 - properties to a constructor
 - New method objects to a constructor
- You can modify only your own prototypes
 - Do not modify pre-build javascript objects
 - i.e. date

Creating objects – prototype

```
function computearea() {  
    var area = this.width*this.height;  return area;  
}  
  
function Rectangle(w, h) {  
    this.width = w;  this.height = h;  this.area = computearea;  
}  
  
function getPerimeter() { return (this.width+this.height)*2; }  
  
var rect1 = new Rectangle(2,3);  
var rect2 = new Rectangle(2,4);  
document.write("<p>rect1 area = " + rect1.area() + "<\p>");  
document.write("<p>rect2 area = " + rect2.area() + "<\p>");  
  
Rectangle.prototype.perimeter = getPerimeter;  
  
document.write("<p>rect1 perimeter = " + rect1.getPerimeter() + "<\p>");  
document.write("<p>rect2 perimeter = " + rect2.getPerimeter() + "<\p>");
```

**rect2.perimeter now
exists because we have
modified the prototype**

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
```

Functions are objects

- Functions may be passed to other functions
- The `sort()` method (of the Array object) will sort the elements alphabetically by default.

```
function cmp(x,y) {
    return x < y ? -1 : x == y ? 0 : 1;
}

function fillArray(data) {
    for (var i=0; i<10; i++) {
        number = Math.floor(Math.random()*10);
        data.push(number)
    }
}

function displayArray(data) {
    document.write(data);
    document.write("<br \\/>");
}

var data = [];
fillArray(data);
document.write("unsorted: ");
displayArray(data);
data.sort(cmp);
document.write("sorted: ");
displayArray(data);
```

Built-in objects in JavaScript

- Array, Boolean, Date, Error, Function, Math, Number, Object, RegExp, String
- The Date object methods
 - getDate, getDay, getHours, ..., getTime, ..., setDate, setHours, ...
- String
 - charAt, substring, toLowerCase, ...
- Array
 - join, sort, ...
- There are also objects that relate to the DOM hierarchy and to the browser – see later

Javascript Date

- Date object lets you work with dates
- Format: Wed Feb 28 2018 11:25:52 GMT+0000 (GMT)

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Home page</title>
  <link rel="stylesheet" href="mystyle.css">
</head>
<body>

<h1>My First Web Page</h1>
<p>My First Paragraph</p>

<p id="demo"></p>

<script language="javascript" type="text/javascript">
  var d = new Date();
  document.getElementById("demo").innerHTML = d;
</script>

</body>
</html>
```

Javascript Date

- Watch out for some of the methods?
- What does `getTime` return?



The
University
Of
Sheffield.

Event Handling

Event handling

- Dealing with user interaction on a web page
- Some example events:
 - Loading a document
 - Clicking a button or other form control
 - Browser screen changing size
- A function/script is assigned to an event handler (element property)
 - E.g. `element.onclick = clickHandler;`
 - `clickHandler` can be some JavaScript code, e.g. a function call
- There are lots of event handlers for different HTML elements
 - All these properties begin with '**on**'

Event handling

- A function/script is assigned to an event handler, which is an element property
- The system automatically calls the relevant handler when the user interacts with the element

Event Name	Meaning	Restrictions
load	All the content of a page has been loaded.	body element only
unload	The document has been removed from the window	
click	The mouse was clicked with the cursor over the element.	
dblclick	The mouse was double-clicked with the cursor over the element.	Not a DOM event
mousedown	The mouse button was pressed with the cursor over the element.	
mouseup	The mouse button was released with the cursor over the element.	
mouseover	The cursor was moved onto the element.	
mousemove	The cursor was moved while it was over the element.	
mouseout	The cursor was moved away from the element.	
focus	The element has received the focus, i.e. it will accept input.	a, area and form control elements
blur	The element has lost the focus.	
keypress	A key was pressed and released while the cursor was over the element.	Not a DOM event
keydown	A key was pressed while the cursor was over the element.	
keyup	A key was released while the cursor was over the element.	
submit	The form was submitted.	form elements only
reset	The form was reset.	
select	The user selected some text in a text field.	input and textarea elements only
change	The element has lost the focus and its value has been modified since it received the focus.	input, select and textarea elements only



To create an event Handler add “on” in Front of the event name
click -> onclick

Example 1

```
<head>...
<script type="text/javascript">
    function printDateInfo() {
        var now = new Date();
        alert(now);
    }
</script>
</head>

<body>

<form name="myform">
<p><input type="button"
    name="dateinfo" id="dateinfo"
    value="Date info"
    onclick="printDateInfo();" /></p>
</form>

</body>
</html>
```

Date info

Thu Nov 10 2011 14:28:47 GMT+0000 (GMT Standard Time)

OK

Note: this form has no method or action because it has no submit button. The action is calling the JS method when pressing the button

- When button is clicked the event handler is called

The Document Object Model

- “An API that defines how to access the objects that compose a document”

David Flanagan, “JavaScript: The Definitive Guide”, 5th edition, O’Reilly, 2006
- We’ve briefly looked at the “Level 0” DOM (the legacy DOM)
 - Provides a document object
 - Also includes legacy properties,
 - e.g. `document.write(document.lastModified);`
- Now: The W3C DOM standard
 - Defines a Document API (for any XML document)
 - Defines a specialized HTMLDocument API (includes most of the features of the Level 0 DOM)
 - Rather than use `document.write()`, use W3C DOM to insert content into any part of the document, even after it has been parsed

A hierarchy

- The structure of a document's element hierarchy is modelled as a tree
 - Indentation shows the levels→
- This contains **element nodes**
 - h1, p, ...
 - Shown in **blue**→
- And **text nodes**
 - 'A list', 'one', etc
 - Shown in **orange**→

```
html
.. head
... sub-tree for head contents
.. body
... text[\n]
... h1
... text[A list]
... text[\n]
... ul
... text[\n]
... li
... text[one]
... text[\n]
... li
... text[two]
... text[\n]
... text[\n]
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <title>test</title>
</head>
<body>
<h1>A list</h1>
<ul>
<li>one</li>
<li>two</li>
</ul>
</body>
</html>
```

A list

- one
- two

The document object

- The *document object* represents the entire document. Its methods include:
- createElement
 - make new element objects
- createTextNode
 - make new text objects
- getElementById
 - access individual element objects
- getElementsByTagName
 - access all elements of a specific **type**.

Object	Property or Method	Meaning
document	createElement(n)	Make a new Element object of type n.
	createTextNode(t)	Make a new text node with the text t.
	getElementsByTagName(n)	Find all the Element objects of type n in the document and return them in a NodeList.
Element	getAttribute(n)	Return the string value of this element's attribute with name n.
	setAttribute(n, v)	Set the value of this element's attribute with name n to v.
	getElementsByTagName(n)	Find all the Element objects of type n in this element's children and return them in a NodeList.
	appendChild()	Add the Element e to this element's children after all its existing children.
	insertBefore(e, c)	Add the Element e to this element's children, before the child c.
	removeChild(c)	Remove the Element c from among this element's children.
	replaceChild(c, d)	Replace the Element's child d with c.
	firstChild	This element's first child Element.
	lastChild	This element's last child Element.
	previousSibling	The Element to the left of this element in the tree.
	nextSibling	The Element to the right of this element in the tree.
	parentNode	The Element above this element in the tree.
	childNodes	A NodeList containing all this element's children.
	nodeType	Always equal to 1.
NodeList	item(i)	Get the object at position i in the list.
	length	The number of items in the list.
Text	data	The string of text.
	length	The number of characters in the text.
	nodeType	Always equal to 3.

Element objects

- *Element* objects correspond to the element nodes.
- Methods include:
 - `getAttribute`
 - `setAttribute`
 - `appendChild`
- Properties include:
 - `childNodes`, which holds a `NodeList` object;
 - `firstChild`
 - `lastChild`

Object	Property or Method	Meaning
Element	<code>getAttribute(n)</code>	Return the string value of this element's attribute with name <code>n</code> .
	<code>setAttribute(n, v)</code>	Set the value of this element's attribute with name <code>n</code> to <code>v</code> .
	<code>getElementsByTagName(n)</code>	Find all the Element objects of type <code>n</code> in this element's children and return them in a <code>NodeList</code> .
	<code>appendChild(e)</code>	Add the Element <code>e</code> to this element's children after all its existing children.
	<code>insertBefore(e, c)</code>	Add the Element <code>e</code> to this element's children, before the child <code>c</code> .
	<code>removeChild(c)</code>	Remove the Element <code>c</code> from among this element's children.
	<code>replaceChild(c, d)</code>	Replace the Element's child <code>d</code> with <code>c</code> .
	<code>firstChild</code>	This element's first child Element.
	<code>lastChild</code>	This element's last child Element.
	<code>previousSibling</code>	The Element to the left of this element in the tree.
	<code>nextSibling</code>	The Element to the right of this element in the tree.
	<code>parentNode</code>	The Element above this element in the tree.
	<code>childNodes</code>	A <code>NodeList</code> containing all this element's children.
	<code>nodeType</code>	Always equal to 1.

Principal DOM objects, from Chapman and Chapman, 06

Functions to reach an element in the page

Let's look at the following two methods for the document object

- `getElementById("elementID")`
 - returns the element with the id elementID as an object
- `getElementsByTagName("tag")`
 - returns all elements with the name tag as an array
- Example:

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</body>
```

getElementById('elementID')

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
```

```
<script>
  var mylist = document.getElementById("mylist");
  document.write(mylist.innerHTML);
</script>
</body>
```

- returns the element with the id elementID as an object
- Elements labelled with an id are easy to access and are very useful in conjunction with JavaScript
- innerHTML gives us the content of the tag

getElementsByTagName('tag')

```
<body>
<ul id="mylist">
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
```

- `getElementsByTagName('tag')`
 - returns **all elements** with the name **tag** as an array of elements
- Here, all the li tags are returned.
- `innerHTML` gives us the content of the tag

```
<script>
var listItems = document.getElementsByTagName('li');

for (var i=0; i<listItems.length; i++) {
    alert(listItems[i].innerHTML);
}
;

</script>
</body>
```

More examples

- `document.getElementById('navigation').getElementsByTagName('a')[3];`
- `// returns the fourth link inside the element with the ID 'navigation'`

- `var e = document.getElementById('navigation');`
- `var fourthLink = e.getElementsByTagName('a')[3];`
- `// returns the fourth link inside the element with the ID 'navigation'`

- `document.getElementsByTagName('div')[2].getElementsByTagName('p')[0];`
- `// returns the first paragraph inside the third div in the document.`

<http://www.onlinetools.org/articles/unobtrusivejavascript/>

Navigate from a certain element

- `childNodes`
 - returns an array of all the nodes inside the current one.
 - There is also `firstChild` and `lastChild`, which are shorter versions for `childNodes[0]` and `childNodes[childNodes.length-1]`.
- `parentNode`
 - The element containing this one
- `nextSibling`
 - the next element on the same level in the document tree
- `previousSibling`
 - the previous element on the same level in the document tree

childNodes

- returns an array of all the nodes inside the current one
- Use array notation or method 'item' to access each child node

```

<body>
<div id="mylist">
<ul>
<li>one</li>
<li>two</li>
<li>three</li>
</ul>
</div>
<script>
    var mylist = document.getElementById("mylist");
    document.write(mylist);           // [object HTMLDivElement]
    document.write("<br \\/>");
    document.write(mylist.firstChild.innerHTML);    // [object
Text] \n
    document.write(mylist.childNodes[0]); // [object Text] \n
    var nodes = mylist.childNodes;
    document.write(nodes.item(0));        // [object Text] \n
    document.write(nodes.item(1));        // [object HTMLUListElement]
ul
    document.write(nodes[2]);             // [object Text] \n
</script>

```

Attributes and functions for elements

- In the HTML DOM, every **Element** object has properties corresponding to the element's **attributes**

attributes	returns an array of all the attributes of this element
data	returns or sets the textual data of the node
nodeName	returns the name of the node (the HTML element name)
nodeType	returns the type of the node — 1 is an element node, 2 attribute and 3 text
nodeValue	returns or sets the value of the node. This value is the text when the node is a textnode, the attribute if it is an attribute or null if it is an element
getAttribute(attribute)	returns the value of the attribute

<http://www.onlinetools.org/articles/unobtrusivejavascript/>

Example

```

<ul id="mylist">
  <li>Guns n' Roses</li>
  <li>Nirvana</li>
  <li id="3">Stone Roses</li>
  <li>Metallica</li>
</ul>
</div>

<script language="javascript" type="text/javascript">
  var mylist = document.getElementById("mylist");
  var listItems = mylist.getElementsByTagName('li');

  document.write(mylist);
  document.write("<br \\/>");

  for (var i=0; i<listItems.length; i++) {
    var e = listItems[i];
    document.write(e.nodeType+" "+e.nodeName);
    document.write(" " + e.getAttribute("id"));
    document.write("<br \\/>");
  }

</script>

```

childNodes - access value

```
<div id="content">
  <h2>Welcome</h2>
  <p>Here is a list of my favourite bands</p>

  <ul id="mylist">
    <li>Guns n' Roses</li>
    <li>Nirvana</li>
    <li>Stone Roses</li>
    <li>Metallica</li>
  </ul>
</div>
```

- Use nodeValue

```
<script language="javascript" type="text/javascript">
  var mylist = document.getElementById("mylist");
  var listitem = mylist.getElementsByTagName('li')[0];
  document.write(mylist);
  document.write("<br \\/>");
  document.write(listitem.firstChild.nodeValue);

</script>
```

More examples

- `var other = document.getElementById('nav').childNodes[3].firstChild;`
- `// returns the 4th element's first sub element inside the element with the ID nav`
- `var prevlink = o.parentNode.previousSibling.firstChild.childNodes[2];`
- `// returns the third node inside the previous element`
- `// that is on the same level as the parent element of o`

<http://www.onlinetools.org/articles/unobtrusivejavascript/>

Creating new content

- `createElement(element)` – creates a new element
- `createTextNode(string)` – creates a new text node with the value string

```
<head>...  
  <script>  
    ...  
  </script>  
</head>  
  
<body>  
<h1>Results</h1>  
<div id="results">  
<p>New text will be added after this...</p>  
</div>  
<h1>Some other heading</h1>  
</body>
```



Insert new text
here using
JavaScript

Creating new content

```
<div id="content">
  <h2>Welcome</h2>
  <p>Here is a list of my favourite bands</p>

  <ul id="mylist">
    <li>Guns n' Roses</li>
    <li>Nirvana</li>
    <li id="3">Stone Roses</li>
    <li>Metallica</li>
  </ul>
</div>
<script language="javascript" type="text/javascript">

var newItem = null;

function updateResults() {
  newItem = document.getElementById("mylist");
  var element = document.createElement("li");
  var str = "Thirty seconds to Mars";
  var textNode = document.createTextNode(str);
  element.appendChild(textNode);
  newItem.appendChild(element);
}

if (document.getElementById) window.onload=updateResults;

</script>
```

Note! No
brackets in
“”



Altering the existing content

<code>setAttribute(attribute,value)</code>	Adds a new attribute with the value to the object
<code>appendChild(child)</code>	Adds child as a childNode to the object. child needs to be an object, you cannot use a string.
<code>cloneNode()</code>	Copies the whole node with all childNodes.
<code>hasChildNodes()</code>	Checks if an object has childNodes, and returns true if that is the case.
<code>insertBefore(newchild,oldchild)</code>	Adds newchild before oldchild to the document tree.
<code>removeChild(oldchild)</code>	Removes the childnode oldchild.
<code>replaceChild(newchild,oldchild)</code>	Replaces oldchild with newchild.
<code>removeAttribute(attribute)</code>	Removes the attribute from the object.

<http://www.onlinetools.org/articles/unobtrusivejavascript/>

4. Unobtrusive JavaScript

<http://www.onlinetools.org/articles/unobtrusivejavascript/chapter1.html>

1. Never, under any circumstances, add JavaScript directly to the document (always use a separate file)
 - `<script src="scripts.js"></script>`
2. JavaScript is an enhancement, not a secure functionality
3. Check the availability of an object before accessing it
4. Create JavaScript, not browser specific dialects
5. Don't hijack other script's variables
6. Keep effects mouse independent

Summary

- The Web browser provides a programming environment with the following features:
 - A global window object
 - A client-side object hierarchy
 - An event-driven programming model
- The DOM contains many methods to manipulate the HTML document
- We are aiming for unobtrusive JavaScript
- Further reading: https://developer.mozilla.org/en/Gecko_DOM_Reference
- *Next lecture:* Graphics on the Web



The
University
Of
Sheffield.

78

Forms

Forms

- Forms make it possible for Web sites to collect information from their visitors
 - e.g. order form on Amazon
- Pre HTML5:
 - Can use JavaScript to check and process the data entered – client side
 - Can validate data on the Web server – server side
- HTML5:
 - Validation inbuilt for common data types

Product Care Programme

If you have a problem with any Desperate Software product, please fill in this form below. Fields marked with a * are required.

Your name: *

Your email address: *

Country of residence:

Telephone number:

Customer ID number: *

Please describe your problem as clearly as possible. *

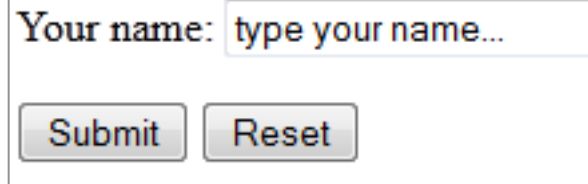
Form

- **Element:** `<form>...</form>`

```
<form name="myName" action="howToHandleTheData"  
      method="howToSendTheData">  
  ...form contents...  
</form>
```

- The **action** attribute is the URL of the method or program which will process the data
- The **method** attribute determines how data is sent to the server:
 - get – use when processing of data has no side-effects
 - post – used if there are side-effects, e.g. update a database

Example 1



```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="get">
<input type="hidden" name="recipient" value="s.maddock@dcs.shef.ac.uk" />
<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your name..."
maxlength="40" size="20" onclick="this.value=''" />
</p>
<p>
<input type="submit" name="submit" id="submit" value="Submit" />
<input type="reset" />
</p>
</form>
```

Before we continue and discuss how to execute code when the form is submitted we need to see some parts of this code

Aside: Type 'hidden'

- Input elements with the type set to hidden are useful in two ways:
- Information not explicitly entered by the user can be included in the data sent to the server
 - Could be used to identify the page the user is on or the product described on that page
- Can be used by the server to save data in a page that it sends to a user, so that it can subsequently be sent back when a form is submitted
 - Keeps track of a user during a session

Example 1

Your name:

Submit

Reset

```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="post">
```

```
<input type="hidden" name="s.maddock"
value="s.maddock" />
```

- Each form input should have a label
- The for attribute of the <label> tag should be equal to the id attribute of the related element to bind them together

```
<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your
name..." maxlength="40" size="20" onclick="this.value=''" />
</p>
```

```
<p>
<input type="submit" name="submit" id="submit" value="Submit"
/>
<input type="reset" />
</p>
</form>
```

Example 1

Your name:

- There are a number of different input elements
- Both 'name' and 'id' attributes should be supplied and set to the same value
 - name is used in submission of form
 - id can be used to find the specific field in the form (e.g. with a #)

```
<p>
<label for="from">Your name:</label>
<input type="text" name="from" id="from" value="type your
name..." maxlength="40" size="20" onclick="this.value=''" />
</p>
```

- The 'value' attribute defines the default value for the input field
- There are also attributes that respond to user interaction, i.e. event handlers, e.g. onclick – we'll revisit this later

Example 1

Your name:

```
<form name="myform"
      action="http://www.dcs.shef.ac.uk/cgi-bin/FormMail.pl"
      method="get">
```

```
<input type="hidden" name="recipient"
value="s.maddock@dcs.shef.ac.uk" />
```

```
<p>
```

```
<label for="f
```

```
<input type="t
```

```
name..." maxle
```

```
</p>
```

```
<p>
```

```
<input type="submit" name="submit" id="submit" value="Submit"
/>
```

```
<input type="reset" />
```

```
</p>
```

```
</form>
```

- Three kinds of button can be created
 - submit button – submits the form
 - reset button – resets all controls to their initial values
 - push button – attach a client-side script to these

Example 2: A search form

- A radio button set share the same name so that only one of them can be 'on'
- The initial one that is 'on' is indicated by the attribute 'checked'
 - `checked="checked"` or just `checked`

```
<form action="http://www.google.com/search" method="get">
  <input type="text" name="q" size="31" maxlength="255" value="" />
  <input type="submit" value="Google Search" /><br/>
  <input type="radio" name="sitesearch" value="" />Search WWW<br/>
  <input type="radio" name="sitesearch"
    value="http://staffwww.dcs.shef.ac.uk/people/S.Maddock"
    checked />Search dcs.shef.ac.uk<br />
</form>
```

the University of Sheffield. If you are visiting this department for the first time you may find it more help information in the sections below, some links are restricted to local users only.

Google™

☒ Search WWW ☐ Search dcs.shef.ac.uk

Same name so
it is same field

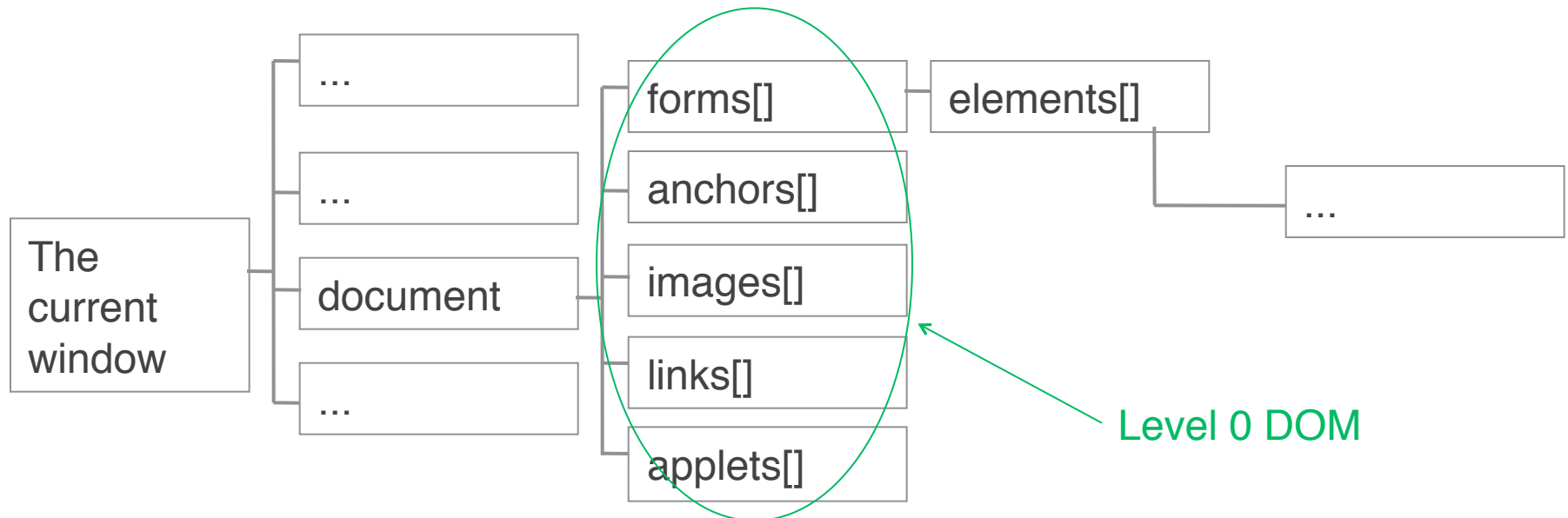
The Window object

```
var answer = 42;  
window.answer = 42;
```

- The following are equivalent:
- As another example, the first element in the first form on a web page can be accessed using:

```
window.document.forms[0].elements[0]
```

- Can also use name attribute to access a form element – see later



Example

Example: Room area

Rectangle length in cm?

Rectangle width in cm?

Rectangle Area

```
<form id="calcarea" name="calcarea" onsubmit="area(); return false;">
<label for="mylength">Rectangle length in cm?</label>
<input type="text" id="mylength" name="mylength" required="required"
>
<br />
<label for="mywidth">Rectangle width in cm?</label>
<input type="text" id="mywidth" name="mywidth" required="required" />
<br />
<input type="submit" value="Calculate" />
<label for="output">Rectangle Area</label>
<textarea rows="1" cols="10" id="output" name="output" >result here</
textarea>
</form>
```

Before we continue and discuss how to execute code when the form is submitted we need to see some parts of this code

Example

```
<head>...
<script>
  function area() {
    var length
      = document.forms["calcarea"].elements["mylength"].value;

    var width = document.calcarea.mywidth.value;

    document.forms["calcarea"]["output"].value=length*width;
  }
</script>
</head>
```

Example: Room area

Rectangle length in cm?

Rectangle width in cm?

Rectangle Area

More frequently however

- you will assign an id to the field
 - `<textarea rows="1" cols="10" id="output" name="output" >result here</textarea>`
- and you will identify the element directly
 - `var elem=document.getElementById("output");`
 - `elem.innerHTML= "whatever";`

Summary

- Functions turn useful code into reusable commands
- JavaScript is an ‘object oriented programming language’
 - does not use classes, but does have objects
- Forms make it possible for Web sites to collect information from their visitors
- The Web browser provides a programming environment with the following features:
 - A global Window object
 - A client-side object hierarchy – the DOM
 - An event-driven programming model
- Next lecture: Events, the DOM



The
University
Of
Sheffield.

92

Bootstrap

Designing for multiple architectures

- As we said in the first lecture, mobile phones are the most used device for browsing the Web
- All websites *must* be designed for both computers AND mobiles
 - For all operating systems as well!
- This means
 - Strict use of standards (e.g. HTML5)
 - Test and display on all architectures
- A good way of doing it is to adopt libraries that allow this flexibility as a built in property
 - One of them is Bootstrap

Aww yash, Bootstrap 4 is coming!

Bootstrap

Getting started

CSS

Components

JavaScript

Customize

Themes

Expo

Blog



Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web.

Download Bootstrap

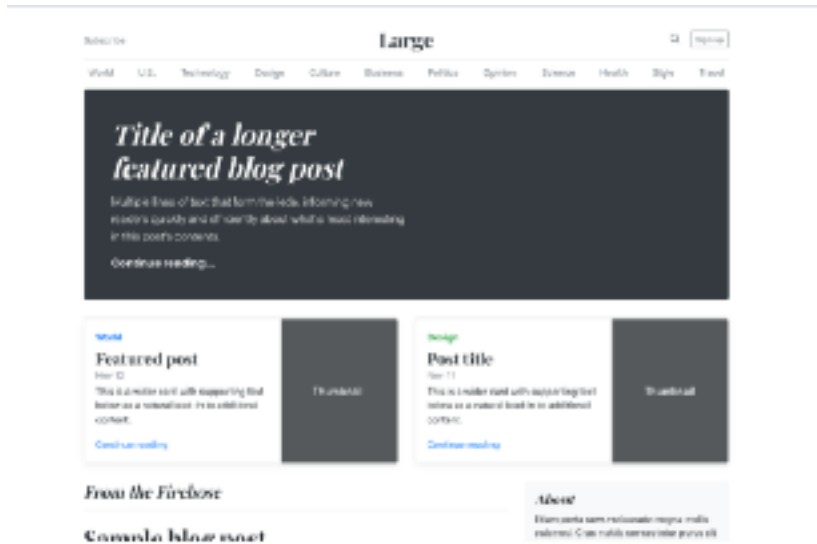
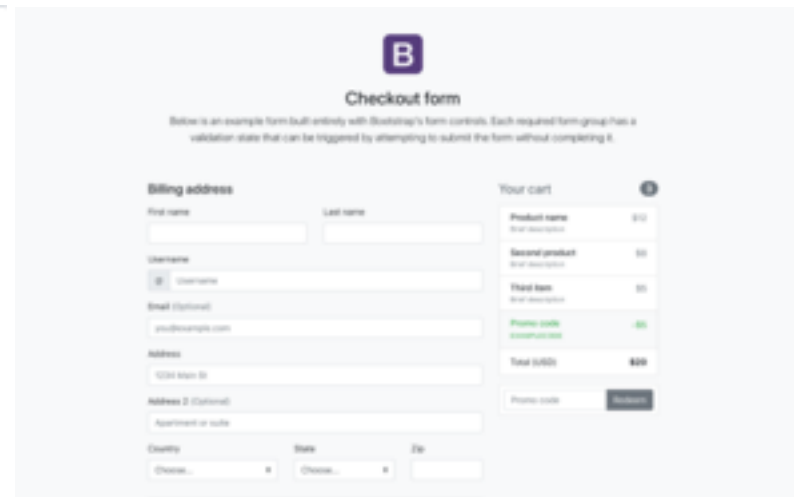
Currently v3.3.8

- <http://getbootstrap.com/>

What is Bootstrap?

- Bootstrap is a layer over CSS and Javascript
 - It defines a set of pre-defined templates that will behave seamlessly in both mobile and desktop environments.
- It contains templates for
 - typography
 - forms
 - Buttons
 - navigation
 - other interface components
 - JavaScript extensions
- Open source

What is Bootstrap?

This image shows a Bootstrap checkout form. It includes a header with a logo and a title. Below the header is a form with two main sections: 'Billing address' and 'Your cart'. The 'Billing address' section contains fields for first name, last name, username, email (optional), address, address 2 (optional), country, state, and zip. The 'Your cart' section displays a table with columns for product name, price, and quantity. It lists three items: 'Product name', 'Second product', and 'Third item'. The total price is shown as '\$20'. There is a 'Proceed to checkout' button at the bottom of the form.

