

CT214H – Web Programming Fundamentals

Chapter 3

JavaScript

Tran Cong An
(tcan@cit.ctu.edu.vn)

Content

1. Introduction
2. Using JavaScript in an HTML documents
3. JS language basics
4. JS functions
5. DOM
6. OOP in JS
7. Appendix

Introduction to JavaScript

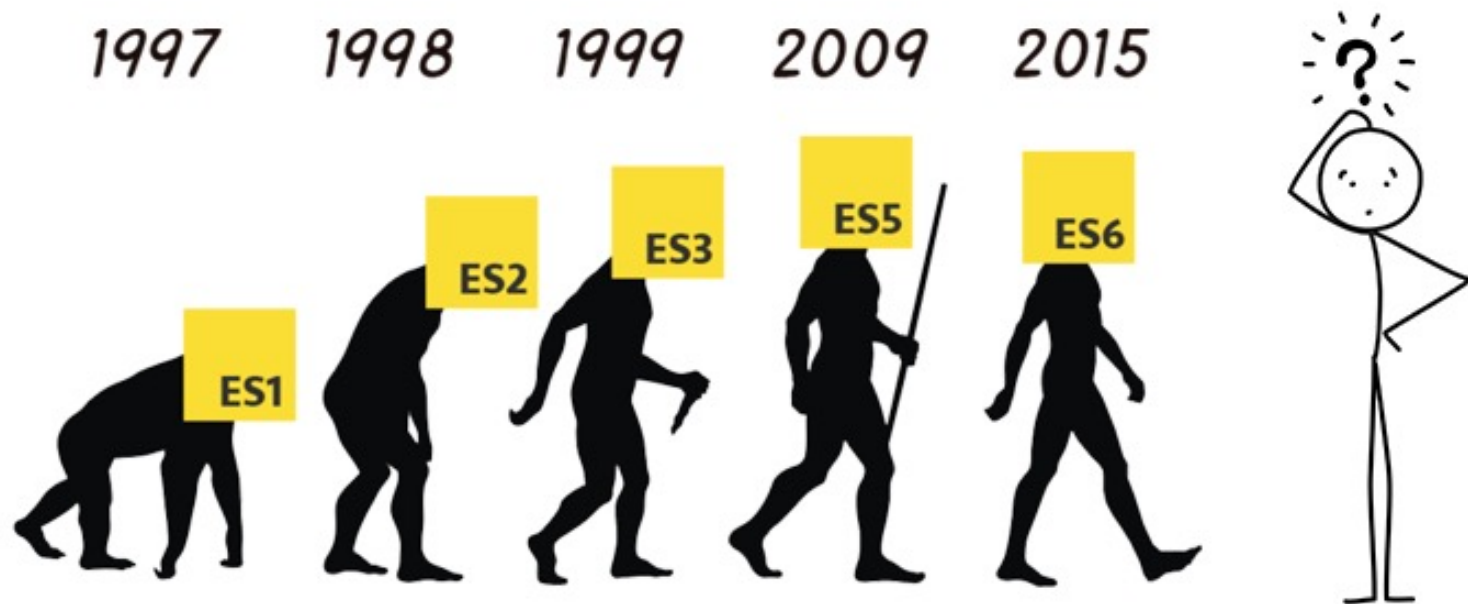
What is JavaScript?

- JS is a programming language (no relation to Java, it is named just for marketing reason 😊)
- Currently is **the only programming language for browsers**
- History:
 - Originally developed by Brendan Eich at Netscape (named LiveScript at that time)
 - Became a joint venture of Netscape and Sun in 1995 (renamed JavaScript)
 - Now standardized by the ECMA (ECMA-262, also ISO 16262)

ECMA (European Computer Manufacturers Association) is an industry association dedicated to the standardization of information and communication systems

What is JavaScript?

- ECMAScript (ES): a **standard** for scripting languages
- JavaScript: a programming language based on ECMAScript (the most popular implementation of ES)



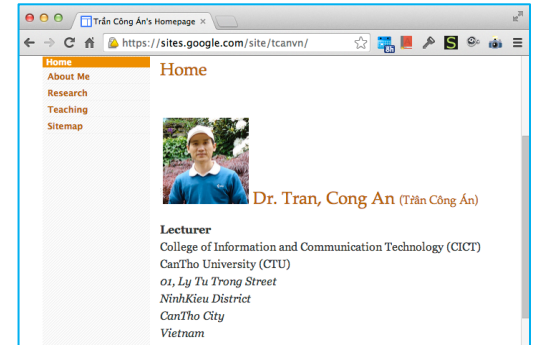
Why JavaScript?



+



produces



Describes the **content and structure** of the page

Describes the **appearance and style** of the page

A web page... that doesn't do anything, except:

- **Shows** the information
- **Links** to other webpages

- If we want **dynamic**, more **friendly** and **interactive** webpages (automation, animation, interactivity, etc.)

JavaScript comes to help!

What JS Can Do?

- User **interaction** through form are easy
- Document Object Model (DOM) makes it possible to create **dynamic HTML** with JS:
 - can change the HTML page content
 - can show/hide HTML page elements
 - can change the HTML element's attributes
 - etc.
- And much **more features** that we can do with **event-driven** computation...

Basic Features of JS

- An **interpreter** language (no compilation by developers, it is compiled and executed on-the-fly by browser)
- **Dynamic typed** (no need to declare variables, auto type-casting)
- **Supports the OO approach** (but lack of some basic features of OO such as polymorphism)

Using JS in HTML Documents

Internal JS Code

- JS code can be embedded directly in an HTML page using `<script>` tag:

```
<script>
```

```
    ... JavaScript code ...
```

```
</script>
```

- There may be a number of scripts in an HTML page
- Scripts can be placed in the `<head>` or in the `<body>` section

❖ **Note:** Placing scripts at the **bottom** of the `<body>` element **improves the display speed**, because script compilation slows down the display

External JS Code

- JS script (code) can also be placed in **external files** and linked to HTML pages
- JS script files have the extension **.js**
- To link a JS script, use the **<script>** tag with the **src** attribute:
`<script src="<script URL>"></script>`
- External JS script **advantages**:
 - **Separate** JS code and HTML code (easy to read and maintain)
 - JS code can be **reused/shared** in HTML pages
 - **Cached** JS files can **speed-up** page load, and much more...

Recommended!

External JS Code

- External scripts can be referenced with a full URL (placed in other sites or the same site) or with a relative path (placed in the same site)
- Example:

```
<script src="scripts/myscript1.js"></script>
```

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
```

❖ Notes:

- External JS scripts cannot contain `<script>` tags
- The `<script>` tag can be placed **anywhere** in the HTML page
- An HTML page can use **multiple script files**: use multiple `<script>` tags to link to multiple script files

How Does JS Code Get Loaded?

- If a browser receive an HTML page with a reference to a JS file:
 - 1) It will **make a request** to the web server for the **script file**
 - 2) After receiving the script files, it will **execute** the script file
- JS script execution:
 - There is **no main function** as other programming languages
 - ⇒ The script code is executed from the **top to the bottom**
 - JS **functions** in the script file are only executed when called

Output of JS

- JS can display data in different ways:
 - Writing into an **HTML element**, using **innerHTML** property
 - Writing into **HTML page**, using **document.write()**
(**Note:** *Using document.write() after an HTML document is loaded, will delete all existing HTML*)
 - Writing into an **alert box**, using **window.alert()**
 - Output to browser **console**, using **console.log()**
(*usually used for debugging purpose*)

Output of JS

- Writing to an HTML element:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Today is: <span id="curdate"></span></h1>
    <p>5 + 6 = <span id="calc"></span></p>
    <script>
      var d = (new Date()).toString().substring(0, 15);
      document.getElementById("curdate").innerHTML = d;
      document.getElementById("calc").innerHTML = 5 + 6;
    </script>
  </body>
</html>
```

Today is: Mon Feb 18 2019

5 + 6 = 11

Output of JS

- Writing into an alert box:

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JS Output</h2>
    <button onclick="alert('Hello!')">Click me to say Hello</button>
  </body>
</html>
```

JS Output

Click me to say Hello

An embedded page on this page says

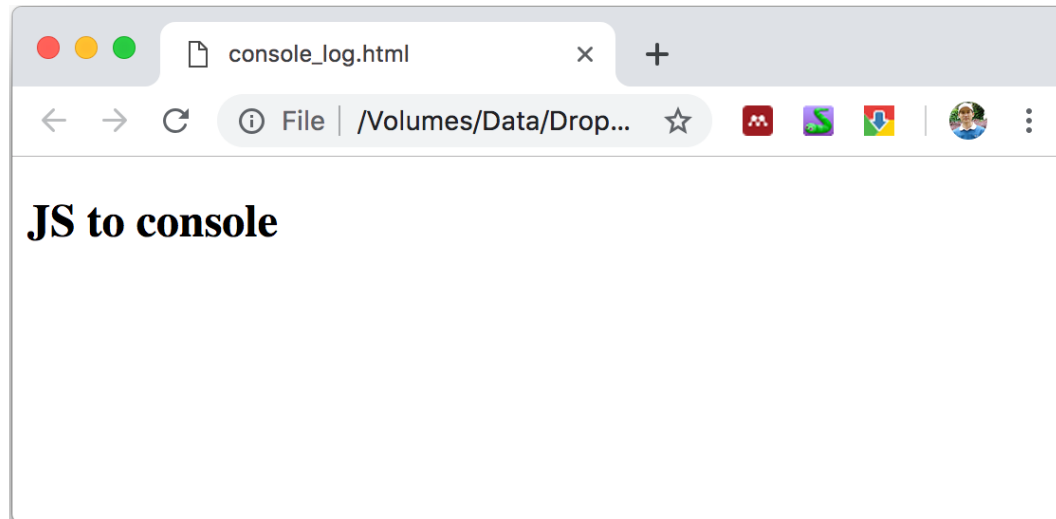
Hello!

OK

Output of JS

- Output to browser console:

```
<!DOCTYPE html>
<html>
  <body>
    <h2>JS to console</h2>
    <script>
      console.log('Hello, world!');
    </script>
  </body>
</html>
```



Output of JS

- Output to browser console (cont.):

The diagram illustrates the process of viewing JavaScript output in the browser console. It features two browser window screenshots. The first screenshot shows a right-click context menu on the page content, with a red arrow pointing from the instruction '1. Right click' to the menu. The second screenshot shows the 'Inspect' option selected in the menu, with a red arrow pointing from the instruction '2. Click Inspect' to it. A third red arrow points from the instruction '3. Choose Console' to the 'Console' tab in the browser's developer tools. A blue starburst graphic with the text 'for debugging' is positioned above the second screenshot. The browser window displays 'console_log.html' and the text 'JS to console'. The developer tools console shows the message 'Hello, world!' with the source 'console_log.html:6'.

JS to console

1. Right click

2. Click Inspect

3. Choose Console

for debugging

console_log.html

JS to console

Elements Console Sources Network Performance

top Filter Default

Hello, world! console_log.html:6

>

Console

JS Language Basics

Statement, syntax, comment, variable, operators, arithmetic, etc.

Statements

- Script/Program = { set of instructions/statement }
- A **statement** is composed of: values (constant/literals or variable), operators, expressions, keywords, and comments
- A statement **ends** with a semicolon ;
- Multiple **while-space** are ignored (so we can add as much as while-space for readability)
- A statement can be **broken** into multiple lines
- A set of statements can be grouped in **code block**, using {...}

Comments

- For code readability and maintenance
- Will be **ignored** by JS (doesn't interpret and execute)
- **Single line** comments:
 - Start with //
 - ... to the end of line
- **Multiline** comments:
 - Start with /*
 - End with */
- Good practice to use comments as much as possible

Variables

- **Declaration:** `var/let <var_name> [= <expression>];`
- **Example:** `var/let name = 'Jerry', age = 40;`
- **Datatype** is not necessary in the variable declaration
(however, JS does has datatypes: number, boolean, string, object, function, undefined, Array, Object, Date)
- JS variables can also be used **without declaration**
- Variable **naming rules** (same as identifier naming rules):
 - Names can **contain** letters, digits, underscores, and dollar signs
 - Names must **begin** with a letter, **\$** and **_**
 - Names are **case sensitive**

Variables

- Declaring a variable **without assigning value**: the variable will have the value **undefined**
`var age; // ⇒ age = undefined`
- **Re-declaring** a variable doesn't erase its value
`var myAge = 40;`
`var myAge; // ⇒ age still = 40`
- Get the **datatype** of a variable: use the **typeof** operator
 - Example: `typeof myAge` returns a string **"number"**
- We can **declare multiple variables** in one statement (using comma to separate variable names)

Operators

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation (<u>ES2016</u>)
/	Division
%	Modulus (Division Remainder)
++	Increment
--	Decrement

Arithmetic

Operator	Example	Same As
=	x = y	x = y
+=	x += y	x = x + y
-=	x -= y	x = x - y
*=	x *= y	x = x * y
/=	x /= y	x = x / y
%=	x %= y	x = x % y

Assignment

Operators

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to
?	ternary operator

Comparison

Operator	Description
&&	logical and
	logical or
!	logical not

Logical

Operator	Description
+	Addition

String

JS also provides **bitwise operators** such as & (AND), | (OR), ~ (NOT), etc.

Datatypes

- JS variables don't have types, but the values do
- Primitive datatypes:
 - Boolean: `true` or `false`
 - Number: everything is double (no integers)
 - String: in 'single' or "double" quotes
 - Null: `null`, a value meaning "this has no value" (null is an object)
 - Undefined: `undefined`, i.e. "the variable has no value assigned"
- Object types: Array, Date, String (wraps the primitive datatype)

Numbers

- All numbers in JS are **real numbers**, no integer
- **Operators** are like C/C++ or Java
- **Precedence** like C/C++ or Java
- Some **special values**:
 - NaN (not-a-number)
 - +Infinity
 - -Infinity
- There is a **Math** class, which provide basic math functions (such as `Math.ceil()`, `Math.floor()`, etc.)

Strings

- Object type
- Can be defined with single quote (preferred) or double quote
- Immutable
- No char type: letters are strings with length 1
- Methods: `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `join`, `substring`, `toLowerCase`, `toUpperCase`
- Property: `length`
- The only operator: `+`

```
var s = "Lion Messi";  
var fname = s.substring(0, s.indexOf(" "));  
var len = s.length;  
var s2 = 'Cristian Ronaldo';
```

String and Number Conversion

- Converse between string and number:

```
var count = 10;  
var s1 = "" + count; //"10"  
var s2 = count + "student"; //"10 student"  
var n1 = parseInt("10 student"); //10  
var n2 = parseFloat("blablabla"); //NaN  
var s = count.toString(); //"10"  
var n3 = Number(true); //1  
var n4 = Number(""); //0
```

- Get a character in a string at a specified position:

```
var s = "Hello World";  
var firstLetter = s[0]; //"H"  
var firstLetter = s.charAt(0); //"H"
```

Boolean

- **Literal** values: `true` and `false`
- **Operators**: `&&`, `||` and `!`
- Non-boolean values can be used as a boolean value with the following (implicitly) **conversion**:
 - `null`, `undefined`, `0`, `NaN`, `"`, and `""` are evaluated to `false`
 - Everything else is evaluated to `true`
- We can also explicitly convert an arbitrary value to boolean using the `Boolean()` constructor or `!!` operator

Boolean

```
var complete = true;
var age = 40;
var old = age > 60; //false
var boolValue = Boolean("to ambiguous"); //true
var boolValue = !!(""); //false

var username = "...";

if (username) {
    // username is defined
}
else {
    //username is not defined
}
```

Equality

- JavaScript operators `==` and `!=` are basically broken: they do an **implicit type conversion** before the comparison

```
' ' == '0' // false
' ' == 0   // true ('' ~ false, 0 ~ false)
0 == '0'   // true (0 => '0' = '0')
NaN == NaN // false
[''] == '' // true
false == undefined // false
false == null      // false
null == undefined  // true
```


Equality

- ECMAScript standard added `===` and `!==` operators with the better behavior

```
' ' === '0' // false
```

```
' ' === 0 // false
```

```
0 === '0' // false
```

```
NaN === NaN // still weirdly false
```

```
[] === '' // false
```

```
false === undefined // false
```

```
false === null // false
```

```
null === undefined // false
```

```
' ' == '0' // false
```

```
' ' == 0 // true
```

```
0 == '0' // true
```

```
NaN == NaN // false
```

```
[] == '' // true
```

```
false == undefined // false
```

```
false == null // false
```

```
null == undefined // true
```

Null and Undefined

- `null` is a value representing the **absence of a value** (similar to `null` value in Java and `nullptr` in C++)
- `undefined` is the value given to a variable that **has not had a value**
- However, we can also set a variable's value to `undefined` 😞

```
let x = null;  
let y;  
console.log(x);  
console.log(y);
```

```
let x = null;  
let y = undefined;  
console.log(x);  
console.log(y);
```

<code>null</code>
<code>undefined</code>
<code>></code>

Arrays

- An Object type used to store **multiple values** in a **single variable**
- 0-based indexing
- Mutable
- Can check size via **length** property

```
var list = []; // Creates an empty array
```

```
//create a pre-filled array
```

```
var groceries = ['milk', 'cocoa puffs'];
```

```
groceries[1] = 'kix'; //access an element of the array
```

Arrays

- An array can be used as a list, queue, stack, etc.
- Basic methods:
 - **push/pop**: add/remove the element at the end, return an element
 - **shift/unshift**: remove/add an element at the head, return an element
 - **concat**: joints two or more arrays, return the jointed array
 - **splice**: add/remove elements from an array, return the array after adding/removing
 - **sort/reserve**: sort/reverse the order of elements in the array, return the result after sorted, reserved
 - **slice**: select a part of an array, return the selected part

https://www.w3schools.com/jsref/jsref_obj_array.asp

Arrays

```
var a = ["Truc", "Lan"];  
a.sort();           //a=["Lan", "Truc"]  
a.push("Cuc");      //a=["Lan", "Truc", "Cuc"]  
a.unshift("Mai");   //a=["Mai", "Lan", "Truc", "Cuc"]  
  
var b = a.slice(1, 3);  
a.splice(2, 1, "Tung", "Dao"); //a=["Mai","Lan","Tung","Dao","Cuc"]  
a.pop();            //a=["Mai", "Lan", "Tung", "Dao"]  
a.shift();          //a=["Lan", "Tung", "Dao"]  
  
var i = a.indexOf("Tung"); //1
```

Array Iteration Methods

- Simple for loop (usually combined with the length property)
- for ... of statement
- Array.forEach():
 - calls a **callback function** once for each array element
 - The callback function takes **3 parameters**: item value, the index and the array itself

```
let iterable = [10, 20, 30];  
  
for (const value of iterable) {  
  console.log(value);  
}
```

10
20
30

```
let iterable = [10, 20, 30];  
  
for (let value of iterable) {  
  value += 1;  
  console.log(value);  
}
```

11
21
31

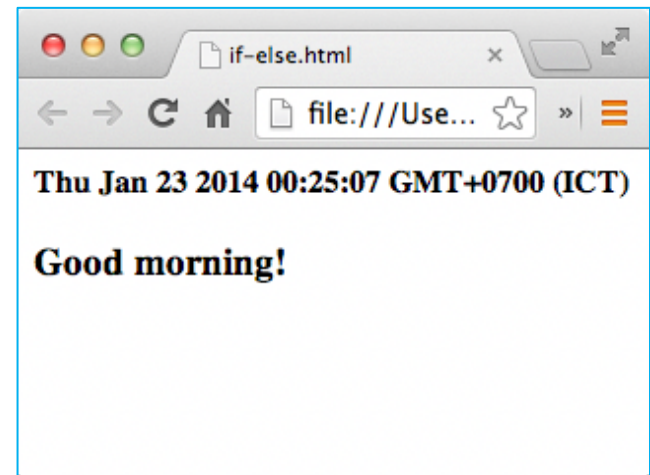
Condition Statements – if/else

- Syntax:

```
if (condition1) {  
    //block of code to be executed if condition1 is true  
} else if (condition2) {  
    //block of code to be executed if the condition1 is false and condition2 is true  
}  
...  
else {  
    //block of code to be executed if the condition1 is false and condition2 is false  
}
```

Condition Statements – if/else

```
var d = new Date();
var t = d.getHours();
document.write("<h4>" + d + "</h4>");
document.write("<h3>");
if (t < 12) {
    document.write("Good morning!"); }
else if (t < 18) {
    document.write("Good afternoon!"); }
else {
    document.write("Good evening!");
}
document.write("</h3>");
```



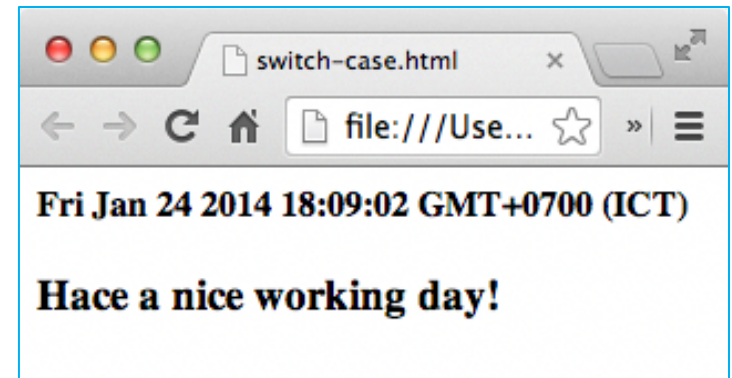
Condition Statement – switch/case

- Syntax:

```
switch (expression) {  
  case x:  
    //code block to be executed when expression = x  
    break;  
  case y:  
    // code block to be executed when expression = y  
    break;  
  ...  
  default:  
    // code block to be executed when all above cases  
}
```

Condition Statement – switch/case

```
var d = new Date();
document.write("<h4>" + d + "</h4>");
document.write("<h3>");
switch (d.getDay()) {
    case 0:
    case 6:
        document.write("Have a lovely weekend!");
        break;
    default:
        document.write("Hace a nice working day!");
}
document.write("</h3>");
```



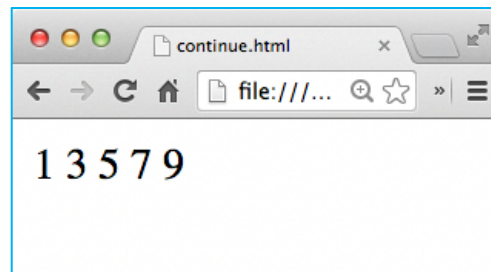
Loop – for

- Syntax:

```
for (statement 1; statement 2; statement 3) {  
    //code block to be executed  
}
```

- Example:

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



Loop – while, do/while

- Syntax:

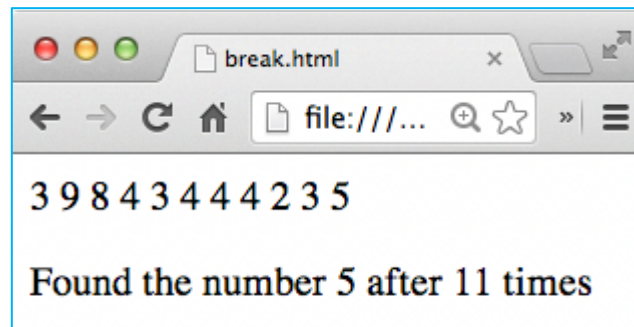
```
while (condition) {  
    // code block to be executed  
}
```

```
do {  
    //code block to be executed  
} while (condition);
```

break statement

- Used to jump out a loop or a switch statement and continue executing the code after the loop or switch

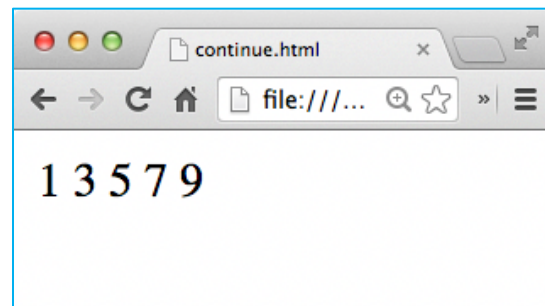
```
var count = 0;
while (true) {
    count++;
    var r = Math.floor(Math.random() * 10);
    document.write(r + " ");
    if (r == 5) {
        document.write("<p>Found '5' after "
            + count + " times</p>");
        break;
    }
}
```



continue statement

- The continue statement jumps over an iteration of a loop
- The condition of the loop is checked for a new iteration

```
for (var i = 0; i <= 10; i++) {  
    if (i % 2 == 0)  
        continue;  
    document.write(i + " ");  
}
```



JS Functions

Function Declaration

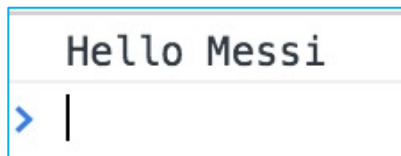
- **Syntax:**

```
function func_name(parameters) {  
    //function body (statements)  
}
```

- A function hasn't been executed until it is **called**
- A **function call** can be made **before** its declaration (**hoisting**)

```
function hello(name) {  
    console.log("Hello " + name);  
}
```

```
hello("Messi");
```



Function Expression

- A JS function can also be defined using an **expression**

```
var x = function (a, b) {return a * b};  
var z = x(4, 3);  
alert(z);           //alert "12"
```

- The above function is actually an anonymous function (no name)
- The function is referenced by the variable and it can **be called using the variable**
- This also allow to use the **function as a parameter** or **return value** of a function

Function Expression

```
function sort(lessthan, arr) { //pass function as a parameter
  for (i=0; i<arr.length-1; i++) {
    for (j=i+1; j<arr.length; j++) {
      if (lessthan(arr[j], arr[i]))
        swap(arr[i], arr[j]);
    }
  }
}
```

```
function d() {
  function e() {
    alert('E');
  }
  return e; //returns function e()
}

d(); //alerts 'E'
```

Function Parameters

- **Function parameters** (formal parameter) are the names **listed** in the function definition
- **Function arguments** (actual parameter) are the real values **passed** to (and received by) the function
- Parameter rules:
 - JS function definitions do not specify **datatype** for parameters
 - JS functions don't check the **datatype** of the passed arguments
 - JS functions do not check the **number of arguments** received
- The built-in object **arguments** contains an **array** of the arguments

Function Parameters

```
function findMax() {  
    var i;  
    var max = -Infinity;  
    for (i = 0; i < arguments.length; i++) {  
        if (arguments[i] > max) {  
            max = arguments[i];  
        }  
    }  
    return max;  
}
```

Pass-by-Reference & Pass-by-Value

- **Objects** are pass by reference:
 - The change of the object properties are reflected outside the function
- **Arguments** are pass by value:
 - The function only gets to know the value, not the argument's location
 - Change to arguments are not reflected outside the function

Document Object Model (DOM)

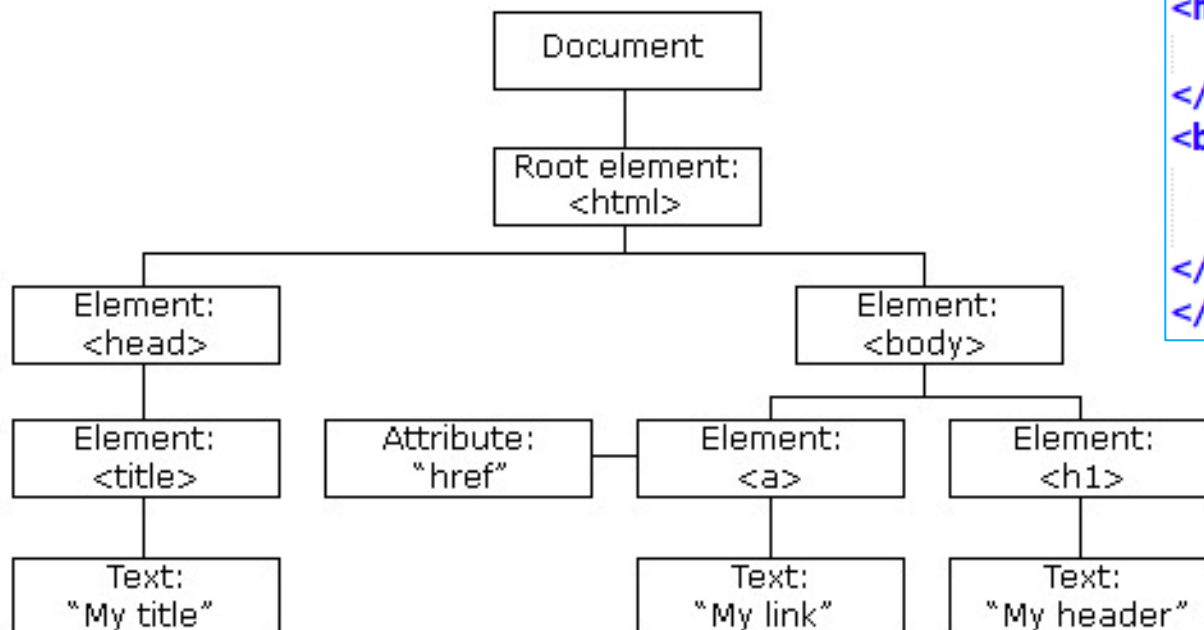
What is HTML DOM?

- The **DOM** is a **W3C standard** for accessing documents, including core, XML and HTML DOM
- The **HTML DOM** is a standard **object model** and **programming interface** for HTML, which defines:
 - The HTML elements as **objects**
 - The **properties** of all HTML elements
 - The **methods** to access all HTML elements
 - The **events** for all HTML elements

HTML DOM: a standard for getting, changing, adding, or deleting HTML elements

HTML DOM Model

- The HTML DOM model is constructed as a **tree of objects**
- When a web page is loaded, the browser creates a Document Object Model of the page

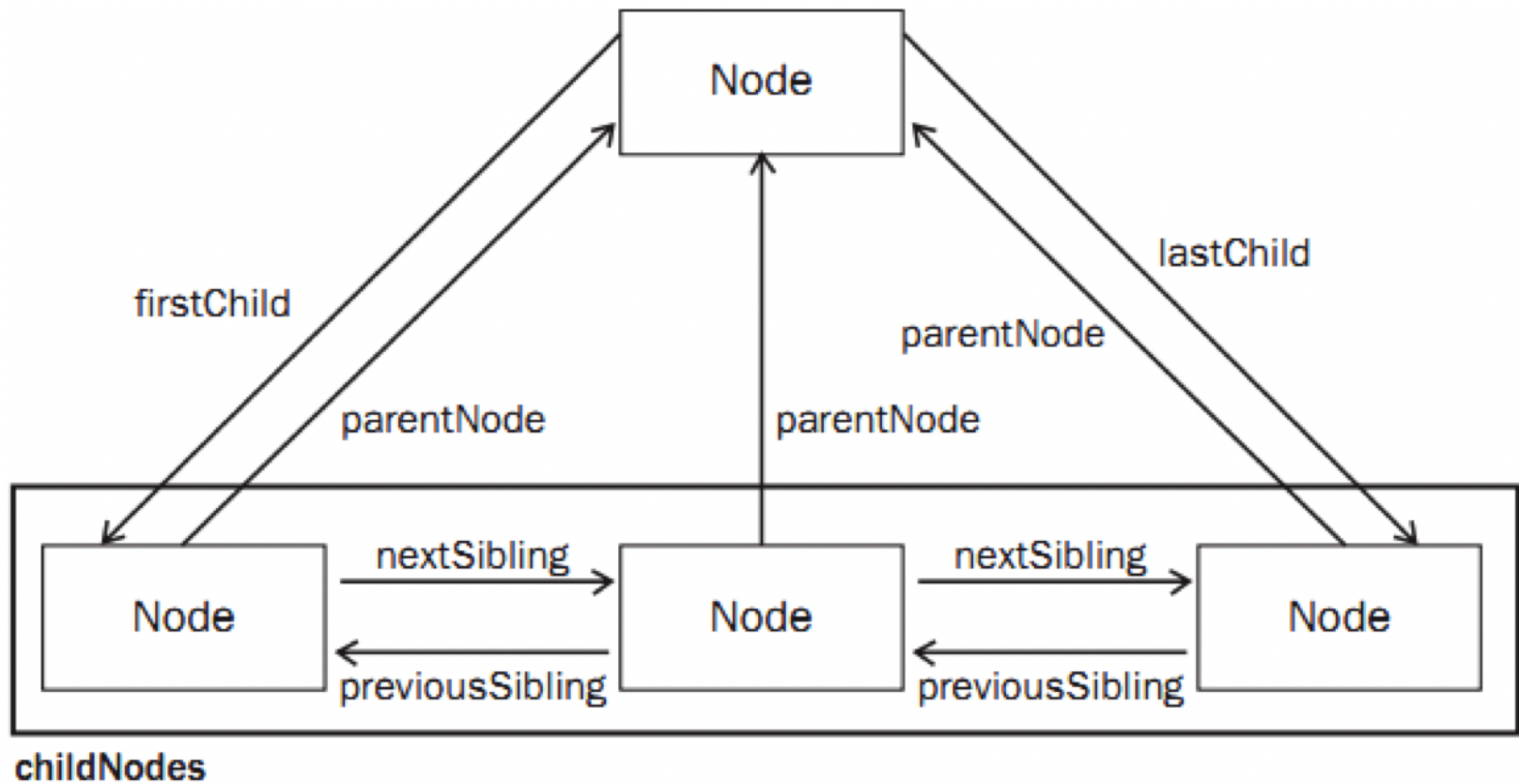


```
<html>
<head>
  <title>My title</title>
</head>
<body>
  <a href="...">My link</a>
  <h1>My header</h1>
</body>
</html>
```


HTML DOM Model

- The objects in the hierarchical tree are called **nodes**
- Basic **node types**:
 - **Document node** (`Node.DOCUMENT_NODE`, 9): represents the entire document (the root-node of the DOM tree, children: Elements)
 - **Element node** (`Node.ELEMENT_NODE`, 1): represents an element (children: Elements, Texts, Comments)
 - **Text node** (`Node.TEXT_NODE`, 3): represents textual content in an element or attribute (children: none)
 - **Attribute node** (`Node.ATTRIBUTE_NODE`, 2): represents an attribute (children: none)

HTML DOM Model



HTML DOM Methods

- A method is an **action** that we can perform on **HTML elements**

Methods	Description
Selecting HTML element	
<code>document.getElementById(id)</code>	Find an element by element id
<code>document.getElementsByTagName(tag)</code>	Find elements by tag name
<code>document.getElementsByClassName(class)</code>	Find elements by class name
<code>document.querySelector(css selectors)</code>	Returns the first child element that matches the given CSS selector(s)
<code>document.querySelectorAll(css selectors)</code>	Returns all elements that matches the given CSS selector(s)

HTML DOM Methods

- A method is an **action** that we can perform on **HTML elements**

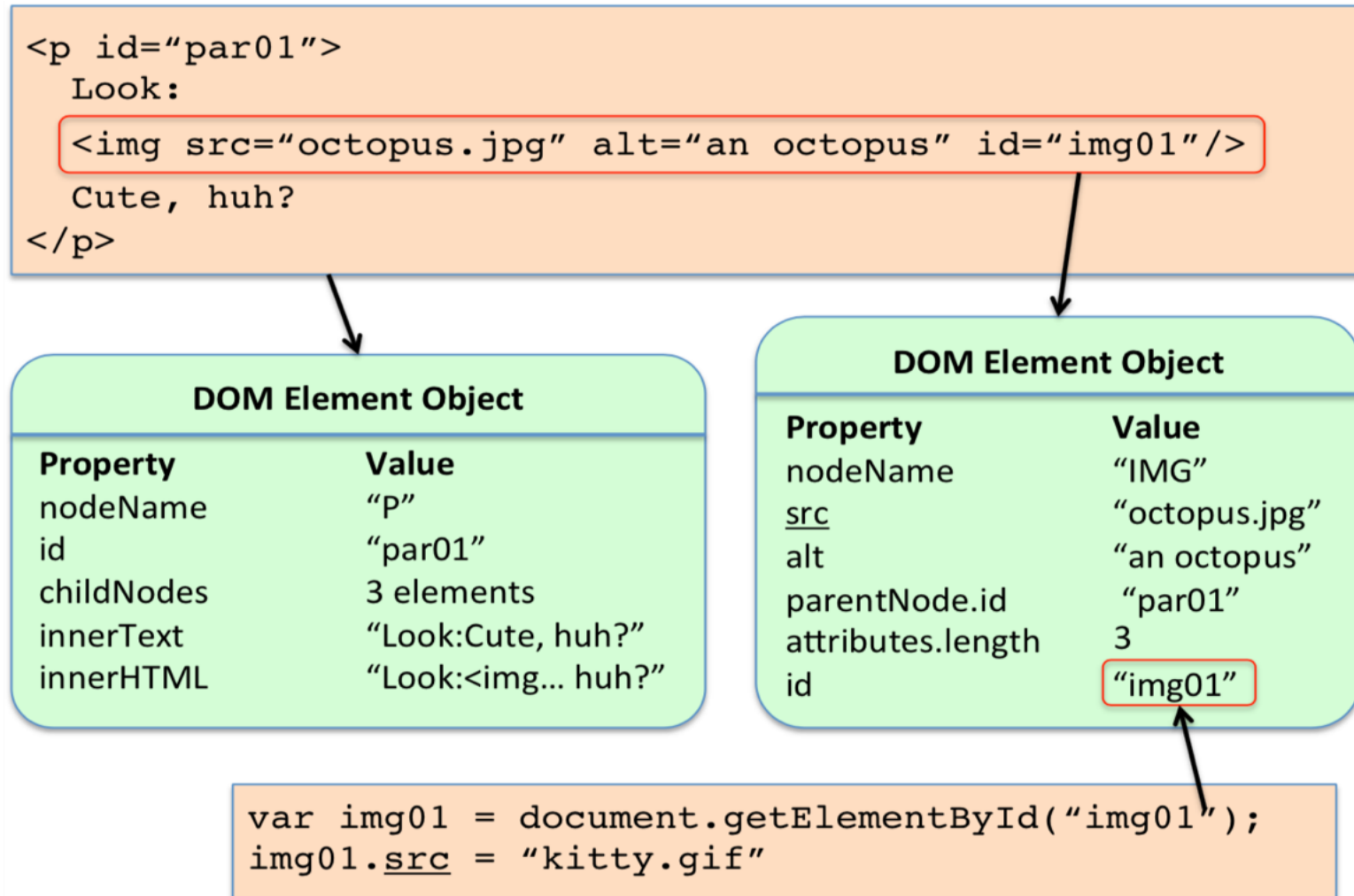
Methods	Description
Adding and Deleting elements	
<code>document.createElement(element)</code>	Create an HTML element
<code>parentNode.appendChild(node)</code>	Add an HTML element
<code>parentNode.removeChild(node)</code>	Remove an HTML element
<code>parentNode.replaceChild(new, old)</code>	Replace an HTML element
<code>document.write(text)</code>	Write into the HTML output stream

HTML DOM Properties

- Properties: values (of HTML elements) that we can **set** or **change**
- Each type of node has its own **set of attributes**
 - .innerText**: the text content of the specified node (and all its descendants)
 - .innerHTML**: the HTML content of the specified node
 - .nodeValue**: the node value of the specified node
 - .nodeName**: the name of the specified node
 - .parentNode**: the parent of the specified node (Node object)
 - .childNodes**: the child nodes (NodeList object)
 - .attributes**: attributes of the specified node (NamedNodeMap obj.)

Full list: https://www.w3schools.com/jsref/dom_obj_all.asp

HTML DOM Properties

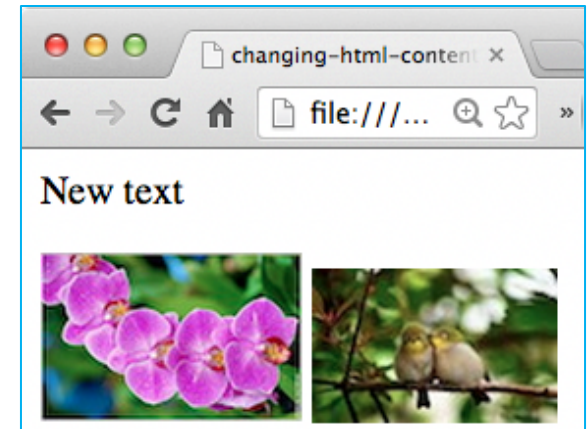


HTML DOM Properties

- nodeName:
 - Element node: tag name (e.g. "P", "IMG", etc.)
 - Attribute node: attribute name (e.g. "src", "href", etc.)
 - Text/Document node: "#text", "#document"
- nodeValue:
 - Element node: "undefine"
 - Text node: text value of the node
 - Attribute node: value of the attribute
- Note: to access a property value, using the syntax
 <node>.<attribute_name>

Change Element Content and Attribute

```
<html>
  <body>
    <p id="par1">Hello World</p>
    <p>
      
    </p>
    <script type="text/javascript">
      var p1 = document.getElementById("par1");
      p1.innerHTML = "New text";
      var m2 = document.getElementById("img2");
      m2.src = "birds.jpg";
    </script>
  </body>
</html>
```



Change Element Style

- Syntax: `<node>.style.<property> = <value>`

```
<body>
```

```
<p id="par1">The first paragraph</p>
```

```
<p id="par2">The second paragraph</p>
```

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

```
  var p1 = document.getElementById("par1");
```

```
  p1.style.fontSize = "x-large";
```

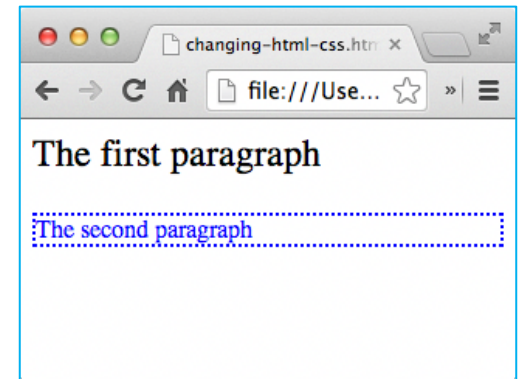
```
  var p2 = document.getElementById("par2");
```

```
  p2.style.color = "blue";
```

```
  p2.style.border = "2px dotted #0000FF";
```

```
</script>
```

```
</body>
```



Change Element Class

- Syntax: `<node>.className = <classname>`

```
<head>
```

```
  <style>
```

```
    .blue {color: blue; border: 2px dotted #0000FF;}
```

```
    .big {font-size: x-large;}
```

```
  </style>
```

```
</head>
```

```
<body>
```

```
  <p id="par1">The first paragraph</p>
```

```
  <p id="par2">The second paragraph</p>
```

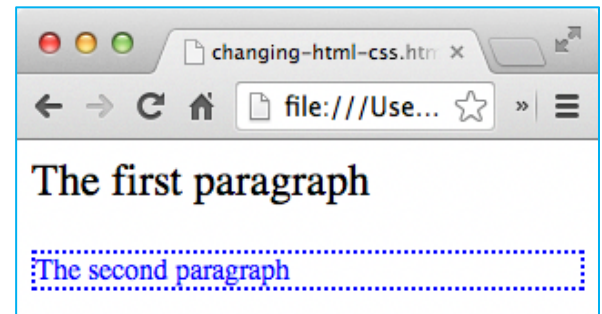
```
  <script type="text/javascript">
```

```
    document.getElementById("par1").className = "big";
```

```
    document.getElementById("par2").className = "blue";
```

```
  </script>
```

```
</body>
```



Events

HTML Events

- HTML provides event handler attributes to use JS code to handle the events

```
<element event='some JS code'>
```

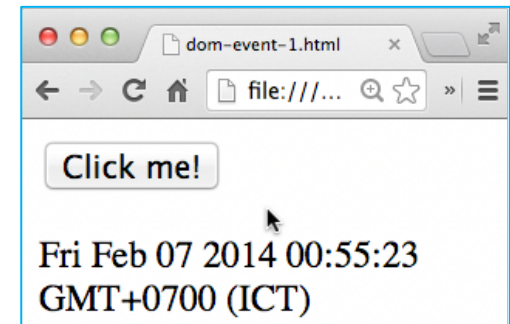
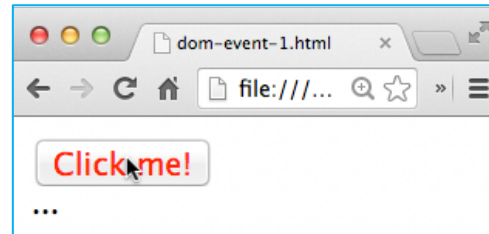
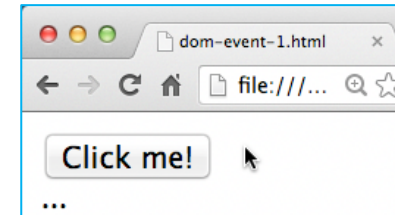
- Some common HTML events:
 - Mouse: onclick, ondblclick, onmousedown, onmousemove, onmouseover, onmouseout, onmouseup
 - Keyboard: onkeypress, onkeyup, onkeydown
 - Form: onblur, onchange, onfocus, onsubmit, onselect
 - Frame/Object: onload, onresize, onscroll

HTML Events

```

<html>
  <head>
    <script>
      function displayDate(eid) {
        var e = document.getElementById(eid);
        e.innerHTML = Date();
      }
    </script>
  </head>
  <body>
    <input type="button" value="Click me!" id="bt1"
      onclick="displayDate('demo');"
      onmouseover="this.style.color='red';" onmouseout="
        this.style.color='black';"/>
    <p id="demo">...</p>
  </body>
</html>

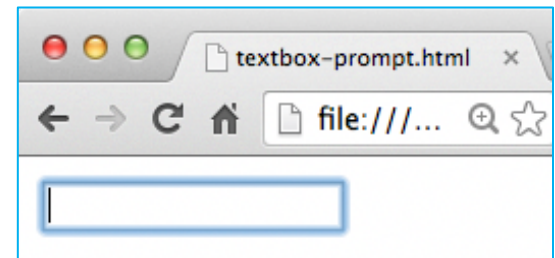
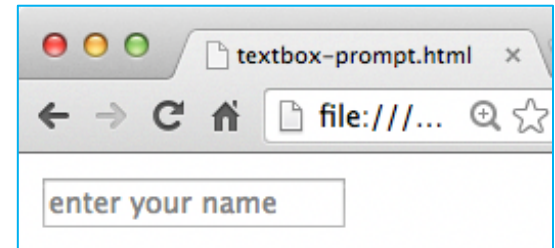
```



HTML Events

```
<script>
function prompttext(getf, inp) {
  if (getf && (inp.value == "enter your name")) {
    inp.value = "";
    inp.style.color = "black";
  }
  if (!getf && (inp.value == "")) {
    inp.value = "enter your name";
    inp.style.color = "gray";
  }
}
</script>

<input id="uname" type="text" style="color:gray;"
value="enter your name" onfocus="prompttext(true, this);"
onblur="prompttext(false, this);"/>
```



DOM Event Listeners

- Each DOM object has the following method for **event handling**:
`addEventListener(event name, function name);`
 - **event name**: the string of name of the event to listen to
 - **function name**: the name of the JS function that will be executed when the event fires (this function gets an **Event object** as param)

Pros: **separate** JS code from HTML markup
An event can be handled by **multiple functions**

- To **stop** listen to an event use the following function:
`removeEventListener(event name, function name);`
(the parameters are similar to the `addEventListener()`)

HTML DOM Listeners

```

<script>
  document.getElementById("myButton").addEventListener("click", validate);

  function validate(event) {
    alert("You click the object: " + event.target);
    var x = document.getElementsByName("fname")[0].value;
    if (x == "")
      alert("Name must be filled out");
    else
      alert("Hello: " + x);
  }
</script>
<body>
  <p> Name <input type="text" name="fname">
  <input id="myButton" type="button" value="Click me!">
  </p>
</body>

```

Name

An embedded page on this page says

Hello: Jerry

OOP in JS

JS Objects

- In JS, almost "everything" is an object, except primitives
 - Booleans can be objects (if defined with the new keyword)
 - Numbers can be objects (if defined with the new keyword)
 - Strings can be objects (if defined with the new keyword)
 - Dates are always objects
 - Maths are always objects
 - Regular expressions are always objects
 - Arrays are always objects
 - Functions are always objects
 - Objects are always objects

JS Objects

- A variable can contain a single value
- JS objects:
 - A JS object can contain many **named values** (**properties**)
 - The values are written in pair as **name : value**
 - An object can also contain **methods** (actions that the object can performed)
 - A method can be considered as a property that contains a function definition

Create a JS Object

- Three ways to create an object:

- Define and create a single object, using an **object literal**

```
var person = {firstName:"John", lastName:"Doe", age:50,  
  showInfo: function() {  
    alert(this.name);  
  }  
};
```

- Define and create a single object, with the **keyword new**

```
var person = new Object();  
person.firstName = "John";  
person.age = 50;  
showInfo: function() { alert(this.name); };
```

Create a JS Object

- Three ways to create an object (cont):
 - Define an **object constructor function**, and then create objects of the constructed type

```
function User(uname, pass) {  
    this.uname = uname; this.pass = pass;  
    this.showInfo = function() {  
        alert(this.uname + ":" + this.pass);  
    } //showInfo()  
} //User  
var alibaba = new User("alibaba", "vomc");  
alibaba.showInfo(); //alert "alibaba:vomc"
```

- **Note:** new property **cannot be added** to an existing object constructor (i.e. cannot inherit)

Add Object Properties

- To add a property to an object, name the property and give it a value:

```
scores = {  
  peach: 100,  
  mario: 88,  
  luigi: 91  
};  
scores.toad = 72;  
let name = 'wario';  
scores[name] = 102;  
console.log(scores);
```

► *Object {peach: 100, mario: 88, luigi: 91, toad: 72, wario: 102}*

Remove Object Properties

- To remove a property of an object, use delete:

```
const scores = {  
  peach: 100,  
  mario: 88,  
  luigi: 91  
};  
scores.toad = 72;  
let name = 'wario';  
scores[name] = 102;  
delete scores.peach;  
console.log(scores);
```

```
► Object {mario: 88, luigi: 91, toad: 72, wario: 102}
```

Iterate through Object Properties

- Use the **for ... in** loop (for each property in the object):

```
for (key in object) {  
    //do something with object[key]  
}
```

- Example:

```
for (let name in scores) {  
    console.log(name + ' got ' + scores[name]);  
}
```

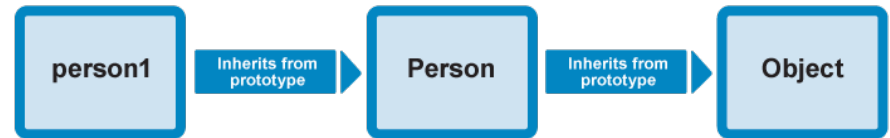

Prototype

- JS is described as a **prototype-based language**
- Prototype may be considered as a “class” of an object created by an object constructor function
- A prototype is created for every object constructor function
- It is used to:
 - **create static** (common) **properties** for objects
 - **add properties** (inherit) to an existing object created by an object constructor function

Prototype

```
function Person(first, last) {  
  this.first = first;  
  this.last = last;  
}
```

```
var person1 = new Person("John", "Doe");
```



- To add a method for Person:

```
Person.prototype.showInfo = function() {  
  alert(this.first + " " + this.last);  
};
```

```
var person1 = new Person("John", "Doe");  
person1.showInfo();
```

Prototype

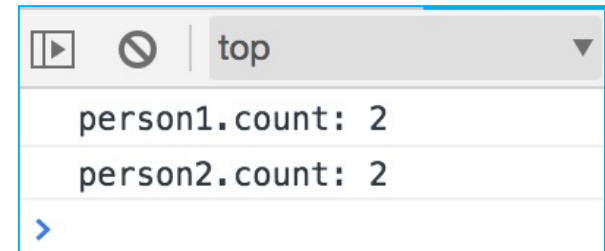
- Add a “static” property for a prototype:

```
function Person(first, last) {  
  this.firstName = first;  
  this.lastName = last;  
  Person.prototype.count++;  
};
```

```
Person.prototype.count = 0;
```

```
var person1 = new Person("John", "Doe");  
var person2 = new Person("Doe ", "John");
```

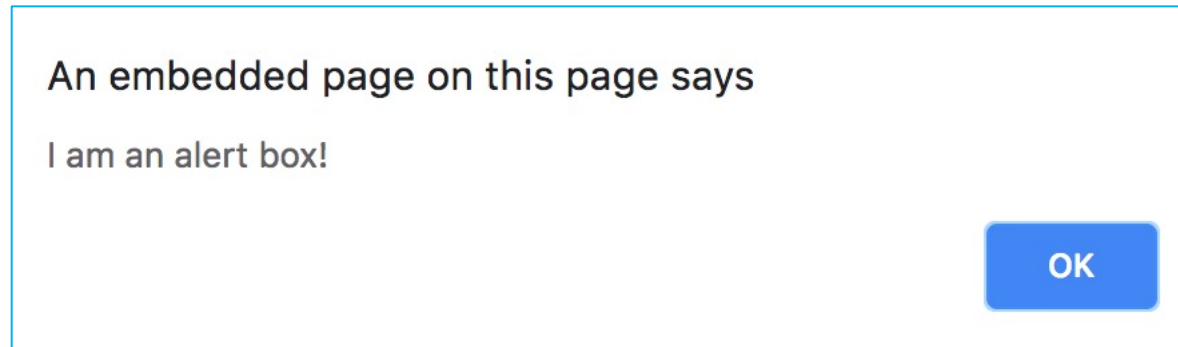
```
console.log("person1.count: " + person1.count);  
console.log("person2.count: " + person1.count);
```



Appendix

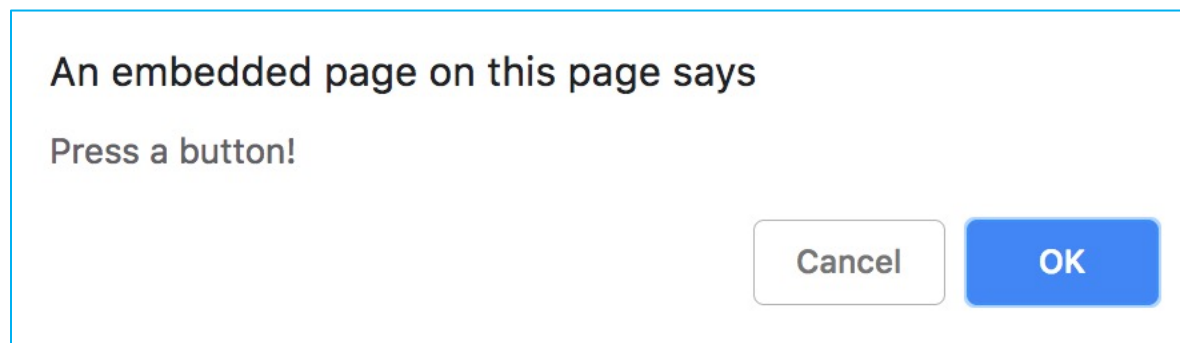
Alert Box

- Used to make sure information comes through to the user
- The user will have to click "OK" to proceed
- Syntax: `[window.]alert("sometext");`



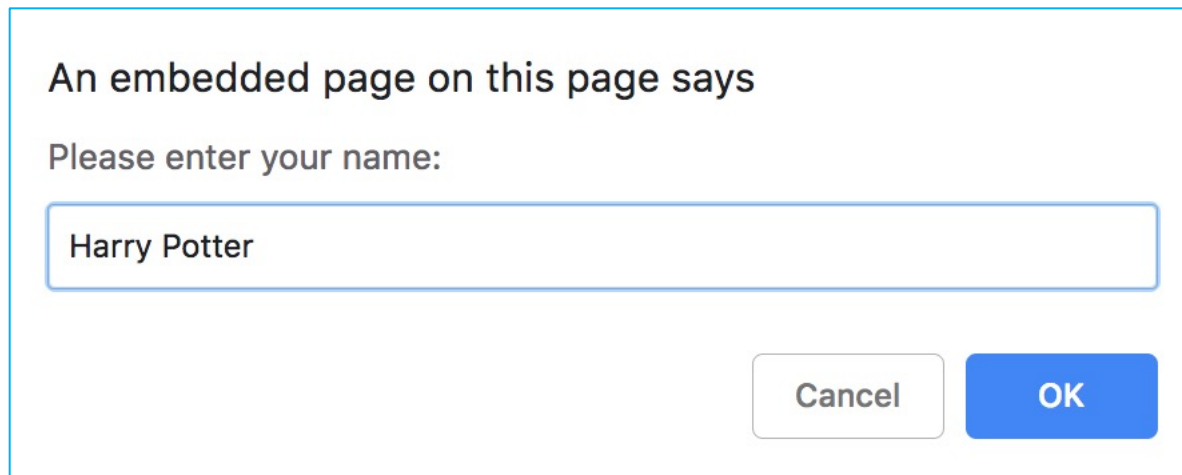
Confirm Box

- Used verify that the user accept something or not
- The user will have to click either "OK" or "Cancel" to proceed
 - If the user clicks "OK", the box returns **true**.
 - If the user clicks "Cancel", the box returns **false**
- Syntax: `[window.]confirm("sometext");`



Prompt Box

- Used to get an input value from users
- The user will have to click either "OK" or "Cancel" to proceed
 - If the user clicks "OK" the box returns the input value.
 - If the user clicks "Cancel" the box returns **null**
- Syntax: `[window.]prompt("sometext", "defaultText");`



An embedded page on this page says

Please enter your name:

Cancel OK

Further Reading

- Methods of JS basic datatypes:
 - String: `indexOf()`, `lastIndexOf()`, `slice()`, `split()`, `trim()`, etc.
 - Number: `isFinite()`, `isInteger()`, `toString()`, etc.
 - Math: `abs()`, `min()`, `max()`, `sqrt()`, `round()`, `random()`, etc.
- JS regular expression: https://www.w3schools.com/js/js_regexp.asp
- JS debugging: https://www.w3schools.com/js/js_debugging.asp
- JS style guide: https://www.w3schools.com/js/js_conventions.asp
- JS Browser Object Model (BOM, allows JS to "talk to" the browser):
Window, Screen, Location, History, Navigation, Cookies

Question?

Chapter 3 – JavaScript