The University Of Sheffield.

Javascript Continued

Prof Fabio Ciravegna
The University of Sheffield
fabio.ciravegna@shef.ac.uk
COM6517

The University Of Sheffield.

What is JQuery?

http://www.w3schools.com/jquery/jquery_intro.asp

- jQuery is a lightweight, "write less, do more", JavaScript library.
- its purpose is to make it much easier to use JavaScript on your website.
- jQuery takes a lot of common tasks that require many lines of JavaScript code to accomplish,
  - and wraps them into methods that you can call with a single line of code.
- It simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
  - HTML/DOM manipulation
  - CSS manipulation
  - HTML event methods
  - Effects and animations
  - AJAX
  - Utilities

download it from jQuery.com
include it in the <head> of your HTML page

```
<head>
<script src="jquery.min.js"></script>
</head>
```

The University Of Sheffield.

Syntax

http://www.w3schools.com/jquery/jquery_syntax.asp

- The jQuery syntax is tailor-made for selecting HTML elements and performing some action on the element(s).
- Basic syntax is: **`$(selector).action()`**
  - A $ sign to define/access jQuery
  - A (selector) to "query (or find)" HTML elements
  - A jQuery action() to be performed on the element(s)
- Examples:
- `$(this).hide()` - hides the current element.
- `$("p").hide()` - hides all <p> elements.
- `$(".test").hide()` - hides all elements with class="test".
- `$("#test").hide()` - hides the element with id="test".

The University Of Sheffield.

Document Ready Event

- Always wait for the document to be fully loaded and ready before working with it.
- Examples of actions that can fail if methods are run before the document is fully loaded:
  - Trying to hide an element that is not created yet
  - Trying to get the size of an image that is not loaded yet
- In JQuery:

```
$(document).ready(function(){

    // jQuery methods go here...

});
```

- This is to prevent any jQuery code from running before the document is finished loading (is ready)

The University Of Sheffield.

Query Selectors: Element

http://www.w3schools.com/jquery/jquery_selectors.asp

- Element Selector:
  - selects all <p> elements on a page
  - e.g. $("p")
    - Example: hide all <p> elements when a button is clicked

```
$(document).ready(function(){
    $("button").click(function(){
        $("p").hide();
    });
});
```

This is equivalent to document.getElementsByTagName('p');

The University Of Sheffield.

#id Selector

- An id should be unique within a page,
  - so you should use the #id selector when you want to find a single, unique element
  - e.g. `$("#test")` (where test is the id of an element)
  - Example:

```
$(document).ready(function(){
    $("button").click(function(){
        $("#test").hide();
    });
});
```

This is equivalent to document.getElementById('test');

The University Of Sheffield.

.class Selector

- Use the .class Selector to find elements with a specific class
  - Example: When a user clicks on a button, the elements with class="test" will be hidden:

```
$(document).ready(function(){
    $("button").click(function(){
        $(".test").hide();
});});});
```

Equivalent to:

```
<body onload= function(){
    var elems = document.getElementsByTagName('*'), i;
    for (i in elems) {
        if((' ' + elems[i].className + ' ').indexOf(' test ')
                > -1) {
            elems[i].style.visibility = ‘hidden;
```

More Examples of jQuery Selectors

Syntax	Description	Example
$("*")	Selects all elements	Try it
$(this)	Selects the current HTML element	Try it
$("p.intro")	Selects all <p> elements with class="intro"	Try it
$("p:first")	Selects the first <p> element	Try it
$("ul li:first")	Selects the first element of the first	Try it
$("ul li:first-child")	Selects the first element of every	Try it
$("[href]")	Selects all elements with an href attribute	Try it
$("a[target='_blank']")	Selects all <a> elements with a target attribute value equal to "_blank"	Try it
$("a[target!='_blank']")	Selects all <a> elements with a target attribute value NOT equal to "_blank"	Try it
$(":button")	Selects all <button> elements and <input> elements of type="button"	Try it
$("tr:even")	Selects all even <tr> elements	Try it
$("tr:odd")	Selects all odd <tr> elements	Try it

Functions In a Separate File

If your website contains a lot of pages, and you want your jQuery functions to be easy to maintain, you can put your jQuery functions in a separate .js file.

When we demonstrate jQuery in this tutorial, the functions are added directly into the <head> section. However, sometimes it is preferable to place them in a separate file, like this (use the src attribute to refer to the .js file):

Example

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js">
</script>
<script src="my_jquery_functions.js"></script>
</head>
```

The University Of Sheffield.

Events in JQuery

http://www.w3schools.com/jquery/jquery_events.asp

- Events in Javascript have a corresponding event in JQuery
- e.g.

Mouse Events	Keyboard Events	Form Events	Document/Window Events
click	keypress	submit	load
dblclick	keydown	change	resize
mouseenter	keyup	focus	scroll
mouseleave		blur	unload

- events must be combined with the selectors, e.g. to associate a function to <button id='**mybutton**'>

```
$("#mybutton").click(function(){
  // action goes here!!
alert('button pressed!');});});
```

Commonly Used jQuery Event Methods

$(document).ready()

The $(document).ready() method allows us to execute a function when the document is fully loaded. This event is already explained in the jQuery Syntax chapter.

click()

The click() method attaches an event handler function to an HTML element.

The function is executed when the user clicks on the HTML element.

The following example says: When a click event fires on a <p> element; hide the current <p> element:

Example

```
$("p").click(function(){
    $(this).hide();
});
```

Try it yourself »

dblclick()

The dblclick() method attaches an event handler function to an HTML element.

The function is executed when the user double-clicks on the HTML element:

Example

```
$("p").dblclick(function(){
    $(this).hide();
});
```

Try it yourself »

mouseenter()

The mouseenter() method attaches an event handler function to an HTML element.

The function is executed when the mouse pointer enters the HTML element:

Example

```
$("#p1").mouseenter(function(){
    alert("You entered p1!");
});
```

Try it yourself »

The University Of Sheffield.

Multiple events with ‘on’

The on() Method

The on() method attaches one or more event handlers for the selected elements.

Attach a click event to a <p> element:

Example

```
$("p").on("click", function(){
    $(this).hide();
});
```

Try it yourself »

Attach multiple event handlers to a <p> element:

Example

```
$("p").on({
    mouseenter: function(){
        $(this).css("background-color", "lightgray");
    },
    mouseleave: function(){
        $(this).css("background-color", "lightblue");
    },
    click: function(){
        $(this).css("background-color", "yellow");
```

The University Of Sheffield.

Hide and Show

http://www.w3schools.com/jquery/jquery_hide_show.asp

- With jQuery, you can hide and show HTML elements with the hide() and show() methods:

Example

```
$("#hide").click(function(){
    $("p").hide();
});

$("#show").click(function(){
    $("p").show();
});
```

The University Of Sheffield.

Before we continue…

- Please always remember that when Javascript is used on client side (e.g. on a browser), it will always need to be in a webpage!
- So our previous example will be:

```
<!DOCTYPE html>
<html>
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js">
</script>
<script>
$(document).ready(function(){
    $("#hide").click(function(){
        $("p").hide();
    });
    $("#show").click(function(){
        $("p").show();
    });
});
</script>
</head>
<body>

<p>If you click on the "Hide" button, I will disappear.</p>

<button id="hide">Hide</button>
<button id="show">Show</button>

</body>
```

The University Of Sheffield.

Hide/Show with speed

Syntax:

```
$(selector).hide(speed,callback);

$(selector).show(speed,callback);
```

The optional speed parameter specifies the speed of the hiding/showing, and can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the hide() or show() method completes (you will learn more about callback functions in a later chapter).

The following example demonstrates the speed parameter with hide():

Example

```
$("button").click(function(){
    $("p").hide(1000);
});
```

The University Of Sheffield.

Toggle between hide/show

jQuery toggle()

With jQuery, you can toggle between the hide() and show() methods with the toggle() method.

Shown elements are hidden and hidden elements are shown:

Example

```
$("button").click(function(){
    $("p").toggle();
});
```

Try it yourself »

Syntax:

```
$(selector).toggle(speed,callback);
```

The optional speed parameter can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after toggle() completes.

jQuery fadeIn() Method

The jQuery fadeIn() method is used to fade in a hidden element.

Syntax:

```
$(selector).fadeIn(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeIn() method with different parameters:

Example

```
$("button").click(function(){
    $("#div1").fadeIn();
    $("#div2").fadeIn("slow");
    $("#div3").fadeIn(3000);
});
```

Try it yourself »

jQuery fadeOut() Method

The jQuery fadeOut() method is used to fade out a visible element.

Syntax:

```
$(selector).fadeOut(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeOut() method with different parameters:

Example

```
$("button").click(function(){
    $("#div1").fadeOut();
    $("#div2").fadeOut("slow");
    $("#div3").fadeOut(3000);
});
```

Try it yourself »

jQuery fadeToggle() Method

The jQuery fadeToggle() method toggles between the fadeIn() and fadeOut() methods.

If the elements are faded out, fadeToggle() will fade them in.

If the elements are faded in, fadeToggle() will fade them out.

Syntax:

```
$(selector).fadeToggle(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the fading completes.

The following example demonstrates the fadeToggle() method with different parameters:

Example

```
$("button").click(function(){
    $("#div1").fadeToggle();
    $("#div2").fadeToggle("slow");
    $("#div3").fadeToggle(3000);
});
```

Try it yourself »

jQuery Sliding Methods

With jQuery you can create a sliding effect on elements.

jQuery has the following slide methods:

- slideDown()
- slideUp()
- slideToggle()

---

jQuery slideDown() Method

The jQuery slideDown() method is used to slide down an element.

Syntax:

```
$(selector).slideDown(speed,callback);
```

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the sliding completes.

The following example demonstrates the slideDown() method:

Example

```
$("#flip").click(function(){
    $("#panel").slideDown();
});
```

The University Of Sheffield.

Animations

- Animations are done by manipulating css features
  - e.g. you move an element to the right by assigning style="left:250px" to an element

jQuery Animations - The animate() Method

The jQuery animate() method is used to create custom animations.

Syntax:

```
$(selector).animate({params},speed,callback);
```

The required params parameter defines the CSS properties to be animated.

The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.

The optional callback parameter is a function to be executed after the animation completes.

The following example demonstrates a simple use of the animate() method; it moves a element to the right, until it has reached a left property of 250px:

Example

```
$("button").click(function(){
    $("div").animate({left: '250px'});
});
```

please note that despite these elements tend to use element types (e.g. $("div")) you will normally use an id selector (e.g. `$("#mysquare")`)

The University Of Sheffield.

Animation of multiple properties

jQuery animate() - Manipulate Multiple Properties

Notice that multiple properties can be animated at the same time:

Example

```
$("button").click(function(){
    $("div").animate({
        left: '250px',
        opacity: '0.5',
        height: '150px',
        width: '150px'
    });
});
```

Is it possible to manipulate ALL CSS properties with the animate() method?

Yes, almost! However, there is one important thing to remember: all property names must be camel-cased when used with the animate() method: You will need to write paddingLeft instead of padding-left, marginRight instead of margin-right, and so on.

Also, color animation is not included in the core jQuery library.
If you want to animate color, you need to download the Color Animations plugin from jQuery.com.

jQuery animate() - Using Relative Values

It is also possible to define relative values (the value is then relative to the element's current value). This is done by putting += or -= in front of the value:

Example

```
$("button").click(function(){
    $("div").animate({
        left: '250px',
        height: '+=150px',
        width: '+=150px'
    });
});
```

jQuery animate() - Uses Queue Functionality

By default, jQuery comes with queue functionality for animations.

This means that if you write multiple animate() calls after each other, jQuery creates an "internal" queue with these method calls. Then it runs the animate calls ONE by ONE.

So, if you want to perform different animations after each other, we take advantage of the queue functionality:

Example 1

```
$("button").click(function(){
    var div = $("div");
    div.animate({height: '300px', opacity: '0.4'}, "slow");
    div.animate({width: '300px', opacity: '0.8'}, "slow");
    div.animate({height: '100px', opacity: '0.4'}, "slow");
    div.animate({width: '100px', opacity: '0.8'}, "slow");
});
```

The University Of Sheffield.

Callbacks

- Javascript is event based
- This means that some operations are performed when an event happens
  - For example when the user clicks a button
- Many languages do not have this
- There are two types of events
  - When the user performs an action
    - e.g. click on button
  - When part of the program completes
    - e.g. $(document).ready(…)
- Javascript handles these situations with callback functions

The University Of Sheffield.

Callbacks (ctd)

- A callback is a function that is called only when the action is performed AND finished
  - So if the event takes time to finish, the callback is called only at the end
- As we will see, callbacks can be used to perform and action at the end of an asynchronous action
  - i.e. an action that is executed outside the current flow
    - e.g. an action that requires to go to the server to get data
- It is important to understand the concept of callbacks as they are the main feature of the javascript that we will see this semester

The University Of Sheffield.

Example

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<button id='mybutton'>press me</p>
<script>
    var myVariable=0;
    $(#mybutton).onclick(function() { myVariable=100});
    alert(myVariable);
</script>
</body>
</html>
```

- The callback function is called only when the button is pressed
- So when this page is loaded the alert will show 0 because the button has not been pressed

The University Of Sheffield.

Example (ctd)

- If we want the alert to print the value of 100 we must write:
- i.e. the action (the alert) must be inserted into the callback!

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<button id='mybutton'>press me</p>
<script>
    var myVariable=0;
    $(#mybutton).onclick(function() {
          myVariable=100;
          alert(myVariable);
          });
</script>
</body>
</html>
```

The University Of Sheffield.

Using this in Javascript

http://www.quirksmode.org/js/this.html

- Javascript has a powerful element: this
- 'This' refers to the element "owner" of the function, i.e. the element that holds the function

Owner

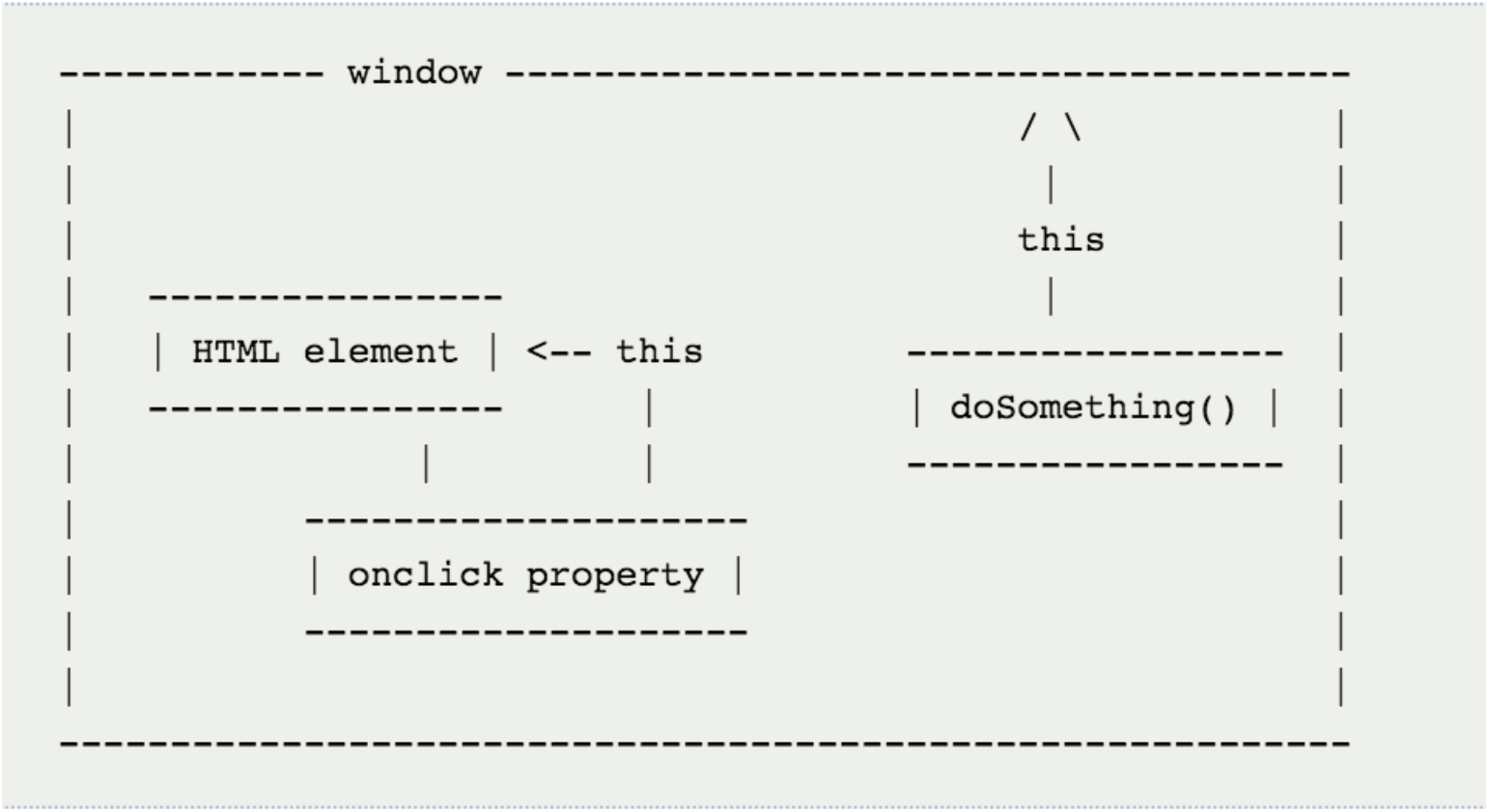
The question that we'll discuss for the remainder of the page is: What does `this` refer to in the function `doSomething()`?

```
function doSomething() {
   this.style.color = '#cc0000';
}
```

In JavaScript `this` always refers to the "owner" of the function we're executing, or rather, to the object that a function is a method of. When we define our faithful function `doSomething()` in a page, its *owner* is the page, or rather, the `window` object (or global object) of JavaScript. An `onclick` property, though, is owned by the HTML element it belongs to.

The University Of Sheffield.

Function owned by window



If we execute `doSomething()` without any more preparation the `this` keyword refers to the window and the function tries to change the `style.color` of the window. Since the window doesn't have a `style` object the function fails miserably and produces JavaScript errors.

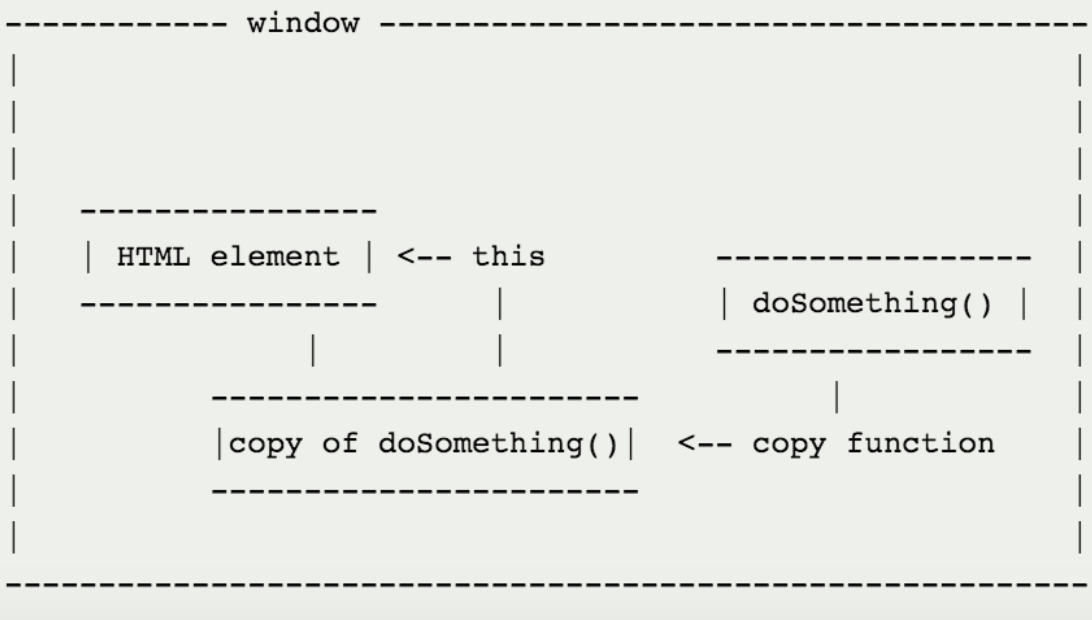
The University Of Sheffield.

Copying

So if we want to use `this` to its full extent we have to take care that the functio that uses it is "owned" by the correct HTML element. In other words, we have to *copy* the function to our onclick property. Traditional event registration takes care of it.

```
element.onclick = doSomething;
```

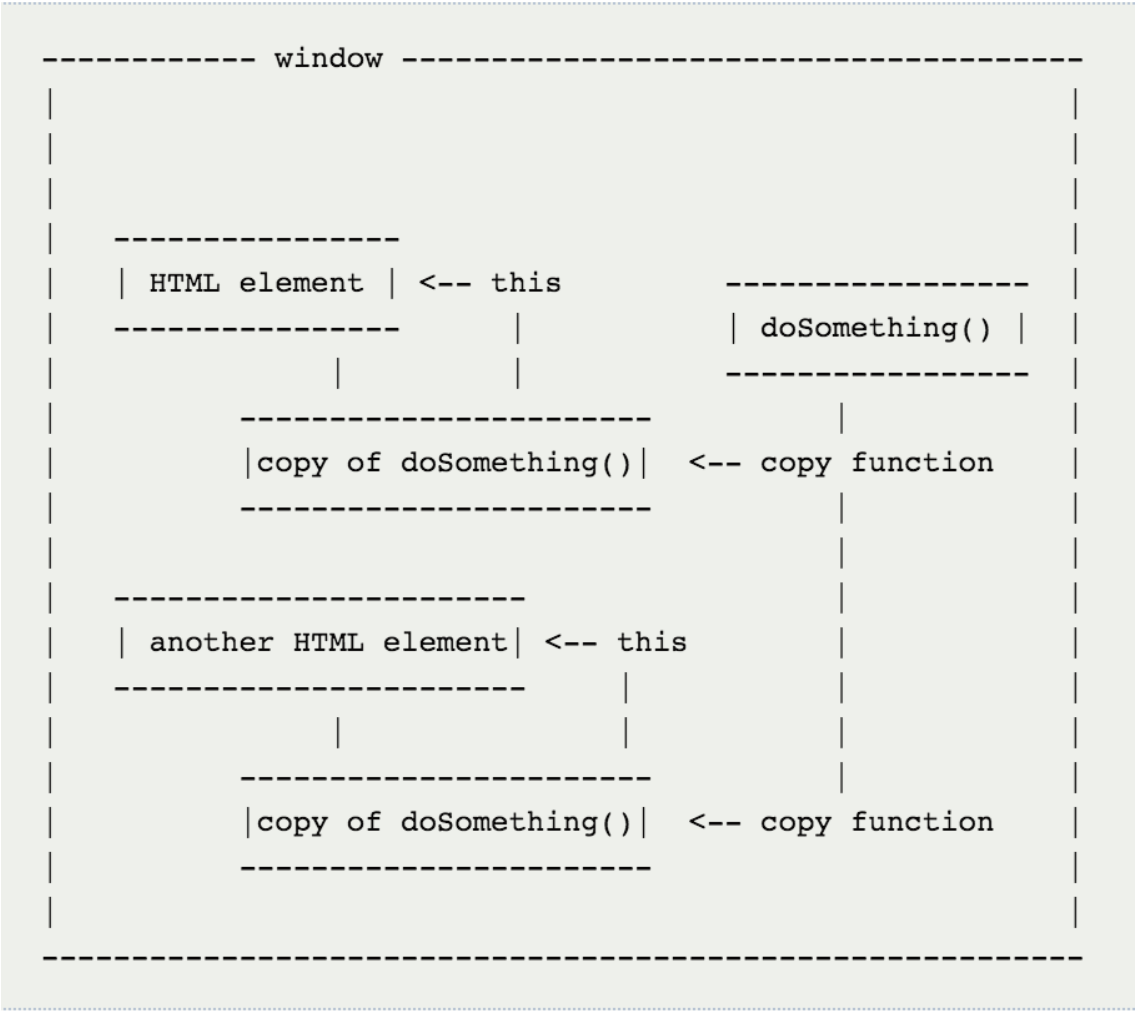
The function is copied in its entirety to the `onclick` property (which now becomes a method). So if the event handler is executed `this` refers to the HTML element and its `color` is changed.



The University Of Sheffield.

```
<script>
    function doSomething() {
       this.style.color = '#cc0000';
    }
    anotherElement.onclick = doSomething;
</script>
```

The trick is of course that we can copy the function to several event handlers. Each time `this` will refer to the correct HTML element:



Thus you use `this` to the fullest extent. Each time the function is called, `this` refers to the HTML element that is currently handling the event, the HTML element that "owns" the copy of `doSomething()`.

The University Of Sheffield.

Referring

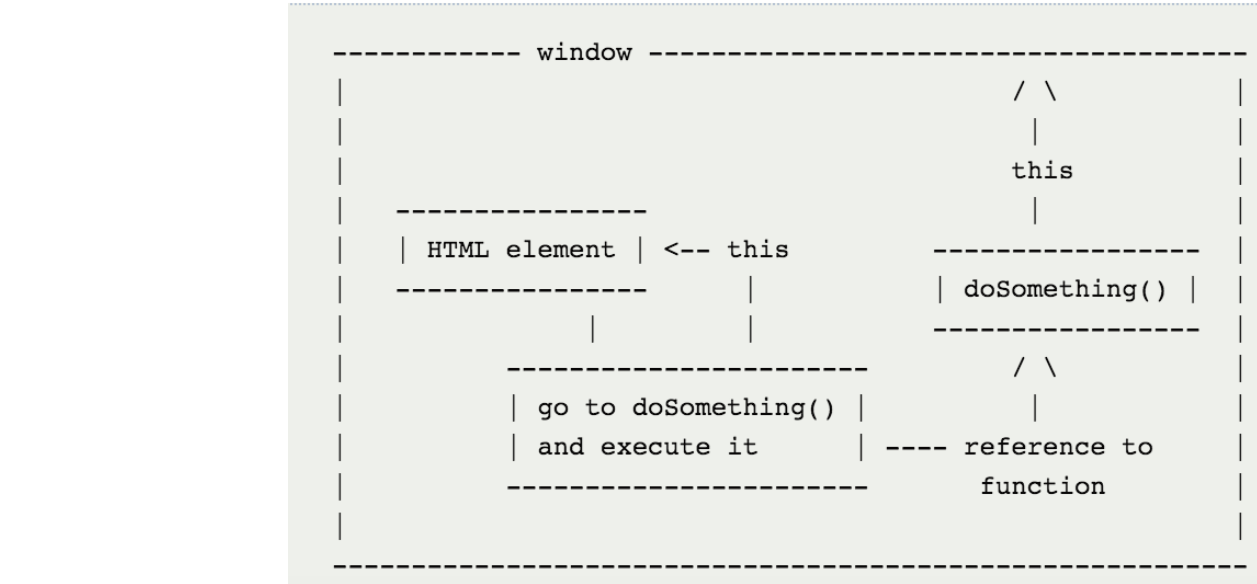
However, if you use inline event registration

```
<element onclick="doSomething()">
```

you do not copy the function! Instead, you refer to it, and the difference is crucial. The `onclick` property does not contain the actual function, but merely a function call:

```
doSomething();
```

So it says “Go to doSomething() and execute it.” When we arrive at `doSomething()` the `this` keyword once again refers to the global window object and the function returns error messages.



The University Of Sheffield.

Always use
element.onclick= functionName;

DO NOT Insert parameters or parenthesis

i.e. do NOT write
element.onclick= functionName();

The latter will call the function and expect it to return a function!

For example:
element.onclick= assignAction();

```
function assignAction(){
    if (someConditions)
        return printNo;
    else return printYes;
    }

function printNo(){
    alert("no");
    }

function printYes(){
    alert("yes");
    }
```

If you want to use `this` for accessing the HTML element that is handling the event, you must make sure that the `this` keyword is actually written into the `onclick` property. Only in that case does it refer to the HTML element the event handler is registered to. So if you do

```
element.onclick = doSomething;
alert(element.onclick)
```

you get

```
function doSomething()
{
        this.style.color = '#cc0000';
}
```

As you can see, the `this` keyword is present in the `onclick` method. Therefore it refers to the HTML element.

But if you do

```
<element onclick="doSomething()">
alert(element.onclick)
```

you get

```
function onclick()
{
        doSomething()
}
```

The University Of Sheffield.

Timing events

http://www.w3schools.com/js/js_timing.asp

- The window object allows execution of code at specified time intervals.
  - These time intervals are called timing events.
- The two key methods to use with JavaScript are:
  - setTimeout(function, milliseconds)
    - Executes a function, after waiting a specified number of milliseconds.
  - setInterval(function, milliseconds)
  - Same as setTimeout(), but repeats the execution of the function continuously.

The University Of Sheffield.

setTimeout

The setTimeout() Method

```
window.setTimeout(function, milliseconds);
```

The **window.setTimeout()** method can be written without the window prefix.

The first parameter is a function to be executed.

The second parameter indicates the number of milliseconds before execution.

Example

Click a button. Wait 3 seconds, and the page will alert "Hello":

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
    alert('Hello');
}
</script>
```

How to Stop the Execution?

The clearTimeout() method stops the execution of the function specified in setTimeout().

```
window.clearTimeout(timeoutVariable)
```

The **window.clearTimeout()** method can be written without the window prefix.

The clearTimeout() method uses the variable returned from setTimeout():

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

If the function has not already been executed, you can stop the execution by calling the clearTimeout() method:

Example

Same example as above, but with an added "Stop" button:

```
<button onclick="myVar = setTimeout(myFunction, 3000)">Try it</button>

<button onclick="clearTimeout(myVar)">Stop it</button>
```

SetInterval

The setInterval() Method

The setInterval() method repeats a given function at every given time-interval.

```
window.setInterval(function, milliseconds);
```

The **window.setInterval()** method can be written without the window prefix.

The first parameter is the function to be executed.

The second parameter indicates the length of the time-interval between each execution.

This example executes a function called "myTimer" once every second (like a digital watch).

Example

Display the current time:

```
var myVar = setInterval(myTimer, 1000);

function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
```

How to Stop the Execution?

The clearInterval() method stops the executions of the function specified in the setInterval() method.

```
window.clearInterval(timerVariable)
```

The **window.clearInterval()** method can be written without the window prefix.

The clearInterval() method uses the variable returned from setInterval():

```
myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```

Example

Same example as above, but we have added a "Stop time" button:

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

<button onclick="clearInterval(myVar)">Stop time</button>

<script>
var myVar = setInterval(myTimer, 1000);
function myTimer() {
    var d = new Date();
    document.getElementById("demo").innerHTML = d.toLocaleTimeString();
}
</script>
```

The University Of Sheffield.

Now take two minutes

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<button id='mybutton'>press me</p>
<script>

    function myFunction() {
        ix=123;
    }

    var ix=0;
    myFunction();
    alert(ix);

</script>
</body>
</html>
```

Question:
- what is the role of the button?
-what value will be display by the alert?

Write the answer down (including a short explanation)

The University Of Sheffield.

Now take two minutes

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<script>

    function myFunction() {
        ix=123;
    }

    var ix=0;
    setTimeout(myFunction, 3000);
    alert(ix);

</script>
</body>
</html>
```

Question: what value will be display by the alert?

Write the answer down (including a short explanation)

The University Of Sheffield.

Now take two minutes

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<script>

    function myFunction() {
        ix=ix*2;
        alert(ix);
    }

    var ix=1;
    setTimeout(myFunction, 3000);


</script>
</body>
</html>
```

Question: what value will be display by the alert?

Write the answer down (including a short explanation)

The University Of Sheffield.

Now take two minutes

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<script>

    function myFunction() {
        ix=ix*2;
        setTimeout(myFuction2, 3000);
        alert(ix);
    }

    function myFunction2() {
        ix=ix=1234;
    }
    var ix=1;
    setTimeout(myFunction, 3000);
</script>
</body>
</html>
```

Question:

- how many alerts will you get?
- what value will be display by the alert?

Write the answer down (including a short explanation)

The University Of Sheffield.

Now take two minutes

```
<!DOCTYPE html>
<html>
<body>
<h1>My Callback example</h1>
<script>

    function myFunction() {
        ix=ix*2;
        setInterval(myFunction2, 3000);
        alert(ix);
    }

    function myFunction2() {
        ix=ix=1234;
    }
    var ix=1;
    setTimeout(myFunction, 3000);
</script>
</body>
</html>
```

Question:

- how many alerts will you get?
- what value will be display by the alert?

Write the answer down (including a short explanation)

The University Of Sheffield.

Cookies

http://www.w3schools.com/js/js_cookies.asp

What are Cookies?

Cookies are data, stored in small text files, on your computer.

When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user.

Cookies were invented to solve the problem "how to remember information about the user":

- When a user visits a web page, his name can be stored in a cookie.
- Next time the user visits the page, the cookie "remembers" his name.

Cookies are saved in name-value pairs like:

```
username=John Doe
```

When a browser request a web page from a server, cookies belonging to the page is added to the request. This way the server gets the necessary data to "remember" information about users.

Create a Cookie with JavaScript

JavaScript can create, read, and delete cookies with the **document.cookie** property.

With JavaScript, a cookie can be created like this:

```
document.cookie="username=John Doe";
```

You can also add an expiry date (in UTC time). By default, the cookie is deleted when the browser is closed:

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC";
```

With a path parameter, you can tell the browser what path the cookie belongs to. By default, the cookie belongs to the current page.

```
document.cookie="username=John Doe; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

Read a Cookie with JavaScript

With JavaScript, cookies can be read like this:

```
var x = document.cookie;
```

document.cookie will return all cookies in one string much like: cookie1=value; cookie2=value; cookie3=value;

Change a Cookie with JavaScript

With JavaScript, you can change a cookie the same way as you create it:

```
document.cookie="username=John Smith; expires=Thu, 18 Dec 2013 12:00:00 UTC; path=/";
```

The old cookie is overwritten.

Delete a Cookie with JavaScript

Deleting a cookie is very simple. Just set the expires parameter to a passed date:

```
document.cookie = "username=; expires=Thu, 01 Jan 1970 00:00:00 UTC";
```

The Cookie String

The document.cookie property looks like a normal text string. But it is not.

Even if you write a whole cookie string to document.cookie, when you read it out again, you can only see the name-value pair of it.

If you set a new cookie, older cookies are not overwritten. The new cookie is added to document.cookie, so if you read document.cookie a
you will get something like:

cookie1=value; cookie2=value;

The University Of Sheffield.

Setting and reading cookies

```
function setCookie(cname, cvalue, exdays) {
    var d = new Date();
    d.setTime(d.getTime() + (exdays*24*60*60*1000));
    var expires = "expires="+d.toUTCString();
    document.cookie = cname + "=" + cvalue + "; " + expires;
}

function getCookie(cname) {
    var name = cname + "=";
    var ca = document.cookie.split(';');
    for(var i=0; i<ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0)==' ') c = c.substring(1);
        if (c.indexOf(name) == 0) return c.substring(name.length, c.length);
    }
    return "";
}

function checkCookie() {
    var user = getCookie("username");
    if (user != "") {
        alert("Welcome again " + user);
    } else {
        user = prompt("Please enter your name:", "");
        if (user != "" && user != null) {
            setCookie("username", user, 365);
        }
```

The University Of Sheffield.

Local Storage

- With HTML5, there is another way of storing data in a browser which is more powerful and more in line with Javascript as a language
  - Local Storage
- With local storage, web applications can store data locally within the user's browser.
  - Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance.
- Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.
- Local storage is per origin (per domain and protocol). All pages, from one origin, can store and access the same data
  - So no one else can read it

The University Of Sheffield.

Local and session storage

- HTML local storage provides two objects for storing data on the client:
  - window.localStorage - stores data with no expiration date
  - window.sessionStorage - stores data for one session (data is lost when the browser tab is closed)
- Before using local storage, check browser support for localStorage and sessionStorage

The localStorage Object

The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.

Example

```
// Store
localStorage.setItem("lastname", "Smith");
// Retrieve
document.getElementById("result").innerHTML = localStorage.getItem("lastname");
```

The example above could also be written like this:

```
// Store
localStorage.lastname = "Smith";
// Retrieve
document.getElementById("result").innerHTML = localStorage.lastname;
```

The syntax for removing the "lastname" localStorage item is as follows:

```
localStorage.removeItem("lastname");
```

The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.