Client Side Scripting

in HTML5 pages

Version 2013/2014

Sources

- W3Scools, *http://www.w3schools.com*
- Support de cours “Client side scripting”, Laurent Haan, *http://www.haan.lu/*
- JS Tutorial, Gilles Everling, *http://foxi.ltam.lu/COURS/cliss/*
- JavaScript Kit, *http://www.javascriptkit.com/javatutors/arraysort.shtml*
- jQuery, *http://jquery.com/*

Author

- Robert Fisch

Thanks to all students of the class T1IF1/2012-2013 as well ass the class T1IF2/2013-2014 for detecting and telling me about all errors and mistakes that sneaked off to this document.

The present document is under permanent development depending on your contributions. Ideas, hints and suggestions can be submitted to Robert Fisch who will take care to integrate them as soon as possible.

Table of contents

1. Getting started

1.1. Hello world

Let's start with some fact:

- JavaScript is a programming language which is executed in the browser, thus on the client-side.
- JavaScript code is being embedded into HTML files using the **<script>** tag.
- The **<script>** element either contains scripting statements, or it points to an external script file through the src attribute.
- The **<script>** tag may occur anywhere inside the **<head>** or the **<body>** of a document.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Hello world</title>
        <script src="jquery-1.7.2.min.js"></script>
    </head>
    <body>
        <script>
            alert("Hello world");
        </script>
    </body>
</html>
```

This <script> tag loads an external file called jquery-1.7.2.min.js

This <script> tag executes the contained instructions. In this example a text message box is being displayed using the alert method.

Text must be written between single or double quotes.

Notes

- Whether the **<script>** tag is being added to the **<head>** or the **<body>** of a document always depends on the needs to be satisfied.

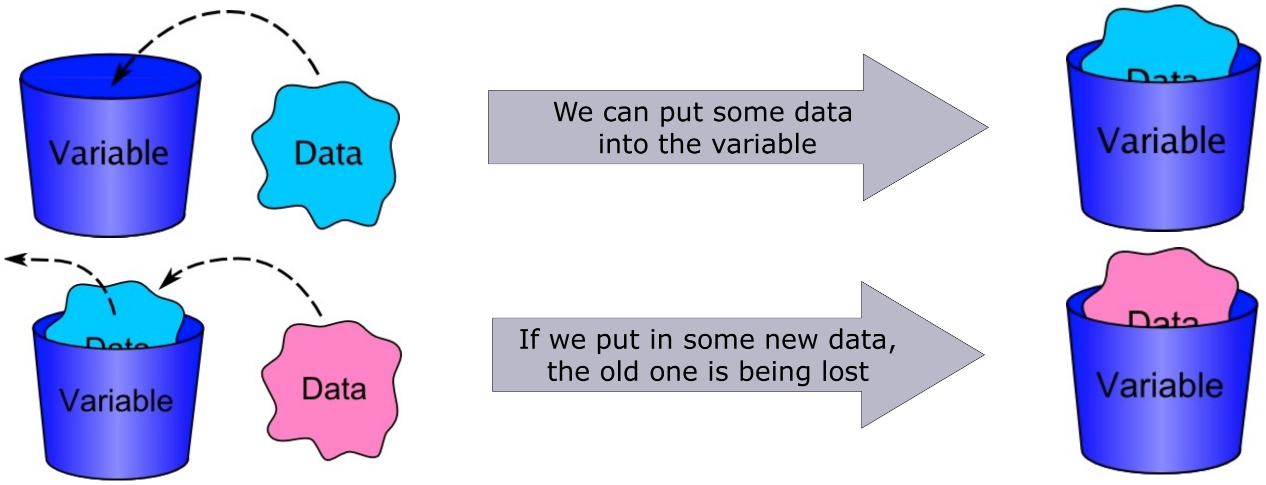
Practice

1. Write a single HTML page which display the message "My first JS script" inside a message box.
2. Write a HTML page which loads the file **hello.js**. This file contains the code to display the message "Hello world" inside a message box.

1.2. Defining variables

Just few facts about variables:

- A variable is a container which can hold some information.
- In order to access it, a variable has a ***name***. This name can be chosen by the developer, but it has to follow some standard rules:
  - it must start with a letter,
  - it must not contain specials signs,
  - it is generally being written in lowercase.



We can put some data into the variable

If we put in some new data, the old one is being lost

Example

In the code, we use the word `var` in order to define a variable.

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Defining variables</title>
     </head>
     <body>
          <script>
               var firstname = "Bob";
               alert("Hello " + firstname);
          </script>
     </body>
</html>
```

The = operator is used to put some data into the variable.

Text can be combined using the + operator.

Practice

1. Try the example given above.
2. Write a single HTML document with a script in which you define your name and your town and which then pops up a message box with the message like this: “Hi *Peter* from *New York*” (The italic words must be replaced by your values of course …)

1.3. Getting user input

The previous example would be much nicer if the user could enter his name by itself. In order to achieve this task, we have to make our script prompt the user for his name. In JavaScript there exists already a method called **`prompt`**, which we are going to use.

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Getting user input</title>
     </head>
     <body>
          <script>
               var firstname = prompt("Please enter you name","");
               alert("Hello " + firstname);
          </script>
     </body>
</html>
```

The first parameter is the displayed message.

The second parameter is the pre-filled text to display.

Practice

1. Write a single HTML document with a script that asks the user for it's name and town and which then pops up a message box with the message like this: “Hi *Peter* from *New York*”.

1.4. Working with numbers

Let's go:

- In one of the previous chapters, we defined variables which contained some text. In this chapter, we are going to work with numbers.
- In JavaScript, you can also calculate with number using the standard operators like **+**, **-**, ***** and **/**.

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Working with numbers</title>
     </head>
     <body>
          <script>
               var n1 = 7;
               var n2 = 3;
               var result = n1+(n2*n2);
               alert("The result is " + result);
          </script>
     </body>
</html>
```

`n1` and `n2` are initialized with constant values.

`result` is initialized with an expression.

Practice

1. Write a HTML document with a script that displays the square of the number 33.
2. Write a HTML document with a script that displays the result of “2+3*4”.

Combined practice

3. Write a HTML document with a script that asks the user for a number and which displays the square of the entered number.
4. Write a HTML document with a script that asks the user for two numbers and which displays the sum of the entered numbers.

   Why isn't your programm working as expected? Detect the problem and solve it!
5. Write a HTML document with a script that asks the user for his birth year and which tells him his age.

   If you want your script to work even in the upcoming years, you can use the following line of code to get the actual year into the variable **`thisYear`**:

   ```
   var thisYear = (new Date()).getFullYear();
   ```

1.5. Taking decisions

Read all the text until the bottom of the page before starting with the practice!

Practice

1. Write a HTML document with a script that asks the user for two numbers and which displays the smallest one.

   Which number is smaller? 19 or 7? Detect the problem and solve it!

2. Write a HTML document with a script that asks the user for his birth year and which tells him if he is of full age or not.
3. Write a HTML document with a script that asks the user to enter the mark of his last exam and which tells him if he passed it or not.

If you have read attentively the above exercises, you must have noticed that in each of them the script has to take a decision. Decisions are base upon questions that can be answered with either “yes” or “no”.

1. Is the first number smaller than the second one?
2. Is he aged 18 or more?
3. Is the mark greater than or equals 30 points?

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Taking decisions</title>
     </head>
     <body>
          <script>
               var number = prompt("Please enter a number","");
               if (number>0)
               {
                    alert(number + " is positive");
               }
               else
               {
                    alert(number + " is negative");
               }
          </script>
     </body>
</html>
```

The first block of instructions is executed if the answer to the question is “yes”.

The `else` block is not mandatory.

The second block of instructions is executed if the answer to the question is “no”.

The used structure is called: if-statement

Now you should be able to resolve the problems stated above.

1.6. Taking multiple decisions

In all the above exercises, we had to distinguish only ***two*** cases, but what to do if there are more? As a mater of fact, it is possible to combine multiple if-statements.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Taking multiple decisions</title>
    </head>
    <body>
        <script>
            var number = prompt("Please enter a number","");
            if (number===0)
            {
                alert(number + " is zero");
            }
            else
            {
                if (number>0)
                {
                    alert(number + " is positive");
                }
                else
                {
                    alert(number + " is negative");
                }
            }
        </script>
    </body>
</html>
```

The second block of the outer if-statement contains another if-statement.

In order to save space and increase the readability of the code, this is generally written like this.

```
            if (number===0)
            {
                alert(number + " is zero");
            }
            else if (number>0)
            {
                alert(number + " is positive");
            }
            else
            {
                alert(number + " is negative");
            }
```

Do you spot the difference? As a matter of fact, one pair of “{ ..}” has ben omitted and the interior IF statement is indented in the same way than the exeterior.

Go on reading the next chapter ...

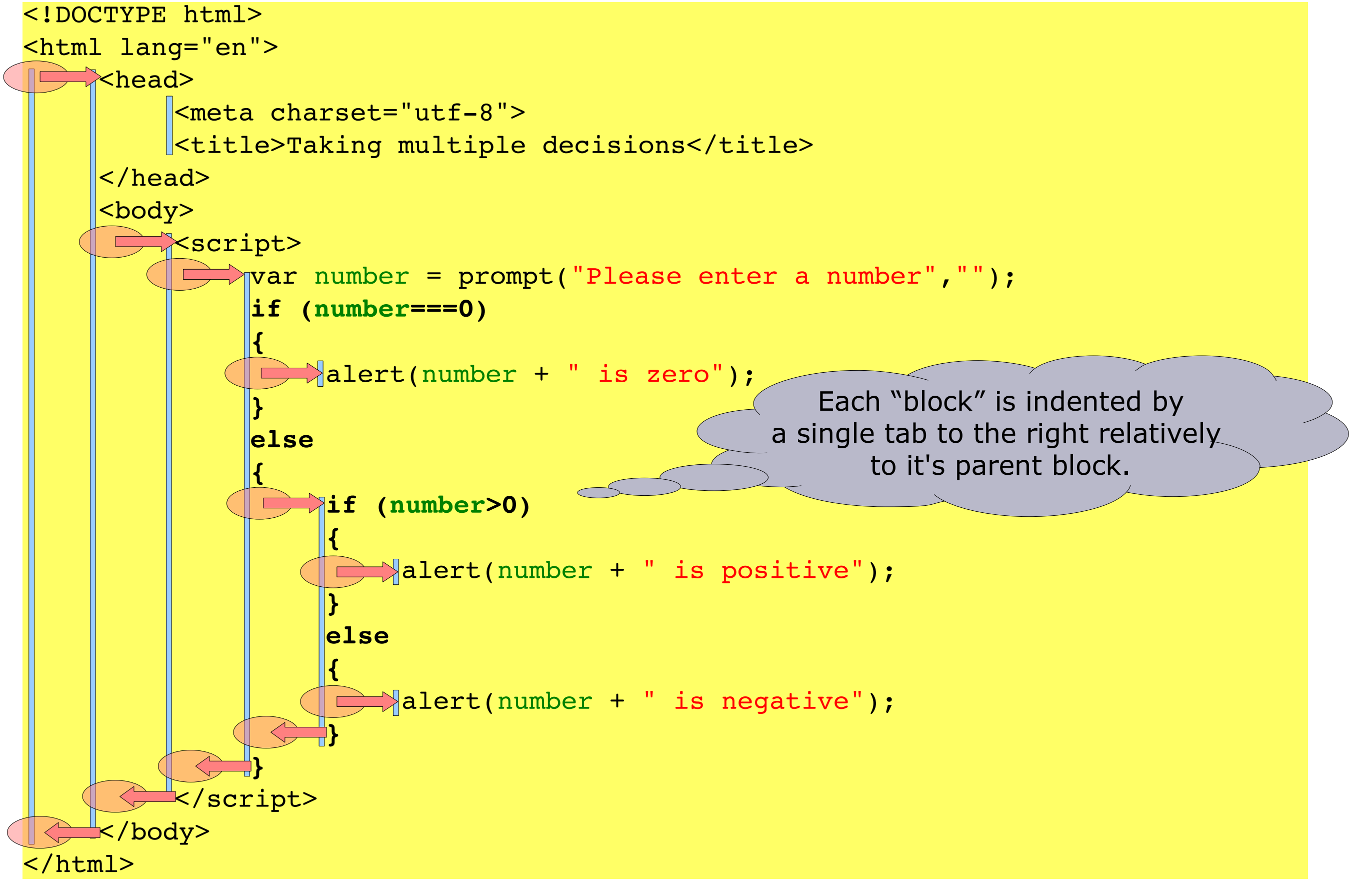
1.7. Writing readable code

Did you understand the previous examples? Was the code easy to read?

It generally should be the case!

In fact, all the code in the present document is correctly indented.

Example



```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Taking multiple decisions</title>
    </head>
    <body>
        <script>
            var number = prompt("Please enter a number","");
            if (number===0)
            {
                alert(number + " is zero");
            }
            else
            {
                if (number>0)
                {
                    alert(number + " is positive");
                }
                else
                {
                    alert(number + " is negative");
                }
            }
        </script>
    </body>
</html>
```

Each “block” is indented by a single tab to the right relatively to it's parent block.

Hint

In nearly all editors you can select multiple rows at once and use the **`<Tab>`** keys to shift the code to the right. Using **`<Shift>+<Tab>`**, you generally can shift all the rows to the left.

Practice

1. Review all your document you wrote until now and check your indentation. Correct it if necessary.

1.8. Getting user confirmation

If you need the user to confirm something, you can use the **`confirm`** method.

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Getting user confirmation</title>
     </head>
     <body>
          <script>
               var answer = confirm("Do you want me to say hello?");
               if (answer)
               {
                    alert("Hello");
               }
               else
               {
                    alert("Goodbye");
               }
          </script>
     </body>
</html>
```

The result returned by the `confirm` method is either `true` or `false`, so you can use it immediately in an `if` statement.

Practice

1. Using Borca's formula, the ideal weight of a men is:

   weight (kg) = (size (cm) – 100) * 0.9

   For women, the formula is slightly different:

   weight (kg) = (size (cm) – 100) * 0.85

   Write a HTML document with a script that asks the user for it's size and whether he is male (using the **`confirm`** method) and returns him his ideal weight using Borca's formula.

1.9. Comparison operators

In order to compare thinks, we can use one of the following operators:

<	less than
<=	less or equal than
>	more than
>=	more or equal than
===	equal than (strict)
!==	not equal than (strict)

Practice

1. Write a HTML document with a script that asks the user for ***three*** numbers and which displays the biggest one.
2. Write a HTML document with a script that asks the user for ***two*** numbers called “a” and “b” and which display the result of the division $\frac{a}{b}$.
3. Write a HTML document with a script that asks the user for ***two*** exam marks and which indicates whether the average is sufficient or not.
4. Write a HTML document with a script that asks the user for ***three*** numbers and which displays these three numbers in ascending order.
5. Write a HTML document with a script that asks the user for a single number and which displays the absolute value of it.

2. Interacting with HTML and CSS

2.1. *Interacting with HTML objects*

As you probably know, HTML is responsible for the structure and data of a web page, whereas CSS is used to format this data. The objets, also called DOM elements, of an HTML document can be accessed in JavaScript using the **`document.getElementById(...)`** method.

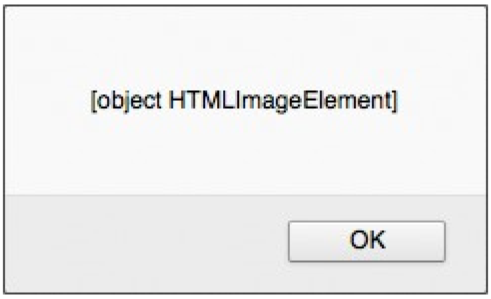
Example

Let's define an ID for this image.
The ID must be unique inside the page.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Interacting with HTML objects</title>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish">
        <script>
            var fish = document.getElementById("fish");
            alert(fish);
        </script>
    </body>
</html>
```

We now put the fish inside a
variable called fish in order to “speak” to it.

Did you notice the output?



[object HTMLImageElement]

As you can see, the variable fish is an object, to be more precise, it's an HTMLImageElement.

Practice

1. Find out of what type the **`body`** is.
2. Get the precise type of a paragraph.
3. What is the type of a **`button`**?

2.2. Interacting with CSS properties

Yes, your are completely right. Knowing how to get access to an HTML object without knowing how to do something with it is useless and boring. Fortunately, JavaScript allows to access – this means to read and also to modify – the CSS properties of such an element. As a matter of fact, an HTML object also owns non-CSS properties, but we are not going to work with them right now.

Example

We need to define the dimensions of the image explicitly

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Interacting with CSS properties</title>
     </head>
     <body>
          <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
                style="width:50px; height:30px;">
          <script>
               // get the fish
               var fish = document.getElementById("fish");
               // double the width of the fish
               fish.style.width="100px";
          </script>
     </body>
</html>
```

This is a comment!

CSS properties do use units, so we need to tell it that the new width is "100px".

CSS properties are being accessed via the style property of an object.

Practice

1. Find out what happens if you forget in the above example to specify the unit!
2. Modify the above example so that the fish is not being stretched, meaning that the height if the image has also to be changed.
3. Write a new HTML document with a script that asks the user for the dimensions (width and height) the fish should be displayed in.
4. Write a new HTML document with a script that asks the user for the width the fish should be displayed in. It has to calculate the height by itself so that the image is not being stretched.
5. If you set the CSS property "position" to the value "absolute", you can use the properties "left" an "top" to change the position of the image. Modify the script so that the fish appears at the following positions:
   1. bottom left
   2. top right
   3. bottom right
   4. absolute center

2.3. *Text to number transformation*

Right now, we are able to read CSS properties of DOM elements an we can even modify them. Unfortunately most of these properties contain units, e.g. **`50px`**

In oder to be able to *calculate* with the values of the CSS properties, we need to get rid of the units. For this, we are going to use the following two functions:

- **`parseInt(<value>)`** Function to convert a text to an integer number.
- **`parseFloat(<value>)`** Function to convert a text to a decimal number.

Hint

If you don't know what a function is, please read the next chapter …

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Text to number transformation</title>
     </head>
     <body>
          <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
                style="width:50px; height:30px; position:absolute; left:100px;">
          <script>
               // get the fish
               var fish = document.getElementById("fish");
               // get the fish's width
               var fishWidth = fish.style.width;
               // display the width
               alert(fishWidth);
               // transform the width into a number
               fishWidth = parseInt(fishWidth);
               // display the width again
               alert(fishWidth);
          </script>
     </body>
</html>
```

This line displays “50px”.

This line cuts of the unit “px”.

This line displays “50”.

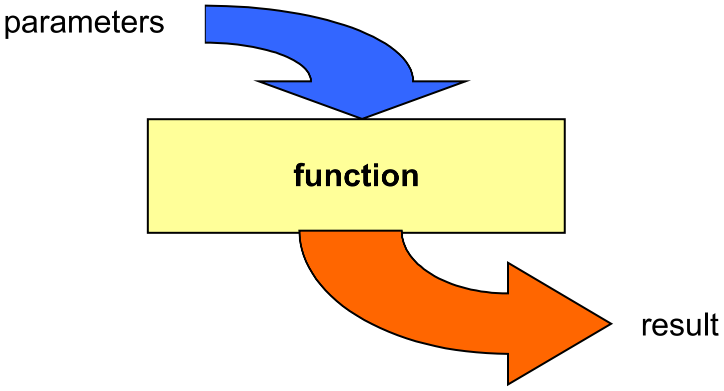
Practice

1. Create a new HTML page with a script that:
   - defines a variable with an integer value and another variable with a decimal value,
   - displays both values,
   - transforms their values using **`parseFloat`** and displays the results and
   - finally transforms their values using **`parseInt`** and displays the results.

   Compare the different results. What did you notice?

2.4. Functions

A function, also called sub-routine, is a small, mostly independent, program. The developer can pass data to the function by specifying parameters. Most functions do something with these parameters and then return a result.



Let's take the line from the example above:

```
fishWidth = parseInt(fishWidth);
```

Notes

- **parseInt** is the name of the function
- **(fishWidth)** is the parameter. As a matter of fact the function **parseInt** has only one parameter.
- The result of the call to **parseInt(fishWidth)** is then stored back into the variable **fishWidth**.

Hints

We already used some other functions!

- **alert(<text>)**

  Function with one parameter to display a text. This function does not *return* something.
- **prompt(<text>,<text>)**

  Function with two parameters to query the user for an input. The input of the user is *returned* as text.
- **document.getElementById(<id>)**

  Function with one parameter that *returns* the DOM element with a given ID.

For the moment we are fine with *using existing* functions. Later on, we are going to write our own ones.

2.5. Reducing the code size

The previous example was this one:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Text to number transformation</title>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
              style="width:50px; height:30px; position:absolute; left:100px;">
        <script>
            // get the fish
            var fish = document.getElementById("fish");
            // get the fish's width
            var fishWidth = fish.style.width;
            // display the width
            alert(fishWidth);
            // transform the width into a number
            fishWidth = parseInt(fishWidth);
            // display the width again
            alert(fishWidth);
        </script>
    </body>
</html>
```

If you want to, you may combine things:

- Get the width of the fish immediately into a variable:

  ```
  var fishWidth = document.getElementById("fish").style.width;
  ```

- Get the *parsed* width of the fish immediately into a variable:

  ```
  var fishWidth = parseInt(document.getElementById("fish").style.width);
  ```

Practice

1. Write a new HTML document that contains the fish image as well as a script that asks the user for how many pixels the image should be moved to the right.
2. What happens in the above example if the user enters a negative value?
3. Modify your program in order to make the fish move down if the user enters a negative value.
4. You can get the dimensions of a page with **`window.innerWidth`** and **`window.innerHeight`**. Now go and write a new HTML document that contains the fish image as well as a script that positions the image at the center of the page.

3. Reacting on user input

3.1. Clicking on an image

If we want do develop interactive dynamic pages, we need to react on the event generated by the user. The most known and popular event is the one that is triggered if a user clicks on something.

Example 1

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Clicking on an image #1</title>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
              onclick="alert('Aua');">
    </body>
</html>
```

Each time the user clicks on the fish, the message "Aua" is being displayed.

Now if we want to do more than only executing a single instruction, it is not really convenient to add all of them to the IMG tag. The better way to achieve this is being shown here:

Example 2

Functions are being written inside the HEAD of a document.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Clicking on an image #2</title>
        <script>
            function fishClick()
            {
                alert("Aua");
                alert("You clicked me!");
            }
        </script>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
              onclick="fishClick()">
    </body>
</html>
```

The funciton `fishClick` has no parameters and does not return any result. In this example displays two messages.

The `onclick` event is bound to a function called `fishClick`. This is a self-made function!

Practice

1. Write a new HTML document that contains the fish image. If the user clicks on the image, the fish has to be moved 20 pixels to the right.
2. Add another image! This one has to be moved down by 20 pixels it is being clicked on!

3.2. Clicking on a button

As already said, we can add the onclick-event to many different HTML elements. A very popular and intuitive is the button.

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Clicking on a button</tit
          <script>
               function fishMoveDown()
               {
                    // get the fish
                    var fish = document.getElementById("fish");
                    // get the Y coordinate of the fish
                    var fishTop = parseInt(fish.style.top);
                    // increase it by 10
                    fishTop=fishTop+10;
                    // move the fish down
                    fish.style.top=fishTop+"px";
               }
          </script>
     </head>
     <body>
          <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
               style="position:absolute; top:0px; left:100px;">
          <button onclick="fishMoveDown()">move down</button>
     </body>
</html>
```

This function moves the fish 10 pixels downwards. It is executed each time the user clicks on the button.

Don't forget to add the unit!

Practice

1. Write a new HTML document that contains the fish image as well as a script that positions the image at the center of the page.
2. Add two buttons to your page. They have to be positioned in the upper left corner. One button makes the fish move to the left, the other one makes it move to the right. Each movement has to be 30 pixels in distance.
3. In the previous task the fish can be moved *outside* the page. Modify your script so that the user can no longer move the fish outside the page.
4. Add two more buttons to the *upper right* corner of your page. These buttons should move the fish up or down by a certain distance which is prompted to the user. Pay attention that the fish is not leaving the visible area of the page!

3.3. Well structured HTML documents

Each document has to be well structured. Besides a correct indentation, this increases it readability.

Example

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Clicking on a button</title>
          <script>
               // Functions are being writing inside the HEAD.
          </script>
     </head>
     <body>
          <!-- Put your HTML tags here -->
          <script>
               // Code that has to be executed immediately when the page
               // is loaded has to be put AFTER the HTML tags.
          </script>
     </body>
</html>
```

3.4. Well structured JS code

To most of the JavaScript code we write, we can apply the “input-processing-output”, short IPO[1], method.

Example

```
function fishMoveDown()
{
     // INPUT
     // get the fish
     var fish = document.getElementById("fish");        INPUT
     // get the Y coordinate of the fish
     var fishTop = parseInt(fish.style.top);

     // PROCESSING
     // increase it by 10                               PROCESSING
     fishTop=fishTop+10;

     // OUPUT
     // move the fish down                              OUTPUT
     fish.style.top=fishTop+"px";
}
```

This increases not also the readability of the code but make it also easier to debug.

[1] http://en.wikipedia.org/wiki/IPO_Model

3.5. Reading from textfields

Until now, the only way to get textual input from a user was to use the **prompt** function, which pops up a window where the user can enter some text. This is not really convenient especially if the user will have to enter several values. As a matter of fact, he cannot return to a previous input in order to correct it. Another way of letting the user enter data is using a textfield, also referred to as “input box”.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Reading from textfields</title>
        <script>
            function fishMoveDown()
            {
                // get the fish
                var fish = document.getElementById("fish");
                // get the Y coordinate of the fish
                var fishTop = parseInt(fish.style.top);
                // get the distance
                var distance = parseInt(document.getElementById("distance").value);
                // increases the fish's top position
                fishTop=fishTop+distance;
                // move the fish down
                fish.style.top=fishTop+"px";
            }
        </script>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="a fish"
            style="position:absolute; top:40px; left:100px;">
        <input type="text" id="distance">
        <button onclick="fishMoveDown()">move down</button>
    </body>
</html>
```

An input box allows a user to enter some “text”, so we need to transform it to a number using parseInt.

Practice

1. Write a web page asking the user for the dimensions (length, width & height) of a box using the **prompt** function and which calculates and displays it's volume. Use the string "m\xB3" to indicate the unit “m³”.
2. Write a web page asking the user for the dimensions (length, width & height) of a box using input boxes and which calculates and display it's volume.
3. The formula to calculate the body mass index (BMI) of a person is the following:

$$\mathrm{BMI} = \frac{\mathrm{mass(kg)}}{(\mathrm{height(m)})^2}$$

   Write a page which gives the user the possibility to calculate it's BMI.

3.6. Writing to HTML elements

Outputting every single result using a message dialog (**`alert`**) can be very annoying, especially if multiple messages pop up. A far more elegant way is to write result into HTML objects. This can be achieved using the **`innerHTML`** property.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Writing to HTML elements</title>
        <script>
            function calculate()
            {
                // INPUT
                var number = parseFloat(document.getElementById("number").value);
                // PROCESSING
                var result = number*number;
                // OUTPUT
                document.getElementById("result").innerHTML=
                    number + "m\xB2 = "+ result;
            }
        </script>
    </head>
    <body>
        Number: <input type="text" id="number" value="33">
        <div id="result">
            Result will be displayed here …
        </div>
        <button onclick="calculate()">calculate</button>
    </body>
</html>
```

Use `parseFloat` because the number may be decimal.

This modifies the content of the DIV. You here also place HTML code!

In order to "talk" to the DIV, it needs an id.

Practice

1. Rewrite the practice (2) from the previous chapter using a DIV tag as output.
2. Rewrite the practice (3) from the previous chapter using a DIV tag as output.
3. Write a little calculator which allows the user to enter two numbers. The user can clic on one of the four buttons representing the four operations: add, subtract, multiply by and divide by. The result is being writing into a div.
4. Copy and modify the previous script so that the result is not being written into a DIV but into another textfield, so that copying the result into another application is easier.

3.7. Pressing keys

As for clicking on something, there exists two important events that deal with key interaction:

- **onkeydown** fires when a key is being held down
- **onkeyup** fires when a key is being released
- **onkeypress** fires if the key has been pressed

Both event can be set on different objects, but for now we just use them on the **body** tag.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Pressing keys</title>
        <script>
            function showKey(event)
            {
                alert("keyCode = "+event.keyCode + "\n" +
                      "charCode = "+event.charCode);
            }
        </script>
    </head>
    <body onkeydown="showKey(event)">
    </body>
</html>
```

Here we have to specify "event" if we want to know what key has been pressed.

The event object has two attributes we use.

Here we have to specify "event" if we want to know what key has been pressed.

You can chose the name of the function by yourself.

Notes

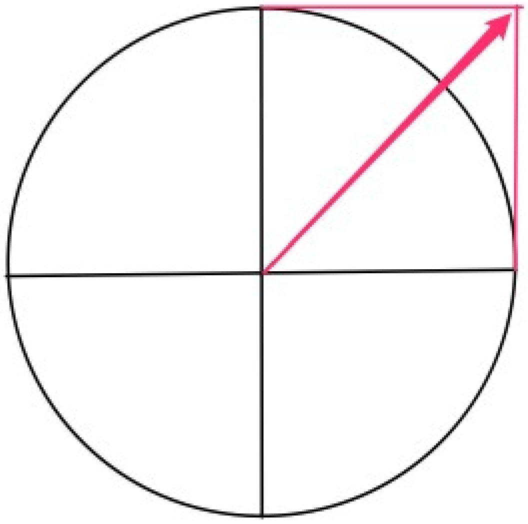
- **charCode** has a value for alphanumeric keys, e.g. "A", "-", "4", ...depending on the used event.
- **keyCode** has a value for control keys, e.g. tab, arrow keys, ...

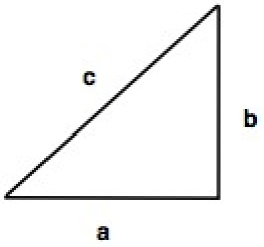
Practice

1. Test the example given above with the three indicated events. Notice the differences!
2. Write a new HTML document that contains the fish image as well as a script that allows the fish to move 10 pixels to the right if the user presses the key "p" and 10 pixels to the left he pressed the key "o". *Hint: Use the script above to determine the code you have to react on!*
3. Write a new HTML document that contains the fish image as well as a script that allows the fish to move in any direction using the arrow keys.
4. Modify your previous script so that the fish cannot move out of the window.
5. Your fish can only move into one direction at once. Wouldn't it be nice if he could also follow the diagonals? In order to react on two keys, you have to remember what key has been pressed (**onkeydown**) an release its state if the key is being released (**onkeyup**). Change your script so that you can move your fish also following the diagonals! *Hint: the fish cannot move at the same time to the right and to the left...*

3.8. *Doing some maths*

After the last exercise, you probably noticed, that the fish's movement is faster if he follows the diagonals. This is because the vertical movement combined with the horizontal movement results in a greater distance relative to the starting point of the fish:





Using Pythagorus and the fact that $a=b$, it is possible to resolve the equation and calculate the correct vertical (b) and horizontal (a) movements so that their combination results in the same distance as if we had moved the fish into a single direction:

$$
\begin{aligned}
&c^2=a^2+b^2\\
&\Leftrightarrow c^2=2\cdot a^2\\
&\Leftrightarrow a^2=\frac{c^2}{2}\\
&\Leftrightarrow a=\sqrt{\frac{c^2}{2}}
\end{aligned}
$$

Now let's put this into code:

```
var c = 10;
var a = Math.sqrt(Math.pow(c,2)/2);
```

Notices

- The function **`Math.sqrt(...)`** allows us to calculate the square root of number.
- The function **`Math.pow(... , ...)`** allows us to calculate the power of a given number.

Of course you are not provided the direct solution to the problem. It's up to you to put it into the right context and use it at the right place.

If you need other mathematical functions, please take a look at the following list:

http://www.w3schools.com/jsref/jsref_obj_math.asp

4. Looping around

Until now, our scripts were executed either immediately after or while loading the page or when the user clicked on something. The scripts where quite simple and always executed from the top to the bottom, but sometimes this is not enough.

Now let's face the following simple mathematical problem: *Test is a number is prime or not.*

As far as you should know, a prime number is a number that has only two dividers: 1 and the number itself.

A simple, but yet not optimized way is to test the number to analyze against all numbers smaller or equal itself and to count the number of found dividers.

Example

Let's take the number "3":

- Is 1 a divider of 3? (yes)
- Is 2 a divider of 3? (no)
- Is 3 a divider of 3? (yes)

As you can see, only the first and the last questions are answered with yes, so we count two of them. This means, that "3" is prime.

Probably you already noticed that if we want to check a number like "1001", we would need to do 1001 tests! That would be a real huge number of lines to write, so let's simplify things by using a loop, because each test is in fact nearly the same!

In this document, we are going to see two different types of loops. Although both of them offer the same main functionally, namely to repeat one or more instructions a certain number of times, they both have different characteristics.

- the FOR loop

  This type of loop uses a counter. You can use it each time you exactly know how many steps you need to execute.
- the WHILE loop

  This type of loop uses a condition to determine if it should continue or not. It can be used in any condition!

In the next chapters we are going to test if a number is prime of not using both loop types...

4.1. The FOR loop

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>The FOR loop</title>
    </head>
    <body>
        <script>
            // INPUT
            // define the number to test
            var number = parseInt(prompt("What number do you want to test?","8"));
            // initialize the counter
            var count = 0;

            // PROCESSING
            // loop through all number between 1 and number
            for (var i=1 ; i<=number ; i++)
            {
                // test if i is a divider of number
                if (number % i === 0)
                {
                    // increment the counter
                    count++;
                }
            }

            // OUTPUT
            // test if we found exactly two dividers
            if (count===2)
            {
                alert("prime");
            }
            else
            {
                alert("not prime");
            }
        </script>
    </body>
</html>
```

The variable i is being defined.
Its initial value is 1.
The loop continues as long
as its value is less or equal
the value of `number`.
`i` is incremented each step by one.

The `%` operator, also
called "modulo" operator,
calculates the rest of the
division `number/i`.

Practice

1. Copy the above example and modify it so that the user can enter the number to test in a textfield.
2. Write a HTML document with a script that calculates the sum from A to B. A and B are two integer numbers which have to be entered via textfields by the user.

4.2. *The WHILE loop*

The example given here represent the same program as the one for the FOR loop but it uses a WHILE loop instead.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>The WHILE loop</title>
    </head>
    <body>
        <script>
            // INPUT
            // define the number to test
            var number = parseInt(prompt("What number do you want to test?","8"));
            // initialize the counter
            var count = 0;

            // PROCESSING
            // loop through all number between 1 and number
            var i=1;
            while(i<=number)
            {
                // test if i is a divider of number
                if (number % i === 0)
                {
                    // increment the counter
                    count++;
                }
                i++;
            }

            // OUTPUT
            // test if we found exactly two dividers
            if (count===2)
            {
                alert("prime");
            }
            else
            {
                alert("not prime");
            }
        </script>
    </body>
</html>
```

The loop is executed as long as the condition is true. Once is is false, the loop stop.

You probably notice that both versions have a lot in common. The main difference between the two is, that the number of iterations of a WHILE loop is not necessarily predictable, whereas this statement is mainly true for a FOR loop.

4.3. *Practicing loops*

Practice

1. What type of loop would you recommend? Could you give the pseudo code of these sentences? ("Pseudo code" is code that is more close to english, so no compiler would understand it but we, humans, do so).
   - Shot 20 times at the goal and count the number of hits.
   - Play football until the sun goes down.
   - Do not close the umbrella while it rains!
   - Check the speed of the next 100 cars that pass by.
   - Reload the web page as long as the news about the iPhone 7 is not being displayed.
2. Write a script that applies the following calculation to a number given by the user until the result is 1:
   - if the number is even, divide it by 2
   - else, meaning if the number is odd, multiply it by 3 and add 1.

   Your script has to count the steps needed for the initial number to reach 1.

   Example: (10) → 5 → 16 → 8 → 4 → 2 → 1 = 6 steps
3. Write a HTML document with a script that allows the user to calculate the average of **x** numbers. **x** is also provided by the user.

   Example:
   - How many number do you want to enter? 3
   - Please enter number 1: 10
   - Please enter number 2: 12
   - Please enter number 3: 15
   - The average is 12.3
4. Write a HTML document with a script that allows the user to enter two integer numbers and which calculates and displays the greatest common divider (FR: PGCD)

   Example:
   - Please enter the first number: 100
   - Please enter the second number: 350
   - The greatest common divider of 100 and 350 if 50.
5. Write a HTML document with a script that ask the use to enter a number from the interval [-10,10]. The scripts must repeat to ask for a number until the user enters a correct value!

4.4. *The timer*

Besides loop structures, there is also another possibility for repeating events at certain intervals: the timer.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>The timer</title>
        <script>
            function count()
            {
                // get the counter element
                var counter = document.getElementById("counter");
                // get it's value
                var value = parseInt(counter.innerHTML);
                // increase it
                value = value + 1;
                // write the new value back
                counter.innerHTML=value;
            }

            function start()
            {
                stop();
                countTimer = setInterval("count()",1000);
            }

            function stop()
            {
                clearInterval(countTimer);
            }

            function reset()
            {
                document.getElementById("counter").innerHTML="0";
            }
        </script>
    </head>
    <body>
        <button onclick="start()">start</button>
        <button onclick="stop()">stop</button>
        <button onclick="reset()">reset</button>
        <div style="font-size:10em" id="counter">0</div>
        <script>
            var countTimer = setInterval("count()",1000);
        </script>
    </body>
</html>
```

Make sure the timer is not running anymore before starting a new one.

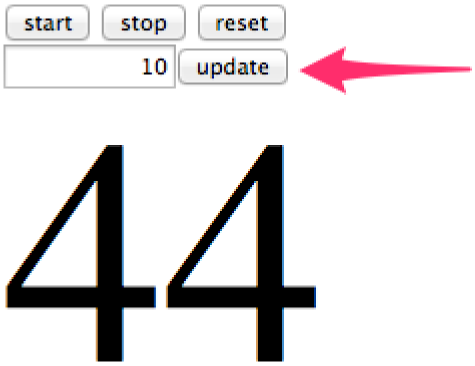
This stops (clears) the timer.

It is important that the counter is started after the definition of the div#counter in order that the latter exists!

The function `count()` is executed each 1000ms = 1 s.

Practice

1. Try the previous example and answer the following questions:
   - What do you need to change in order to count half seconds instead of seconds?
   - Why is it important to make sure that the previous timer is cleared before starting a new one? Try to comment out the call to **`stop()`** and try what happens when you click "Start" button when the timer already is started.
2. Make a copy of the previous page and so that the timer is not started automatically when the page is loaded.
3. Make another copy of the previous page an modify it in order that the user can change the speed of the counter by entering an integer number into a text field.



4. Create a HTML document containing the fish image, which has to move each 200ms 10 pixels to the right.
5. Copy the previous page and modify it, so that the fish changes his direction (it moves then from the left to the right) as soon as it's next step would be outside the window. Then is changes its direction again before touching the left border … and so on.

6. In the previous exercise, the fish only does a horizontal movement. Copy your page and modify it in order to allow the fish also to do a vertical movement! Remember that vertical and horizontal movements are independent of each other.

For advanced only

7. When loading the page, the fish is displayed in its center. When one of the arrow keys is pressed, the fish starts to move into that direction. It continues its movement even when the key is being released. To halt the fish, the same key or the opposite key has to be pressed. Vertical and horizontal movements are independent, so that the fish can also do diagonal movements. When reaching a border, the fish "bounces" back and continues to move on.

5. Important things

5.1. *Generating random numbers*

There are plenty of situations where applications need to generate a random number. Okay, computers cannot really generate a random number, but, based on different parameter settings, like the time, the mouse movement, the pressed keys, they can generate pseudo random numbers.

For this, we can use the the following function:

```
var randomNumber = Math.random()
```

In fact, this generates a floating point number from the interval [0,1[. Sadly, this is not what we need the most. Mostly we need to have a integer number from any interval, let's say from the interval [min,max]. In that case, the correct formula is:

```
var randomNumber = Math.floor(Math.random()*(max-min+1)) + min;
```

Practice

1. Write a script, that generates 50 random floating point number from the interval [0,1[and displays them in a container in the page.
2. Write a script, that generates 50 random integer number from the interval [0,10] and displays them in a container in the page.
3. Write a script, that generates 50 random integer number from the interval [5,15] and displays them in a container in the page.
4. Write a script, that generates 50 random integer number from the interval [-10,10] and displays them in a container in the page.
5. Write a script, that generates 50 random integer number from the interval [**min**,**max**] and displays them in a container in the page. **min** and **max** are two values which are entered by the user itself. The number are generated after he clicked on a button.
6. Write a script, that simulates to throw 10 dices (DE: Würfel) and displays their values in a container in the page.
7. Write a page that moves the well know fish image each time a button is pressed to a random position on the screen. Pay attention that the fish is never being positioned outside of the screen!
8. Write a page that moves the well know fish image each 10 milli seconds to a random position on the screen. Pay attention that the fish is never being positioned outside of the screen!

5.2. *Getting element dimensions*

We already noticed, that the attributes **`style.width`** and **`style.height`** of a DOM element only have a valid value if the respective style attributes have been set using CSS definitions. However, there are several situations were the dimensions of an element are dynamic and not know while coding.

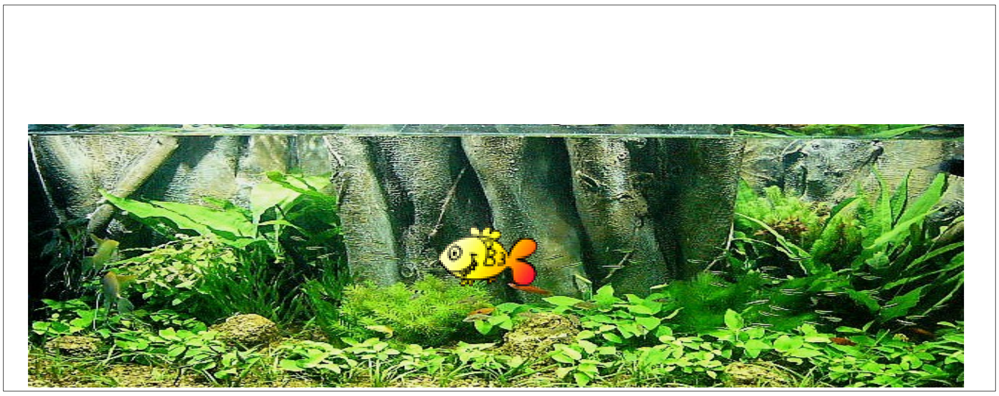
Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Getting element dimensions</title>
        <script>
            function getSize()
            {
                var width  = document.getElementById("area").offsetWidth;
                var height = document.getElementById("area").offsetHeight;
                alert("Width = "+width+"\nHeight = "+height);
            }
        </script>
    </head>
    <body>
        <button onclick="getSize()">getSize</button>
        <div id="area" style="background-color:green; position:absolute;
                          left:10px; top:100px; right:10px; bottom:10px;"></div>
    </body>
</html>
```

The width and height of the entire element, including borders

Practice

1. Copy the above example and run it. Try to change the dimensions of your browser window and click again on the button! Do you notice the change?
2. Write a page that simulates an aquarium. The aquarium 500 pixel height, sticks to the bottom of the page and is spaced by 20 pixels to the right, left and bottom border. When the page is being resized, the aquarium's width will change as well. Add all necessary elements to your page to mage the fish image swim inside the aquarium.



6. Structuring data in lists

6.1. *Visual lists*

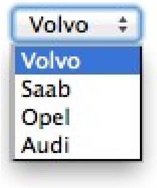
In opposite to non visual lists, visual lists can be displayed on the screen. In HTML we call them “select lists”, but sometimes they are also referred to as “combo box” or “select box”.

Depending on the set attributes, the look of such a list may change.

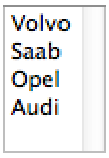
Example:

```
<select id="cars">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="opel">Opel</option>
  <option value="audi">Audi</option>
</select>
```

```
<select id="cars" size="5">
  <option value="volvo">Volvo</option>
  <option value="saab">Saab</option>
  <option value="opel">Opel</option>
  <option value="audi">Audi</option>
</select>
```



“combo box”



“select box”

Practice

1. Create a new document with a combo that contains 7 fruits. The value of each entry must be identical to it's text but with lower case letters only.
2. Create a copy of your previous document and convert the lists into a select box. What HTML attribute do you have to specify to allow the selection of multiple items?
3. Modify your previous document so that the the fifth fruit is always preselected if the page is being reloaded (hard reload!).
4. Using the **`innerHTML`** property of the select list DOM element, write a page with a script that displays a combo box with the numbers from 1 to 100.
5. Using the same technique, write a page with a select box that contains all odd numbers (DE: ungerade Zahlen) between 1 and 100.

That sound cool, but what can we do using a select box? As usual, let's try this example:

Example:

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Visual lists</title>
          <script>
               function updateLog()
               {
                    // get object
                    var cars = document.getElementById("cars");
                    // generate output code
                    var code = "There are "+cars.length+" cars in the list<br>";
                    code += "<ul>";
                    for(var i=0 ; i<cars.length ; i++)
                    {
                         code += '<li>Text: "'+cars.options[i].text+'" '+
                                 'with value: "'+cars.options[i].value+'" </li>';
                    }
                    code += "</ul>";
                    code += "Actually the entry on index "+
                            cars.selectedIndex+" is selected,<br>";
                    code += "which is the text: "+
                             cars.options[cars.selectedIndex].text+"<br>";
                    code += "with the value: "+
                            cars.options[cars.selectedIndex].value+"<br>";
                    // write to the output element
                    document.getElementById("log").innerHTML=code;
               }
          </script>
     </head>
     <body>
          <select id="cars" onChange="updateLog()">
               <option value="volvo">Volvo</option>
               <option value="saab">Saab</option>
               <option value="opel">Opel</option>
               <option value="audi">Audi</option>
          </select>
          <div id="log"></div>
          <script>
               updateLog();
          </script>
     </body>
</html>
```

The number of elements inside the list.

The index of the selected item. It's value is -1 if no option is selected.

The function to be executed when the selection changes.

You may want to read the following page as well:

- http://de.selfhtml.org/javascript/objekte/options.htm

Practice

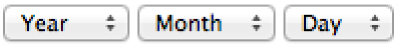
6. Find an answer to the following questions:
   a) What represents the attribute **`length`** of a select box?
   b) What does **`selectedIndex`** stand for?
   c) What's the index of the first item of a list?
   d) What's the index of the last item of a list?
   e) How can we access the **`text`** of an item at a specified position?
   f) What's the difference between the **`text`** and the **`value`** of an item?
7. What is the **`value`** being reported by JavaScript if you omit the **`value="..."`** in the HTML code?
8. Using the **`innerHTML`** property of the select list DOM element, write a page with a script that displays a combo box with the numbers from 1 to 100. If the user selects the entry with the value 7, the message "7 up!" has to be displayed.
9. The following line gives you the actual year:

   ```
   var thisYear = (new Date()).getFullYear();
   ```

   Write a page with a combo box that contains the last 30 years. The first entry of the list has to be "Year" with an empty value!
10. Make a copy of your previous page and add another combo box that represents the month, thus a number of the interval [1,12]. The first entry of the list has to be "Month" with an empty value!
11. Make a copy of your previous page and add a third combo box for the days. This combo box has to be filled with the correct number of days as soon as a year and a month has been selected.

    In order to determine if a year is a leap year (DE: Schaltjahr) or not, please use the following function:

    ```
    function isLeapYear(year)
    {
        return ((year%400===0)||((year%4===0)&&(year%100!==0)));
    }
    ```



12. On the previous written page, you will notice, that the selection of the "days" is lost each time the user selects a new month or year. Modify your script so that the selected index of the list of days is maintained even if it is being regenerated!

13. Write a page that contains a text box, a button and a list box. Each time the user clicks on the button, the content of the text box is being added to the list box.
14. Make a copy of the previous document and add an output area to your page. From now on, we assume that the user only enters numbers into the text box. Each time the user clicks on the button, the content of the text box is being added to the list box and the scripts calculates the minimum and maximum of the numbers contained in the list.
15. Make a copy of the previous document. Each time the user clicks on the button, the content of the text box is being added to the list box and the scripts calculates the sum and the average of the numbers contained in the list.

6.2. Non visual lists

Non visual lists, as the name already says, cannot be displayed inside the page. They act as data structure in the background, but, if we need or want to, we can display their content by programmatically put it somewhere in the page. We call them usually “array”.

There are three ways to define and fill an array in JavaScript:

Example:

```
//1: Regular:
var myCars=new Array();       // this defines an empty array/list
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";

//2: Condensed:
var myCars=new Array("Saab","Volvo","BMW");

//3: Literal:
var myCars=["Saab","Volvo","BMW"];
```

As opposed to a list, the elements of an array can only hold a single item. Remember: The items of a listbox have a **`text`**, which is visible to the user, and a **`value`**, which is not visible to the user. Using arrays, we could create an array of an array, but for the moment this is out of scope of the present document.

Of course, you want to know what kind of things we can do using arrays. So just jump to the next page and test the given example!

Example 1

```
<!DOCTYPE html>
<html lang="en">
     <head>
          <meta charset="utf-8">
          <title>Non visual lists</title>
          <script>
               var numbers = new Array();

               function addSorted()
               {
                    var number =
                              parseInt(document.getElementById("number").value);
                    // add the number to the array
                    numbers.push(number);
                    // sort the array
                    numbers.sort();
                    // calculate the sum of all
                    var sum = 0;
                    for(var i=0 ; i<numbers.length ; i++)
                    {
                         sum += numbers[i];
                    }
                    // calculate the average
                    var avg = sum / numbers.length;
                    // output everything
                    var code = '<ul>';
                    for(var i=0 ; i<numbers.length ; i++)
                    {
                         code += '<li>'+numbers[i]+'</li>';
                    }
                    code += '</ul>Sum = '+sum+'<br>';
                    code += 'Average = '+avg+'<br>';
                    // display
                    document.getElementById("output").innerHTML=code;
               }
          </script>
     </head>
     <body>
          Number: <input type="text" id="number">
          <button onClick="addSorted();">Add and sort</button>
          <div id="output"></div>
     </body>
</html>
```

The method push is used to add a new item at the end of the list.

The method sort() is used to sort the list in lexicographical order.

The attribute length tells us how much items are in the list.

Use square brackets to access the items of the list.

This example allows the user to enter numerical values and display them sorted in lexicographical order on screen. It also calculates and displays the sum and average of the items of the non visual list.

Practice

1. Try to enter the following numbers: 10, 1, 4

   Can you explain why they are not ordered correctly?

2. Goto to the following page and read it!

   http://www.javascriptkit.com/javatutors/arraysort.shtml

   Modify the above example so that the numbers are displayed no in numerical order!

3. Add a button to the above example which gives the user the possibility to shuffle the elements of the list.

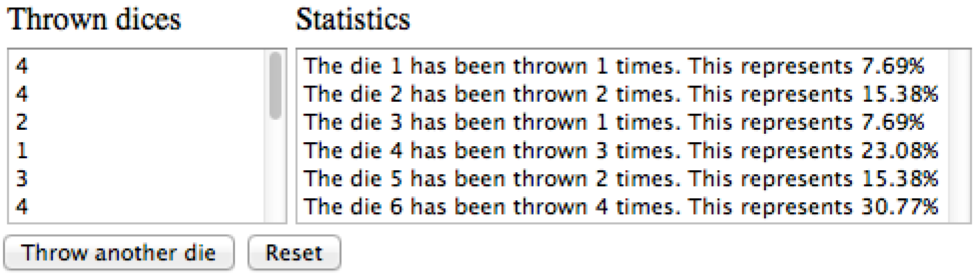
4. Modify the previous exercise by adding the minimum and maximum value to the output.

5. Modify the previous exercise so that positive numbers are displayed in green and negative numbers in red.

6. Make a copy of the previous exercise and add the code to display even numbers in blue.

7. Next, add all necessary to not only show the numbers in the output area, but also inside a listbox! Take care to make sure that the size of the listbox is always the same as the number of the elements contained in your array of numbers!

8. Create a new page that looks like this:



Thrown dices

4
4
2
1
3
4

Statistics

The die 1 has been thrown 1 times. This represents 7.69%
The die 2 has been thrown 2 times. This represents 15.38%
The die 3 has been thrown 1 times. This represents 7.69%
The die 4 has been thrown 3 times. This represents 23.08%
The die 5 has been thrown 2 times. This represents 15.38%
The die 6 has been thrown 4 times. This represents 30.77%

   - Each time the button "Throw another die" is pressed, a value from the interval [1,6] is being added to the left list and the content of the right list is being re-calculated.
   - Clicking on the "Reset" button empties the left list and re-calculates the content of the right list as well.

Now to start, let's assume we want to make an image follow a predefined path. For example, let's take something ease, like a ring.

In the next example, you will find a script, that make our well-known fish image follow a perfect circle around the center of the page. Each position where the fish has to move to is pre-calculated immediately when the page is loaded. Next, a timer is in charge of making the fish move.

Example 2

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>Non visual lists</title>
        <script>
            function moveFish()
            {
                var fish = document.getElementById("fish");
                fish.style.left = x[index]-fish.offsetWidth/2+"px";
                fish.style.top = y[index]-fish.offsetHeight/2+"px";
                // increase the index
                index=(index+1)%x.length;
            }
        </script>
    </head>
    <body>
        <img id="fish" src="http://www.fisch.lu/f.gif" alt="fish" title="fish"
             style="position:absolute; left:0px; top:0px;">
        <script>
            // calculate the center of the page
            var cx = window.innerWidth / 2;
            var cy = window.innerHeight / 2;
            // now let's pre-calculate the positions of the image
            var x = new Array();
            var y = new Array();
            // the raidus of the circle
            var radius = 250;
            // the number of positions to pre-calculate
            var steps = 2*360;
            // pre-calculate the positions
            for(var i=0;i<steps;i++)
            {
                x[i]=cx+radius*Math.cos(i/180*Math.PI);
                y[i]=cy+radius*Math.sin(i/180*Math.PI);
            }
            // define the actual position of the fish
            var index = 0;
            // start the timer
            var movetimer = setInterval("moveFish()",10);
        </script>
    </body>
</html>
```

Read the item at the position index of the array x.

```
index=index+1;
if(index>=x.length) index=0;
```

We need one array to store the x-coordinates and another one for the y-coordinates.

Fill the arrays with values.

The non visual list objets have a bunch of interessing and very usful methods.

`.pop()`	Remove the last element of an array, and returns that element:
	`var fruits = ["Banana", "Orange", "Apple", "Mango"];`
	`var last = fruits.pop();`
	
	The result of fruit will be:
	`Banana,Orange,Apple`
	
	And the variable **`last`** will hold:
	`Mango`

`.shift()`	Remove the first item of an array, and returns that element:
	`var fruits = ["Banana", "Orange", "Apple", "Mango"];`
	`var first = fruits.shift()`
	
	The result of fruits will be:
	`Orange,Apple,Mango`
	
	And the variable **`first`** will hold:
	`Banana`

`.toString()`	Converts the array into a string.
	`var fruits = ["Banana", "Orange", "Apple", "Mango"];`
	`fruits.toString();`
	
	The result of fruits will be:
	`Banana,Orange,Apple,Mango`

`.reverse()`	Reverse the order of the elements in an array:
	`var fruits = ["Banana", "Orange", "Apple", "Mango"];`
	`fruits.reverse();`
	
	The result of fruits will be:
	`Mango,Apple,Orange,Banana`

7. Functions

7.1. Parameterless functions

Until this point of time, you've already used functions and you even wrote some on your own. The functions you wrote did not have any parameters and they did not return a result. They looked like this:

Name of the function

```
function moveSomething()
{
     // code to be executed
}
```

I said "*... and they did not return a result.*", but what does that mean? Actually, a function may return something, mostly a value. The above function executes some instructions, but it does not **return** anything. Here's an example of a function that returns a value:

```
function getPI()
{
     return 3.141592;
}
```

Of course, such a function could contain an infinite number of other instructions and/or structures (if statements, loops, ...). **The most important** to know is, that the function is being terminated immediately after the keyword **`return`** occurred. This means, that anything that is being written after that line of code will not be executed!

```
var a = 5;
var b = 7;

function getMin()
{
     if(a<b) return a;
     return b;
}
```

If `a` is smaller that `b`, `a` is returned and the function is terminated.

This statement is only reached if the previous `if` is false!

You probably noticed, that it is not quite nice nor well done to use global variables (in the above example **`a`** and **`b`**) inside functions. Your are right about that, so please go on and read the next chapter!

7.2. Parameterised functions

A function is actually something like a small, mostly independent program and thus it follows the IPO[2] method. The inputs of a function are called **parameters**.

This function has two parameters.
They are here called "a" and "b".

```
function getSum(a,b)     // input
{
     var sum = a + b;    // processing
     return sum;         // output
}
```

Using the parameters, we can "give" the function some values. Here is an example of how our self-created function could be tested:

```
alert(getSum(3,8));
```

Be conscious, that the parameters do not have to be numbers. They can also be entire expressions using other function:

```
alert(getSum(parseInt(prompt("Please enter a number","")),10));
```

Notes

- **prompt** is also a function. It has 2 parameters and returns the input from the user as text.
- **parseInt** is a function that takes a text a parameter and transforms (=returns) the numerical equivalent of it.

You see, functions are nothing new to you. You already used them since your very first steps in JavaScript. What is new, is that you are now able to write your own functions!

[2] input, processing, output

Practice

1. Write a page which contains and tests the following functions:
   - **`getSum`** with two parameters to calculate the sum of two numbers.
   - **`getMin`** with two parameters to determine the minimum of two numbers.
   - **`getMax`** with two parameters to determine the maximum of two numbers.
   - **`getSumBetween`** with two parameters to calculate the sum of all numbers between the two entered parameters. eg. **`getSumBetween(3,6) = 3+4+5+6 = 18`**
2. Write a page that contains at least 10 fish images positioned randomly all over the screen. If the user clicks on any of the fishes, it moves 10 pixel upwards. If the upper border is being hit by a fish, it reappears at the bottom of the screen.
3. Write a page that complies with the following constraints:
   - At the very beginning, the page contains a edit box, that allows the user to specify the number of fish images he want's to be added as well as a button.
   - After hitting that button, the edit box and the button itself will disappear and the given number of fish images will put positioned randomly all over the screen as well as a select box containing the numbers of the fishes (n° 1, n° 2, ….).
   - Using the arrow keys, the user can move around the screen the fish that is selected in the select box.
4. Write a page that contains and tests the following JavaScript functions:
   - **`getSum`** with one parameter that calculates the sum of all values of an array.
   - **`getAverage`** with one parameter that calculates the average of all values of an array.
   - **`getMin`** and **`getMax`** with one parameter that calculates the smallest, respectively the biggest value of all values of an array.
5. Write a page that contains and tests the following JavaScript functions:
   - **`generateTable`** with two parameters that generates the HTML code for a table with N rows and M columns.
   - **`generateCombobox`** with one parameter that transforms the values of an array into the corresponding HTML code of a combo box.
6. Write a page that contains at least 10 fish images positioned randomly all over the screen. A fish can be “selected” of “not selected”. A selected fish has a red border. A non selected fish has no border at all. Fishes can be selected or de-selected by clicking on them. All selected fishes can be moved using the arrow keys on the keyboard.

8. jQuery

8.1. What is "jQuery"?[3]

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript.

8.2. How to use?

In order to user jQuery, you have to include it into your page. You can either download it from the official site or user some of the many CDN sources. A good choice is Google. To inlcude that version, just put this line of code into the head of your HTML file.

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
```

8.3. The jQuery object

Each call has to be made on a special jQuery object, which is referred to as "**$**". In fact, it is something like a function. The passed parameter is a CSS selector, used in the same way as you are used to from your CSS course.

Example

Get the **<button>** element with the class '**continue**' and change its HTML to 'Next Step...'

```
$("button.continue").html("Next Step...")
```

8.4. Adding events to objets

This is the usual way you know to add events to objets.

```
<!DOCTYPE html>
<html lang="en">
      <head>
            <meta charset="utf-8">
            <title>jQuery demo</title>
            <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
            <script>
                  function clickme()
                  {
                        $("button.continue").html("Next Step...");
                  }
            </script>
      </head>
      <body>
            <button class="continue" onclick="clickme()">a button</button>
      </body>
</html>
```

[3] http://jquery.com/

While using jQuery, these events are mostly added danamically to the objects. This is done at the very beginning, so when the page has been loaded and thus when it is ready to receive user input. Anyway, the user will or should not interact with a page when it has not fully been loaded into the browser.

8.5. ***Executing a script when the page has been loaded***

We really often need to execute some instructions when the page has finished loading. This can be done by adding the following code into a page:

```
    $(document).ready(function()
    {
        // do something
    });
```

8.6. ***Back to the event-attaching***

Let's take the previous example in order to put the "onclick" event dynamically onto the button:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>jQuery demo</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
        <script>
            function clickme()
            {
                $("button.continue").html("Next Step...");
            }

            $(document).ready(function()
            {
                $("button.continue").click(clickme);
            });
        </script>
    </head>
    <body>
        <button class="continue">a button</button>
    </body>
</html>
```

Take all buttons of the class "continue" and make their "onclick" event the "`clickme`" funtion

Of course, there are, beside the **`click`**, many other events that can be attached to an object. A complete list can be found here:

http://api.jquery.com/category/events/

8.7. Anonymous functions

Now let's get back to the previous example. As a matter of fact, in the jQuery world, this would rather be written like this, thus using an “anonymous” function.

```
          <script>
               $(document).ready(function()
               {
                    $("button.continue").click(function()
                    {
                         $(this).html("Next Step...");
                    });
               });
          </script>
```

An “anonymous” function is a function that has no name.

The CSS selector “button.continue” selects **all** button of the class “continue”. The keyword “this” inside the anonymous click function refers always to the current element.

An anonymous function is a function that is embeded immediately at the place where it is being needed. A opposite to a “normal” function, anonymous function cannot be used somewhere else. As a matter of fact, it has no name we could use to call it …

Anonymous function are not something special of jQuery. They are part of the regular JavaScript. Some chapters earlier, we were working with timers. Here an example of a script that displays each 10 seconds the message “Hello world!”.

With “normal” function

```
function helloWorld()
{
     alert('Hello world!');
}

setInterval("helloWorld()",10000);
```

With “anonymous” function

```
setInterval(function() {
     alert('Hello world!');
},10000);
```

8.8. Animations

Using jQuery, it is really simple to create animations. Let's take the following code, which displays a simple button with a height of 200 pixel. Each time the button is clicked on, is heigh is slide up during a second (= 1000 milli seconds), then there is a pause of 800 ms and finally the button is being faded in duing half a second.

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>jQuery demo</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
        <script>
            $(document).ready(function()
            {
                $("button.continue").click(function()
                {
                    $(this).slideUp(1000).delay(800).fadeIn(500);
                });
            });
        </script>
    </head>
    <body>
        <button class="continue" style="height:200px">a button</button>
    </body>
</html>
```

You probaley notice, that the effets are simple concatenated one after each other. If we want to, we can add as many as we want. Let's try this one:

```
        <script>
            $(document).ready(function()
            {
                $("button.continue").click(function()
                {
                    $(this).slideUp(1000)
                           .slideDown(900)
                           .delay(800)
                           .fadeOut(500)
                           .delay(500)
                           .fadeIn(1000);
                });
            });
        </script>
```

Of course there is many many more in jQuery. The easiest way to get into it, is to google for the word “jquery xyz” where “xyz” is whatever you want to do. Let say, we would like to move an object, just google for “jQuery move”. At some point you sould find the following official jQuery page:

http://api.jquery.com/animate/

Using **`animate`** it is possible to animate a multitude of CSS properties of an object. Let's analyse one of the official examples where two buttons a programmed to make a squre object move to the left and to the right.

Example

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8">
        <title>jQuery demo</title>
        <script src="https://ajax.googleapis.com/ajax/libs/jquery/2.0.0/jquery.min.js"></script>
        <script>
            $(document).ready(function()
            {
                $("#right").click(function()
                {
                    $(".block").animate({"left": "+=50px"}, "slow");
                });

                $("#left").click(function()
                {
                    $(".block").animate({"left": "-=50px"}, "slow");
                });

        </script>
        <style>
            div {
                position:absolute; background-color:#abc;
                left:50px; width:90px; height:90px; margin:5px;
            }
        </style>
    </head>
    <body>
        <button id="left">&laquo;</button>
        <button id="right">&raquo;</button>
        <div class="block"></div>
    </body>
</html>
```

The “left” property is decreased by 50px.

Practice

1. Copy the previously given example and add two more buttons to make the square also move up and down.
2. Copy again the previously example and add everthing you need in order that:
   - the block get 10px larger when moving to the right,
   - the width of the block is reduced by 10px when moving to the left.
3. Find and ansewer to the following questions:
   1. By what instruction can we get the height of the browser window?
   2. How can we be informed about the height of the HTML document?
   3. How can we detect if the size of the browser windows has been modified?
4. Write a page that contains 10x10 buttons. All the buttons must have the same size. The 10x10 buttons have to fill up the entire page. If the browser windows is being resized, the size of the buttons has to be changed accordingly.
5. Write a page with 6 objects (button, images or a “div”) in a row. Each object is a 100x100px square. Each time the mouse moves over an object, it size must be increase to 150x150px. When the mouse leaves the element, is size is set back to 100x100px.
6. Just continue with the exercices on the following pages:
   - http://jqexercise.droppages.com/
   - http://www.coursdewebdesign.com/javascript/exercices/