Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Training Materials
Subchapter 4 – Advanced JavaScript

WP3: Code4SP Training Materials

Prepared by: CDDR

Co-funded by the
Erasmus+ Programme
of the European Union

Subchapter 4 – Advanced JavaScript

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Date and Time

You can create a new Date object by passing the string representation of a date to the Date() constructor, or by passing the individual components of a date as parameters to the Date() constructor (year, month, day, hour, minute, second, millisecond).

```
// Create a date object for today's date:
const now = new Date();
// Outputs today's date and time in standard format
console.log(now);
```

Play around with the various possibilities of Date:
https://developer.mozilla.org/de/docs/Web/JavaScript/Reference/Global_Objects/Date

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Math Operations

You can perform numerical operations using JavaScript. This can be done with the help of the + (addition), - (subtraction), * (multiplication), / (division) and % (modulus) operators.

In the example below, we use the addition operator to add two numbers:

```
var x = 10; var y = 5; var z = x + y;
```

This will give the value of z as 15.

Play around with the various possibilities of Math Operations:
https://www.w3schools.com/js/js_arithmetic.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Type Conversions

You can convert a value to a specific data type using built-in methods, such as parseInt() for converting a value to an integer, or parseFloat() for converting a value to a float.

You can also use the Number() method to convert a value to a number, or the Boolean() method to convert a value to a boolean.

In the following example we will convert a string to a number using the Number() method:

```
var x = "100"; var y = Number(x); console.log(y); // 100.
```

Play around with the various possibilities of Type Conversions:
https://www.w3schools.com/js/js_type_conversion.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Event Listeners

JavaScript Event Listeners allow you to define functions that will run when a specific event occurs on an element in the DOM.
There are a number of different events that can occur on a DOM element, such as when the element is clicked, hovered over, or even when the element's contents are changed.
To add an event listener to an element, you first need to select the element using one of the DOM selection methods, such as document.querySelector() or document.getElementById() .
Once the element is selected, you can use the addEventListener() method to attach an event listener to the element.
The addEventListener() method takes two arguments: the name of the event to listen for, and a function to run when the event occurs.
For example, to add a click event listener to a button element, you would use the following code:

```
button.addEventListener ( "click" , function ( ) { console.log ( "The button was clicked!" ) ; } );
```

Play around with the various possibilities of Event Listeners:
https://www.w3schools.com/js/js_htmldom_eventlistener.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Event Propagation

When an event occurs on an element, that event can be recognized by JavaScript and acted upon.

For example, if you click on a button on a web page, you can program JavaScript to recognize that event and take some action.
The event that occurred is a click event.
When you click on an element, the click event is fired on the element that was clicked.
The event then propagates up the DOM tree.
This means that if there is an event handler on a parent element, that event handler will be executed.
If there is an event handler on a grandparent element, that event handler will be executed.
This propagation up the DOM tree continues until the event reaches the root element of the DOM tree or until the event is stopped.

Some examples of Event Propagation:
https://developer.mozilla.org/en-US/docs/Web/API/Event/stopPropagation

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Borrowing Methods

We will use an example of an object that contains a property we need to borrow.

```
var myObject = {

someProperty: "foo",

};
```

We can borrow the someProperty property from myObject like this:

```
var myProperty = myObject.someProperty;
```

myProperty will now contain the value “foo”.

More information on Borrowing Methods:
https://www.tutorialrepublic.com/javascript-tutorial/javascript-borrowing-methods.php

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Hoisting

Hoisting is a behavior that is unique to JavaScript. It is a concept that is often misunderstood. Many developers believe that JavaScript hoists variables and function declarations to the top of the scope.
This is not entirely accurate. What JavaScript actually does is move declarations to the top of the scope, but not initialization. This can lead to some unexpected behavior.

Consider the following code:

```
var foo = 1; function bar() { if (!foo) { var foo = 10; } console.log(foo); } bar();
```

What do you think the output of this code will be?
If you guessed 10, you would be wrong. The output of this code is 1.
This is because the declaration of foo is hoisted to the top of the scope, but the initialization is not. When the if statement is executed, foo is undefined and is thus set to 10.
This can be a bit confusing, but it is important to understand how hoisting works in JavaScript. It can help you to avoid some common mistakes.

More information on Hoisting:
https://www.w3schools.com/js/js_hoisting.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Closures

JavaScript closure is an inner function that has access to the variables in the outer (enclosing) function's scope chain. The closure has three scope chains: it has access to its own scope (variables defined between its curly brackets), it has access to the outer function's variables, and it has access to the global variables.
JavaScript closures are created when the inner function is made within the outer function. Closures are used extensively in JavaScript libraries such as jQuery.
Here is a simple example of a closure in JavaScript:

```
function outerFunction(x) {
  var innerFunction = function(y) {
    return x + y;
  }
  return innerFunction;
}

var add5 = outerFunction(5);
var add10 = outerFunction(10);

console.log(add5(2)); // 7
console.log(add10(2)); // 12
```

More information on Closures:
https://www.w3schools.com/js/js_function_closures.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Error Handling

JavaScript is a very flexible language, which means that there are many ways to write code. This flexibility can lead to errors in your program if you're not careful about how you structure it.

One common way to deal with errors is to use try/catch blocks. These blocks allow you to "try" some code and "catch" any errors that occur while it's executing. The syntax for a try/catch block looks like this:

```
try { // Code to try goes here } catch (error) { // Code to handle errors goes here }
```

The code inside the try block will be executed first. If no errors occur, the code in the catch block will never be executed. However, if an error does occur, execution will jump directly to the catch block and any subsequent code in the try block will be skipped.

Play around with the various possibilities of Error Handling:
https://www.w3schools.com/js/js_errors.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Regular Expressions

Regular expressions are a powerful tool used to perform pattern matching and "search-and-replace" functions on text.

A regular expression is simply a sequence of characters that defines a particular search pattern. For example, the following regex would match any string containing an uppercase letter A: /A/.

A regular expression literal is simply a string that contains the pattern you want to match, enclosed within two forward slashes (/). For example:

```
const regex = /abc/;
console.log(regex); // => /abc/
```

Play around with the various possibilities of Regular Expressions:
https://www.w3schools.com/jsref/jsref_obj_regexp.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Form Validation

JavaScript is a client-side scripting language, which means that the script runs on your visitor's computer. This allows you to do things like check if an email address has been entered correctly before it gets sent off to the server. It also reduces strain on your server because form data doesn't have to be submitted until after it has been checked for errors by JavaScript.

Play around with the various possibilities of Form Validation:
https://www.w3schools.com/js/js_validation.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Cookies

Creating a cookie in JavaScript is very simple. You just have to use the document object and its method createCookie(). The syntax of this function looks like:

```
document.cookie = "name=value; expires=date";
```

The name parameter represents the name of your cookie, while value stands for its content (string). If you want to set an expiration date on it, then add another attribute called expires with a valid Date() string as value or simply pass null if you don't need one.

More information on Cookies:
https://www.w3schools.com/js/js_cookies.asp

Co-funded by the
Erasmus+ Programme
of the European Union

JavaScript Ajax Requests

Ajax is a web development technique for creating interactive web applications. The goal of Ajax is to make web pages feel more responsive by exchanging small amounts of data with the server behind the scenes, so that the entire page does not have to be reloaded each time the user makes a change.

Once you have included the Ajax library in your web page, you can start making requests to the server. For example, if you wanted to load some data from a file on the server, you would use the following code:

```
$.ajax({ url: 'data.json', success: function(data) { // do something with the data } });
```

This code will make an Ajax request to the URL 'data.json'. If the request is successful, the function in the 'success' callback will be executed with the data from the server as its argument.

Play around with the various possibilities of Ajax Requests:
https://www.w3schools.com/xml/ajax_intro.asp

Co-funded by the
Erasmus+ Programme
of the European Union

THANK YOU!

Co-funded by the
Erasmus+ Programme
of the European Union