[Lesson 1]

Roi Yehoshua 2018

[DAN.IT EDUCATION]

Our targets for today

- What is JavaScript and where is it used
- How to add scripts to your HTML page
- How to perform basic interactions with the user in Browser
- JavaScript basics

[JavaScript]

- → Created In 1995 by Brendan Eich as a scripting language for Netscape Navigator
- → Standardized as ECMAScript in 1997
- → JavaScript (often abbreviated as JS) enables interactive web pages and thus is an essential part of web applications
- → Initially created as a browser-only language, but now it is used in many other environments as well:
  - → HTML5 mobile apps
  - → Server side development (NodeJS)
  - → JS on devices – the internet of things
  - → Huge potential of running JavaScript on embedded devices

[DAN.IT] EDUCATION

JavaScript Main Features

- → Interpreter based (no compilation) scripting language
- → Loosely typed and dynamic language
- → Uses Syntax influenced by that of Java
  - → However, has very different semantics than Java
- → Main components
  - → The Core (ECMAScript)
  - → The DOM (Document Object Model)
  - → The BOM (Browser Object Model)

In-Browser JavaScript

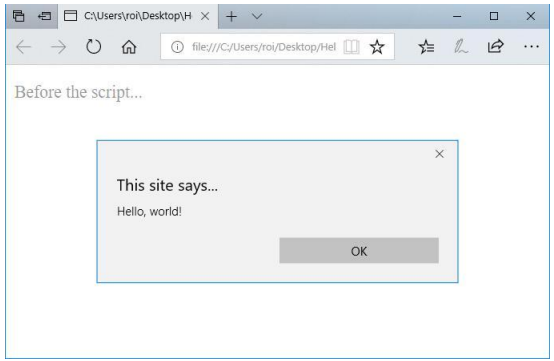
- In-browser JavaScript can do everything related to webpage manipulation, interaction with the user and the web server. For instance, in-browser JS is able to:
  - Add new HTML to the page, change the existing content, modify styles
  - React to user actions, run on mouse clicks, pointer movements, key presses
  - Send requests over the network to remote servers, download and upload files (AJAX)
  - Remember the data on the client-side (“local storage”)
- However, JavaScript in the browser is limited for the sake of the user’s safety:
  - JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs
  - JavaScript has no direct access to OS system functions
  - JavaScript from one page may not access another page if they come from different sites
  - This is called the “Same Origin Policy”

[DAN.IT] EDUCATION

The <script> tag

→ JavaScript programs can be inserted in any part of an HTML document with the help of the <script> tag

```
<html>
<head>
      <title>My First Script</title>
</head>
<body>
      <p>Before the script...</p>
      <script>
            alert('Hello, world!');
      </script>
      <p>...After the script.</p>
</body>
</html>
```



→The <script> tag contains JavaScript code which is automatically executed when the browser meets the tag

→ Current practice often places it just before the closing body tag

[DAN.IT] EDUCATION

Developer Console

→ Code is prone to errors

→ But in the browser, a user doesn't see the errors by default. So, if something goes wrong in the script, we won't see what's broken and can't fix it.

→ To see errors and get a lot of other useful information about scripts, browsers have embedded "developer tools"

→ Most often developers lean towards Chrome or Firefox for development, because those browsers have the best developer tools

[Developer Console]

→ Create the following page:

→ Open it in the browser

→ There's an error in the JavaScript code on it

→ It's hidden from a regular visitor's eyes, so let's open developer tools to see it

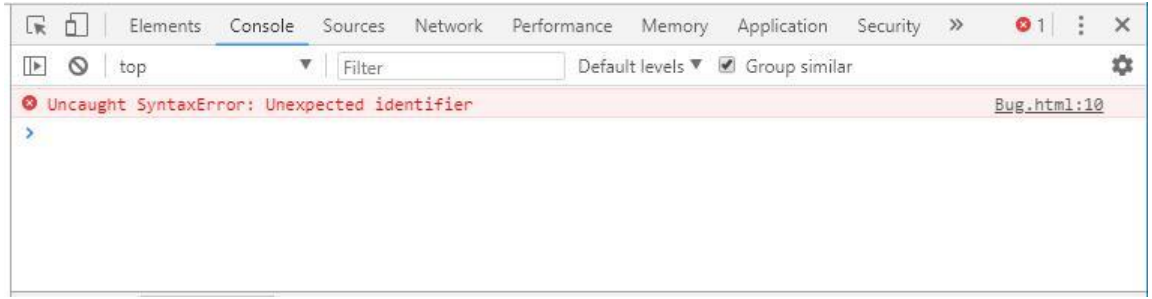
→ Press F12 or, if you're on Mac, then Cmd+Opt+J

→ The developer tools will open on the Console tab by default

```
<html>
<head>
    <title>Buggy page</title>
</head>
<body>
    There is an error in the script on this page.
    <script>
        bla bla
    </script>
</body>
</html>
```

[DAN.IT] EDUCATION

Developer Console



→ Here we can see the red-colored error message
  → In this case the script contains an unknown “bla bla” command
→ On the right, there is a clickable link to the source bug.html:10 with the line number where the error has occurred
→ Below the error message there is a blue > symbol.
  → It marks a “command line” where we can type JS commands and press Enter to run them

[DAN.IT] EDUCATION

External Scripts

- → As a rule, only the simplest scripts are put into HTML
- → More complex ones reside in separate .js file
- → The benefit of a separate file is that the browser will download it and then store in its cache
- → The script file is attached to HTML with the src attribute:

```
<html>
  <head>
    <title>External Script</title>
  </head>
<body>
    <script src="/scripts/myapp.js"></script>
</body>
</html>
```

- → Here /scripts/MyScript.js is an absolute path to the script file (from the site root)
- → It is also possible to provide a path relative to the current page
- → For instance, src="myapp.js" would mean a file "myapp.js" in the current folder

[DAN.IT] EDUCATION

External Scripts

→ To attach several scripts, use multiple tags:

```
<script src="/scripts/script1.js"></script>
<script src="/scripts/script2.js"></script>
```

→ A single <script> tag can't have both the src attribute and the code inside

→ This won't work:

```
<script src ="file.js">
     alert(1); // the content is ignored, because src is set
</script>
```

→ The example above can be split into two scripts to work:

```
<script src="file.js"></script>
<script>
     alert(1);
</script>
```

[DAN.IT] EDUCATION

Code Structure

→ Statements in JavaScript are separated by a semicolon

```
alert('Hello'); alert('World');
```

→ Usually each statement is written on a separate line – thus the code becomes more readable:

```
alert('Hello');
alert('World');
```

→ A semicolon may be omitted in most cases when a line break exists. This would also work:

```
alert('Hello')
alert('World')
```

[DAN.IT] EDUCATION

Comments

→ Comments can be put into any place of the script
→ One-line comments start with two forward slash characters //
→ The rest of the line is a comment. It may occupy a full line of its own or follow a statement.

```
//   This comment occupies a line of its own
alert('Hello');
alert('World'); // This comment follows the
statement
```

→ Multiline comments start with a forward slash and an asterisk /*, and end with an asterisk and a forward slash */

```
/* An example with two messages.
This is a multiline comment.
*/
alert('Hello');
alert('World');
```

→ Please, don't hesitate to comment your code

[DAN.IT] EDUCATION

Variables

→ A variable is a “named storage” for data
→ To create a variable in JavaScript, we need to use the let keyword
→ The statement below creates (*declares* or *defines*) a variable named “message”:

```
let message;
```

→ Now we can put some data into it by using the assignment operator =

```
let message;
message = 'Hello!';
```

→ The string is now saved into the memory area associated with the variable
→ We can access it using the variable name:


```
alert(message); // shows the
variable content
```

[DAN.IT] EDUCATION

Variables

→ To be concise we can merge the variable declaration and assignment into a single line:

```
let message = 'Hello!';
```

→ We can also declare multiple variables in one line:

```
let user = 'John', age = 25,
message = 'Hello';
```

→ That might seem shorter, but it's not recommended. For the sake of better readability, please use a single line per variable.

[The Old "var"]

→ In older scripts you may also find another keyword: var instead of let

```
var message = 'Hello!';
```

→ The var keyword is *almost* the same as let

→ There are two main differences of var:

  → Variables have no block scope, they are visible minimum at the function level

  → Variable declarations are processed at function start

→ These differences are actually a bad thing most of the time

→ So, for new scripts var is used exceptionally rarely

[DAN.IT] EDUCATION

Variable Naming

- There are two limitations for a variable name in JavaScript:
  - The name must contain only letters, digits, symbols $ and _
  - The first character must not be a digit
- When the name contains multiple words, camelCase is commonly used
  - i.e., words go one after another, each word starts with a capital letter: myVeryLongName
- JavaScript is case-sensitive, e.g., variables named apple and Apple are two different variables
- There is a list of reserved words, which cannot be used as variable names, because they are used by the language itself
- For example, the words let, class, return, function are reserved
- Please name the variables sensibly. Take time to think if needed
- Variable naming is one of the most important and complex skills in programming

[DAN.IT] EDUCATION

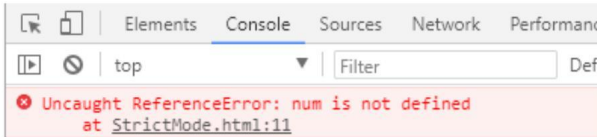
[Strict Mode]

→ Normally, we need to define a variable before using it
→ But in the old times, it was technically possible to create a variable by a mere assignment of the value, without let

```
num = 5; // the variable "num" is created if didn't
exist alert(num); // 5
```

→ **Strict mode** is a way to introduce better error-checking into your code
→ You can declare strict mode by adding "use strict"; at the beginning of a file, a program, or a function
→ When you use **strict mode**, you cannot, for example, use implicitly declared variables

```
"use strict";
num = 5; // error: num is not defined
```



[DAN.IT] EDUCATION

Constants

To declare a constant (unchanging) variable, use const instead of let:

```
const message = 'hello';
message = 'bye'; // error, can't reassign the constant!
```

There is a widespread practice to use constants as aliases for difficult-to-remember values that are known prior to execution

Such constants are named using capital letters and underscores

Example:

```
const COLOR_GREEN = '#0F0';
const COLOR_BLUE = '#00F';
const COLOR_ORANGE = '#FF7F00';
//...when we need to pick a color
let color = COLOR_ORANGE; alert(color); // #FF7F00
```

[Exercise (1)]

→ Declare two variables: admin and name

→ Assign the value "John" to name

→ Copy the value from name to admin

→ Show the value of admin using alert (must output “John”)

[Interaction: alert, prompt, confirm]

→ The browser supplies a few user-interface functions: alert, prompt and confirm

→ alert(message) - shows a message and pauses the script execution until the user presses “OK”

→ The mini-window with the message is called a modal window

→ The word “modal” means that the visitor can’t interact with the rest of the page, press other buttons etc, until they have dealt with the window

```
alert('Hello');
```

[DAN.IT] EDUCATION

[Interaction: alert, prompt, confirm]

→ prompt shows a modal window with a text message, an input field for the visitor and buttons OK/CANCEL

→ It accepts two arguments:

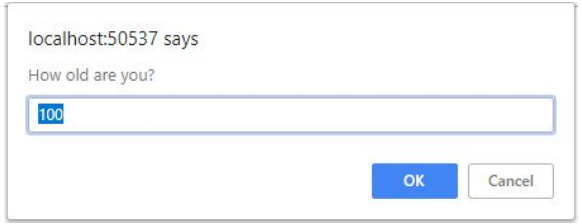
```
result = prompt(title[, default]);
```

→ title - the text to show to the visitor

→ default - an optional second parameter, the initial value for the input field

→ The call to prompt returns the text from the field or null if the input was canceled

```
let age = prompt('How old are you?', 100);
alert(`You are ${age} years old!`); // You are 100 years old!
```



[DAN.IT] EDUCATION

[Interaction: alert, prompt, confirm]

→ The result is true if OK is pressed and false otherwise

```
result = confirm(question);
```

→ confirm shows a modal window with a question and two buttons: OK and CANCEL

```
let isBoss = confirm("Are you the boss?");
alert(isBoss); // true if OK is pressed
```

[DAN.IT] EDUCATION

[Control questions]

1. What is JavaScript and where is it used?
2. How can we add JavaScript to the HTML page?
3. What is the difference between let and var variables?
4. What is the naming convention for variables in Javascript?
5. What is constant?
6. Why do we need strict mode?
7. Name simple ways to enter the data in Browser and access it in JavaScript.

[DAN.IT] EDUCATION