The JavaScript language

Stanisław Polak, Ph.D.

Computer Systems Group

https://www.icsr.agh.edu.pl/~polak/

INFORMATYKA
Akademia Górniczo-Hutnicza w Krakowie

Notatki

Outline

Notatki

Organizational information

- Lectures every week — each of them is preparation for laboratory exercises
- Laboratory exercises — every week
- The final grade is calculated on the basis of the number of points:
  - A set of programming tasks to be performed in the classroom
  - A set of homework — the deadline for the solution: next classes
- Subject URL:
  `https://www.icsr.agh.edu.pl/~polak/jezyki/js/`
- Laboratory exercises URL:
  `https://polak.icsr.agh.edu.pl/`

Outline of laboratory exercises

1. CSS3 and creating responsive websites
2. JavaScript — data types, creating 2D graphics
3. DOM
4. NodeJS
5. Basics of the "Express.js" framework
6. AJAX
7. The "TypeScript" language

Notatki

The first JS script

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <title>Pierwszy skrypt JS</title>
5    </head>
6    <body>
7      <h1>Pierwszy skrypt JS</h1>
8      <script>
9        document.write("Witaj w dokumencie");
10       console.log("Witaj w konsoli");
11     </script>
12   </body>
13 </html>
```

HTML document with the content of JS script

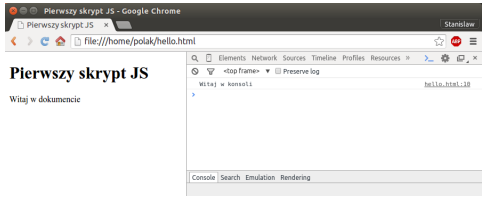


Figure: The result of the JS script execution after rendering the web page

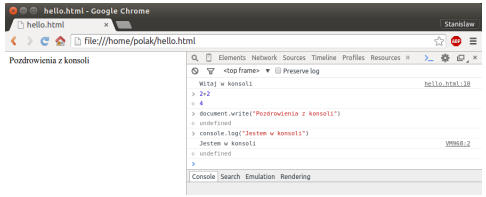


Figure: Using the JS console

Notatki

Embedding JS code

```
<!-- <script type="text/javascript;version=x.x"> -->
<script type="text/javascript">
<!--
  Treść skryptu JavaScript
  The content of the JavaScript script
-->
</script>
<noscript>
Twoja przeglądarka nie obsługuje JavaScript!<br>
Your browser does not support JavaScript!
</noscript>
```

JS internal script embedded in HTML

```
<script src="URL"></script>
<noscript>
Twoja przeglądarka nie obsługuje JavaScript!<br>
Your browser does not support JavaScript!
</noscript>
```

Referring to an external script

Notatki

The order in which scripts are executed

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<script>
function info(arg){
  var img = document.getElementsByTagName("img");
  var names='';
  var img_msg = '\nThe "img" element is unavailable\n'
  if(img.length != 0)
    img_msg = '\nThe "img" element is available\n';
  if(document.body){
    x=document.body.childNodes
    for (i=0;i<x.length;i++){
      if(x[i].nodeType == 1)
        names+=x[i].nodeName+", ";
    }
    console.log(arg+img_msg+"'document.body' contains elements: "+names);
  }
  else
    console.log(arg+img_msg+"'document.body' = NULL");
}
</script>
<script>info("Script1 1");</script>
<script src="a.js"></script>
</head>
<body onLoad="info('Script 4');">
<script>info("Script 2");</script>
<script src="b.js"></script>
<img src="image.jpg">
<script>info("Script 3");</script>
<script src="c.js"></script>
</body>
</html>
```

```
info('<file name>');
```

a.js, b.js, c.js

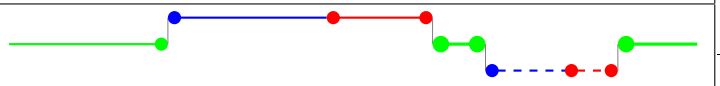
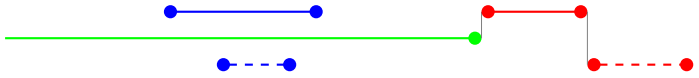
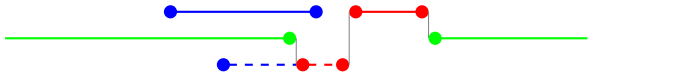
Console output:

1. Script 1
   The "img" element is unavailable
   'document.body'=NULL
2. a.js
   The "img" element is unavailable
   'document.body'=NULL
3. Script 2
   The "img" element is unavailable
   'document.body' contains elements: SCRIPT,
4. b.js
   The "img" element is unavailable
   'document.body' contains elements: SCRIPT, SCRIPT,
5. <Displaying the image 'image.jpg'>
6. Script 3
   The "img" element is available
   'document.body' contains elements: SCRIPT, SCRIPT, IMG, SCRIPT,
7. c.js
   The "img" element is available
   'document.body' contains elements: SCRIPT, SCRIPT, IMG, SCRIPT, SCRIPT,
8. Script 4
   The "img" element is available
   'document.body' contains elements: SCRIPT, SCRIPT, IMG, SCRIPT, SCRIPT,

Notatki

Asynchronous or deferred execution of external scripts

```
1 ...
2 <script async src="https://.../js1.js">
3 ...
4 <script async src="https://.../js2.js">
5 ...
```

Tag	Document processing schema
<script>	
<script defer>	
<script async>	

Parsing Downloading Execution

Source: https://www.growingwiththeweb.com/2014/02/async-vs-defer-attributes.html

Notatki

Defining variables

```
1  var x = 42;
2  //or
3  y = 42; // Not recommended
4  var _y = 42;
5  var $if = 42;
6  var róża = 42;
7  var 1a = 42; // identifier starts immediately after numeric literal
8  console.log(y); // 42
9  console.log(Y); // Y is not defined
10 if = 42; //missing variable name
11 var y = 42
12 console.log(typeof(y)); // number
13 y = "42";
14 console.log(typeof(y)); // string
```

Notatki

Defining constants

```
1  const PI = 3.1415926;
2  console.log(PI); // 3.1415926
3  console.log(typeof(PI)); // number
4  const PI = 3.14; // redeclaration of const PI
5  PI="3.24" // An attempt to overwrite a constant value
6  console.log(PI); // 3.1415926 that is, the attempt to overwrite failed
7  console.log(typeof(PI)); // number
8
9  var PI = 3.14 // redeclaration of const PI
10 var zmienna = 1;
11 const zmienna = 1; // redeclaration of var zmienna
```

Notatki

Special types of JS

Type: "null"

```
1  var empty = null;
2  console.log(typeof(empty)); // object
3
4  var empty1 = NULL; //NULL is not defined
5
6  if(empty)
7    console.log("true");
8  else
9    console.log("false");
10 // false
11
12 console.log(empty-1); // -1
```

Type: "undefined"

```
1  console.log(typeof(abc)); // undefined
2
3  var def;
4  console.log(typeof(def)); // undefined
5
6  function f(arg){
7    console.log("arg="+arg)
8  }
9
10 var result = f()         // arg=undefined
11 console.log(result)  // undefined
12
13
14 if(def === undefined)
15   console.log("Undefined");
16 else
17   console.log("Defined");
18 // Undefined
19
20 if(def)
21   console.log("true");
22 else
23   console.log("false");
24 // false
25
26 console.log(def-1); // NaN
```

Notatki

Simple types and their object-related equivalents

Simple types	Object equivalent (Prototype)
boolean	Boolean
number	Number
string	String

```
1  console.log("2+2"); // "2 + 2" is of type (simple) 'string' => will write: 2 + 2
2  console.log("2+2".length); // Implicit conversion to the type (prototype) 'String' => will write: 3
3  /*
4  The above line is equivalent:
5      var objString = new String("2+2")
6      console.log(objString.length)
7  */
8  var str = "2+2";
9  console.log(str.charAt(1)); // +
10 console.log(str[1]); // +
11 console.log(str.charCodeAt(1)); // 43
12
13 var num = 1.987654
14 console.log(num.toPrecision(3)) //Implicit conversion to the type (prototype) 'Number' => will write:
       1.99
```

Notatki

Type: "boolean"

```
1 var dead = false;
2 var married = true;
3 console.log(typeof(dead)); // boolean
4 married = FALSE; //FALSE is not defined
```

Notatki

Type: “number”

```
var price = 10.5;
var num1 = 2;
var num2 = 2.0;
var binary = 0b101; //0B101;
var octal = 0o77; //0O77
var hexadecimal = 0xFF; //0XFF

console.log(typeof(price)); // number
console.log(typeof(num1)); // number
console.log(typeof(num2)); // number
console.log(num2); // 2
console.log(typeof(binary)); // number
console.log(binary); // 5
console.log(typeof(octal)); // number
console.log(octal); // 63
console.log(typeof(hexadecimal)); // number
console.log(hexadecimal); // 255
```

Notatki

Conversion to type: “number”

```
console.log(parseInt("3.14")); // 3
console.log(parseInt("3.94")); // 3
console.log(parseInt("3.94.1")); // 3
console.log(parseInt("3.94a")); // 3
console.log(parseInt("a3.94")); // NaN
console.log(parseFloat("3.14")); // 3.14
console.log(parseFloat("3.14.1")); // 3.14
console.log(parseFloat('0x10')); // 0
console.log(parseFloat('')); // NaN
console.log(parseFloat(' \r\n\t')); // NaN


console.log(parseInt("101",2)); // 5
console.log(parseInt("FF")); // NaN
console.log(parseInt("FF",16)); // 255

console.log(parseInt("FF - Firefox")); // NaN
console.log(parseInt("FF - Firefox",16)); // 255
console.log(parseInt("false")); // NaN
console.log(parseInt("false",16)); // 250 - "fa" has been changed to a number!

console.log(Number(null)); // 0
console.log(Number(undefined)); // NaN
console.log(Number(false)); // 0
console.log(Number(true)); // 1
console.log(Number("3.14")); // 3.14
console.log(Number("3.14.1")); // NaN
console.log(Number("3")); // 3
console.log(Number("3a")); // NaN
console.log(Number('0x10')); // 16
console.log(Number('')); // 0
console.log(Number(' \r\n\t'); // 0
```

Notatki

Type: “string”

```
1  var last_name = "Polak";
2  var first_name = 'Stanisław';
3
4  console.log(typeof(last_name)); // string
5  console.log(typeof(first_name));  // string
6
7  console.log("First name=${first_name} Last name=${last_name}"); // First name=${first_name} Last name=${last_name}
8  console.log('First name=${first_name} Last=${last_name}'); // First name=${first_name} Last name=${last_name}
9
10 var a = 11 - "1" ;
11 console.log(a); // 10
12 var b = 11 + "1";
13 console.log(b); // 111
14
15 console.log(typeof(a)); // number
16 console.log(typeof(b)); // string
17
18 last_name[0]='W';
19 console.log(last_name); //"Polak" instead of "Wolak"
20
21 last_name = 'W' + last_name.substr(1);
22 console.log(last_name); // and now "Wolak"
```

Notatki

Type: “string”

Template strings

Untagged

```
1 var a = 2;
2 var str = `Variable 'a' has value ${a}, 2+2=${2+2}\n`
3 console.log(str);
4 /*********************************/
5 var str = `Line 1
6 Line 2`;
7 console.log(str);
```

Tagged

```
1  function tag(strings,val1,val2){
2    result = "String 1:\t\t'"+strings[0]+"'\n";
3    result += "Raw string 1:\t'"+strings.raw[0]+"'\n";
4    result += "Value 1:\t\t"+val1+"\n";
5    result += "String 2:\t\t'"+strings[1]+"'\n";
6    result += "Raw string 2:\t'"+strings.raw[1]+"'\n";
7    result += "Value 2:\t\t"+val2+"\n";
8    return result;
9  }
10 /*************************************/
11 var a = 2;
12 var b = 3;
13 str = tag`a+b=\t${a+b}\n, a*b=${a*b}`;
14 console.log(str);
```

Notatki

Typ „string”

Sekwencje specjalne

- \b
- \f
- \n
- \r
- \t
- \v
- \’
- \"
- \\
- \xXX
- \uXXXX

```
1 console.log("a'\"\x63\'\u0105") // a'"c'ą
```

Example of use

Notatki

Conversion to type: “string”

```
1  var dead = true;
2  console.log(typeof(dead)); // boolean
3  var lancuch=dead.toString();
4  console.log(typeof(lancuch)); // string
5  console.log(lancuch); // true
6  var liczba = 0xFF;
7  console.log(0xFF.toString()); // "255"
8  liczba = 11;
9  console.log(liczba.toString()); // "11"
10 liczba = 11.9;
11 console.log(liczba.toString()); // "11.9"
12
13 var liczba=255;
14 console.log(liczba.toString(2)); // 11111111
15 console.log(liczba.toString(4)); // 3333
16 console.log(liczba.toString(8)); // 377
17 console.log(liczba.toString(16)); // ff
18
19 console.log(String(null)); // "null"
20 console.log(String(undefined)); // "undefined"
21 console.log(String(false)); // "false"
22 console.log(String(true)); // "true"
23 console.log(String(255)); // "255"
24 console.log(String(3.14)); // "3.14"
```

Notatki

Type: “symbol”

```
symbol1 = Symbol();
symbol2 = Symbol();
console.log(typeof(symbol1));   //symbol
console.log(symbol1 == symbol2) //false

symbol3 = Symbol('Symbol description');
symbol4 = Symbol('Symbol description');
console.log(symbol3); //Symbol(Symbol description)
console.log(symbol4); //Symbol(Symbol description)
console.log(symbol3 == symbol4); //false

symbol5 = Symbol.for("symbol3");
symbol6 = Symbol.for("symbol3");
console.log(symbol5); //Symbol(symbol3)
console.log(symbol6); //Symbol(symbol3)
console.log(symbol5 == symbol6); //true

var symbol7 = Symbol.for("uid");
console.log(Symbol.keyFor(symbol7)); // "uid"
var symbol8 = Symbol.for("uid");
console.log(Symbol.keyFor(symbol8)); // "uid"
var symbol9 = Symbol("uid");
console.log(Symbol.keyFor(symbol9)); // undefined
```

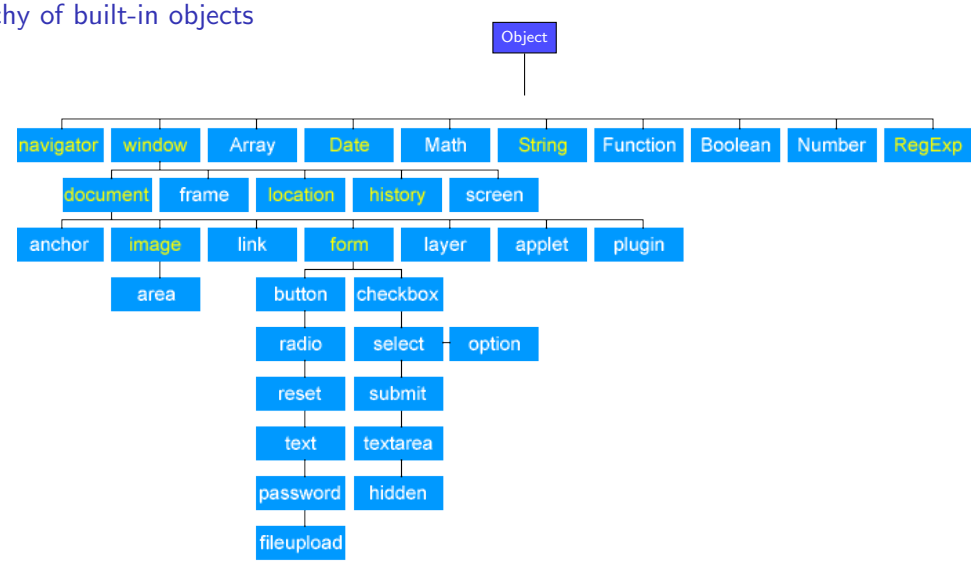
Notatki

Objects

```
//Create an instance of the (built-in) type 'Object'
var object1 = new Object();
var object2 = {a:1, b:10};
console.log(typeof(object1)); // object
console.log(typeof(object2)); // object

//Access to object properties
console.log(object1.constructor);    // function Object() { [native code] }
console.log(object1['constructor']); // function Object() { [native code] }
console.log(object2['constructor']); // function Object() { [native code] }
console.log(object2.a); // 1
console.log(object2['b']); // 10

const {a,b} = object2; // Object destructing
console.log(a); // 1
console.log(b); // 10

//Use the symbol as the object's property
a = Symbol();
var object3 = {[a]:1, b:10}
console.log(object3[a]);    // 1
console.log(object3.a);     // undefined
console.log(object3['a']); // undefined
console.log(object3['b']); // 10
console.log(object3.b); // 10
```

Notatki

Hierarchy of built-in objects



Source: `http://www.visualtech.ca/javascript/javascript_object_model.php`

Notatki

The "Array" object

```
1  var tab1 = new Array(1,2,3);//equivalent to: var tab1=Array(1,2,3)
2  var tab2a = new Array(10);  //equivalent to: var tab2a = []; tab2a.length=10;
3  var tab2b = new Array("10");
4  var tab3 = [4,'abc',6];
5  var a, rest;
6
7  console.log(tab1.length); // 3
8  console.log(tab2a.length); // 10
9  console.log(tab2b.length); // 1
10 console.log(tab3.length); // 3
11 console.log(tab1[0]); // 1
12 console.log(tab1.0); //missing ) after argument list
13 console.log(tab2a[0]); // undefined
14 console.log(tab2b[0]); // 10
15 console.log(tab3[1]); // abc
16 console.log(tab3[5]); // undefined
17
18 [a, ...rest] = tab3; // Array destructing
19 console.log(a); // 4
20 console.log(rest); // [ 'abc', 6 ]
21
22 var tab4 = new Array(new Array(1,null,undefined),new Array('A','B','C'),new Array('x','y','z'));
23 console.log(tab4.length); // 3
24 console.log(tab4[0]); // 1,,
25 console.log(tab4[0][1]); // null
26 var tab1 = new Array(1,2,3);
27 var tab2 = [4,5,6];
28 console.log(tab1); // [1, 2, 3]
29
30 console.log(Array.from("JavaScript")); //['J','a','v','a','S','c','r','i','p','t']
31
32 var tab3 = tab1.concat(tab2);
33 console.log(tab3); // 1, 2, 3, 4, 5, 6
34
35 console.log(tab3.splice(1,2)); // 2, 3
```

Notatki

The “Map” object

```
var map = new Map();
emptyObject = {};
map.set("string","The value associated with the string");
map.set(1,{a:10});
map.set(emptyObject,1);

console.log(map); //Map { string: "The value associated with the string", 1: Object, Object: 1 }
console.log(map.get(1)); //Object { a: 10 }
console.log(map.get(2)); //undefined
console.log(map.get("string")); //"The value associated with the string"
console.log(map.get({})); //undefined
console.log(map.get(emptyObject)); //1
console.log(map.size); //3
map.delete("string");
console.log(map.size); //2

//Iteration of the hash
map.forEach((value,key,map) => {console.log("map["+key+"]="+value)});
/*
"map[1]=[object Object]"
"map[[object Object]]=1"
*/
//Conversion of the array into hash
var tab = [["key1","String"],["key2",5]];
map = new Map(tab);
console.log(map); //Map { key1: "String", key2: 5 }
```

Notatki

The “RegExp” object

```
function check(number) {
  var re = /\d{7}/g; //'g' - return all matching fragments, not just the first one
  //lub
  var re=new RegExp("\\d{7}","g");

  if(re.test(number))
    console.log("The correct phone number");
  else
    console.log("The telephone number should consist of seven digits");
}

var number1="1234567";
var number2="12-34";

check(number1); // The correct phone number
check(number2); // The telephone number should consist of seven digits
```

Validation of phone number format

Notatki

The "Function" object

```
1 var adder = new Function("a", "b", "return a + b");
2 var result = adder(1,2);
3 console.log(result); // 3
4 console.log(adder.length); // 2
5 var obj = {x: 1, y:2};
6 adder = new Function("message",  "console.log(message+' ');return this.x + this.y");
7 console.log(adder.call(obj)); // undefined 3
8 console.log(adder.call(obj,'Value=')); // Value=3
```

Notatki

The "Math" object

```
1  ...
2  <canvas id="canvas" width="400" height="100">
3  Your browser does not support the "canvas" element
4  </canvas>
5
6  <script>
7  var canvas = document.getElementById("canvas");
8  if (canvas.getContext) {
9    var ctx = canvas.getContext('2d');
10   var ox = 0, oy = 50;
11   var t_min = 0, t_max = 10*Math.PI;
12   var scale = 20, step = 200, inc = t_max/step;
13
14   ctx.beginPath();
15   for (var t=t_min; t<=t_max; t+=inc){
16     y = scale * Math.sin(t);
17     x = (t / t_max) * canvas.width;
18     ctx.lineTo(ox+x, oy-y);
19   }
20   ctx.stroke();
21 </script>
22 ...
```

Drawing a function $y = \sin(x)$ for $x \in [0, 10\pi]$

Notatki

The "Date" object

```
1  function JSClock() {
2     var time = new Date()
3     var hour = time.getHours()
4     var minute = time.getMinutes()
5     var second = time.getSeconds()
6     var temp = "" + ((hour > 12) ? hour - 12 : hour)
7     if (hour == 0)
8        temp = "12";
9     temp += ((minute < 10) ? ":0" : ":") + minute
10    temp += ((second < 10) ? ":0" : ":") + second
11    temp += (hour >= 12) ? " P.M." : " A.M."
12    return temp
13 }
14 document.write(JSClock()); // 2:21:47 P.M.
```

The current time is displayed in a 12-hour format

Notatki

The "navigator" object

```
1  //We assume that 'navigator.userAgent' contains the string "Mozilla/4.0 (compatible; MSIE 7.0; Windows
       NT 5.1; .NET CLR 2.0.50727)"
2
3  if (/MSIE (\d+\.\d+);/.test(navigator.userAgent)){ //check if the browser is MSIE x.x;
4   var ieversion=new Number(RegExp.$1) // $1 contains the version number, here: 7.0
5   if (ieversion>=8)
6    document.write("You're using IE8 or above")
7   else if (ieversion>=7)
8    document.write("You're using IE7.x")
9   else if (ieversion>=6)
10   document.write("You're using IE6.x")
11  else if (ieversion>=5)
12   document.write("You're using IE5.x")
13 }
14 else
15  document.write("n/a")
```

Identification of the IE browser version

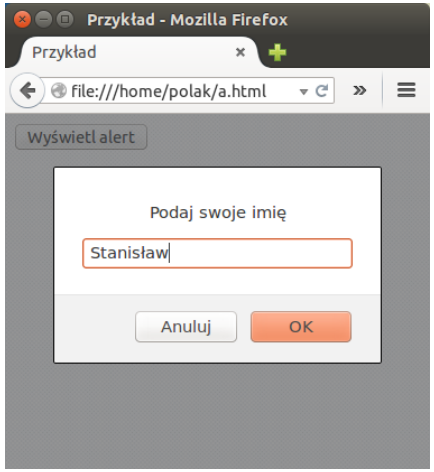
Notatki

The "window" object

The prompt() and alert() methods

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;charset=UTF
       -8">
4  <title>Przykład</title>
5  <script>
6  function load_and_display_the_name(){
7    var first_name=window.prompt('Podaj swoje imię','');
8    window.alert("Witaj "+first_name); //Display welcome text
9  }
10 </script>
11 </head>
12 <body>
13 <form>
14 <button type="button" onClick="load_and_display_the_name();">Wy
       świetl alert</button><br>
15 </form>
16 </body>
17 </html>
```

Opening the input / output data window



Notatki

The "window" object

The setTimeout() and clearTimeout() methods

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
4  <title>An example</title>
5  <script>
6  function delayedAlert(time){
7    timeoutID = window.setTimeout(display, 2000);
8  }
9
10 function display(){
11   window.alert("Hello World!");
12   //timeoutID = window.setTimeout(display, 2000);
13 }
14
15 function stopExecution(){
16   window.clearTimeout(timeoutID);
17 }
18 </script>
19 </head>
20 <body>
21 <form>
22 <button type="button" onClick="delayedAlert();">Show alert</button><br>
23 <button type="button" onClick="stopExecution();">Cancel</button>
24 </form>
25 </body>
26 </html>
```

After 2 seconds, the alert window is displayed

Notatki

The "window" object

The setInterval() and clearInterval() methods

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
4  <title>An example</title>
5  <script>
6  function start_cyclic_execution() {
7    timeoutID = window.setInterval(display, 1000);
8  }
9
10 function display(){
11   console.log("Hello World!");
12 }
13
14 function stop_cyclic_execution(){
15   window.clearInterval(timeoutID);
16 }
17 </script>
18 </head>
19 <body>
20 <form>
21 <button type="button" onClick="start_cyclic_execution()">Start</button><br>
22 <button type="button" onClick="stop_cyclic_execution();">Stop</button>
23 </form>
24 </body>
25 </html>
```

Every second, a message is displayed on the console

Notatki

The "window" object

The requestAnimationFrame() and cancelAnimationFrame() methods

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
4  <title>An example</title>
5  <script>
6  function start_cyclic_execution() {
7    requestID=window.requestAnimationFrame(display);
8  }
9
10 function display(){
11   console.log("Hello World!");
12   requestID=window.requestAnimationFrame(display);
13 }
14
15 function stop_cyclic_execution(){
16   window.cancelAnimationFrame(requestID);
17 }
18 </script>
19 </head>
20 <body>
21 <form>
22 <button type="button" onClick="start_cyclic_execution()">Start</button><br>
23 <button type="button" onClick="stop_cyclic_execution()">Stop</button>
24 </form>
25 </body>
26 </html>
```

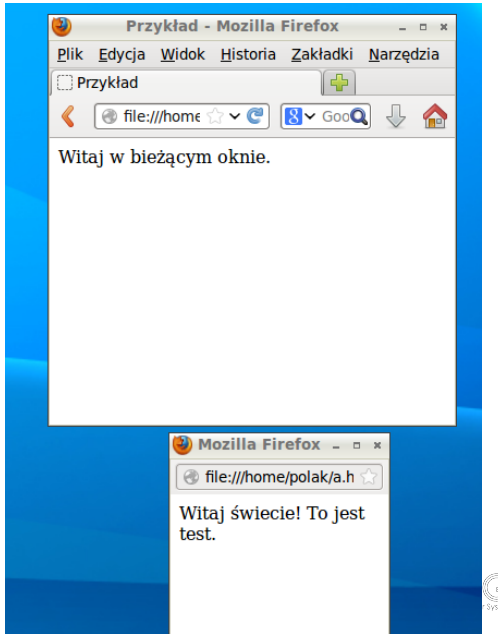
Periodically, a message is printed when the screen is refreshed

Notatki

The “document” object

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html;
       charset=UTF-8">
4  <title>Przykład</title>
5  </head>
6  <body>
7  Witaj
8  <script>
9  var newWindow=window.open('','','toolbar=no,
       scrollbars=no,width=200,height=150');
10 newWindow.document.open("text/html", "replace");
11 newWindow.document.writeln("Witaj świecie!");
12 newWindow.document.write("To jest test.");
13 newWindow.document.close();
14 document.write(" w bieżącym oknie.");
15
16 </script>
17 </body>
18 </html>
```

The text is displayed in a new window

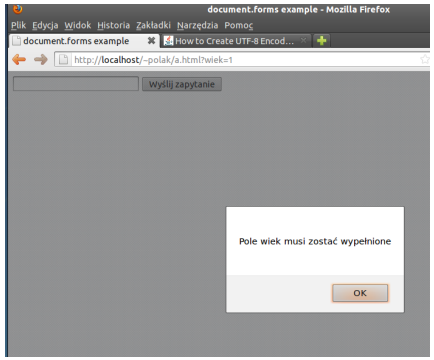


Notatki

The “form” object

```
1  <html>
2  <head>
3  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
4  <title> document.forms example</title>
5  <script>
6  function check() {
7  var form = document.forms[0];
8  //var form = document.forms.form1;
9  //var form = document.forms['form1'];
10 var element  = form.elements[0];
11 //var element  = form.elements.wiek;
12 //var element  = form.elements['wiek'];
13 if (element.value == ""){
14   window.alert("Pole wiek musi zostać wypełnione");
15   return false;
16 }
17 else
18   return true;
19 }
20 </script>
21 </head>
22 <body>
23 <!-- <form ...  onSubmit="return false;"> -->
24 <form id="form1" action="" onSubmit="return check();">
25  <input name='wiek' type='text'>
26  <input type='submit'>
27 </form>
28 </body>
29 </html>
```

Suspending the submission of the form



Notatki

The "Image" object

```
<html>
<head>
<script language = "JavaScript">

function preloader(){
  bigImage = new Image();
  bigImage.src = "bigImage.jpg";
}
</script>
</head>
<body onLoad="javascript:preloader()">
<a href="#" onMouseOver="javascript:document.images[0].src='bigImage.jpg'">
<!--
lub tak:
<a href="#" onMouseOver="javascript:document.img01.src='bigImage.jpg'">
-->
<img name="img01" src="differentImage.jpg"></a>
</body>
</html>
```

Pre-loading a large image

Notatki

The "location" object

```
<script>
if (window.location.protocol == "http:") {
  var restUrl = window.location.href.substr(5);
  location.href = "https:" + restUrl;
}
</script>
```

Redirecting to the secure HTTPS protocol

Notatki

The “history” object

```
history.back(); // equivalent to clicking the 'Back' button
history.go(-1); // equivalent to history.back();
```

Notatki

The “screen” object

```
if (screen.pixelDepth < 8) {
  // use of the "low-color" version of the page
} else {
  // use of the "full-color" version of the page
}
```

Notatki

Operators, "inherited" from Java

Exactly the same as in Java

- Arithmetic (+, -, *, /, ++, --, %)
- Conditional (?:)
- Bitwise (&, |, ^, ~, <<, >>, >>>)
- Logical (&&, ||, !)
- String (+)
- Assignment (=, *=,/=, %=, +=, -=, <<=, >>=, >>>=, &=, ^=, |=)
- Type (`instanceof`)
- `new` and `this`

Almost the same as in Java

- Comparison — additional: === and !==

```
1 document.write(true == 1); // true, because 'true' is
      converted to 1 and then compared
2 document.write("1" ==  1); // true, because 1 is converted to
      "1" and then compared
3 document.write([] == true);            // false, because [] is
      converted to 0 and then compared
4 document.write([] && true);            // true, because [] is
      not converted and Boolean([])  has value 'true'
5 document.write("1" === 1); // false
6 document.write("1" !== 1); // true
7 document.write(1 === 1);   // true
```

Additional comparison operators

Notatki

Comma

```
1  a=1 , b=2;
2  document.write(a); // 1
3  document.write(b); // 2
4  for (var i = 0, j = 9; i <= 9; i++, j--)
5    document.writeln("a[" + i + "][" + j + "]"); /* Output:
6  a[0][9]
7  a[1][8]
8  a[2][7]
9  a[3][6]
10 a[4][5]
11 a[5][4]
12 a[6][3]
13 a[7][2]
14 a[8][1]
15 a[9][0]
16 */
```


Notatki

The `in` operator

```
var trees = new Array("redwood", "cedar", "oak", "maple");
0 in trees;        // true
3 in trees;        // true
6 in trees;        // false
"maple" in trees;  // false (the index number should be given, not the value after the given index)
"length" in trees; // true ('length' is the property of the object 'Array')

//Predefined objects
"PI" in Math;          // true
var myString = new String("coral");
"length" in myString;  // true

// User objects
var car = {brand: "Honda", model: "Accord", year: 1998};
"brand" in car; //  true
"model" in car; //  true
```

Notatki

The `delete` operator

```
x = 42;
var y = 43;
obj = new Number();
obj.h = 4;
delete x;       // returns true (you can delete a variable if it is implicitly defined)
delete y;       // returns false (the variable can not be deleted if it is defined using 'var')
delete Math.PI; // returns false (you can not delete predefined properties)
delete obj.h;   // returns true (you can delete user-defined properties)
delete obj;     // returns true (you can delete a variable if it is implicitly defined)

var trees = new Array("redwood", "cedar", "oak", "maple");
document.write(trees.length); // 4
document.write(trees[2]);     // oak
delete trees[2];
document.write(trees.length); // 4
document.write(trees[2]);     // undefined

trees[3] = undefined;
if (2 in trees)
  document.write("The element with index 2 exists in the table");
if (3 in trees)
  document.write("The element with index 3 exists in the table");
//Output: The element with index 3 exists in the table
```

Notatki

The typeof operator

```
var myFun = new Function("5 + 2");
var shape = "round";
var size = 1;
var today = new Date();

typeof myFun;           // returns "function"
typeof(shape);          // returns "string"
typeof size;            // returns "number"
typeof today;           // returns "object"
typeof does_not_exists; // returns "undefined"
typeof true;            // returns "boolean"
typeof null;            // returns "object"
typeof 62;              // returns "number"
typeof 'Hello world';   // returns "string"
typeof Math;            // returns "object"
typeof Array;           // returns "function"
```

Notatki

The void operator

```
<A HREF="javascript:void(0)">Click here so that nothing will happen</A>
<A HREF="javascript:void(document.form.submit())">Click here for the form to be approved</A>
```

Notatki

Functions

- Defining (dissimilar to Java) — using the keyword `function`
- Calling functions and returning values by function — similar to Java
- They let you define objects' prototype

```
function multiply(a, b=1){
  var c = a * b;
  a = 0;
  b = 0;
  return c;
}
function f() { return [1, 2, 3] }

a = 2;
b = 2;
var result = multiply(a,b);
console.log(result);//  4
var result = multiply(a);
console.log(result);// 2

var x, y, z;
[x, y, z] = f(); // Returning many values
[x, y, z] = (function f() { return [1, 2, 3] })(); //
      Simultaneous defining and calling functions
console.log(x); // 1
/*****************************/
const constant = 1;
var variable = 2;
function constant(){} //Redeclaration of const constant
function variable(){}
variable(); //variable is not a function
```

Example functions

```
function change(x,object1,object2){
  x = 2;
  object1.brand = "Fiat";
  object2 = {brand: "Skoda"};
}

var car1 = {brand: "Ferrari"};
var car2 = {brand: "Ferrari"};
var variable = 1;
console.log(variable);      // 1
console.log(car1.brand);    // Ferrari
console.log(car2.brand);    // Ferrari
change(variable,car1,car2);
console.log(variable);      // 1
console.log(car1.brand);    // Fiat
console.log(car2.brand);    // Ferrari
```

Passing simple and complex types

Notatki

Anonymous functions

```
//Procedural function
function hello1(who) {
  return 'Hello '+who;
}
/**********************/
console.log(hello1('world'));// "Hello world"
/**********************/
/**********************/
// Function as a variable
var hello2 = function (who) {return 'Hello '+who};
// or
var hello2 = (who) => {return 'Hello '+who};
// or
var hello2 = (who) => 'Hello '+who;
/*****************************/
var hello3 = function() {
  console.log('Hello');
  console.log('World');
}
// or
var hello3 = () => {
  console.log('Hello');
  console.log('World');
}
console.log(hello2('world'));// "Hello world"
hello3(); // "Hello"
          // "World"
```

```
function Person() {
  // The Person () constructor defines 'this' as an
       instance of itself
  this.age = 0;
  this.salary = 0;

  setInterval(function () {
    //Here 'this' ⟺ object 'window' that is, it is
         different from 'this' defined in the Person
         constructor
    this.age++;
    console.log("Age="+this.age);
  }, 1000);

  setInterval(() => {
    this.salary++;//Here 'this' is a Person object
    console.log("Salary="+this.salary);
  }, 1000);
}
var person = new Person();
```

Lexical `this`

Notatki

Closures

```
function init() {
  var name = "Polak";

  function displayName() {
    console.log(name);
  }
  displayName();
}
init(); // Polak
```

```
function createFunction() {
  var name = "Polak";

  function displayName() {
    console.log(name);
  }
  return displayName;
}

var myFunction = createFunction();
myFunction(); // Polak
```

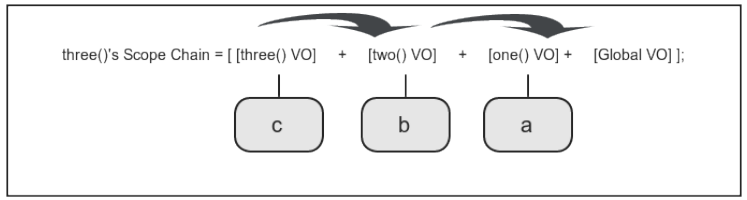
```
function multiply_by(x) {
  return function(y) {
    /* the function uses two variables:
       y - available to the user
       x - defined only inside the 'multiply_by()' function
    */
    return x * y;
  };
}

var product_5_by = multiply_by(5); //the parameter 'x' is assigned the value 5

console.log(product_5_by(12)); // will be written 5 * 12 or 60
```

Example of closure use

Notatki

Scope chain

```
function one(){
  var a = 1;
  two();

  function two(){
    var b = 2;
    three();

    function three() {
      var c = 3;
      console.log(a + b + c); //6
    }
  }
}
one();
```



Source: http://davidshariff.com/blog/javascript-scope-chain-and-closures/

Notatki

Variables in functions

```
1  function fun(x){
2    a = ++x;
3    b = 10;
4  }
5  /*************************/
6  a = 0; // <=> var a = 0;
7  console.log(a); // 0
8  console.log(b); //b is not defined
9  fun(a);
10 console.log(a); // 1
11 console.log(b); // 10
```

```
1  function fun(x){
2    var a = ++x;
3    var b = 10;
4  }
5  /*************************/
6  a = 0; // <=> var a = 0;
7  console.log(a); // 0
8  console.log(b); //b is not defined
9  fun(a);
10 console.log(a); // 0
11 console.log(b); //b is not defined
```

```
1  let a=1;
2  console.log(a); // 1
3  /*******************************/
4  for (let i = 0; i<10; i++) {
5    console.log(i);
6  // 1, 2, 3, 4 ... 9
7  }
8  console.log(i); // i is not defined
```

Expression 'let'

```
1  function fun()
2  {
3    var a = 3;
4    var b = 4;
5    if (a === 3) {
6      let a = 10; // another variable 'a'. Range - interior of the 'if
           ' block
7      var b = 11; // the same variable 'b' as above. Range - interior
           of the 'fun' function
8      console.log(a);  // 10
9      console.log(b);  // 11
10   }
11   console.log(a); // 3
12   console.log(b); // 11
13 }
14 fun();
```

'let' vs. 'var'

Notatki

Functions with variable number of arguments

```
1  function write() {
2    // go after all arguments
3    for (var i=0; i<arguments.length; i++)
4      console.log(arguments[i]);
5  }
6
7  write("A","B","C");
```

```
1  function write(a, ...others) {
2    console.log(a);
3    console.log(Array.isArray(others));
4    for(counter in others)
5      console.log("others["+counter+"]="+others[
           counter]);
6
7    console.log(arguments.length); //SyntaxError: '
         arguments' object may not be used in
         conjunction with a rest parameter
8  }
9
10 write("A","B","C");
```


Notatki

Object of the `Object` type

Defining the methods

```
1 const person = {
2   firstname: 'John',
3   lastname: 'Doe',
4   printThis1: function () { console.log(this) },
5   printThis2: () => { console.log(this) },
6 }
7 person.printThis1();
8 console.log('-------------');
9 person.printThis2();
```

Notatki

The `call()`, `apply()` and `bind()` methods

```
1  const person1 = {
2    firstname: 'John',
3    lastname: 'Doe',
4    printFullName: function (text) { console.log(`${text} ${this.firstname} ${this.lastname}`) }
5  }
6
7  const person2 = {
8    firstname: 'James',
9    lastname: 'Bond',
10 }
11 person1.printFullName('My name is'); // My name is John Doe
12
13 person1.printFullName.call(person2, 'My name is Bond,'); // My name is Bond, James Bond
14
15 const args = ['My name is Bond,'];
16 person1.printFullName.apply(person2, args);  // My name is Bond, James Bond
17
18 const myNameIs = person1.printFullName.bind(person2, 'My name is Bond,');
19 myNameIs(); // My name is Bond, James Bond
```

Notatki

Defining classes

The JavaScript syntax

```
1  function Human(age, name) {
2    this.age = age;
3    this.name = name;
4  }
5  Human.prototype.getInfo = function () { return `name=$
       {this.name} age=${this.age}`; }
6
7  var john = new Human(10, 'John');
8  console.log(john.getInfo()); // name=John age=10
9
10 /*************************************/
11 function Infant(name) {
12   Human.call(this, 0, name);
13 }
14 Infant.prototype = Object.create(Human.prototype);
15
16 var anna = new Infant('Anna');
17 console.log(anna.getInfo()); // name=Anna age=0
```

The ES6 syntax

```
1  class Human {
2    constructor(age, name) {
3      this.age = age;
4      this.name = name;
5    }
6    getInfo() {
7      return `name=${this.name} age=${this.age}`;
8    }
9  }
10
11 var john = new Human(10, 'John');
12 console.log(john.getInfo()); // name=John age=10
13 /*************************/
14 class Infant extends Human {
15   constructor(name) {
16     super(0, name)
17   }
18 }
19
20 var anna = new Infant('Anna');
21 console.log(anna.getInfo()); // name=Anna age=0
```

Notatki

Document Object Model

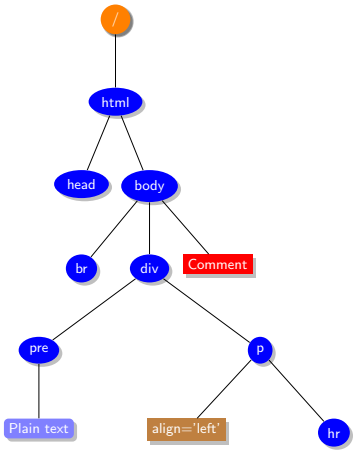
General characteristics

- DOM — Document Object Model
- Document — tree of objects
- Software interface (API) for HTML and XML documents
- A set of properties and methods for manipulating the above mentioned documents

```
1  <html>
2    <head></head>
3    <body>
4      <br>
5      <div>
6        <pre>Plain text</pre>
7        <p align='left'>
8          <hr/>
9        </p>
10     </div>
11     <!-- Comment -->
12   </body>
13 </html>
```

HTML document

DOM tree



Notatki

Basic properties of nodes

The type of node	nodeType	nodeName	nodeValue
Element	Node.ELEMENT_NODE (1)	Name of the tag in upper-case letters	null
Attribute	Node.ATTRIBUTE_NODE (2)	The name of the attribute	The value of the attribute
Text	Node.TEXT_NODE (3)	#text	The text
Comment	Node.COMMENT_NODE (8)	#comment	The comment
Document	Node.DOCUMENT_NODE (9)	#document	null

Notatki

Displaying information about a single 'element' node

Methods: `getElementById()` / `querySelector()`

```
1  <html>
2    <head>
3      <title>An example</title>
4      <script>
5  function start(){
6    var elem1 = document.getElementById("elem1");
7    # or
8    var elem1 = document.querySelector("#elem1");
9
10   window.alert(elem1);            // [object HTMLBodyElement]
11
12   window.alert(elem1.nodeType);   // 1
13   window.alert(elem1.nodeName);   // BODY
14   window.alert(elem1.nodeValue);  // null
15 }
16     </script>
17   </head>
18   <body id="elem1" onLoad="start();">
19   </body>
20 </html>
```

Notatki

Displaying information about several 'element' nodes

Methods: `getElementsByTagName()` / `querySelectorAll()`

```
<html>
  <head>
    <title>An example</title>
    <script>
function start(){
  var tdElements = document.getElementsByTagName("td");
  # or
  var tdElements = document.querySelectorAll("tr td");
  window.alert(tdElements);               // [object HTMLCollection]
  window.alert(tdElements.length);        // 4
  window.alert(tdElements[0]);            // [object HTMLTableCellElement]
  window.alert(tdElements[0].nodeType);   // 1
  window.alert(tdElements[0].nodeName);   // TD
  window.alert(tdElements[0].nodeValue); // null
  window.alert(tdElements[1]);            // [object HTMLTableCellElement]
  window.alert(tdElements[1].nodeType);   // 1
  window.alert(tdElements[1].nodeName);   // TD
  window.alert(tdElements[1].nodeValue); // null
}
    </script>
  </head>
  <body onLoad="start();">
    <table>
      <tr><td>a</td><td>b</td></tr>
      <tr><td>c</td><td>d</td></tr>
    </table>
  </body>
</html>
```

Notatki

Retrieving descendants of 'element' and 'text' nodes

The `childNodes` property

```
<html>
  <head>
    <title>An example</title>
    <script>
function start(){
  var trElements = document.getElementsByTagName("tr");
  for(var i=0 ; i<TRelements.length ; i++){
    var tdElements = trElements[i].childNodes; // the type of the 'tdElements' object is "NodeList"
    for (var j=0;j<tdElements.length;j++){
     var string = tdElements[j].nodeName + ": " + tdElements[j].childNodes[0].nodeValue;
     window.alert(string);
    }
  }
}
/* Output:
TD: a
TD: b
TD: c
TD: d
*/
    </script>
  </head>
  <body onLoad="start();">
  <table border="1">
    <tr><td>a</td><td>b</td></tr>
    <tr><td>c</td><td>d</td></tr>
  </table>
  </body>
</html>
```

Notatki

Support for element attributes

The attributes property, and the setAttribute() & the removeAttribute() methods

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
   <title>An examples</title>
   <script>
function change(thickness){
  var elem1 = document.getElementById("elem1"); // "HTMLTableElement" type object
  window.alert(elem1.getAttribute('border')); // 1
  window.alert(elem1.getAttribute('id')); // elem1
  elem1.setAttribute('border',thickness);

  //You can do it anyway
  var attributes = elem1.attributes; //obiekt typu "NamedNodeMap" type object
  window.alert(attributes.border.value); // 1
  window.alert(attributes.id.value);     // elem1
  attributes.border.value = thickness;
}

function delete(){
  var elem1 = document.getElementById("elem1");
  elem1.removeAttribute('border');
}
   </script>
  </head>
  <body>
    <table border="1" id="elem1">
      <tr><td>a</td><td>b</td></tr>
      <tr><td>c</td><td>d</td></tr>
    </table>
   <form>
     <input type="button" value="Change the thickness" onClick="change(2);">
     <input type="button" value="Delete" onClick="delete();">
   </form>
  </body>
</html>
```

Notatki

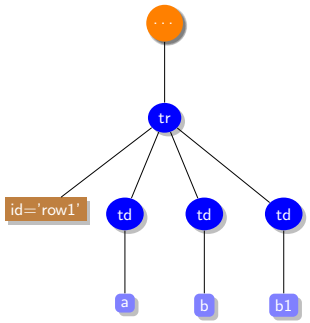
Inserting a new table cell at the end of a row

Methods: createElement(), createTextNode() and appendChild()

```
<html>
  <head>
    <title>An example</title>
    <script>
function insert(){
  var newTD = document.createElement("td");
  var newTextNode = document.createTextNode("b1");
  newTD.appendChild(newTextNode);
  var row1 = document.getElementById("row1");
  row1.appendChild(newTD);
}
    </script>
  </head>
  <body>
    <table border="1">
      <tr id="row1"><td>a</td><td>b</td></tr>
      <tr><td>c</td><td>d</td></tr>
    </table>
   <form>
     <input type="button" value="Insert" onClick="insert();">
   </form>
  </body>
</html>
```

a	b
c	d

a	b	b1
c	d	



Notatki

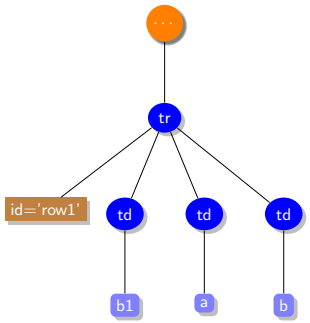
Inserting a new table cell at the beginning of a row

The `insertBefore()` method

```
1  <html>
2    <head>
3      <title>An example</title>
4      <script>
5  function insert(){
6    var newTD = document.createElement("td");
7    var newTextNode = document.createTextNode("b1");
8    newTD.appendChild(newTextNode);
9    var row1 = document.getElementById('row1');
10   var cell1 = row1.getElementsByTagName("td").item(0);
11   row1.insertBefore(newTD,cell1);
12 }
13     </script>
14   </head>
15   <body>
16     <table border="1">
17       <tr id='row1'><td>a</td><td>b</td></tr>
18       <tr><td>c</td><td>d</td></tr>
19     </table>
20     <form>
21       <input type="button" value="Insert" onClick="insert();">
22     </form>
23   </body>
24 </html>
```

a	b
c	d

b1	a	b
c	d	



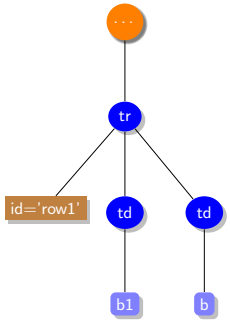
Notatki

Replacing a table cell

The `replaceChild()` method

```
1  <html>
2    <meta http-equiv="Content-Type" content="text/html; charset=
         UTF-8">
3    <head>
4      <title>An example</title>
5      <script>
6  function replace(){
7    var newTD = document.createElement("td");
8    var newTextNode = document.createTextNode("b1");
9    newTD.appendChild(newTextNode);
10   var row1 = document.getElementById('row1');
11   var cell1 = element.getElementsByTagName("td").item(0);
12   row1.replaceChild(newTD,cell1);
13 }
14     </script>
15   </head>
16   <body>
17     <table border="1">
18       <tr id='row1'><td>a</td><td>b</td></tr>
19       <tr><td>c</td><td>d</td></tr>
20     </table>
21     <form>
22       <input type="button" value="Replace" onClick="replace();
             ">
23     </form>
24   </body>
25 </html>
```

a	b
c	d



Notatki

Access to CSS styles

The style object

- CSS feature $\longmapsto$ Property of the style object
  - background-color $\longmapsto$ style.backgroundColor
  - border-top-width $\longmapsto$ style.borderTopWidth
  - Exceptions:
    - float $\longmapsto$ style.cssFloat
    - class $\longmapsto$ style.className
    - for $\longmapsto$ style.htmlFor

red a	b
c	d

```
1  <html>
2    <head>
3      <meta http-equiv="Content-Type" content="text/
           html; charset=UTF-8" />
4      <title>An example</title>
5      <script>
6  function color(value){
7    var cell1 = document.getElementById("cell1");
8    cell1.style.backgroundColor = value;
9  }
10     </script>
11   </head>
12   <body>
13     <table border="1">
14       <tr><td id="cell1">a</td><td>b</td></tr>
15       <tr><td>c</td><td>d</td></tr>
16     </table>
17     <form>
18       <input type="button" value="Red" onClick="
             color('#FF0000');">
19       <input type="button" value="Green" onClick="
             color('#00FF00');">
20     </form>
21   </body>
22 </html>
```

Notatki

Event handling

Changing the background color after clicking a table cell

```
1  <!DOCTYPE html>
2  <meta charset="UTF-8">
3  ...
4      <script>
5    function changeColor(){
6      var cell = document.getElementById("cell");
7      cell.style.backgroundColor='red';
8    }
9    /*********************************/
10   function displayAlert(){
11       alert("displayAlert()");
12   }
13   /*********************************/
14   function load() {
15     var table = document.getElementById("table");
16     table.addEventListener("click", displayAlert, false); //First, 'changeColor()' will be executed,
           followed by 'displayAlert ()'. If the third parameter is 'true' then 'displayAlert()' will be
           executed first, then 'changeColor ()'
17     var cell = document.getElementById("cell");
18     cell.addEventListener("click", changeColor, false);
19   }
20     </script>
21     ...
22   <body onload="load();">
23     <table id="table" border="1">
24       <tr><td id="cell">One</td></tr>
25       <tr><td>Two</td></tr>
26     </table>
27   </body>
28 </html>
```

Notatki

Own events

```
...
<script>
  function load(){
    const form = document.querySelector('form');

    // Listen for the 'userLogin' event
    form.addEventListener("userLogin", function(e) {
      console.info("Event is: ", e);
      console.info("Custom data is: ", e.detail);
    });

    const button = document.getElementById("button");
    button.addEventListener('click', function() {

      // Create the 'userLogin' event
      const myEvent = new Event('userLogin');
      const myEvent = new CustomEvent("userLogin", {
        detail: {
          username: document.getElementById("login").value
        }
      });

      // Trigger it!
      form.dispatchEvent(myEvent);
    });
  }
  </script>
```

```
...
<body onLoad="load()">
  <form>
    <input
      id="login"
      type="text">
    <button
      id="button"
      type="button">Log in</button>
  </form>
<body>
...
```


Notatki

General characteristics

- Library for creating graphical interfaces
- Characteristics:
  - Virtual DOM
  - JSX

Notatki

The "Hello World" example 1

The functional component named 'Hello'

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
    // Create a component named 'Hello'
      function Hello() {
        return <h1>Hello World!</h1>; // ⇔ React.createElement('h1', null, 'Hello World!');
      }
      const container = document.getElementById('root');
      const root = ReactDOM.createRoot(container);
      root.render(<Hello />);
    </script>
  </body>
</html>
```

index.html

Notatki

The "Hello World" example 1

The class component named 'Hello'

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
    <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
    <script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
  </head>
  <body>
    <div id="root"></div>
    <script type="text/babel">
    // Create a component named 'Hello'
    class Hello extends React.Component {
      render() {
        return <h1>Hello World!</h1>;
      }
    }
    const container = document.getElementById('root');
    const root = ReactDOM.createRoot(container);
    root.render(<Hello />);
  </script>
  </body>
</html>
```

index.html

Notatki

The "Hello World" example 2

```
$ npx create-react-app myapp
$ cd myapp
$ npm start
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-
        width, initial-scale=1" />

    <title>React App</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

public/index.html

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import Hello from './Hello';

const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<Hello />);
```

src/index.js

```
import React from 'react';

function Hello() {
  return (
  <h1> Hello World! </h1>
  );
}

export default Hello;
```

src/Hello.js

Notatki

JSX

Example 1

```
class Hello extends React.Component {
  render() {
    return <div>Hello World 1+2={1+2}</div>;
  }
```

Generated HTML

```
<div>Hello World 1+2=3</div>
```

Example 2

```
class Hello extends React.Component {
  render() {
    const elements = ["Element  1", "Element 2"];
  const multiLine = (<ul className='name'>
      <li>{elements[0]}</li>
      <li>{elements[1]}</li>
    </ul>);

    return multiLine;
 }
```

Generated HTML

```
<ul class='name'>
  <li>Element 1</li>
  <li>Element 2</li>
</ul>
```

Notatki

Component life cycle

```
1  class Hello extends React.Component {
2
3    constructor(props) {
4      super(props);
5      console.log('constructor()');
6    }
7
8    componentDidMount() {
9      console.log('componentDidMount()');
10   }
11
12   componentWillUnmount() {
13     console.log('componentWillUnmount()');
14   }
15
16   render() {
17       console.log('render()');
18
19       return <div>Hello World</div>;
20   }
21 }
22
23 const container = document.getElementById('root');
24 const root = ReactDOM.createRoot(container);
25 root.render(<Hello />);
```

Console

```
constructor()
render()
componentDidMount()
componentWillUnmount()
```

The full list of methods can be found at `https://en.reactjs.org/docs/react-component.html`

Notatki

Passing data to a component

```
1 class Hello extends React.Component {
2   render() {
3     return <h1>Hello {this.props.welcome}</h1>;
4   }
5 }
6
7 const container = document.getElementById('root');
8 const root = ReactDOM.createRoot(container);
9 root.render(<Hello welcome="Hello World"/>);
```

Generated HTML

```
<h1>Hello World</h1>
```

Notatki

Conditional rendering

```
1  class Even extends React.Component {
2    render() {
3      return <div>The number is even</div>
4    }
5  }
6
7  class Odd extends React.Component {
8    render() {
9      return <div>The number is odd</div>
10   }
11 }
12 ...
13 const container = document.getElementById('root');
14 const root = ReactDOM.createRoot(container);
15 root.render(<Number value={Math.floor(Math.random() * 10)} />);
```

```
1 function Number(props) {
2   if (props.value % 2 == 0)
3     return <Even />;
4   else
5     return <Odd />;
6 }
```

Function component

```
1  class Number extends React.Component {
2    constructor(props) {
3      super(props);
4    }
5
6    render() {
7      if (this.props.value % 2 == 0)
8        return <Even />;
9      else
10       return <Odd />;
11   }
12 }
```

Class component

Notatki

Lists and keys

```
1  class Elements extends React.Component {
2    render() {
3      const elements = [
4        { id: 'e1', name: 'Element1' },
5        { id: 'e2', name: 'Element2' }
6      ];
7      const items = elements.map((element) =>
8        <li key={element.id}>
9          {element.name}
10       </li>
11     );
12
13     return (
14       <ul>
15         {items}
16       </ul>
17     );
18   }
19 }
20 const container = document.getElementById('root');
21 const root = ReactDOM.createRoot(container);
22 root.render(<Element />);
```

Generated HTML

```
<ul>
<li key='e1'>Element1</li>
<li key='e2'>Element2</li>
</ul>
```

Notatki

Events

The first way

```
1  class Hello extends React.Component {
2    render() {
3      return <button onClick={(function (event) {
4        console.log(this.props.welcome);
5        console.log(event._reactName);
6      }).bind(this)}>Click me</button>
7    }
8  }
9
10 const container = document.getElementById('root');
11 const root = ReactDOM.createRoot(container);
12 root.render(<Hello welcome="Hello World"/>);
```

The second way

```
1  class Hello extends React.Component {
2    print(event) {
3      console.log(this.props.welcome);
4      console.log(event._reactName);
5    }
6    render() {
7      return <button onClick={this.print.bind(this)}>
             Click me</button>
8    }
9  }
10 const container = document.getElementById('root');
11 const root = ReactDOM.createRoot(container);
12 root.render(<Hello welcome="Hello World"/>);
```

After pressing the button

```
Hello World
onClick
```

Notatki

Component data storage

```
1  class Hello extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { firstName: "John", lastName: "Doe"
             };
5    }
6
7
8    clear = () => {
9      this.setState({ firstName: '', lastName: '' })
10   }
11   render() {
12     return (<div><h1>Hello {this.state.firstName} {
           this.state.lastName} </h1>
13     <button onClick={this.clear}>Clear</button></div
           >)
14   }
15 }
16 const container = document.getElementById('root');
17 const root = ReactDOM.createRoot(container);
18 root.render(<Hello />);
```

Notatki

Form handling

```
1  class Form extends React.Component {
2    constructor(props) {
3      super(props);
4      this.state = { value: '' };
5    }
6
7    handleChange = (event) => {
8      this.setState({ value: event.target.value });
9    }
10
11   handleSubmit = (event) => {
12     alert('A name was submitted: ' + this.state.value);
13     event.preventDefault();
14   }
15   h
16   render() {
17     return (
18     <form onSubmit={this.handleSubmit}>
19       <label>
20         Name:
21         <input type="text" value={this.state.value} onChange={this.handleChange} />
22     </label>
23     <input type="submit" value="Submit" />
24     </form>
25     );
26   }
27 }
28
29 const container = document.getElementById('root');
30 const root = ReactDOM.createRoot(container);
31 root.render(<Form />);
```

Notatki

Component composition

```
1  class Hello extends React.Component {
2    render() {
3      return (
4        <div>
5          <Welcome text="Hello " />
6          <Welcome text="World" />
7        </div>
8      );
9    }
10 }
11
12 class Welcome extends React.Component {
13   render() {
14     return <span>{this.props.text}</span>
15   }
16 }
17
18 const container = document.getElementById('root');
19 const root = ReactDOM.createRoot(container);
20 root.render(<Hello />);
```

Generated HTML

```
<div>
<span>Hello </span>
<span>World</span>
</div>
```

Notatki

Fragments

```
class Hello extends React.Component {
  render() {
    return (
      <div>
        <Welcome text="Hello " />
        <Welcome text="World" />
      </div>
    );
  }
}

class Welcome extends React.Component {
  render() {
    return <span>{this.props.text}</span>
  }
}

...
```

Generated HTML

```
<div>
<span>Hello </span>
<span>World</span>
</div>
```

```
class Hello extends React.Component {
  render() {
    return (
      <div>
        <Welcome text="Hello " />
        <Welcome text="World" />
      </div>
    );
  }
}

class Welcome extends React.Component {
  render() {
    return <React.Fragment>{this.props.text}</React.
        Fragment>
    // return <>{this.props.text}</>
  }
}

...
```

Generated HTML

```
<div>
Hello
World
</div>
```

Notatki

Lifting State Up

```
class Echo extends React.Component {
  constructor(props) {
    super(props);
    this.handleTextChange = this.handleTextChange.
        bind(this);
    this.state = { text: '' };
  }

  handleTextChange(newText) {
    this.setState({ text: newText });
  }

  render() {
    return (
      <React.Fragment>
        <EchoInput text={this.state.text}
          handleTextChange={this.handleTextChange} />
        <EchoOutput text={this.state.text} />
      </React.Fragment>
    );
  }
}
const container = document.getElementById('root');
const root = ReactDOM.createRoot(container);
root.render(<Echo />);
```

```
  class EchoOutput extends React.Component {
    render() {
      return (
      <div>Output: {this.props.text}</div>
      );
    }
  }
```

```
class EchoInput extends React.Component {
  constructor(props) {
    super(props);
    this.handleTextChange = this.handleTextChange.
        bind(this);
  }

  handleTextChange(e) {
    this.props.handleTextChange(e.target.value);
  }

  render() {
    return (
    <input value={this.props.text}
      onChange={this.handleTextChange} />
    );
  }
}
```

Source: https://www.geeksforgeeks.org/lifting-state-up-in-reactjs/

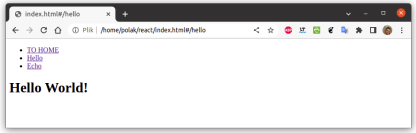
Notatki

Creating subpages

The 'React Router' library

```
...
<head>
...
  <script src='https://unpkg.com/react-router-dom@5.0.0/umd/react-router-dom.min.js'></script>
</head>
...
class Route extends React.Component {
  render() {
    return (<ReactRouterDOM.HashRouter>
    <ul>
      <li><ReactRouterDOM.Link to="/">TO HOME</ReactRouterDOM.Link></li>
      <li><ReactRouterDOM.Link to="/hello">Hello</ReactRouterDOM.Link></li>
      <li><ReactRouterDOM.Link to="/echo">Echo</ReactRouterDOM.Link></li>
    </ul>
      <ReactRouterDOM.Switch>
        <ReactRouterDOM.Route path="/echo">
          <Echo />
        </ReactRouterDOM.Route>
        <ReactRouterDOM.Route path="/hello">
          <Hello />
        </ReactRouterDOM.Route>
        <ReactRouterDOM.Route path="/">
          <Home />
        </ReactRouterDOM.Route>
      </ReactRouterDOM.Switch>
    </ReactRouterDOM.HashRouter>
    );
  }
}
```

```
function Home() {
  return <h2>Home</h2>;
}
...
```



Notatki

Hooks

Hooks

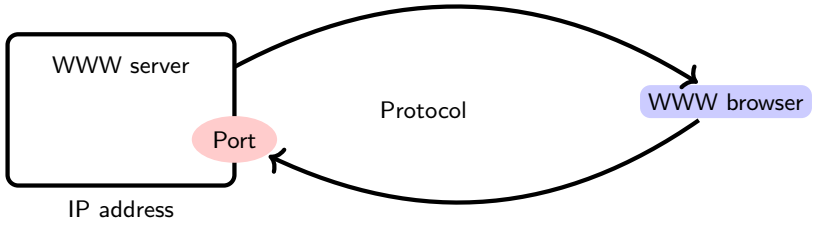
- useState
- useReducer
- useEffect
- useRef
- useLayoutEffect
- useContext
- useImperativeHandle
- useMemo
- useCallback

```
function Component() {
  const [count, setCount] = React.useState(0);

  return (
    <div>
      <p>Pressed {count} times</p>
      <button onClick={() => setCount(count + 1)}>Press me</button>
    </div>
  );
}
```

Notatki

Client-Server Model

WWW service



Notatki

Selected HTTP protocol commands (methods)

The "GET" command

Request URL:
`http://www.icsr.agh.edu.pl/index.html`

```
1 GET /index.html HTTP/1.1
2 Host: www.icsr.agh.edu.pl
3
```

Request

```
1  HTTP/1.1 200 OK
2  Date: Mon, 09 Aug 2013 17:02:08 GMT
3  Server: Apache/2.4.4 (UNIX)
4  Content-Length: 1776
5  Content-Type: text/html; charset=utf-8
6
7  <!DOCTYPE html>
8  <html>
9  ...
10 </html>
```

Response

The "POST" command

Request URL: `http://www.icsr.agh.edu.pl/cgi-bin/search.cgi`

```
1 POST /cgi-bin/search.cgi HTTP/1.1
2 Host: www.icsr.agh.edu.pl
3 Content-Length: 46
4
5 query=alpha+complex&casesens=false&cmd=submit
```

Request

```
1  HTTP/1.1 200 OK
2  Date: Mon, 09 Aug 2013 17:02:20 GMT
3  Server: Apache/2.4.4 (UNIX)
4  Content-Length: 1776
5  Content-Type: text/html; charset=utf-8
6  Connection: close
7
8  <!DOCTYPE html>
9  <html>
10 ...
11 </html>
```

Response

Notatki

Sending data from the HTML form

Notatki

Approving the form ⟼ data encoding ⟼ sending the data to a web server

```
1 <form method="..." enctype="..." action="...">
2 ...
3 </form>
```

- GET
- POST

- application/x-www-form-urlencoded
- multipart/form-data

Encoding procedure „application/x-www-form-urlencoded"

Example

Notatki

```
1 <form action="http://www.serwer.com/script">
2 Login:    <input name="login" type="TEXT"><br>
3 Password: <input name="password" type="PASSWORD">
4 </form>
```

HTML document

Login: Jan

Password: Kowalski (Nowak)

```
1 login=Jan&password=Kowalski+%28Nowak%29
```

Encoded data

Encoding procedure „multipart/form-data”

Example

```
1 <form  action="..." method="POST" enctype="multipart/form-data">
2 <input name="login" type="TEXT">
3 <input name="password" type="PASSWORD">
4 <input name="file" type="FILE"  accept="image/jpeg,image/gif">
5 </form>
```

Jan

Kowalski (Nowak)

image.jpg

```
1  POST /skrypt HTTP/1.0
2  Content-Length: 775
3  Content-Type: multipart/form-data; boundary=-------------------------815276501818664599101790669 2
4
5  ---------------------------815276501818664599101790669 2
6  Content-Disposition: form-data; name="login"
7
8  Jan
9  ---------------------------815276501818664599101790669 2
10 Content-Disposition: form-data; name="password"
11
12 Kowalski (Nowak)
13 ---------------------------815276501818664599101790669 2
14 Content-Disposition: form-data; name="file"; filename="image.jpg"
15 Content-Type: image/jpeg
16 Content-Transfer-Encoding: binary
17
18 The content of 'image.jpg'
19 ---------------------------815276501818664599101790669 2
```

Notatki

Node.js

General characteristics

- Provides JavaScript on the server side
- Uses V8 JavaScript Engine
- System for creating network services with asynchronous I/O
- Uses the event-driven programming paradigm
- It is well-suited for writing applications that require real-time communication between the browser and the server
- A single instance of Node.js acts as a single thread

Event loop — entity that handles / processes external events and converts them to callback functions

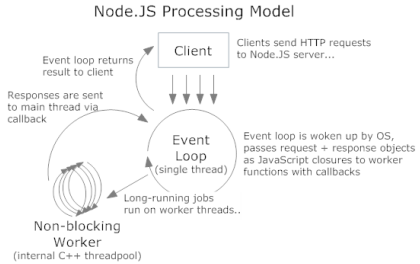


Figure: The diagram of the event loop operation in Node.js

Source: http://www.aaronstannard.com/post/2011/12/14/Intro-to-NodeJS-for-NET-Developers.aspx

Notatki

The "Hello World" example

```
1 #!/usr/bin/node
2 console.log("Hello World");
```

hello.js

Notatki

Input and output of data

```
1  process.stdout.write('1');
2  process.stdout.write('2');
3  console.log(3)
4
5  /*
6  console.log = function(d) {
7    process.stdout.write(d + '\n');
8  };
9  */
10
11 process.stdin.setEncoding('utf8');
12 process.stdout.write('Enter data - pressing ^ D will finish entering them\n');
13 process.stdin.on('readable', function() {
14   var chunk = process.stdin.read();
15   if (chunk !== null) {
16     process.stdout.write('Read: ' + chunk);
17   }
18 });
19 console.log("The end of the script has been reached");
```

script.js

Notatki

Access to environment variables, command line support

```
1 //Reading the value of the 'HOME' environment variable
2 console.log("Your home directory is: "+process.env['HOME']);
3
4 //Displays the value of the command line arguments
5 console.log("The command line arguments are:");
6 process.argv.forEach(function(value, index, array) {
7   console.log('\t'+index + ': ' + value);
8 });
```

script.js

Notatki

Module

Creation of module

```
1 var myModule = require('myModule');
2 //myModule = require("./myModule");
3 /*********************************/
4 console.log(myModule.variable1); //undefined
5 console.log(myModule.variable2); //2
6 console.log(myModule.variable3); //undefined
7 console.log(myModule.variable4); //4
8 console.log(myModule.fun1());   //"fun1"
9 console.log(myModule.fun2());   //Error
10 /*********************************/
11 console.log(myModule); //{ variable2: 2, variable4: 4, fun1: [Function]
       }
12 console.log(myModule()); //Error
13
```

script.js

```
1 $ export NODE_PATH='dir1:dir2:...:dirN'
```

```
1  variable1 = 1;
2  exports.variable2 = 2;
3  var variable3 = 3;
4  var variable4;
5  module.exports.variable4 = 4;
6  var exports.variable5 = 5; //SyntaxError: Unexpected token .
7  exports.fun1 = function () {
8    return "fun1";
9  };
10 fun2 = function () {
11   return "fun2";
12 };
13 console.log(module);
14 /* Module {
15   ...,
16   exports: { variable2: 2, variable4: 4, fun1: [Function] },
17   ... } */
18 exports = function() {
19   return "fun3";
20 }
21 module.exports = function() {
22   return "fun4";
23 }
24 console.log(module.exports); //{ variable2: 2, variable4: 4, fun1:
       [Function] }
25 console.log(exports);       //{ variable2: 2, variable4: 4, fun1:
       [Function] }
26 console.log(module.exports == exports); //true
```

node_modules/myModule.js

Notatki

Module

Using modules

Package support

```
$ npm install package_name
# Modules ⟼ ./node_modules/
# Executables ⟼ ./node_modules/.bin/
# Manuals ⟼ are not installed

$ npm install --global  package_name
# Modules ⟼ {prefix}/lib/node_modules/
# Executables ⟼ {prefix}/bin/
# Manuals ⟼ {prefix}/share/man/
# {prefix} = e.g. /usr

$ npm link package_name
#  Executes: ln -s {prefix}/lib/node_modules/package_name/ ./node_modules/

$ npx program_name
```

Notatki

File support

```
1  var   fs = require("fs");
2  // Check if the file exists
3  try{ fs.accessSync('file.txt'); }
4  catch(err){ fs.writeFileSync('file.txt', '1'); }
5  fs.readFile('file.txt', 'utf-8', function (error,
       data) {
6      if (error) throw error;
7      console.log("Read value: "+data);
8  });
9
10 fs.writeFile('file.txt', '2', function (error) {
11     if (error) throw error;
12     console.log('The value 2 has been saved');
13 });
```

Invalid version

```
1  var   fs = require("fs");
2  try{ fs.accessSync('file.txt') }
3  catch(err){ fs.writeFileSync('file.txt', '1'); }
4
5  fs.readFile('file.txt', 'utf-8', function (error,
       data) {
6    if (error) throw error;
7    console.log("Read value: "+data);
8
9    fs.writeFile('file.txt', '2', function (error) {
10     if (error) throw error;
11     console.log('The value 2 has been saved');
12   });
13 });
```

Correct version

Notatki

SQLite 3 database support

```
var sqlite3 = require('sqlite3');

var db = new sqlite3.Database(':memory:'); //returns 'Database' object

db.serialize(function() {
  db.run("CREATE TABLE products (info TEXT)");
  var stmt = db.prepare("INSERT INTO products VALUES (?)");//returns  'Statement' object
  for (var i = 0; i < 2; i++) {
    stmt.run("Product " + i);
  }
  stmt.finalize();

  jsonData = { products: [] };
  db.each("SELECT rowid AS id, info FROM products", function(err, row) {
      jsonData.products.push({ id: row.id, info: row.info });
    }, function () {
      console.log(JSON.stringify(jsonData)); //JSON.stringify - built-in JS function
    }
  );
});
db.close();
```

bd.js

Notatki

C++ Addons

The "Hello World" example

```
/*
The following program is equivalent to the following JS code:
  exports.hello = function() { return 'world'; };
  */

#include <node.h>
using namespace v8;

void Method(const FunctionCallbackInfo<Value>& args) {
  Isolate* isolate = args.GetIsolate();
  args.GetReturnValue().Set(String::NewFromUtf8(isolate, "world"));
        //Specify what function returns
}

void init(Local<Object> exports) {
  NODE_SET_METHOD(exports, "hello", Method); //Associate the name '
        hello' with the above C++ method and export it
}

NODE_MODULE(NODE_GYP_MODULE_NAME, init) //there is no semicolon
```

hello.cc

```
{
  "targets": [
    {
      "target_name": "hello",
      "sources": [ "hello.cc" ]
    }
  ]
}
```

binding.gyp

Compilation and program execution

```
$ npx node-gyp configure
$ npx node-gyp build
$ node hello.js
world
```

```
var addon = require('./build/Release/hello');
console.log(addon.hello());
```

hello.js

Notatki

HTTP support

Script skeleton

```
1  var http = require("http");
2
3  function requestListener(request, response) {
4    console.log("A request from the client has appeared");
5    response.writeHead(200, {"Content-Type": "text/plain"});
6    response.write("Hello World");
7    response.end();
8  }
9  var serwer = http.createServer(requestListener);
10 serwer.listen(8080);
11 console.log("Server started");
```

server.js

```
1 var http = require("http");
2
3 http.createServer(function(request, response) {
4   console.log("A request from the client has appeared");
5   response.writeHead(200, {"Content-Type": "text/plain"});
6   response.write("Hello World");
7   response.end();
8 }).listen(8080);
9 console.log("Server started");
```

Alternative version

Testing the operation of the script

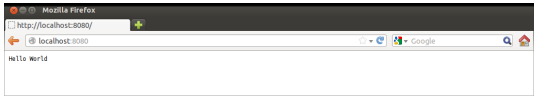


Figure: In the web browser

Notatki

URL parameter support

```
1  ...
2  function requestListener(request,response) {
3    console.log("A request from the client has appeared");
4    var url = new URL(request.url, `http://${request.headers.host}`);
5
6    console.log(url);
7
8    response.writeHead(200, {"Content-Type": "text/plain"});
9    response.write('\n');
10   response.write(url.pathname+'\n');
11   response.write('Login: '+url.searchParams.get('login')+'\n');
12   response.write('Password: '+url.searchParams.get('password')+'\n');
13   response.end();
14 }
15 ...
```

server.js

Output

Notatki

Form support

The "application/x-www-form-urlencoded" encoding support

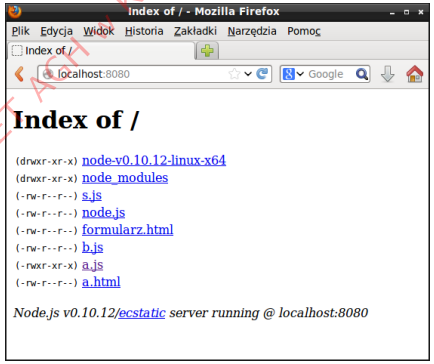
```
var qs = require('querystring');
...
function requestListener(request,response) {
  var url = new URL(request.url, `http://${request.headers.host}`);
  if(url.pathname == '/form'){ //generating the form
    response.writeHead(200, {"Content-Type": "text/html"});
    response.write('<form method="POST" action="/submit">');
    response.write('<input name="login" value="Jan">');
    response.write('<input name="password" value="Kowalski (Nowak) ąę">');
    response.write('<input type="submit">');
    response.write('</form>');
    response.end();
  }
  if(url.pathname == '/submit') { //processing of the form content
    if(request.method=='GET') {
      response.writeHead(200, {"Content-Type": "text/plain; charset=utf-8"});
      response.write(url.searchParams.get('login')+'\n'); //the browser will write: "Jan\n"
      response.write(url.searchParams.get('password')+'\n');; //the browser will write: "Kowalski (Nowak) ąę\n"
      response.end();
    }
    else if(request.method=='POST') {
      var body='';
      request.on('data', function (data) {
        body +=data;
      });
      request.on('end',function(){
        var data =  qs.parse(body); //body contains "login=Jan&password=Kowalski+%28Nowak%29+%C4%85%C4%99"
        response.writeHead(200, {"Content-Type": "text/plain; charset=utf-8
          "});
        response.write(data.login+'\n'); //the browser will write: "Jan\n"
        response.write(data.password+'\n'); //the browser will write: "Kowalski (Nowak) ąę\n"
        response.end();
      });
    }
  }
}
...
```

Notatki

A stand-alone web server

Installation and startup

```
$ npx http-server -h
usage: http-server [path] [options]

options:
  -p                 Port to use [8080]
  -a                 Address to use [0.0.0.0]
  -d                 Show directory listings [true]
  -i                 Display autoIndex [true]
  -e --ext           Default file extension if none supplied [none]
  -s --silent        Suppress log messages from output
  -h --help          Print this list and exit.
  -c                 Set cache time (in seconds). e.g. -c10 for 10
                       seconds.
                     To disable caching, use -c-1.

$ npx http-server
Starting up http-server, serving ./ on port: 8080
Hit CTRL-C to stop the server
```



Notatki

Creating the application skeleton

```
$ mkdir MySite
$ cd MySite
$ vi  package.json
$ npm install # Installing dependencies
```

```
{
  "name": "MySite",
  "version": "0.0.1",
  "private": "true",
  "dependencies": {
    "express": "*",
    "morgan": "*"
  }
}
```

package.json

Notatki

The main application file

```
var express = require('express'),
    logger = require('morgan');

var app = express();

// If you uncomment the code snippet below, then
// the following function will be executed every
     time the app receives a request.
/*
app.use(function (req, res, next) {
  console.log('Time:', Date.now());
  next();
});
*/
app.use(logger('dev'));

app.get('/', function (req, res) {
  res.send('<h1>Hello World!</h1>');
});

app.listen(3000);
```

app.js

Terminal 1

```
$ node app
GET / 200 3.137 ms - 21
```

Terminal 2

```
$ curl http://localhost:3000/
<h1>Hello World!</h1>
```

Notatki

The main application file

```
$ npm install #installing dependencies
```

```
{
    ...
    "dependencies": {
        ...
        "pug": "*",
    }
}
```

package.json

Notatki

The main file

```
var express = require('express'),
    logger = require('morgan');

var app = express();
var router = express.Router();

app.set('view engine', 'pug');
app.set('views', __dirname + '/views');

app.use(logger('dev'));
app.use(express.static(__dirname + '/public'));

router.get('/', function (req, res) {
  res.render('index',
    { title : 'Przykład' }
  )
});

app.use('/', router);

app.listen(3000);
```

app.js

Notatki

The "Pug" file

```
doctype html
html(lang='pl')
  head
    title #{title}
    link(rel='stylesheet', href='/stylesheets/style.
        css')
  body
    header
      h1 Moja strona
    main
      p
        | Witaj Świecie
        | Witaj Świecie
      p
        | Witaj Świecie
    aside
      h1 Nagłówek
      p
        | Treść ramki
    footer
      p Aplikacja stworzona w oparciu o framework
          Express
```

views/index.pug

```
<!DOCTYPE html>
<html lang="pl">
  <head>
    <title>Przykład</title>
    <link rel="stylesheet" href="/stylesheets/style.
        css">
  </head>
  <body>
    <header>
      <h1>Moja strona</h1>
    </header>
    <main>
      <p>
        Witaj Świecie
        Witaj Świecie
      </p>
      <p>Witaj Świecie</p>
    </main>
    <aside>
      <h1>Nagłówek</h1>
      <p>Treść ramki</p>
    </aside>
    <footer>
      <p>Aplikacja stworzona w oparciu o framework
          Express</p>
    </footer>
  </body>
</html>
```

Notatki

"CSS" files

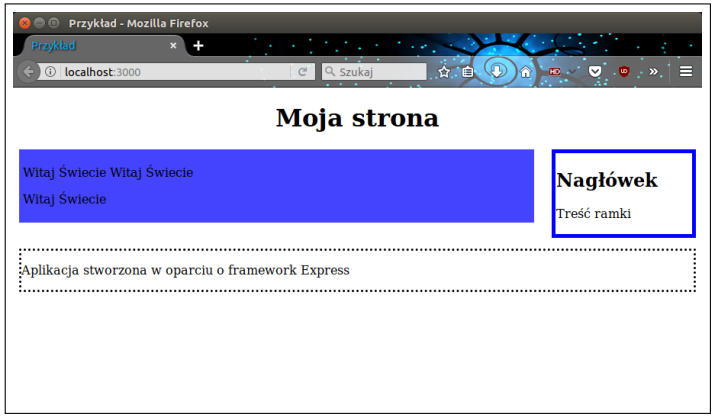
```
aside {
  float: right;
  border: 5px solid blue;
  padding: 1px;
  margin-bottom: 14px;
  width: 20%;
}
main {
  float: left;
  background-color: #44f;
  padding: 5px;
  width: 75%;
}
footer {
  clear:both;
  border-style: dotted;
}
header {
  text-align: center;
}
```

public/stylesheets/style.css

Notatki

Starting the application

```
$ node app
GET / 304 847.588 ms - -
GET /stylesheets/style.css 304 4.478 ms - -
```



Notatki

Route parameters

```
...
app.get('/students/:studentName/departments/:departmentId', function (req, res) {
  res.send(`studentName=${req.params.studentName}, departmentId=${req.params.departmentId}`)
});
...
```

app.js

Terminal 1

```
$ node app.js
GET /students/Jan%20Kowalski/departments/1 200 4.402 ms - 53
```

Terminal 2

```
$ curl http://localhost:3000/students/Jan%20Kowalski/departments/1
studentName=Jan Kowalski, departmentId=1
```

Notatki

Forms handling

- `http://localhost:3000/` — home page with the form
- `http://localhost:3000/submit` — data from the form

The "GET" method

```
1  ...
2  router.get('/', function (req, res) {
3    res.send(`
4    <form method="GET" action="/submit">
5    <input name="login" value="Jan">
6    <input name="password" value="Kowalski (Nowak) ąę"
         >
7    <input type="submit">
8    </form>
9    `);
10 });
11
12 router.get('/submit', function (req, res) {
13   res.send(`
14   login=${req.query.login}
15   <br>
16   password=${req.query.password}
17   `);
18 });
19 ...
```

app.js

The "POST" method

```
1  ...
2  app.use(express.urlencoded({ extended: false }));
3
4  router.get('/', function (req, res) {
5    res.send(`
6    <form method="POST" action="/submit">
7    <input name="login" value="Jan">
8    <input name="password" value="Kowalski (Nowak) ąę"
         >
9    <input type="submit">
10   </form>
11   `);
12 });
13
14 router.post('/submit', function (req, res) {
15   res.send(`
16   login=${req.body.login}
17   <br>
18   password=${req.body.password}
19   `);
20 });
21 ...
```

app.js

Notatki

The express command

```
1  $ npx express-generator --help
2    Usage: express [options] [dir]
3
4    Options:
5
6        --version        output the version number
7    -e, --ejs            add ejs engine support
8        --pug            add pug engine support
9        --hbs            add handlebars engine support
10   -H, --hogan          add hogan.js engine support
11   -v, --view <engine>  add view <engine> support (dust|ejs|hbs|hjs|jade|pug|twig|vash) (defaults to jade
         )
12       --no-view        use static html instead of view engine
13   -c, --css <engine>   add stylesheet <engine> support (less|stylus|compass|sass) (defaults to plain css
         )
14       --git            add .gitignore
15   -f, --force          force on non-empty directory
16   -h, --help           output usage information
```


Notatki

Generating the "Hello World" application

Using the command express

```
$ npx express-generator --view=pug MySite
 create : MySite/
 create : MySite/public/
 create : MySite/public/javascripts/
 create : MySite/public/images/
 create : MySite/public/stylesheets/
 create : MySite/public/stylesheets/style.css
 create : MySite/routes/
 create : MySite/routes/index.js
 create : MySite/routes/users.js
 create : MySite/views/
 create : MySite/views/error.pug
 create : MySite/views/index.pug
 create : MySite/views/layout.pug
 create : MySite/app.js
 create : MySite/package.json
 create : MySite/bin/
 create : MySite/bin/www

 change directory:
 $ cd MySite

 install dependencies:
 $ npm install

 run the app:
 $ DEBUG=mysite:* npm start
```

Notatki

The 'package.json' file

```
{
  "name": "mysite",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www"
  },
  "dependencies": {
    "cookie-parser": "~1.4.4",
    "debug": "~2.6.9",
    "express": "~4.16.1",
    "http-errors": "~1.6.3",
    "morgan": "~1.9.1",
    "pug": "2.0.0-beta11"
  }
}
```

Notatki

The main file (app.js)

```
1 ...
2 var indexRouter = require('./routes/index');
3 var usersRouter = require('./routes/users');
4
5 var app = express();
6 ...
7 app.use('/', indexRouter);
8 app.use('/users', usersRouter);
9 ...
```

Sample address	The object that supports it
http://localhost:3000/	indexRouter
http://localhost:3000/users/	usersRouter
http://localhost:3000/agh/	None — you will get an error
http://localhost:3000/users/agh/	None — you will get an error

Notatki

Files with route definitions

```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET home page. */
5 router.get('/', function(req, res, next) {
6   res.render('index', { title: 'Express' });
7 });
8
9 module.exports = router;
```

routes/index.js

```
1 var express = require('express');
2 var router = express.Router();
3
4 /* GET users listing. */
5 router.get('/', function(req, res, next) {
6   res.send('respond with a resource');
7 });
8
9 module.exports = router;
```

routes/user.js

Notatki

Other generated files

```
extends layout.pug

block content
  h1= title
  p Welcome to #{title}
```

views/index.pug

```
doctype html
html
  head
    title= title
    link(rel='stylesheet', href='/stylesheets/style.
        css')
  body
    block content
```

views/layout.pug

```
body {
  padding: 50px;
  font: 14px "Lucida Grande", Helvetica, Arial, sans
      -serif;
}

a {
  color: #00B7FF;
}
```

public/stylesheets/style.css

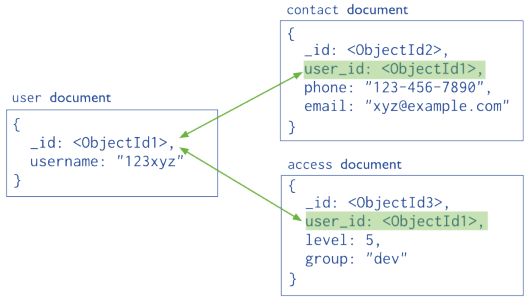
```
$ cd MySite && npm install
$ DEBUG=mysite:* npm start

> mysite@0.0.0 start
> node ./bin/www

mysite:server Listening on port 3000 +0ms
```

Instalowanie zależności i uruchamianie aplikacji

Notatki

The data model in MongoDB

References



Embedded documents

Notatki

Creating database content

```
$ mongo
MongoDB shell version: 4.4.5
connecting to: test
> use testbase1
switched to db testbase1
> db.datacollection.insert({"title":"Express & Mongo"})
> db.datacollection.insert({"title":"Express"})
> db.datacollection.find()
{ "_id" : ObjectId("52f8bb757bade7e2c4741741"), "title" : "Express & Mongo" }
{ "_id" : ObjectId("52f8bd407bade7e2c4741742"), "title" : "Express" }
> db.datacollection.find({"title":"Express"})
{ "_id" : ObjectId("52f8bd407bade7e2c4741742"), "title" : "Express" }
```

Notatki

Installing dependencies

```
$ vi  package.json
$ npm install # Installing dependencies
```

```
...
"dependencies": {
  ...
  "mongodb": "*"
}
...
```

package.json

Notatki

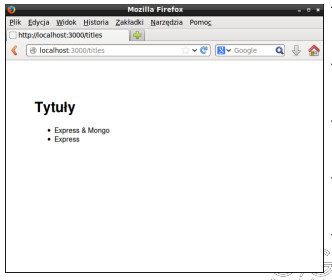
Download and display of data

```
1  ...
2  router.get('/titles', function (req, res, next) {
3    const MongoClient = require('mongodb').MongoClient;
4    const client = new MongoClient('mongodb://localhost:27017', {
         useUnifiedTopology: true });
5    client.connect(function (err) {
6      const db = client.db('testbase1');
7      const collection = db.collection('datacollection');
8      collection.find({}).toArray(function (err, docs) {
9        res.render('titlelist', {
10         "titlelist": docs
11       });
12       client.close();
13     });
14   });
15 });
16
17 module.exports = router;
```

routes/index.js

```
1 extends layout.pug
2
3 block content
4   h1 Tytuły
5   ul
6     each element in titlelist
7       li #{element.title}
```

views/titlelist.pug



Notatki

AJAX

- AJAX (**A**synchronous **Ja**vaScript and **X**ML)
- AJAX = HTML + CSS + DOM + XMLHttpRequest + XML + JavaScript
- Capabilities:
  - Sending a query to the server without reloading the page
  - The application can make quick, incremental updates to the user interface without the need to reload the entire page in the browser
  - Parsing and working with XML documents
- Is it always worth using AJAX?

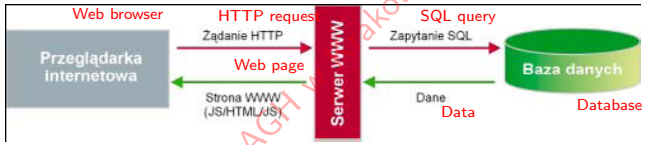


Figure: The scheme of a typical website

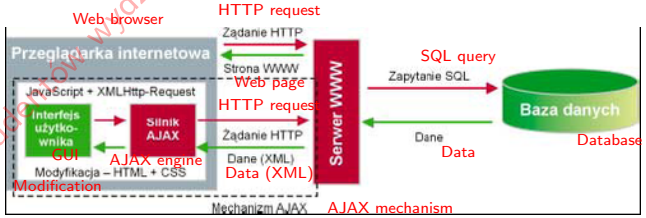


Figure: The scheme of the AJAX website

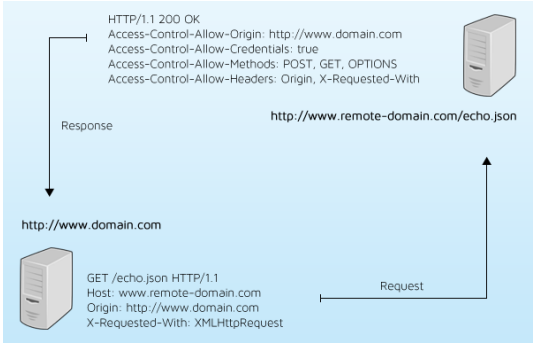
Notatki

Request

```
1  var xhr;
2
3  xhr = new XMLHttpRequest();
4  if (!xhr) {
5    alert('I can not create an XMLHTTP object instance
         ');
6    return;
7  }
8  xhr.onreadystatechange = function() { alertContents(xhr
       ); };
9  // xhr.onreadystatechange = () => alertContents(xhr);
10 xhr.open('GET', "/requested_file.html", true);
11 xhr.send(null);
12 //xhr.open('POST', "/script.cgi", true);
13 //xhr.setRequestHeader('Content-Type','application/x-
       www-form-urlencoded');
14 //xhr.send('field1=value1&field2=value2&...');
```

```
1 <table>
2 <tr><td>Hello</td></tr>
3 </table>
```

requested_file.html

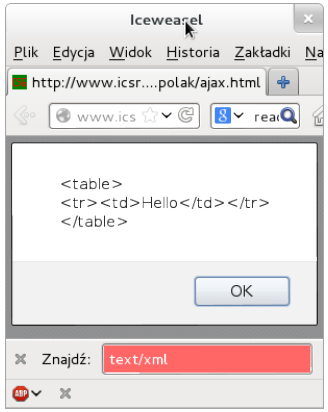


Source: https://zinoui.com/blog/cross-domain-ajax-request

Notatki

Handling the response

```
1  function alertContents(xhr) {
2    if (xhr.readyState === XMLHttpRequest.DONE) { // XMLHttpRequest.DONE ⇔ 4
3      if (xhr.status === 200) {
4        alert(xhr.responseText);
5        var xmldoc = xhr.responseXML;
6        var root_node = xmldoc.getElementsByTagName('td').item(0);
7        alert(root_node.firstChild.data);
8      }
9      else {
10       alert('There was a problem with this task.');
11     }
12 }
```



Notatki

Receiving data in a specific format

```
1  var xhr;
2
3  xhr = new XMLHttpRequest();
4  xhr.responseType = "json";
5  ...
6
7  function alertContents(xhr) {
8    ...
9    alert(xhr.response);
10 }
```

responseType	Received data type
""	DOMString
"text"	DOMString
"arraybuffer"	ArrayBuffer
"blob"	Blob
"document"	Document or XMLDocument
"json"	JSON

Notatki

Handling response through events

```
1  var xhr;
2
3  xhr = new XMLHttpRequest();
4
5  // Called if 'xhr.readyState == XMLHttpRequest.DONE'
6  xhr.addEventListener("load", function (evt) {
7    if (xhr.status === 200) {
8      ...
9    }
10 });
11
12 xhr.addEventListener("error", function (evt) {
13   window.alert('There was a problem with this request.');
14 });
15
16 xhr.open('GET', "/requested_file.html", true);
17 xhr.send(null);
```

Notatki

Promises

- Represent / Store the results of an asynchronous operation[1]
- The results of the operation may not be available yet, but they will be
- Promises are chainable

States of promises

- pending
- fulfilled
- rejected
- settled

[1] Return value (in the case of success) or error (in the case of failure)

Notatki

Promises

Example of use

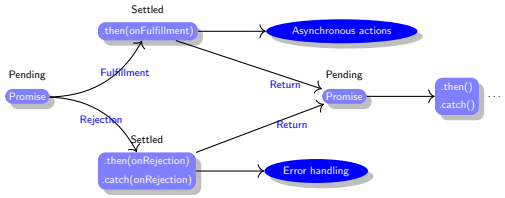
```
1  /*** Creating a promise ***/
2  function createPromise() {
3    return new Promise(function(resolve, reject) {
4      // Calculations performed asynchronously
5      setTimeout(function() {
6        var divider = Math.floor(Math.random() * 3);
7        if(divider != 0)
8          resolve(10/divider); // Fulfill a promise
9        else
10         reject("Attempt to divide by 0"); // Reject a promise
11     }, 2000);
12   });
13 }
14
15 /*** The use of the promise ***/
16 function usePromise() {
17   createPromise() //An instance of the promise has been
         created
18   .then(function(result) {
19     console.log('Division result:', result);
20   })
21   .catch(function(error) {
22     console.log('An error occurred!', error);
23   });
24 }
25
26 usePromise();
```

Output

```
'Division result:' 5
```

Output

```
"An error occurred!" "Attempt to divide by 0"
```



Notatki

Promises

The async / await syntax

That's how promises have been used so far

```
1  function usePromise() {
2    createPromise()
3      .then(function (result) {
4        console.log('Division result : ', result);
5      })
6      .catch(function (error) {
7        console.log('An error occurred! ', error);
8      });
9  }
10
11 usePromise();
```

Using async / await

```
1  async function usePromise() {
2    try {
3      const result = await createPromise();
4      console.log('Division result : ', result);
5    } catch (error) {
6      console.log('An error occurred! ', error);
7    }
8  }
9
10 usePromise();
```

Notatki

Fetch API

General characteristics

- A JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses
- Differences from AJAX:
  - It uses promises and not callbacks
    - A promise that is the result of a query will not reject on HTTP error status even if the response is an HTTP 404 or 500
  - By default, function to execute queries will not send or receive any cookies from the server

Notatki

Fetch API

Example of use

```
...
<script>
  const header = new Headers();
  header.append('Content-Type', 'text/plain');
  const request = new Request('http://localhost:8000/document.html',
    {
      method: 'GET',
      headers: header,
    });
  fetch(request).then(response => {
    if (response.status !== 200)
      return Promise.reject('The query failed');
    else {
      console.log(response); /*Response {
                                          type: "basic",
                                          url: "http://localhost:8000/document.html",
                                          redirected: false,
                                          status: 200,
                                          ok: true,
                                          statusText: "OK",
                                          headers: Headers,
                                          bodyUsed: false
                                        } */
      //console.log(response.arrayBuffer()); //Promise { <state>: "pending" }
      //console.log(response.blob()); //Promise { <state>: "pending" }
      //console.log(response.json()); //Promise { <state>: "pending" }
      //console.log(response.text()); //Promise { <state>: "pending" }
      //console.log(response.formData()); //Promise { <state>: "pending" }
      response.text().then(function (text) {
        console.log(text); //The content of the current file
      });
    } //if (response.status !== 200)
  }).catch(error => console.error(error))
</script>
...
```

document.html

Notatki

The "Hello World" example

```
console.log("Hello World");
```

script.ts

```
console.log("Hello World");
```

script.js

Notatki

Error handling

Typing error

```
1 let alive: boolean = 'abc';
2 console.log(alive);
```

script.ts

```
1 let alive = 'abc';
2 console.log(alive);
```

script.js

Syntax error

```
1 let if = 2;
2 console.log(if);
```

script.ts

```
1 let ;
2 if ( = 2)
3     ;
4 console.log();
5 if ()
6     ;
```

script.js

Notatki

Declaring the type of variable

```
1  let alive:boolean = true;
2  let age:number = 48;
3  let name:string = 'Kowalski';
4
5  let names:string[] = ['Jan','Jerzy'];
6  let children_age:Array<number> = [1, 20, 3];
7  let parents_age:Array<number> = [40, "forty
       one"]; //... Type 'string' is not
       assignable to type 'number'.
8
9  let tuple:[string,number,boolean];
10 tuple = ['1',2,true];
11 console.log(tuple[2]); // true
12 tuple = ['1',2,true,4]; // OK
13 tuple = ['1',2]; //... Type '[string, number]'
        is not assignable to type '[string,
       number, boolean]'. ...
14 tuple = [1,2,3]; //... Type '[number, number,
       number]' is not assignable to type '[
       string, number, boolean]'.
15
16 enum Eyes {Blue, Green = 4, Grey, Brown};
17 let eye_color = Eyes.Blue;
18 console.log(Eyes[0]); //  Blue
19 console.log(Eyes[4]); //  Green
20 console.log(Eyes[5]); //  Grey
```

```
21 let anything:any = 4;
22 anything = "String"; //OK
23 anything = true; //OK
24 anything.x; //OK
25 anything(); //OK
26 new anything(); //OK
27 let string: string = anything; //OK
28
29 let something: unknown = 4;
30 something = "String"; //OK
31 something = true; //OK
32 something.x; //... Object is of type 'unknown'
33 something(); //... Object is of type 'unknown'
34 new something(); //... Object is of type '
       unknown'
35 let string2: string = something; // ... Type '
       unknown' is not assignable to type '
       string'.
36
37 function print(message): void {
38   console.log(message);
39 }
40
41 function exception(message): never {
42   throw new Error(message);
43 }
44
45 function loop(): never  {
46   while(true){}
47 }
```

Notatki

Defining the type of variables

```
let string:string;
string = 1; //Error, you can not assign a number to
     a string type variable

let number = 1;
number = '2'; //Error, you can not assign a string
     to a numeric type variable
```

script.ts

```
let string:string;
string =  <any> 1; //Now it is OK
//or
string =  1 as any;
let number = 1;
number = <any> '2'; //Now it is OK
//or
number = '2' as any; //Now it is OK
```

script.ts

Notatki

Interfaces

Creating a complex type

```
interface Person {
  first_name: string; //mandatory
  last_name: string;  //mandatory
  age?: number;       //optional
}
/******************************/
let user: Person;
/******************************/
user = {
  first_name: 'Jan',
  last_name: 'Kowalski'
}
/******************************/
user = {
  first_name: 'Jan',
  last_name: 'Kowalski',
  age: '40'//Error: wrong value type
}
/******************************/
user = {
 // Error: 'last_name' is not specified
 first_name: 'Jan'
}
/******************************/
user = {
  first_name: 'Jan',
  last_name: 345, //Error: wrong value type
  age: 'teenager'
}
```

script.ts

Notatki

Interfaces

Specifying the types of functions

```
1  interface stringFunction {
2    (param1:string, param2:string): string;
3  }
4  /*******************************************/
5  let addStrings: stringFunction;
6  let addNumbers: stringFunction;
7  /*******************************************/
8  addStrings = function(string1: string,string2: string) {
9    return string1+string2;
10 } //OK
11
12 addNumbers = function(number1: number, number2: number) {
13   return number1 + number2;
14 } /*...
15 error TS2322: Type '(number1: number, number2: number) => number' is not assignable to type '
       stringFunction'.
16   Types of parameters 'number1' and 'param1' are incompatible.
17     Type 'number' is not assignable to type 'string'.
18
19 */
20 /*******************************************/
21 addStrings('Jan','Kowalski'); //OK
22 addStrings('Kowalski'); //... error TS2346: Supplied parameters do not match any signature of call
       target.
23 addStrings(1,2); //... error TS2345: Argument of type 'number' is not assignable to parameter of type '
       string'.
```

script.ts

Notatki

Interfaces

Indexed type

```
1  interface hashString {
2       [index: string]: string;
3  }
4  let parameters: hashString = {};
5
6  parameters['server'] = 'HP'; // OK
7  let bar:number = parameters['server']; //... error TS2322: Type 'string' is not assignable to type '
       number'.
8  parameters['server'] = 234; //... error TS2322: Type 'number' is not assignable to type 'string'.
```

script.ts

Notatki

Interfaces

Class types

```ts
class Person {
  id:number;
}

interface Validator{
  checkIdUniqueness(s: number): boolean;
}

class Employee extends Person implements Validator {}
```

script.ts

Notatki

Classes

Modifiers

```ts
class Person {
  protected _id:number;
  readonly first_name:string;
  readonly last_name:string = "Anonym";

  get id(): number {
    return this._id;
  }

  set id(value: number) {
    this._id = value
  }

  constructor(first_name:string,name:string) {
    this.first_name = first_name;
    this.last_name = last_name;
  }
}
```

```ts
class Employee extends Person  {
    constructor(id:number,first_name:string,
        last_name:string) {
    super(first_name,last_name);
    this.id=id;
  }
}

let Employee = new Employee(1,"Stanisław","
    Polak");
console.log(Employee.id);       //1
console.log(Employee._id);       //Property '
    _id' is protected and only accessible
    within class 'Person' and its subclasses.
console.log(Employee.first_name);     //Stanis
    ław
console.log(Employee.last_name); //Polak
Employee.first_name = "Jan"; //...Left-hand
    side of assignment expression cannot be a
     constant or a read-only property.
```

script.ts

Notatki

Classes and generic types

```typescript
class CustomCollection<T> {
  private itemArray: Array<T>;

  constructor() {
    this.itemArray = [];
  }

  Add(item: T) {
    this.itemArray.push(item);
  }

  GetFirst(): T {
    return this.itemArray[0];
  }
}
/*********************************/
class User {
  public Name;
}
/*********************************/
class Message {
  public Message;
}
```

```typescript
class MyApp {
  constructor() {
    let myUsers = new CustomCollection<User>();
    let myMessages = new CustomCollection<Message>();

    let user: User = myUsers.GetFirst();        // OK
    let message: Message = myUsers.GetFirst(); // Error because of
          the Generic type validation.
    myUsers.Add(new Message());                 // Error because of
          the Generic type validation.
  }
}
```

script.ts

Source: https://gist.github.com/abergs/5817818

Notatki

Decorators

```typescript
function f(): any {
  console.log("f(): evaluated");
  return function (target, propertyKey: string, descriptor:
        PropertyDescriptor) {
    console.log("f(): called");
    console.log(target);
    console.log();
    console.log(propertyKey);
    console.log();
    console.log(descriptor);
  }
}

function g(value): any {
  console.log("g("+value+"): evaluated");
  return function (target, propertyKey: string, descriptor:
        PropertyDescriptor) {
    console.log("g("+value+"): called");
  }
}

class C {
  @f()
  @g('abc')
  method() {}
}
```

script.ts

Source: https://github.com/Microsoft/TypeScript-Handbook/blob/master/pages/Decorators.md

Notatki

Namespace

Single-file

```
namespace A {
  var a:string = 'abc';
  export class Twix {
    constructor() {
      console.log('Twix');
    }
  }

  export class PeanutButterCup {
    constructor() {
      console.log('PeanutButterCup');
    }
  }

  export class KitKat {
    constructor() {
      console.log('KitKat');
    }
  }
}
let o1 = new A.Twix(); // Twix
let o2 = new A.PeanutButterCup(); // PeanutButterCup
let o3 = new A.KitKat(); // KitKat
console.log(A.a); //...error TS2339: Property 'a'
     does not exist on type 'typeof A'.
```

script.ts

Source: http://stackoverflow.com/questions/30357634/how-do-i-use-namespaces-with-typescript-external-modules

Notatki

Namespace

Multiple-file

```
namespace A {
  export class Twix { ... }
}
```

global1.ts

```
namespace A {
  export class PeanutButterCup { ... }
}
```

global2.ts

```
namespace A {
  export class KitKat { ... }
}
```

global3.ts


```
/// <reference path="global1.ts" />
/// <reference path="global2.ts" />
/// <reference path="global3.ts" />
let o1 = new A.Twix();
let o2 = new A.PeanutButterCup();
let o3 = new A.KitKat();
```

script.ts

Notatki

Modules

Exporting

```ts
export class Twix {
  constructor() {
    console.log('Twix');
  }
}
export {Twix as Raider};
```

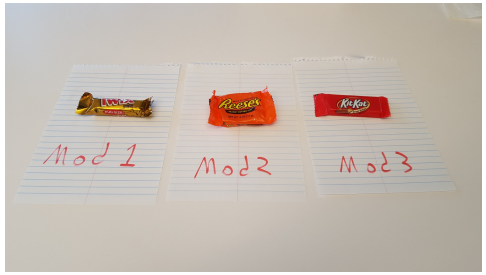
Mod1.ts

```ts
class PeanutButterCup {
  constructor() {
    console.log('PeanutButterCup');
  }
}
export {PeanutButterCup};
```

Mod2.ts

```ts
export  class KitKat {
  constructor() {
    console.log('KitKat');
  }
}
```

Mod3.ts



Source: http://stackoverflow.com/questions/30357634/how-do-i-use-namespaces-with-typescript-external-modules

Notatki

Modules

Importing

```ts
export class Twix {...}
export {Twix as Raider};
```

Mod1.ts

```ts
class PeanutButterCup {...}
export {PeanutButterCup};
```

Mod2.ts

```ts
export class KitKat {...}
```

Mod3.ts

```ts
import {Twix, Raider} from './Mod1';
import {PeanutButterCup} from './Mod2';
import {KitKat} from './Mod3';
import {KitKat as KitKatChunKy} from './Mod3';
import * as  Mars from './Mod1';

let o1 = new Twix(); //  Twix
let o2 = new Raider(); // Twix
let o3 = new PeanutButterCup(); //  PeanutButterCup
let o4 = new KitKat(); //  KitKat
let o5 = new KitKatChunKy(); //  KitKat
let o6 = new Mars.Twix(); //  Twix
let o7 = new Mars.Raider(); //  Twix
```

script.ts

Notatki

Namespaces in modules

```
export namespace A {
  export class Twix { ... }
}
```

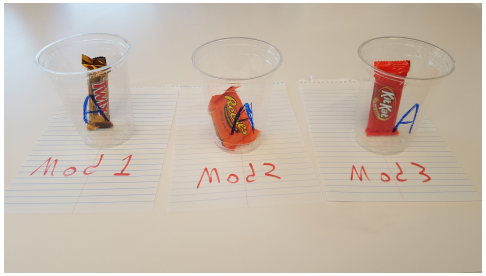
Mod1.ts

```
export namespace A {
  export class PeanutButterCup { ... }
}
```

Mod2.ts

```
export namespace A {
  export class KitKat { ... }
}
```

Mod3.ts



Source: http://stackoverflow.com/questions/30357634/how-do-i-use-namespaces-with-typescript-external-modules

Notatki

Modules

Importing in the "NodeJS" style

```
class Twix {
  constructor() {
    console.log('Twix');
  }
}
export = Twix;
```

Mod.ts

```
import Twix = require('./Mod');
let o = new Twix();
```

script.ts

Notatki

Creating a project

```
{
  "compilerOptions": {
    "module": "commonjs",
    "target": "es5",
    "noImplicitAny": false,
    "sourceMap": false
  },
  "files": [
    "script.ts"
  ]
}
```

tsconfig.json

Notatki

Declaration files

```
import area = require('./area'); //including '
    area.d.ts'
let radius = 2;
console.log(`The area of the circle with
    radius ${radius} is ${area(radius)}`);
```

script.ts

```
function area(radius){
  return Math.PI*Math.pow(radius,2);
}
module.exports = area;
```

area.js

```
declare function area(radius: number) : number
    ;
export = area
```

area.d.ts

```
"use strict";
var area = require("./area");
var radius = 2;
console.log("The area of the circle with
    radius " + radius + " is " + area(radius)
    );
```

script.js

Notatki

Sources I

- Getting Started. URL: https://reactjs.org/docs/getting-started.html.
- Paweł Grzesiak. Ajax w kilka minut. URL: http://internetmaker.pl/artykul/723,1,ajax_w_kilka_minut.html.
- Patrick Hunlock. Functional Javascript. URL: http://www.hunlock.com/blogs/Functional_Javascript.
- MongoDB Inc. The MongoDB Manual. URL: http://docs.mongodb.org/manual/.
- Joyent. Node.js Manual & Documentation. URL: http://nodejs.org/api/.
- Microsoft. TypeScript handbook. URL: http://www.typescriptlang.org/Handbook.
- Mozilla. AJAX. URL: https://developer.mozilla.org/pl/AJAX.
- mozilla.org. JavaScript Guide. URL: https://developer.mozilla.org/en/JavaScript/Guide.
- Basarat Ali Syed. TypeScript Deep Dive. URL: http://basarat.gitbooks.io/typescript/.
- wikibooks.org. JavaScript. URL: http://en.wikibooks.org/wiki/JavaScript.
- Wikipedia. URL: http://pl.wikipedia.org/.

Notatki

Notatki