

# Programming in JavaScript

*dr Michał Kuciapski*  
Higher Banking School

# Overview – Key topics

1. Overview of HTML
2. Overview of CSS
3. Creating a Web Application by Using Visual Studio 2013
4. JavaScript basics
5. Introduction to the Document Object Model
6. Introduction to the Browser Object Model
7. Writing Well-Structured JavaScript Code
8. Object types
9. Creating Custom Objects
10. Extending Objects
11. Introduction to jQuery
12. Sending and Receiving Data by Using the XMLHttpRequest Object
13. Sending and Receiving Data by Using the jQuery Library
14. Validating User Input by Using JavaScript
15. Debugging and Profiling a Web Application

# Overview – Additional topics

1. Reacting to Browser Location and Context
2. Interacting with Files
3. Incorporating Multimedia
4. Reading and Writing Data Locally
5. Adding Offline Support by Using the Application Cache
6. Creating Interactive Graphics by Using SVG
7. Drawing Graphics by Using the Canvas API
8. Introduction to Web Sockets
9. Using the WebSocket API
10. Understanding Web Workers
11. Performing Asynchronous Processing by Using Web Workers

# External sources used for presentation development

Listed in accordance to significance:

- MS20480: Programming in HTML5 with JavaScript and CSS3
- W3Schools - <http://www.w3schools.com>
- Microsoft Developer Network - <https://msdn.microsoft.com/pl-pl/dn308572.aspx>
- jQuery - <https://jquery.com>
- Mozilla Developer Network - <https://developer.mozilla.org/pl/docs/Web/JavaScript>

# Key topics

# Topic 1: Overview of HTML

- The Structure of an HTML Page
- Tags, Elements, Attributes, and Content
- Displaying Text in HTML
- Displaying Images and Linking Documents in HTML
- Gathering User Input by Using Forms in HTML
- Attaching Scripts to an HTML Page

# The Structure of an HTML Page

- All HTML pages have the same structure
  - DOCTYPE declaration
  - HTML section containing:
    - Header
    - Body
- Each version of HTML has its own DOCTYPE
  - The browser uses the DOCTYPE declaration to determine how to interpret the HTML markup
  - For HTML5 pages, specify a DOCTYPE of **html**

# Tags, Elements, Attributes, and Content

- HTML elements define the structure and semantics of content on a web page
- Elements identify their content by surrounding it with a start and an end tag
- Elements can be nested:

```
<p>  
  <strong>Elements</strong> consist of  
  <strong>content</strong> bookended by a  
  <em>start</em> tag and an <em>end</em> tag.  
</p>
```

- Use attributes to provide additional information about the content of an element



# Displaying Text in HTML

Text in HTML can be marked up:

- As headings and paragraphs

```
<h1>An Introduction to HTML</h1>  
<p>In this module, we look at the history of HTML and CSS.</p>  
<h2>In the Beginning</h2>  
<p>WorldWideWeb was created by Sir Tim Berners-Lee at CERN. </p>
```

- With emphasis

To `<strong>emphasize</strong>` is to give extra weight to (a communication); `<em>"Her gesture emphasized her words"</em>`

- In lists

```
<ul>  
  <li>Notepad</li>  
  <li>Textmate</li>  
  <li>Visual Studio</li>  
</ul>
```

# Displaying Images and Linking Documents in HTML

- Use the `<img>` tag to display an image
  - The `src` attribute specifies the URL of the image source:

```

```

- Use the `<a>` tag to define a link
  - The `href` attribute specifies the target of the link:

```
<a href="default.html" alt="Home Page">Home</a>
```

# Gathering User Input by Using Forms in HTML

- The <form> element provides a mechanism for obtaining user input
  - The action attribute specifies where the data will be sent
  - The method attribute specifies how the data will be sent
  - Many different input types are available

First name:

Last name:

Email address:

Choose a password:

Confirm your password:

Website/blog:

# Attaching Scripts to an HTML Page

- HTML is static, but pages can use JavaScript to add dynamic behavior
- Use the `<script>` element to specify the location of the JavaScript code:

```
<script type="text/javascript" src="alertme.js"></script>
```

- The order of `<script>` elements is important
- Make sure objects and functions are in scope before they are used
- Use the `<noscript>` element to alert users with browsers that have scripting disabled.

## Topic 2: Overview of CSS

- Overview of CSS Syntax
- How CSS Selectors Work
- How HTML Inheritance and Cascading Styles Affect Styling
- Adding Styles to An HTML Page

# Overview of CSS Syntax

- All CSS rules have the same syntax:

```
selector {  
  property1:value;  
  property2:value;  
  ..  
  propertyN:value;  
}
```

- Comments are enclosed in `/* ... */` delimiters

```
/* Targets level 1 headings */  
h1 {  
  font-size: 42px;  
  color: pink;  
  font-family: 'Segoe UI';  
}
```

# How CSS Selectors Work

- There are three basic CSS selectors
  - The element selector: `h2{}`
  - The class selector: `.myClass {}`
  - The id selector: `#thisId {}`
- CSS selectors can be combined to create more specific rules
- The wildcard `*` selector returns the set of all elements
- Use `[...]` to refine selectors based on attribute values

# How HTML Inheritance and Cascading Styles Affect Styling

- HTML inheritance and the CSS cascade mechanism govern how browsers apply style rules
- HTML inheritance determines which style properties an element inherits from its parent
- The cascade mechanism determines how style properties are applied when conflicting rules apply to the same element



# Adding Styles to An HTML Page

- Use an element's style attribute to define styles specific to that element:

```
<p style="color:blue;">  
some text </p>
```

- Use the <style> element in the <head> to include styles specific to a page:

```
<style type="text/css">  
  p { color: blue; }  
</style>
```

- Use the <link> element to reference an external style sheet:

```
<link rel="stylesheet" type="text/css" href="mystyles.css" media="screen">
```

## Topic 3: Creating a Web Application by Using Visual Studio 2013

- Developing Web Applications by Using Visual Studio 2013
- Using the Internet Explorer F12 Developer Tools

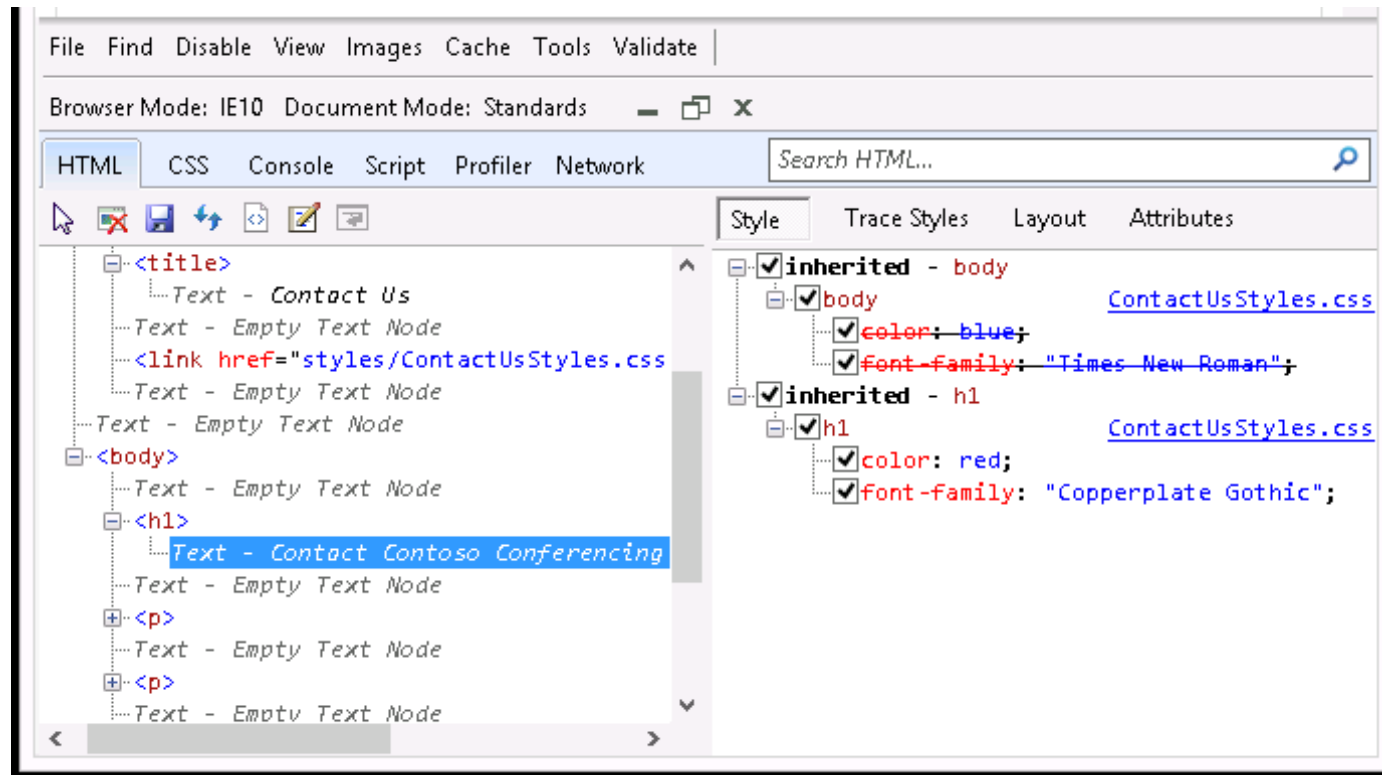
# Developing Web Applications by Using Visual Studio 2013

- Visual Studio 2013 provides tools for:
  - Creating a web application project, and adding folders to structure the content
  - Debugging JavaScript code, examining and modifying variables, and viewing the call stack
  - Deploying a web application to a web server or to the cloud
- Visual Studio 2013 features include:
  - Full support for HTML5
  - IntelliSense for JavaScript code
  - Support for CSS3 properties and values
  - CSS color picker

# Using the Internet Explorer F12 Developer Tools

The F12 Developer Tools enables developers to:

- Inspect and validate HTML and CSS
- Run and debug JavaScript code
- Profile page load times
- View a page as it were in any version of Internet Explorer from v7.0 onwards



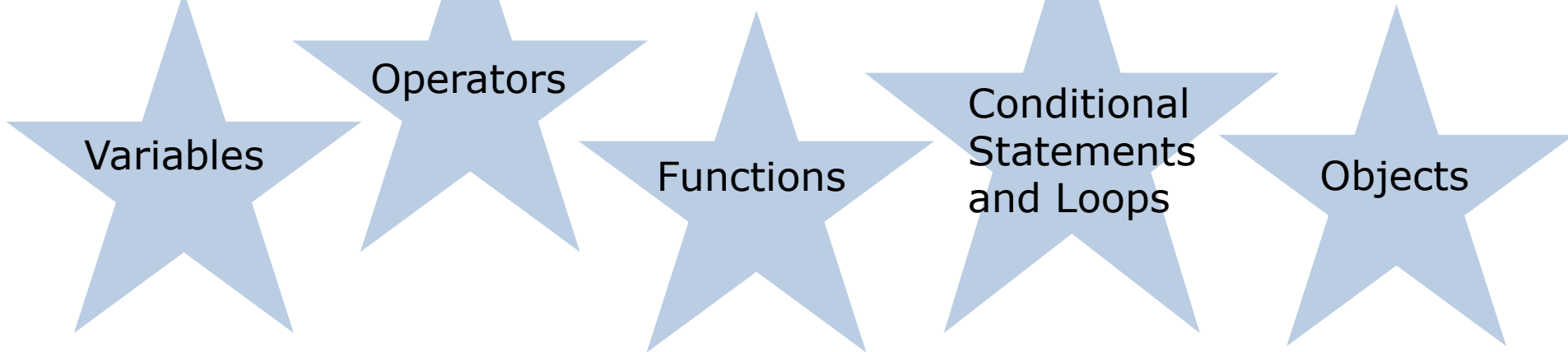
- **Zadanie 1** – *Utworzenie strony*

# Topic 4: JavaScript basics

- What is JavaScript?
- JavaScript Syntax
- Variables and data Types
- Operators
- Constants
- Comments
- Conditional Statements
- Looping Statements
- Functions
- String methods
- Numbers methods
- Dates methods

# What is JavaScript?

- JavaScript is a programming language that supports:



- Use JavaScript with the Document Object Model and Browser Object Model to make web pages dynamic.
- Use the AJAX API to make asynchronous requests to a web server.

# JavaScript Syntax

- A JavaScript statement represents a line of code to be run
- Terminate statements with a semicolon

```
var thisVariable = 3;  
counter = counter + 1;  
GoDoThisThing();  
document.write("An incredibly really \  
very long greeting to the world");
```

- Use comments to add notes to your scripts

```
document.write("I'm learning JavaScript"); // display a message
```

```
/* You can use a multi-line comment  
to add more information */
```



# Variables and data Types

- Use **var** to declare variables

```
var answer = 3;  
var actuallyAsString = "42";
```

- JavaScript has three simple types

- String, Number, and Boolean
- Variables can also be undefined or null

```
var noValue; // noValue has the value undefined  
var nullValue = null; // null is different to undefined
```

- JavaScript Has Dynamic Types

```
var x = 5;           // Now x is a Number  
var x = "John";      // Now x is a String
```

- Use JavaScript **typeof** operator to find the type of a JavaScript variable

# Operators

- JavaScript supports many operators: Arithmetic, assignment, comparison, Boolean, conditional, and string

## Arithmetic

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

## Assignment

Operator	Example	Same As
=	<code>x = y</code>	<code>x = y</code>
+=	<code>x += y</code>	<code>x = x + y</code>
-=	<code>x -= y</code>	<code>x = x - y</code>
*=	<code>x *= y</code>	<code>x = x * y</code>
/=	<code>x /= y</code>	<code>x = x / y</code>
%=	<code>x %= y</code>	<code>x = x % y</code>

# Operators

## Comparison and Logical

Operator	Description
==	equal to
===	equal value and equal type
!=	not equal
!==	not equal value or not equal type
>	greater than
<	less than
>=	greater than or equal to
<=	less than or equal to

## Arithmetic Operator Precedence

Operator	Precedence
( )	Expression grouping
++ --	Increment and decrement
* / %	Multiplication, division, and modulo division
+ -	Addition and subtraction

- **Zadanie 2** – *Zmienne, operatory*

# Comments

- JavaScript comments can be used to explain JavaScript code, and to make it more readable.
- **Syntax** example

```
const tax = 23;  
tax = 22; //single line comment
```

```
/* const tax = 23;  
   tax = 22;  
   multi-line comment  
*/
```

# Constants

- The **const declaration** creates a read-only named constant:
  - constant can be global or local to the function
  - the value of a constant cannot change through re-assignment, and a constant cannot be re-declared.
  - an initializer for a constant is required.
- **Syntax** example

```
const tax = 23;  
tax = 22; //no change
```

- **Zadanie 3** – *State*

# Conditional Statements

- JavaScript provides two conditional constructs

- if: 

```
if (time < 10) {  
    greeting = "Good morning";  
} else if (time < 20) {  
    greeting = "Good day";  
} else {  
    greeting = "Good evening";  
}
```

- switch: 

```
var RoomRate;  
switch (typeofRoom) {  
    case "Suite":  
        RoomRate = 500;  
        break;  
    case "King":  
        RoomRate = 400;  
        break;  
    default:  
        RoomRate = 300;}
```

**switch statements  
use strict comparison**



- **Zadanie 4** – *Wyrażenia warunkowe, komentarze*

# Looping Statements

- JavaScript provides three loop constructs

- while:

```
while (GuestIsStillCheckedIn())  
{  
    numberOfNightsStay += 1;  
}
```

- do while:

```
do {  
    eatARoundOfToast();  
} while (StillHungry())
```

- for:

```
for (var i=0; i<10; i++) {  
    plumpUpAPillow();  
}
```

- **Zadanie 5** – *pętle*

- **Zadanie 6** – *pętle i instrukcje warunkowe*
- **Zadanie 7 [opcjonalne]** – *pętle i instrukcje warunkowe*
- **Zadanie 8 [opcjonalne]** – *pętle i instrukcje warunkowe*

# Functions

- Functions are named blocks of reusable code:

```
function aName( argument1, argument2, ..., argumentN ) {  
    statement1; ... statementN;  
}
```

```
var x = myFunction(3, 5);  
function myFunction(a, b) {  
    return a * b; // Function returns the product of a and b  
}
```

- Arguments are only accessible inside the function
- A function can return a value
- A function can also declare local variables
- Global variables defined outside of a function are available to all functions in scripts referenced by a page
- Self-Invoking Functions – started automatically

```
(function () { var x = "Hello!!";    // I will invoke myself  
})();
```

# Exercises

- **Zadanie 9** – *funkcje*
- **Zadanie 10** – *funkcje, organizacja kodu*
- **Zadanie 11 [opcjonalne]** - *funkcje*

# String methods

- String Length

```
var txt = "Michał Kuciapski";  
var sln = txt.length;
```

- Special Characters

Code	Outputs
\'	single quote
\"	double quote
\\	backslash
\n	new line
\r	carriage return
\t	tab
\b	backspace
\f	form feed

# String methods

Method	Description
<code>charAt()</code>	Returns the character at the specified index (position)
<code>charCodeAt()</code>	Returns the Unicode of the character at the specified index
<code>concat()</code>	Joins two or more strings, and returns a copy of the joined strings
<code>fromCharCode()</code>	Converts Unicode values to characters
<code>indexOf()</code>	Returns the position of the first found occurrence of a specified value in a string
<code>lastIndexOf()</code>	Returns the position of the last found occurrence of a specified value in a string
<code>localeCompare()</code>	Compares two strings in the current locale
<code>match()</code>	Searches a string for a match against a regular expression, and returns the matches
<code>replace()</code>	Searches a string for a value and returns a new string with the value replaced
<code>search()</code>	Searches a string for a value and returns the position of the match
<code>slice()</code>	Extracts a part of a string and returns a new string
<code>split()</code>	Splits a string into an array of substrings
<code>substr()</code>	Extracts a part of a string from a start position through a number of characters
<code>substring()</code>	Extracts a part of a string between two specified positions
<code>toLocaleLowerCase()</code>	Converts a string to lowercase letters, according to the host's locale
<code>toLocaleUpperCase()</code>	Converts a string to uppercase letters, according to the host's locale
<code>toLowerCase()</code>	Converts a string to lowercase letters
<code>toString()</code>	Returns the value of a String object
<code>toUpperCase()</code>	Converts a string to uppercase letters
<code>trim()</code>	Removes whitespace from both ends of a string
<code>valueOf()</code>	Returns the primitive value of a String object



- **Zadanie 12** – *Przetwarzanie tekstu*

# Numbers methods

- Infinity (or -Infinity) calculate a number outside the largest possible number.

```
var myNumber = 2;  
while (myNumber !== Infinity) {           // Execute until Infinity  
    myNumber = myNumber * myNumber;  
}
```

- NaN - indicating that a value is not a number

```
var x = 100 / "Apple";  
isNaN(x);           // returns true because x is Not a  
Number
```

- Number properties

Property	Description
MAX_VALUE	Returns the largest number possible in JavaScript
MIN_VALUE	Returns the smallest number possible in JavaScript
NEGATIVE_INFINITY	Represents negative infinity (returned on overflow)
NaN	Represents a "Not-a-Number" value
POSITIVE_INFINITY	Represents infinity (returned on overflow)

# Numbers methods

Method	Description
Number()	Returns a number, converted from its argument.
parseFloat()	Parses its argument and returns a floating point number
parseInt()	Parses its argument and returns an integer

Method	Description
toString()	Returns a number as a string
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length
valueOf()	Returns a number as a number

# Numbers methods – Math class

Method	Description
<code>abs(x)</code>	Returns the absolute value of x
<code>acos(x)</code>	Returns the arccosine of x, in radians
<code>asin(x)</code>	Returns the arcsine of x, in radians
<code>atan(x)</code>	Returns the arctangent of x as a numeric value between -PI/2 and PI/2 radians
<code>atan2(y,x)</code>	Returns the arctangent of the quotient of its arguments
<code>ceil(x)</code>	Returns x, rounded upwards to the nearest integer
<code>cos(x)</code>	Returns the cosine of x (x is in radians)
<code>exp(x)</code>	Returns the value of $E^x$
<code>floor(x)</code>	Returns x, rounded downwards to the nearest integer
<code>log(x)</code>	Returns the natural logarithm (base E) of x
<code>max(x,y,z,...,n)</code>	Returns the number with the highest value
<code>min(x,y,z,...,n)</code>	Returns the number with the lowest value
<code>pow(x,y)</code>	Returns the value of x to the power of y
<code>random()</code>	Returns a random number between 0 and 1
<code>round(x)</code>	Rounds x to the nearest integer
<code>sin(x)</code>	Returns the sine of x (x is in radians)
<code>sqrt(x)</code>	Returns the square root of x
<code>tan(x)</code>	Returns the tangent of an angle

- **Zadanie 13** – *Operowanie na liczbach*

# Dates methods

- Getting date

```
new Date()  
new Date(year, month, day, hours, minutes, seconds, milliseconds)  
var d = new Date(string);
```

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday as a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

# Dates methods

- Set date

Method	Description
<code>setDate()</code>	Set the day as a number (1-31)
<code>setFullYear()</code>	Set the year (optionally month and day)
<code>setHours()</code>	Set the hour (0-23)
<code>setMilliseconds()</code>	Set the milliseconds (0-999)
<code>setMinutes()</code>	Set the minutes (0-59)
<code>setMonth()</code>	Set the month (0-11)
<code>setSeconds()</code>	Set the seconds (0-59)
<code>setTime()</code>	Set the time (milliseconds since January 1, 1970)

- **Zadanie 14** – *Operowanie na datach*

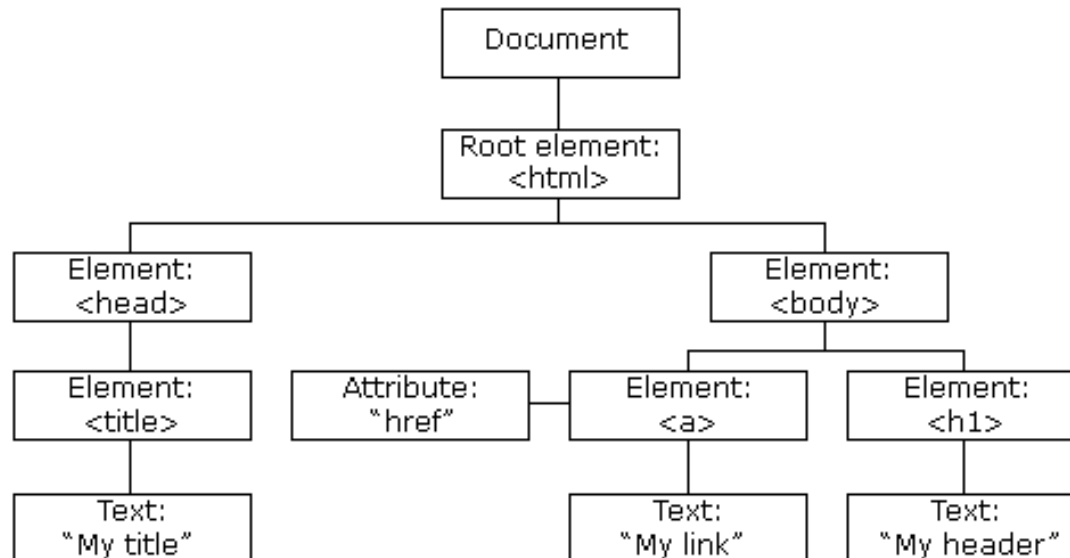


# Topic 5: Introduction to the Document Object Model

- The Document Object Model
- Finding Elements in the DOM
- Adding, Removing, and Manipulating Objects in the DOM
- Handling Events in the DOM
- Common HTML Events

# The Document Object Model

- The DOM provides a programmatic API for controlling a browser and accessing the contents of a web page:
  - Finding and setting the values of elements on a page
  - Handling events for controls on a page
  - Modifying the styles associated with elements
  - Serializing and deserializing a page as an XML document
  - Validating and updating web pages



# Finding Elements in the DOM

- Given the following form:

```
<form name="contactForm">  
  <input type="text" name="nameBox" id="nameBoxId" />  
</form>
```

- You can reference the form by using:

```
document.forms[0] // forms is a zero-based array  
document.forms["contactForm"]  
document.forms.contactForm  
document.contactForm
```

- You can reference the **nameBox** text box by using:

```
document.forms.contactForm.elements[0]  
document.forms.contactForm.elements["nameBox"]  
document.forms.contactForm.nameBox  
document.contactForm.nameBox  
document.getElementById("nameBoxId")  
document.getElementById().getElementsByName("nameBox")  
document.contactForm.childNodes[0]
```

# Finding Elements in the DOM

Property	Description	DOM
document.anchors	Returns all <a> elements that have a name attribute	1
document.applets	Returns all <applet> elements (Deprecated in HTML5)	1
document.baseURI	Returns the absolute base URI of the document	3
document.body	Returns the <body> element	1
document.cookie	Returns the document's cookie	1
document.doctype	Returns the document's doctype	3
document.documentElement	Returns the <html> element	3
document.documentMode	Returns the mode used by the browser	3
document.documentURI	Returns the URI of the document	3
document.domain	Returns the domain name of the document server	1
document.domConfig	Obsolete. Returns the DOM configuration	3
document.embeds	Returns all <embed> elements	3
document.forms	Returns all <form> elements	1
document.head	Returns the <head> element	3
document.images	Returns all <img> elements	1
document.implementation	Returns the DOM implementation	3
document.inputEncoding	Returns the document's encoding (character set)	3
document.lastModified	Returns the date and time the document was updated	3
document.links	Returns all <area> and <a> elements that have a href attribute	1
document.readyState	Returns the (loading) status of the document	3
document.referrer	Returns the URI of the referrer (the linking document)	1
document.scripts	Returns all <script> elements	3
document.strictErrorChecking	Returns if error checking is enforced	3
document.title	Returns the <title> element	1
document.URL	Returns the complete URL of the document	1

# Finding Elements in the DOM

- Finding HTML Elements by CSS Selectors:
  - id, class names, types, attributes, values of attributes, etc

```
var x = document.querySelectorAll("p.intro");
```

- Finding HTML Elements by DOM Nodes:

```
parentNode  
childNodes[nodenumber]  
firstChild  
lastChild  
nextSibling  
previousSibling
```

```
myText =  
document.getElementById("intro").firstChild.nodeValue;  
document.getElementById("demo").innerHTML = myText;
```

# Adding, Removing, and Manipulating Objects in the DOM

To modify an element on a page:

1. Create a new object containing the new data.
2. Find the parent element that should contain the new data.
3. Append, insert, or replace the data in the element with the new data.

To remove an element or attribute:

1. Find the parent element.
2. Use **removeChild** or **removeAttribute** to remove the data.

# Adding, Removing, and Manipulating Objects in the DOM

- Changing HTML Elements

<code>element.innerHTML=</code>	Change the inner HTML of an element
<code>element.attribute=</code>	Change the attribute of an HTML element
<code>element.setAttribute(attribute,value)</code>	Change the attribute of an HTML element
<code>element.style.property=</code>	Change the style of an HTML element

- Adding and Deleting Elements

<code>document.createElement()</code>	Create an HTML element
<code>document.removeChild()</code>	Remove an HTML element
<code>document.appendChild()</code>	Add an HTML element
<code>document.replaceChild()</code>	Replace an HTML element
<code>document.write(text)</code>	Write into the HTML output stream

```
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);  
var child = document.getElementById("p1");  
element.insertBefore(para,child);
```

# Handling Events in the DOM

- The DOM defines events that can be triggered by the browser or by the user
- HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.

```
<button onclick='getElementById("demo").innerHTML=Date()>Get time</button>
```

- Many HTML elements define callbacks that run when an event occurs:

```
var helpIcon = document.getElementById("helpIcon");  
document.images.helpIcon.onmouseover =  
    function() { window.alert('Some help text'); };
```

- You can also define event listeners that run when an event fires - this is useful if the same event needs to trigger multiple actions

```
helpIcon.addEventListener("mouseover",  
    function() { window.alert('Some help text'); }, false);
```

- To remove an event listener:

```
helpIcon.removeEventListener("mouseover", ShowHelpText, false);
```



# Common HTML Events

<b>Event</b>	<b>Description</b>
onchange	An HTML element has been changed
onclick	The user clicks an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user pushes a keyboard key
onload	The browser has finished loading the page

- **Zadanie 15** – *DOM*

# Topic 6: Introduction to the Browser Object Model

- Window Object
- Window Screen
- Window Location
- Window History
- Window Navigator
- Popup Boxes
- Cookies
- Timing Events

# Window Object

- The window object is supported by all browsers. It represents the browser's window.
- All global JavaScript objects, functions, and variables automatically become members of the window object.
- Window Size

```
var w = window.innerWidth  
|| document.documentElement.clientWidth  
|| document.body.clientWidth;
```

```
var h = window.innerHeight  
|| document.documentElement.clientHeight  
|| document.body.clientHeight;
```

- Methods  
window.open() - open a new window  
window.close() - close the current window  
window.moveTo() - move the current window  
window.resizeTo() - resize the current window

# Window Screen

- The window.screen object contains information about the user's screen.
- Methods:
  - screen.width
  - screen.height
  - screen.availWidth
  - screen.availHeight
  - screen.colorDepth
  - screen.pixelDepth

```
document.getElementById("demo").innerHTML =  
    "Available Screen Width: " + screen. availWidth;
```

- **Zadanie 16** – *BOM (okno, zdarzenia)*

# Window Location

- The `window.location` object can be used to get the current page address (URL) and to redirect the browser to a new page.
- Methods:
  - `window.location.href` returns - URL of the current page
  - `window.location.hostname` - domain name of the web host
  - `window.location.pathname` - path and filename of the current page
  - `window.location.protocol` - web protocol used (`http://` or `https://`)
  - `window.location.assign` - loads a new document

```
document.getElementById("demo").innerHTML =  
    "Page hostname is " + window.location.hostname;
```

# Window History

- The window.history object contains the browsers history.
- Methods and properties:
  - history.back() - same as clicking back in the browser
  - history.forward() - same as clicking forward in the browser
  - go() - loads a specific URL from the history list
  - length - Returns the number of URLs in the history list

```
function goBack() {  
    window.history.go(-2);  
}
```



# Window Navigator

- The window.navigator object contains browser information.
- Properties and methods:
  - navigator.appName
  - navigator.appCodeName
  - navigator.platform
  - navigator.appVersion
  - navigator.userAgent
  - navigator.cookieEnabled
  - navigator.language
  - navigator.javaEnabled()

```
document.getElementById("demo").innerHTML =  
    "Cookies Enabled is " + navigator.cookieEnabled;
```

- **Zadanie 17** – *BOM (dane użytkownika)*

# Window Popup Boxes

- JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

```
alert("I am an alert box!");
```

```
var r = confirm("Press a button");  
if (r == true) {  
    x = "You pressed OK!";  
} else {  
    x = "You pressed Cancel!";  
}
```

```
var person = prompt("Please enter your name", "Harry Potter");  
if (person != null) {  
    document.getElementById("demo").innerHTML =  
        "Hello " + person + "! How are you today?";  
}
```

# Cookies

- Cookies are data, stored in small text files, on your computer.
- When a web server has sent a web page to a browser, the connection is shut down, and the server forgets everything about the user. Cookies were invented to solve the problem "how to remember information about the user".
- Create Cookie

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

- Read a Cookie

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

- **Zadanie 18** – *BOM (ciasteczka)*

# Timing Events

- With JavaScript, it is possible to execute some code at specified time-intervals. This is called timing events.
- Methods:
  - setInterval() - executes a function, over and over again, at specified time intervals
  - clearInterval() - is used to stop further executions of the function specified in the setInterval() method.
  - setTimeout() - executes a function, once, after waiting a specified number of milliseconds

```
setInterval(function(){  
    document.getElementById("image").src = "landscape.jpg";  
    document.getElementById("image").setAttribute("Style","border: solid  
    10px");  
}, 3000);
```

- **Zadanie 19** – *BOM (zadania w tle)*

# Topic 7: Writing Well-Structured JavaScript Code

- Scoping and Hoisting
- Managing the Global Namespace
- Singleton Objects and Global Functions in JavaScript
- Style Guide and Coding Conventions



# Scoping and Hoisting

- JavaScript variables have one of two scopes:
  - Global scope
  - Local scope within a function
- JavaScript does not support block scope
  - If you declare a variable inside a block, it is hoisted to function scope

```
var num = 7;

function demonstrateScopingAndHoisting() {
  if (true) {
    var num = 42;
  }
  alert("The value of num is " + num);    // Displays 42, not 7.
}
```

# Managing the Global Namespace

- Global name clashes can be problematic in JavaScript
  - Your global variables might conflict with other global variables elsewhere in the web application
- JavaScript provides several mechanisms to avoid global name clashes
  - Immediate functions
  - Namespaces
  - Strict mode - "use strict"; can also be used inside method (supported in: Internet Explorer from version 10. Firefox from version 4, Chrome from version 13, Safari from version 5.1, Opera from version 12)

# Singleton Objects and Global Functions in JavaScript

- JavaScript defines several singleton objects, such as:
  - **Math**
  - **JSON**
- JavaScript also defines global functions, such as:
  - **parseInt()**
  - **parseFloat()**
  - **isNaN()**

# Style Guide and Coding Conventions

- Variable Names:
  - Use camelCase for identifier names (variables and functions).
  - All names start with a letter.
  - Initialize Variables
  - Global variable written in UPPERCASE
  - Constants written in UPPERCASE
- Spaces Around Operators
- Code Indentation
- Object Rules
  - Place the opening bracket on the same line as the object name.
  - Use colon plus one space between each property and its value.
  - Short objects can be written compressed, on one line, using spaces only between properties.
  - Don't Use new Object()
    - Treat numbers, strings, or booleans as primitive values. Not as objects.
- Declarations on Top

- **Zadanie 20** – *Dobre praktyki pisanania kodu*

## Topic 8: Object types

- Using Object Types
- Arrays
- Defining Arrays of Objects by Using JSON
- Regular Expressions

# Using Object Types

- JavaScript has a number of built-in object types:
  - String, Date, Array, RegExp

```
var seasonsArray = ["Spring", "Summer", "Autumn", "Winter"];  
...  
var autumnLocation = seasonsArray.indexOf("Autumn");
```

```
var re = new RegExp("[dh]og");  
if (re.test("dog")) {...}
```

- JavaScript also provides singleton types providing useful functionality:
  - Math, Global

# Arrays

- Declaration

```
var seasonsArray = ["Spring", "Summer", "Autumn", "Winter"];  
var fruits = ["Banana", "Orange", "Apple", "Mango"];  
var cars = new Array("Saab", "Volvo", "BMW");
```

- Access

```
var name = cars[0];  
cars[0] = "Opel";  
fruits.length;           // the length of fruits is 4  
fruits.push("Lemon");    // adds a new element (Lemon) to fruits  
fruits.pop();           // Removes the last element ("Mango") from fruits  
fruits.shift();          // Removes the first element ("Banana") from fruits  
fruits.unshift("Lemon"); // Adds a new first element to fruits  
delete fruits[0];         // Changes the first element in fruits to undefined  
fruits.splice(2, 0, "Lemon", "Kiwi"); // add new items to an array  
fruits[10] = "Lemon";     // adds a new element (Lemon) to fruits
```



# Arrays

- Methods

```
var autumnLocation = seasonsArray.indexOf("Autumn");
```

```
var y = cars.sort(); // Sorts cars in alphabetical order  
fruits.reverse();    // reverses the order of the elements
```

```
fruits.valueOf(); // returns an array as a string  
fruits.join(" * "); // joins elements into a string, specify the separator  
var myChildren = myGirls.concat(myBoys); // concatenates arrays
```

```
var citrus = fruits.slice(1, 3); // slices an array into a new array
```

# Excercises

- **Zadanie 21** – *Tablice*
- **Zadanie 22 [opcjonalne]** – *Tablice*
- **Zadanie 23 [opcjonalne]** - *Tablice*
- **Zadanie 24 [opcjonalne]** – *Tablice*
- **Zadanie 25 [opcjonalne]** – *Tablice*
- **Zadanie 25 [opcjonalne]** - *Tablice*

# Defining Arrays of Objects by Using JSON

- JSON is a format for serializing objects:

```
var attendees = [  
  {  
    "name": "Eric Gruber",  
    "currentTrack": "1"  
  },  
  {  
    "name": "Martin Weber",  
    "currentTrack": "2"  
  }  
]
```

- JavaScript provides APIs for serializing and parsing JSON data

## **Zadanie 26** – *JSON, tablice, zdarzenia przeglądarka*

# Regular Expressions

- A regular expression is a sequence of characters that forms a search pattern.
- The search pattern can be used for text search and text replace operations.

```
var str = "Visit Gdansk";  
var n = str.search(/gdansk/i); //returns 6
```

```
var str = "Visit Gdansk!";  
var res = str.replace("Gdansk", "Gdynia");
```

- RegExp object is a regular expression object with predefined properties and methods.

```
/e/.test("The best things in life are free!"); //true  
/e/.exec("The best things in life are free!"); //e
```

# Regular Expressions

- **Modifiers**

<u>i</u>	Perform case-insensitive matching
<u>g</u>	Perform a global match (find all matches rather than stopping after the first match)
<u>m</u>	Perform multiline matching

- **Brackets**

<u>[abc]</u>	Find any character between the brackets
<u>[^abc]</u>	Find any character NOT between the brackets
<u>[0-9]</u>	Find any digit between the brackets
<u>[^0-9]</u>	Find any digit NOT between the brackets
<u>(x y)</u>	Find any of the alternatives specified

# Regular Expressions

- Metacharacters (main)

<u>.</u>	Find a single character, except newline or line terminator
<u>\w</u>	Find a word character
<u>\W</u>	Find a non-word character
<u>\d</u>	Find a digit
<u>\D</u>	Find a non-digit character
<u>\s</u>	Find a whitespace character
<u>\S</u>	Find a non-whitespace character

- Quantifiers (main)

<u>n+</u>	Matches any string that contains at least one <i>n</i>
<u>n*</u>	Matches any string that contains zero or more occurrences of <i>n</i>
<u>n?</u>	Matches any string that contains zero or one occurrences of <i>n</i>
<u>n{X}</u>	Matches any string that contains a sequence of <i>X</i> <i>n</i> 's
<u>n{X,Y}</u>	Matches any string that contains a sequence of <i>X</i> to <i>Y</i> <i>n</i> 's
<u>n{X,}</u>	Matches any string that contains a sequence of at least <i>X</i> <i>n</i> 's
<u>n\$</u>	Matches any string with <i>n</i> at the end of it
<u>^n</u>	Matches any string with <i>n</i> at the beginning of it

## **Zadanie 27** – *Wyrażenia regularne*



# Topic 9: Creating Custom Objects

- Creating Simple Objects
- Visibility
- Using Object Literal Notation
- Using Constructors
- Using Prototypes
- Using the Object.create Method
- JSON

# Creating Simple Objects

- There are several ways to create new objects in JavaScript:

```
var employee1 = new Object();
```

```
var employee2 = {};
```

- You can define properties and methods on an object:

```
var employee1 = {};  
employee1.name = "John Smith";  
employee1.age = 21;  
employee1.salary = 10000;  
  
employee1.payRise = function(amount) {  
    // Inside a method, "this" means the current object.  
    this.salary += amount;  
    return this.salary;  
}  
delete employee1.age; // delete property;
```

# Visibility

- Local JavaScript Variables:

- Variables declared within a JavaScript function, become LOCAL to the function.
- Local variables are deleted when the function is completed.

```
// code here can not use carName
function myFunction() {
    var carName = "Volvo";
    // code here can use carName}
```

- Global JavaScript Variables:

- A variable declared outside a function, becomes GLOBAL.
- Global variables are deleted when you close the page.

```
var carName = " Volvo";
// code here can use carName

function myFunction() {
    // code here can use carName
}
```

# Using Object Literal Notation

- Object literal notation provides a shorthand way to create new objects and assign properties and methods:

```
var employee2 = {  
  name: "Mary Jones",  
  age: 42,  
  salary: 20000,  
  
  payRise: function(amount) {  
    this.salary += amount;  
    return this.salary;  
  },  
  
  displayDetails: function() {  
    alert(this.name + " is " + this.age + " and earns " + this.salary);  
  }  
};
```

# Using Constructors

- Constructor functions define the shape of objects
  - They create and assign properties for the target object
  - The target object is referenced by the **this** keyword

```
var Account = function (id, name) {  
  this.id = id;  
  this.name = name;  
  this.balance = 0;  
  this.numTransactions = 0;  
};
```

- Use the constructor function to create new objects with the specified properties:

```
var acc1 = new Account(1, "John");  
var acc2 = new Account(2, "Mary");
```

# Using Prototypes

- All objects created by using a constructor function have their own copy of the properties defined by the constructor
  - All JavaScript objects, including constructors, have a special property named **prototype**
  - Use the prototype to share function definitions between objects:

```
Account.prototype.deposit = function(amount){  
    this.balance += amount;  
    this.numTransactions++;  
};
```

```
Account.prototype = {  
    deposit: function(amount) {  
        this.balance += amount;  
        this.numTransactions++;  
    },  
    // Plus other methods...  
};
```

# Using the Object.create Method

- Use **Object.create()** to create an object based on existing prototype
  - Pass in a prototype object
  - Optionally pass in a properties object that specifies additional properties to add to the new object

```
var obj1 = Object.create(prototypeObject, propertiesObject);
```

- The new object has access to all the properties defined in the specified prototype
  - It forms the basis of the approach used by many JavaScript developers to implement inheritance.

# JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation and is a format for storing and transporting data

```
{  
  "employees": [  
    {"firstName": "John", "lastName": "Doe"},  
    {"firstName": "Anna", "lastName": "Smith"},  
    {"firstName": "Peter", "lastName": "Jones"}  
  ]  
}
```

- Converting a JSON Text to a JavaScript Object

```
var obj = JSON.parse(text);
```

- converting a JavaScript value to a JSON string

```
var obj = JSON.stringify(obj);
```



**Zadanie 28** – *Definiowanie obiektów*

**Zadanie 29** – *Oprogramowanie obiektów*

# Topic 10: Extending Objects

- Implementing Encapsulation
- Implementing Inheritance by Chaining Prototypes
- Adding Functionality to Existing Objects

# Implementing Encapsulation

- To define private members for an object, declare variables in the constructor and omit the **this** keyword
- To define public accessor functions for an object, declare methods in the constructor and include the **this** keyword

```
var Person = function(name, age)
{
    // Private properties.
    var _name, _age;

    // Public accessor functions.
    this.getName = function()
    {
        return _name;
    }
    ...
}
```

# Implementing Inheritance by Chaining Prototypes

```
var Person = function(name, age) {// Base constructor
  this.name = name;
  this.age = age;
}
```

```
Person.prototype = {// Base prototype.
  haveBirthday: function() {
    this.age++;
  }
};
```

```
var Student = function(name, age, subject) {// Derived constructor
  this.name = name;
  this.age = age;
  this.subject = subject;
}
```

```
// Inheritance
```

```
Student.prototype = new Person();
Student.prototype.constructor = Student;
```

```
var aStudent = new Student("Jim", 20, "Physics");
aStudent.subject = "BioChemistry";
aStudent.haveBirthday();
```

# Adding Functionality to Existing Objects

- Get the prototype for an object
- Assign a new property to the object

```
var Point = function(x, y) {  
    this.x = x;  
    this.y = y;  
}
```

```
Point.prototype.moveBy = function(deltaX, deltaY) { ... }  
Point.prototype.moveTo = function(otherPoint) { ... }
```

```
var p1 = new Point(100, 200);  
p1.moveBy(10, 20);  
p1.moveTo(anotherPoint);
```

- Use the **apply** method to resolve references to **this** in generic functions

## **Zadanie 30** – *Rozszerzanie obiektów*

# Topic 11: Introduction to jQuery

- The jQuery Library
- Selecting Elements and Traversing the DOM by Using jQuery
- Adding, Removing, and Modifying Elements by Using jQuery
- Handling Control Events by Using jQuery
- Animation
- Chaining

# The jQuery Library

- jQuery provides portability for JavaScript code, enabling you to easily build cross-browser web applications:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <title>jQuery Example</title>
  <script type="text/javascript" src="Scripts/jquery-1.8.0.min.js"></script>
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js">
  </script>
</head>
<body>
  ...
  <script type="text/javascript">
    $(document).ready(function () {
      // some code
    });
  </script>
</body>
</html>
```



# Selecting Elements and Traversing the DOM by Using jQuery

- jQuery uses the same selector syntax as CSS

```
<script type="text/javascript">
    $(document).ready(function () {
        $("h2").each(function () {
            this.style.color = "red";
        });
    });
</script>
```

- jQuery provides additional functions for traversing and filtering elements

# Selecting Elements and Traversing the DOM by Using jQuery

- jQuery uses the same selector syntax as CSS

Selector	Example	Example description	CSS
<a href="#"><u>.class</u></a>	.intro	Selects all elements with class="intro"	1
<a href="#"><u>#id</u></a>	#firstname	Selects the element with id="firstname"	1
<a href="#"><u>*</u></a>	*	Selects all elements	2
<a href="#"><u>element</u></a>	p	Selects all <p> elements	1
<a href="#"><u>element,element</u></a>	div, p	Selects all <div> elements and all <p> elements	1
<a href="#"><u>element element</u></a>	div p	Selects all <p> elements inside <div> elements	1
<a href="#"><u>element&gt;element</u></a>	div > p	Selects all <p> elements where the parent is a <div> element	2
<a href="#"><u>element+element</u></a>	div + p	Selects all <p> elements that are placed immediately after <div> elements	2
<a href="#"><u>element1~element2</u></a>	p ~ ul	Selects every <ul> element that are preceded by a <p> element	3
<a href="#"><u>[attribute]</u></a>	[target]	Selects all elements with a target attribute	2
<a href="#"><u>[attribute=value]</u></a>	[target=_blank]	Selects all elements with target="_blank"	2
<a href="#"><u>[attribute~=value]</u></a>	[title~=flower]	Selects all elements with a title attribute containing the word "flower"	2
<a href="#"><u>[attribute =value]</u></a>	[lang =en]	Selects all elements with a lang attribute value starting with "en"	2
<a href="#"><u>::after</u></a>	p::after	Insert content after every <p> element	2
<a href="#"><u>::before</u></a>	p::before	Insert content before the content of every <p> element	2
<a href="#"><u>:checked</u></a>	input:checked	Selects every checked <input> element	3
<a href="#"><u>:first-child</u></a>	p:first-child	Selects every <p> element that is the first child of its parent	2
<a href="#"><u>:first-of-type</u></a>	p:first-of-type	Selects every <p> element that is the first <p> element of its parent	3
<a href="#"><u>:hover</u></a>	a:hover	Selects links on mouse over	1
<a href="#"><u>:invalid</u></a>	input:invalid	Selects all input elements with an invalid value	3
<a href="#"><u>:lang(language)</u></a>	p:lang(it)	Selects every <p> element with a lang attribute equal to "it"	2
<a href="#"><u>:last-child</u></a>	p:last-child	Selects every <p> element that is the last child of its parent	3
<a href="#"><u>:nth-child(n)</u></a>	p:nth-child(2)	Selects every <p> element that is the second child of its parent	3
<a href="#"><u>:nth-of-type(n)</u></a>	p:nth-of-type(2)	Selects every <p> element that is the second <p> element of its parent	3

# Adding, Removing, and Modifying Elements by Using jQuery

- Use the **selector** function to specify the elements to change or remove

- Common methods include:

- `addClass` `$("#p").addClass("strike");`
- `css` `$("#p").css("background-color", "yellow");`
- `append` `$("#ul").append("<li>New item</li>");`
- `detach` `$("#Warning").detach();`
- `html` `$("#h1").html("<hgroup>...</hgroup>");`
- `replaceWith` `$("#Warning").replaceWith("<p>Panic over!</p>");`
- `val` `$("#input[type=text]").val();`
- `width/height` `$("#div1").width(500).height(500);`

- Traversing - Ancestors

```
$("#span").parentsUntil("div").css({"color": "red", "border": "2px solid red"});
```

# Adding, Removing, and Modifying Elements by Using jQuery

## • Traversing - Ancestors

Method	Description
<a href="#">add()</a>	Adds elements to the set of matched elements
<a href="#">children()</a>	Returns all direct children of the selected element
<a href="#">closest()</a>	Returns the first ancestor of the selected element
<a href="#">contents()</a>	Returns all direct children of the selected element (including text and comment nodes)
<a href="#">each()</a>	Executes a function for each matched element
<a href="#">eq()</a>	Returns an element with a specific index number of the selected elements
<a href="#">filter()</a>	Reduce the set of matched elements to those that match the selector or pass the function's test
<a href="#">find()</a>	Returns descendant elements of the selected element
<a href="#">first()</a>	Returns the first element of the selected elements
<a href="#">last()</a>	Returns the last element of the selected elements
<a href="#">next()</a>	Returns the next sibling element of the selected element
<a href="#">nextAll()</a>	Returns all next sibling elements of the selected element
<a href="#">nextUntil()</a>	Returns all next sibling elements between two given arguments
<a href="#">not()</a>	Remove elements from the set of matched elements
<a href="#">offsetParent()</a>	Returns the first positioned parent element
<a href="#">parent()</a>	Returns the direct parent element of the selected element
<a href="#">parents()</a>	Returns all ancestor elements of the selected element
<a href="#">parentsUntil()</a>	Returns all ancestor elements between two given arguments
<a href="#">prev()</a>	Returns the previous sibling element of the selected element
<a href="#">prevAll()</a>	Returns all previous sibling elements of the selected element
<a href="#">prevUntil()</a>	Returns all previous sibling elements between two given arguments
<a href="#">siblings()</a>	Returns all sibling elements of the selected element

# Handling Control Events by Using jQuery

- Use the jQuery **selector** function to find the item that raises the event
- Use the **bind** method (or a jQuery shortcut) to bind the event handler to the event

```
<script type="text/javascript">
  $(document).ready(function () {
    $("#submit").click(
      function () {
        var userName = $("#NameBox").val();
        $("#thankYouArea").replaceWith(
          "<p>Thank you " + userName + "</p>");
      })
  });
</script>
```

# Handling Control Events by Using jQuery

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

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

**Zadanie 31** – *jQuery*

**Zadanie 32** – *jQuery*

# Animation

Animate() method is used to create custom animations:

- The required params parameter defines the CSS properties to be animated.
- The optional speed parameter specifies the duration of the effect. It can take the following values: "slow", "fast", or milliseconds.
- The optional callback parameter is a function to be executed after the animation completes.

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



# Chaining

- With jQuery, you can chain together actions/methods.
- Thank to chaining browsers do not have to find the same element(s) more than once.

```
$("#p1").css("color", "red").slideUp(2000).slideDown(2000);
```

**Zadanie 33** – *jQuery (animation)*

**Zadanie 34** – *jQuery (animation, chaining)*

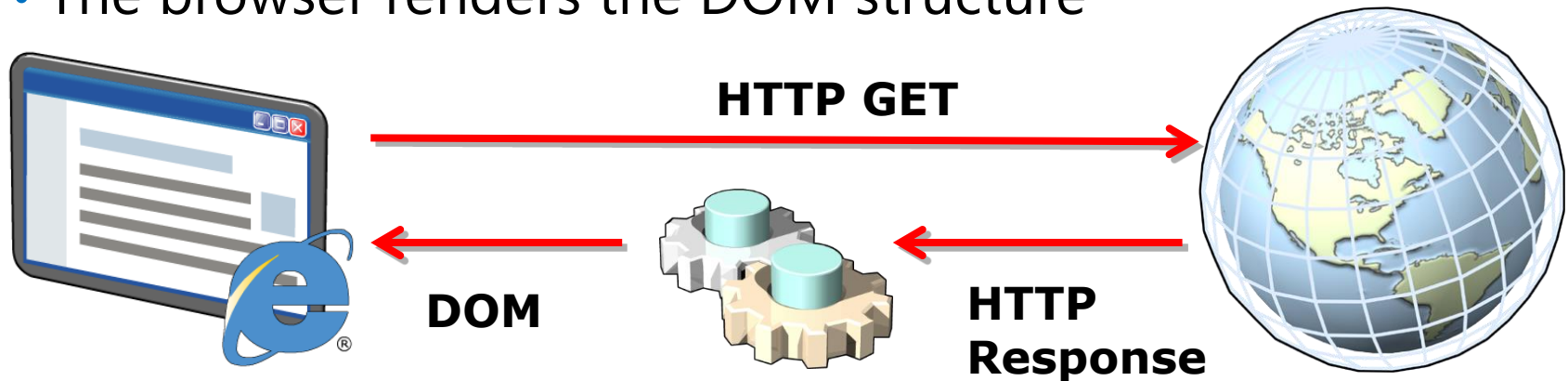
**Zadanie 35 [opcjonalne]** – *jQuery UI*

## Topic 12: Sending and Receiving Data by Using the XMLHttpRequest Object

- How a Browser Retrieves Web Pages
- Using the XMLHttpRequest Object to Access Remote Data
- Handling HTTP Errors
- Consuming the Response
- Handling an Asynchronous Response
- Transmitting Data with a Request

# How a Browser Retrieves Web Pages

- A web browser issues HTTP GET requests to fetch a web page to display
  - The response is parsed into a DOM structure
  - The browser renders the DOM structure



- Elements with a **src** attribute can initiate further HTTP GET requests
- JavaScript code can trigger HTTP GET requests

# Using the XMLHttpRequest Object to Access Remote Data

- To send an HTTP request:
  1. Create a new **XMLHttpRequest** object
  2. Specify the HTTP method and URL
  3. Set the request header
  4. Send the request

```
var request = new XMLHttpRequest();  
var url = "http://contoso.com/resources/...";  
request.open( "GET", url );  
request.send();
```

- Requests are asynchronous by default
  - To block and wait for a response:

```
request.open( "GET", url , false);
```

# Handling HTTP Errors

- Check the status code of the **XMLHttpRequest** object to verify that the request has been sent:

```
var request = new XMLHttpRequest();
request.open("GET", "/luckydip/enter");
request.send( );
...
if( request.status != 200 ) {
    alert( "Error " + request.status + " - " + request.statusText );
}
```

- Wrap your code in a **try...catch** block to handle any unexpected network errors

# Consuming the Response

- Determine the type of data in the response
- Read the response data from the **responseText** property

```
var request = new XMLHttpRequest();  
...  
var type = request.getResponseHeader();  
    switch( type ) {  
        case "text/xml" :  
            return request.responseXML;  
        case "text/json" :  
            return JSON.parse(request.responseText);  
        default :  
            return request.responseText;  
    }
```

# Handling an Asynchronous Response

- Create an event handler for the **readystatechange** event
- Check that the **readyState** of the **XMLHttpRequest** object is set to 4

```
request.onreadystatechange = function () {  
    if (request.readyState === 4) {  
        var response = JSON.parse(request.responseText);  
        ...  
    }  
};
```



# Transmitting Data with a Request

- To send data to a server:
  1. Serialize the data
  2. Set the **Content-Type** property of the request header
  3. Transmit the data by using the HTTP **POST** method

```
var data = JSON.stringify(...);  
var request = new XMLHttpRequest();  
var url = ...;  
request.open("POST", url );  
request.setRequestHeader("Content-Type", "text/plain" );  
request.send(data);
```

**Zadanie 37** – *AJAX (GET)*

**Zadanie 38** – *AJAX (POST)*

## Topic 13: Sending and Receiving Data by Using the jQuery Library

- Using the jQuery Library to Send Asynchronous Requests
- Using the jQuery ajax() Function
- Serializing Forms Data by Using jQuery

# Using the jQuery Library to Send Asynchronous Requests

- The jQuery library provides asynchronous methods for sending requests and handling the response:

```
var response;  
$.get(' http://contoso.com/resources/...', function(data) {  
    response = data;  
}).error(function() {  
    alert("error occurred during get operation");  
});
```

```
$.post("demo_test_post.asp",  
    {  
        name: "Donald Duck",  
        city: "Duckburg"  
    },  
    function(data, status){  
        alert("Data: " + data + "\nStatus: " + status);  
    });
```

```
$.getJSON( url, body, callback );
```

```
$('#container').load( url, body, callback );
```

# Using the jQuery ajax() Function

- The jQuery **ajax()** function provides additional properties and finer control over HTTP requests

```
$.ajax({  
    url: '/luckydip/enter',  
    type: 'GET',  
    timeout: 12000,  
    dataType: 'text'  
}).done(function( responseText ){  
    $('#answer').text( responseText );  
}).fail(function() {  
    alert('An error has occurred – you may not have been entered');  
});
```

# Serializing Forms Data by Using jQuery

- To include forms data in a request, use the **data** property:

```
$.ajax({  
    url: '/luckydip/enterWithName',  
    type: 'POST',  
    timeout: 12000,  
    dataType: 'text',  
    data: {  
        firstName: myForm.fname.value,  
        lastName: myForm.lname.value  
    }  
});
```

- To retrieve input data directly from a form, use the **serializeArray()** function

**Zadanie 39** – *jQuery AJAX (GET)*

**Zadanie 40** – *jQuery AJAX (POST)*

**Zadanie 41 [Opcjonalne]** – *WebApi, Canvas, AJAX*

# Topic 14: Validating User Input by Using JavaScript

- Handling Input Events
- Validating Input
- Validation API
- Ensuring that Fields are Not Empty
- Providing Feedback to the User



# Handling Input Events

- Catch the **submit** event to validate an entire form
  - Return true if the data is valid, false otherwise
  - The form is only submitted if the **submit** event handler returns true
- Catch the **input** event to validate individual fields on a character-by-character basis

# Validating Input

- Use JavaScript code to emulate unsupported HTML5 input types and attributes in a browser:

```
<form id="scoreForm" ... onsubmit="return validateForm();" >
  <div id="scoreField" class="field" >
    <input id="score" name="score" type="number" />
  </div>
</form>
```

```
function isAnInteger( text ){
  var intTestRegex = /^s*(\+|-)?\d+s*$/;
  return String(text).search(intTestRegex) != -1;
}

function validateForm()
{
  if( ! isAnInteger(document.getElementById('score').value))
    return false;  /* No, it's not a number! Form validation fails */

  return true;
}
```

## **Zadanie 42** - *Walidacja*

# Validation API

- Check input validity

```
function myFunction() {  
    var inpObj = document.getElementById("id1");  
    if (inpObj.checkValidity() == false) {  
        document.getElementById("demo").innerHTML = inpObj.validationMessage;  
    }  
}
```

- Custom validation:
  - If the data is not valid display an error message by using the **setCustomValidity** function
  - If the data is valid, reset the error message to an empty string
- Validity properties of controls (main)

patternMismatch	Set to true, if an element's value does not match its pattern attribute.
rangeOverflow	Set to true, if an element's value is greater than its max attribute.
rangeUnderflow	Set to true, if an element's value is less than its min attribute.
tooLong	Set to true, if an element's value exceeds its maxLength attribute.
typeMismatch	Set to true, if an element's value is invalid per its type attribute.
valueMissing	Set to true, if an element (with a required attribute) has no value.

# Ensuring that Fields are Not Empty

Use JavaScript code to ensure that a required field does not contain only whitespace:

```
<form id="scoreForm" ... onsubmit="return validateForm();" >
  <div id="penaltiesField" class="field" >
    <input id="penalties" name="penalties" type="text" />
  </div>
</form>
```

```
function isSignificant( text ){
  var notWhitespaceTestRegex = /^[^\s]{1,}/;
  return String(text).search(notWhitespaceTestRegex) != -1;
}

function validateForm() {
  if( ! isSignificant(document.getElementById('penalties').value))
    return false;  /* No! Form validation fails */

  return true;
}
```

# Providing Feedback to the User

- Provide visual feedback to the user by defining styles and dynamically setting the class of an element:

```
.validatedFine {  
    border-color: #0f0;  
}  
.validationError {  
    border-color: #f00;  
}
```

```
function validateForm() {  
    var textbox = document.getElementById("penalties");  
  
    if( ! isSignificant(textBox.value)) {  
        textbox.className = "validationError";  
        return false; /* No! Form validation fails */  
    }  
    textbox.className = "validatedFine";  
    return true;  
}
```

## **Zadanie 43** – *Walidacja HTML5*

# Topic 15: Exceptions, Debugging and Profiling a Web Application

- Exceptions
- Overview of the F12 Developer Tools in Internet Explorer 10
- Using the F12 Developer Tools to Debug JavaScript Code
- Using the F12 Developer Tools to Profile a Web Application



# Exceptions

- The **try** statement lets you test a block of code for errors.
- The **catch** statement lets you handle the error.
- The **throw** statement lets you create custom errors.
- The **finally** statement lets you execute code, after try and catch, regardless of the result.

```
try {  
    Block of code to try  
}  
catch(err) {  
    Block of code to handle errors  
}  
finally {  
    Block of code to be executed regardless of the try / catch  
    result  
}
```

# Exceptions

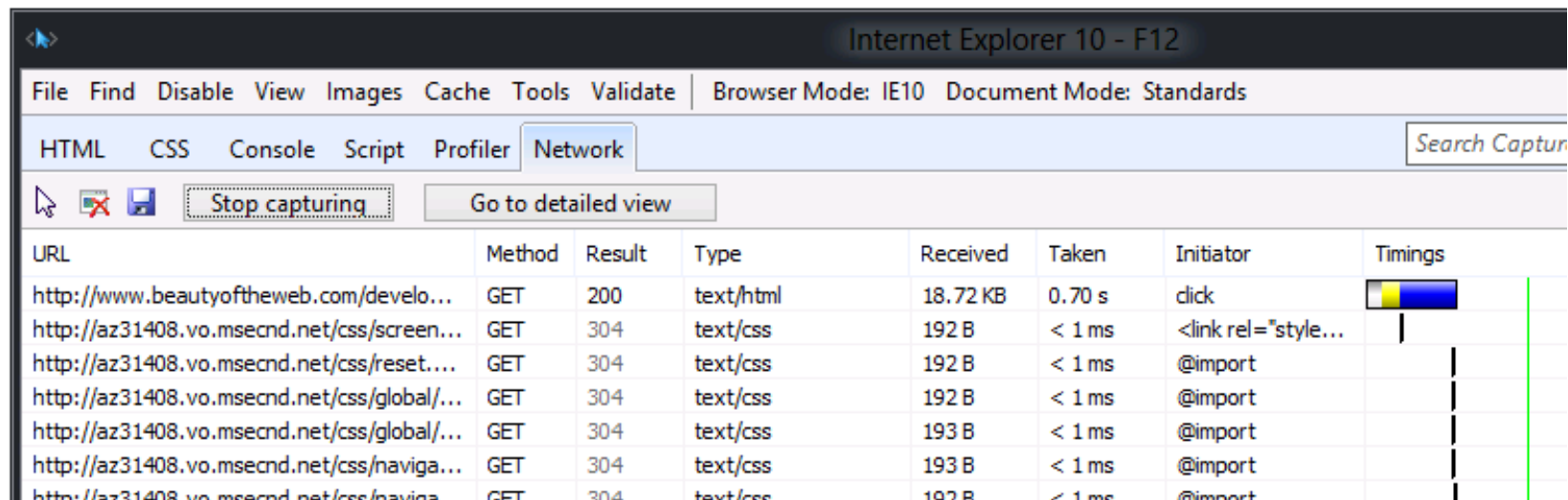
- Example

```
function myFunction() {  
  var message, x;  
  message = document.getElementById("message");  
  message.innerHTML = "";  
  x = document.getElementById("demo").value;  
  try {  
    if(x == "") throw "empty";  
    if(isNaN(x)) throw "not a number";  
    x == Number(x);  
    if(x < 18) throw "too low";  
    if(x > 100) throw "too high";  
  }  
  catch(err) {  
    message.innerHTML = "Input is " + err.message;  
  }  
}
```


## **Zadanie 44** - *Obsługa błędów*

# Overview of the F12 Developer Tools in Internet Explorer

- The Navigation Timing API enables an application to determine the download speed for a web page:
  - **window.performance.navigation**
  - **window.performance.timing**
  - **console.log**
- The F12 Developer Tools provide debugging and profiling capabilities in Internet Explorer



The screenshot shows the Internet Explorer 10 F12 Developer Tools interface. The 'Network' tab is selected, displaying a list of network requests. The table includes columns for URL, Method, Result, Type, Received, Taken, Initiator, and Timings. The first request is a GET request to 'http://www.beautyoftheweb.com/develo...' with a status of 200 and a type of 'text/html'. Subsequent requests are GET requests to 'http://az31408.vo.msecnd.net/css/screen...' with a status of 304 and a type of 'text/css'.

URL	Method	Result	Type	Received	Taken	Initiator	Timings
http://www.beautyoftheweb.com/develo...	GET	200	text/html	18.72 KB	0.70 s	click	
http://az31408.vo.msecnd.net/css/screen...	GET	304	text/css	192 B	< 1 ms	<link rel="style...	
http://az31408.vo.msecnd.net/css/reset....	GET	304	text/css	192 B	< 1 ms	@import	
http://az31408.vo.msecnd.net/css/global/...	GET	304	text/css	192 B	< 1 ms	@import	
http://az31408.vo.msecnd.net/css/global/...	GET	304	text/css	193 B	< 1 ms	@import	
http://az31408.vo.msecnd.net/css/naviga...	GET	304	text/css	193 B	< 1 ms	@import	
http://az31408.vo.msecnd.net/css/naviga...	GET	304	text/css	193 B	< 1 ms	@import	

# Using the F12 Developer Tools to Debug JavaScript Code

You can use the F12 Developer Tools to:

- Set a breakpoint in JavaScript code
- Step through JavaScript code and examine variables

## **Zadanie 45** – *Testowanie*

# Using the F12 Developer Tools to Profile a Web Application

You can use the F12 Developer Tools to:

- Examine the network traffic for a web application
- Capture profile data for a web application

## **Zadanie 46** – *Kompresja*



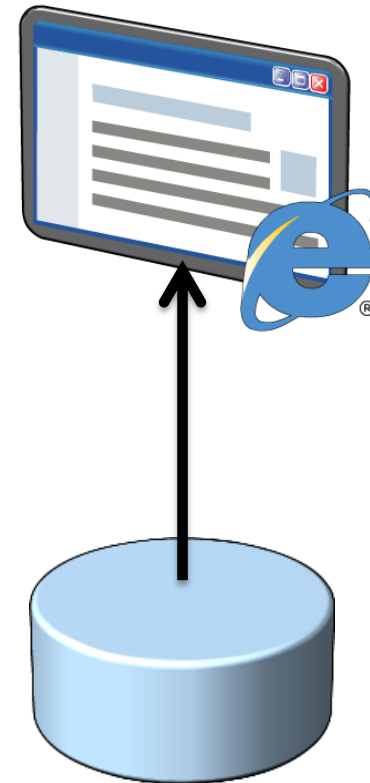
# Additional topics

# Topic 16: Interacting with Files

- HTML5 File Interfaces
- The FileReader Interface
- Reading a Text File
- Reading a Binary File
- Implementing Drag-and-Drop

# HTML5 File Interfaces

- The HTML5 File API enables a web application to access the local file system
- There are four key interfaces:
  - **Blob** – immutable raw binary data
  - **File** - readonly information about a file
  - **FileList** – an array of files
  - **FileReader** – methods for reading data from a file or blob



# The FileReader Interface

- The **FileReader** interface provides methods for reading a file or blob:
  - **readAsText()** – used for reading text files
  - **readAsDataURL()** – used for reading binary files
  - **readAsArrayBuffer()** – used for reading data into a buffer array
- **FileReader** reads data asynchronously and fires events:

**progress**  
**load**

**abort**  
**error**

**loadend**

# Reading a Text File

To read a text file:

1. Get a File or Blob object, either by using an **<input type="file">** element or by drag-and-drop.
2. Create a **FileReader** object and handle events such as **load** and **error**.
3. Invoke **readAsText()** on the **FileReader** object.
4. In the **load** event handler function, access the text content in the **result** property of the event target.
5. In the **error** event handler function, implement appropriate error handling.

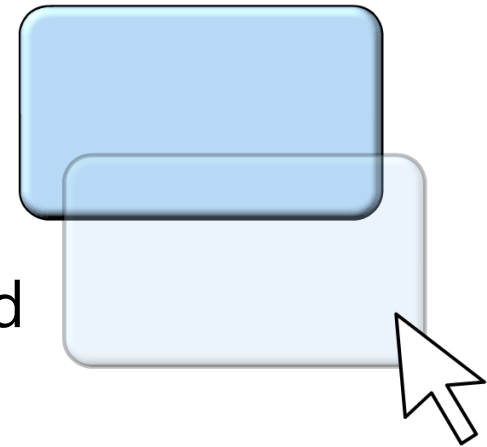
# Reading a Binary File

To read a binary file:

1. Get a File or Blob object, either by using an **<input type="file">** element or by drag and drop.
2. Create a **FileReader** object and handle events such as **load** and **error**.
3. Invoke **readAsDataURL()** on the **FileReader** object.
4. In the **load** event handler function, access the text content in the **result** property of the event target.
5. In the **error** event handler function, implement appropriate error handling.

# Implementing Drag-and-Drop

- HTML5 supports drag-and-drop
  - The user can drag HTML elements, or files from the local file system
  - The user can drop items onto drop-enabled target elements
- To support drag and drop operations
  - Enable drag support on HTML elements, if required
  - Enable drop support on HTML drop target elements
  - Handle dragover and drop events on HTML drop target elements



## **Zadanie 47** – *File System, Drag& Drop*



# Topic 17: Incorporating Multimedia

- Playing Video Content by Using the `<video>` Tag
- Supporting Multiple Video Formats
- Interacting with Video in JavaScript Code
- Playing Audio Content by Using the `<audio>` Tag

# Playing Video Content by Using the <video> Tag

- HTML5 enables a web application to play video files natively, without requiring plugins
- Use the **<video>** tag and set the attributes:
  - **src**
  - **width** and **height**
  - **poster**
  - **controls**
  - **autoplay**
  - **loop**
  - **muted**

```
<video src="MyVideo.mp4"
       width="300" height="200"
       poster="MyPoster.jpg"
       autoplay="autoplay"
       muted="muted"
       controls="controls"
       loop="loop" >
</video>
```

# Supporting Multiple Video Formats

- A **<video>** element can support multiple video formats:

```
<video poster="MyPoster.jpg" autoplay controls>  
  <source src="MyVideos/MyVideo.mp4" type='video/mp4' />  
  <source src="MyVideos/MyVideo.webm" type='video/webm' />  
  <source src="MyVideos/MyVideo.ogv" type='video/ogg' />  
</video>
```

- You can embed Silverlight or Flash content in a **<video>** tag as a fall-back

# Interacting with Video in JavaScript Code

- An application can interact with a **video** object in JavaScript code:

```
var newVideo = document.createElement("video");
newVideo.src = nameOfVideoFile;
newVideo.loop = true;
newVideo.controls = true;
newVideo.poster = "ImageLoading.png";

...

newVideo.addEventListener("loadeddata", function() {
    newVideo.play();
}, false);
```

# Playing Audio Content by Using the <audio> Tag

- Use the **<audio>** tag to play audio files natively, without requiring plugins:

```
<audio src="MyAudio.mp3"></audio>
```

- The JavaScript API for audio is similar to the API for video

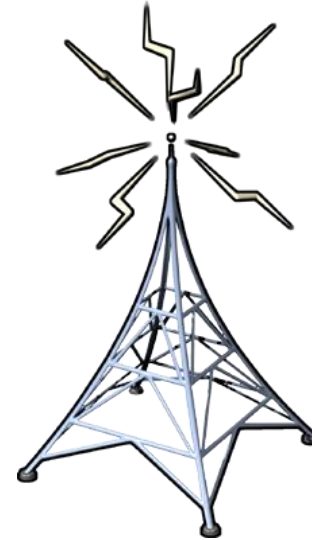
## **Zadanie 48 – *Multimedia***

# Topic 18: Reacting to Browser Location and Context

- The HTML5 Geolocation API
- Requesting Geolocation Information
- Processing Geolocation Information
- Handling Geolocation Errors
- Detecting the Context for a Page

# The HTML5 Geolocation API

- The Geolocation API enables a browser to determine the longitude and latitude of its current location
- A host device can use several techniques to obtain geolocation information:
  - IP address
  - GPS positioning
  - Wi-Fi
  - Cell phone location
  - User-defined location information





# Requesting Geolocation Information

- To make a one-shot request for position information:

```
navigator.geolocation.getCurrentPosition(myPositionCallbackFunction,  
                                         myPositionErrorCallbackFunction,  
                                         {enableHighAccuracy: true, timeout: 5000} );
```

- To receive repeated position information updates:

```
var watchID =  
    navigator.geolocation.watchPosition(myPositionCallbackFunction,  
                                         myPositionErrorCallbackFunction,  
                                         {enableHighAccuracy: true, maximumAge: 10000} );  
...  
navigator.geolocation.clearWatch(watchID);
```

# Processing Geolocation Information

- Geolocation properties include:

- **latitude**
- **longitude**
- **accuracy**



- Geolocation data may include the following optional properties:

- **altitude**
- **altitudeAccuracy**
- **heading**
- **speed**



# Handling Geolocation Errors

- If an error occurs during a geolocation request, the following properties are available:
  - **code**
    - **PositionError.PERMISSION\_DENIED**
    - **PositionError.POSITION\_UNAVAILABLE**
    - **PositionError.TIMEOUT**
  - **message**

```
function myPositionErrorCallbackFunction(error) {  
    var errorMessage = error.message;  
    var errorCode = error.code;  
    // Add code here, to process the information.  
}
```

# Detecting the Context for a Page

- Page Visibility API
  - Enables an application to determine whether a page is currently visible.
- Offline detection
  - Enables an application to detect whether the browser has a live connection to a server.
- User agent information
  - Enables an application to obtain the user agent string for the browser.

## **Zadanie 49** – *Geolocation*

# Topic 19: Reading and Writing Data Locally

- Maintaining Session State Information by Using Cookies
- Persisting Session Data by Using Session Storage
- Persisting Data Across Sessions by Using Local Storage
- Handling Storage Events
- Storing Structured Data by Using the Indexed Database API

# Maintaining Session State Information by Using Cookies

- Cookies:
  - Were designed to implement session tokens
  - Are sent to the server on every page request
  - Are small files of limited size, up to 4 KB
  - Are open to abuse
  - Have no synchronization or concurrency mechanism
- Cookies were not designed for general-purpose data storage

# Persisting Session Data by Using Session Storage

- Use the **sessionStorage** object to store and retrieve text data for a session:

```
sessionStorage.setItem("myKey", "some text value");  
var textFromSession1 = sessionStorage.getItem("myKey");
```

```
sessionStorage["myKey"] = "some text value";  
var textFromSession2 = sessionStorage["myKey"];
```

```
sessionStorage.myKey = "some text value";  
var textFromSession3 = sessionStorage.myKey;
```

- Session data is only available in the session that creates it
  - Session storage is cleared when the user finishes the browser session



# Persisting Data Across Sessions by Using Local Storage

- Use the **localStorage** object to persist data across sessions and web pages:

```
localStorage.setItem("myKey", "some text value");  
var textData = localStorage.getItem("myKey");
```

```
localStorage["myKey"] = "some text value";  
var textData = sessionStorage["myKey"];
```

```
localStorage.myKey = "some text value";  
var textData = sessionStorage.myKey;
```

- Data is persisted until it is explicitly removed

# Handling Storage Events

- Use the **storage** event to notify a web page of changes made to session and local storage:

```
function myStorageCallback( e ) {  
    alert( "Key:" + e.key + " changed to " + e.newValue );  
}  
...  
window.addEventListener("storage", myStorageCallback, true );
```

- Properties of the event object:

key	– name of the value which has changed
oldValue	– the original value
newValue	– the new value
url	– the origin of the event
storageArea	– a reference to the store that has changed

# Storing Structured Data by Using the Indexed Database API

- IndexedDB provides an efficient means for storing structured data on the user's computer
- The API is asynchronous, and includes the following features:
  - Multiple object stores
  - **add()**, **put()**, **get()**, and **delete()** operations on data
  - Transactions
  - Queries by using cursors
  - Indexes to speed up common queries

## **Zadanie 50** - *WebStorage*

## Topic 20: Adding Offline Support by Using the Application Cache

- Configuring the Application Cache
- Monitoring with the Application Cache
- Triggering Resource Updates by Using the Manifest
- Testing for Network Connectivity

# Configuring the Application Cache

- The application cache manifest file specifies the resources to cache, and how they should be updated:
- Each web page that uses cached resources should reference the manifest file:

```
<html manifest="appcache.manifest">
```

```
CACHE MANIFEST
```

```
CACHE:  
index.html  
verification.js  
site.css  
graphics/logo.jpg
```

```
NETWORK:  
/login
```

```
# alternatives paths  
FALLBACK:  
/ajax/account/ /noCode.htm
```

# Monitoring with the Application Cache

- Use the **applicationCache** object to monitor the cache:

```
applicationCache.addEventListener( "error", function() {  
    alert( "Error while downloading resources to the application cache");  
}, true );
```

- Examine the **status** property to determine cache state:

0	UNCACHED	No resources have been downloaded.
1	IDLE	All cached resources have been downloaded.
2	CHECKING	The cache is being checked for updates.
3	DOWNLOADING	Resources are being downloaded to the cache.
4	UPDATEREADY	The cache has been updated with new resources.
5	OBSOLETE	The manifest is missing and no cache is available.

# Triggering Resource Updates by Using the Manifest

- A web page may continue to use cached resources even if newer versions are available
- To force an update:
  - Make a significant change to the manifest file, or
  - Initiate a check for updates by using the **update()** function, and then use the **swapCache()** function

```
applicationCache.update();  
...  
if (applicationCache.status == 4 ) {  
    applicationCache.swapCache();  
}
```



# Testing for Network Connectivity

- Sometimes it is better to disable functionality that requires a network connection
- Use the **onLine** property of the **navigator** object to detect the network status
- Handle the **online** and **offline** events of the window object to track changes to network connectivity

## **Zadanie 51** – *Offline support*

# Topic 21: Creating Interactive Graphics by Using SVG

- What is SVG?
- Creating SVG Graphics
- Drawing Circles and Ellipses
- Drawing Complex Shapes
- Specifying Fill Styles and Stroke Styles
- Using Gradients and Patterns
- Drawing Graphical Text
- Transforming SVG Elements
- Demonstration: Using SVG Transformations and Events

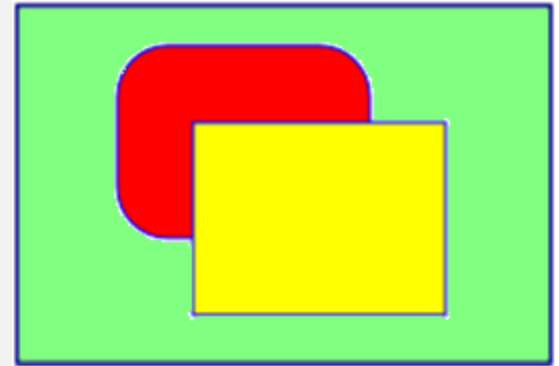
# What is SVG?

- SVG enables you to draw 2D vector graphics
  - It defines XML elements to represent a wide range of shapes
- SVG uses a "retained mode" model
  - The objects tree is kept in memory
  - Rendering speed depends on the number of elements
- You can perform the following operations on SVG-related elements:
  - Access elements through DOM
  - Style elements with CSS
  - Handle user-interaction events

# Creating SVG Graphics

- Use an **<svg>** element and embed child elements that define the graphics:

```
<svg xmlns="http://www.w3.org/2000/svg">  
  <rect x="50" y="50" width="100" height="75"  
    rx="20" ry="20" fill="red" stroke="blue" />  
  
  <rect x="75" y="75" width="100" height="75"  
    fill="yellow" stroke="blue" />  
  
</svg>
```



- Style SVG elements by using CSS:

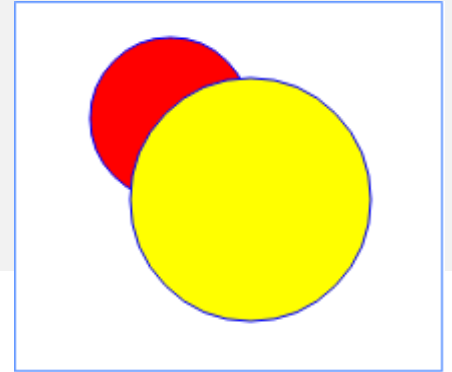
```
<style type="text/css">  
  svg {  
    border: 2px solid darkblue;  
    background-color: lightgreen;  
    width: 300px;  
    height: 200px;  
  }  
</style>
```

# Drawing Circles and Ellipses

To draw circles:

```
<circle cx="120" cy="80" r="40"  
  stroke="blue" fill="red" />
```

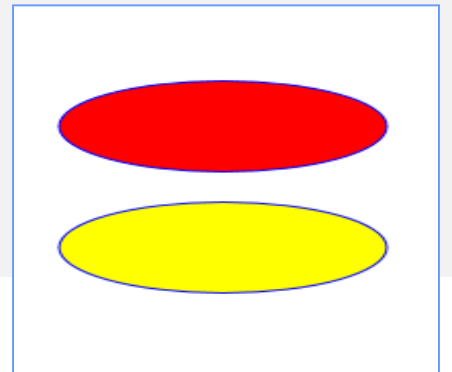
```
<circle cx="160" cy="120" r="60"  
  stroke="blue" fill="yellow" />
```



To draw ellipses:

```
<ellipse cx="150" cy="60" rx="110" ry="30"  
  stroke="blue" fill="red" />
```

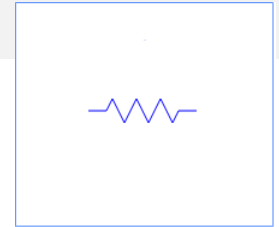
```
<ellipse cx="150" cy="140" rx="110" ry="30"  
  stroke="blue" fill="yellow" />
```



# Drawing Complex Shapes

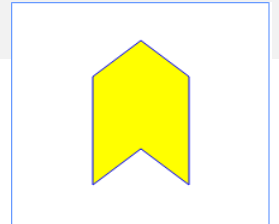
- To draw a polyline:

```
<polyline points="105 100, 120 100, 125 90, 135 110, 145 90, ...  
    fill="none" stroke="blue" />
```



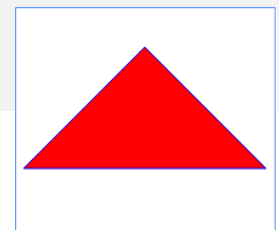
- To draw a polygon:

```
<polygon points="110 70, 150 40, 190 70, 190 160, 150 130, 110 160"  
    fill="yellow" stroke="blue" />
```



- To draw a path:

```
<path d="M 150 50 L 250 150 L 50 150 Z"  
    fill="red" stroke="blue" />
```



# Specifying Fill Styles and Stroke Styles

- You can set the fill style or stroke style for an element

```
stroke="color"
```

```
fill="color"
```

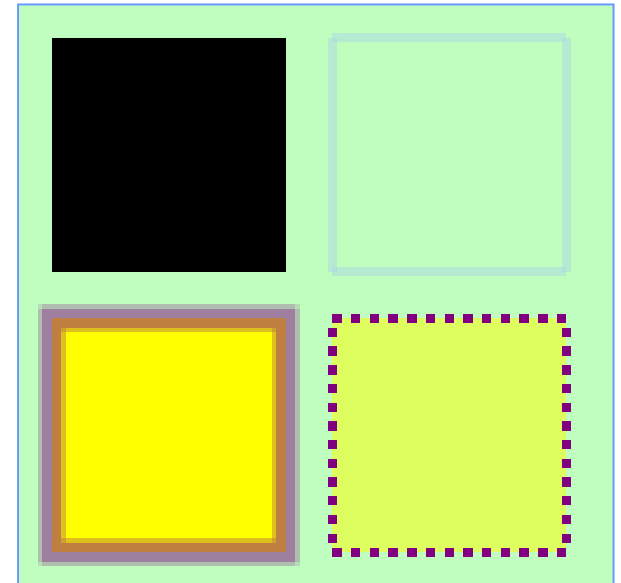
```
stroke-opacity="opacity-fraction"
```

```
fill-opacity="opacity-fraction"
```

```
stroke-width="width"
```

```
fill-rule="nonzero" | "evenodd"
```

```
stroke-dasharray="dash-gap series"
```

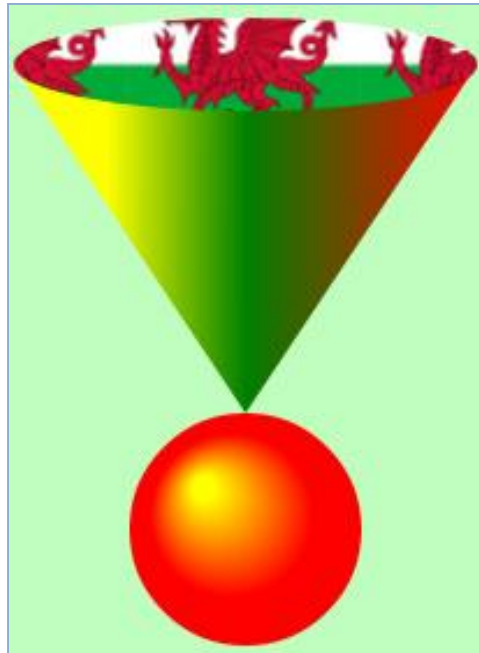




# Using Gradients and Patterns

- You can define a linear or radial gradient for shapes
- Patterns can specify image files

```
<polygon points="50,50 250,50 150,200" fill="url(#gradient1)" />  
<ellipse cx="150" cy="50" rx="100" ry="20" fill="url(#wales)" />  
<circle cx="150" cy="250" r="50" fill="url(#gradient2)" />
```



# Drawing Graphical Text

- To draw text:
- Use the following elements to achieve additional text effects:
  - `<linearGradient>`, `<radialGradient>`, `<pattern>`
  - `<textPath>`
  - `<tspan>`

```
<text x="20" y="100">  
Simple text  
</text>
```



Hello World!!!

Normal text

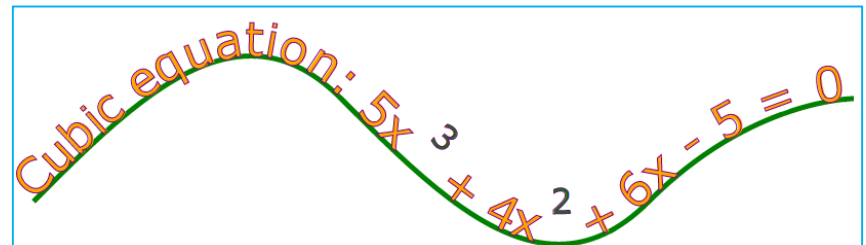
~~Text with line through~~

Underlined text

~~Underlined text with line through~~



Mae hen wlad fy nhadau yn annwyl i mi!



Cubic equation:  $5x^3 + 4x^2 + 6x - 5 = 0$

# Transforming SVG Elements

- To transform SVG elements, set the `transform` attribute to a transformation function

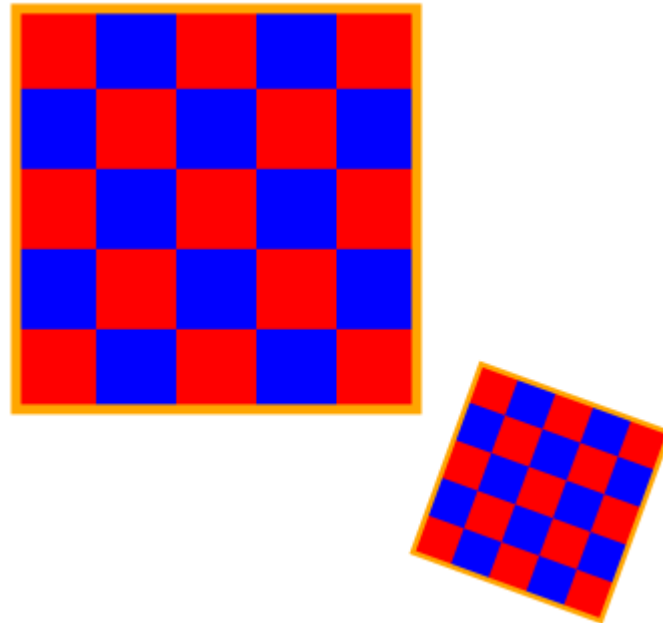
```
rotate(angle, cx, cy)
```

```
translate(dx, dy)
```

```
scale(sx, sy)
```

```
skewX(angle)
```

```
skewY(angle)
```



- To perform a transformation on several elements enclose elements in a `<g>` element, and transform the `<g>`

## **Zadanie 52 – *SVG***

# Topic 22: Drawing Graphics by Using the Canvas API

- What is the Canvas API?
- Using the Canvas API
- Drawing Paths
- Using Gradients and Patterns
- Transforming Shapes

# What is the Canvas API?

- The Canvas API enables you to draw onto a bitmap area
  - Immediate mode: "fire and forget"
  - It does not remember what you drew last
- JavaScript APIs and drawing primitives
  - Simple API: 45 methods, 21 attributes
  - Rectangles, lines, fills, arcs, Bezier curves, ...
- No DOM support
  - A canvas is a "black box"

# Using the Canvas API

- Create a **<canvas>** element

```
<h1>Getting started with canvas</h1>
```

```
<canvas id="myCanvas">No canvas support</canvas>
```

- Define styles for the **<canvas>** element

```
canvas {  
  border: 2px solid darkblue;  
  background-color: lightgreen;  
}
```

Getting started with canvas



- Write JavaScript code to draw on the canvas

```
var canvas = document.getElementById('myCanvas');  
var context = canvas.getContext('2d');  
context.fillStyle = "red";  
context.fillRect(20, 20, canvas.width - 40, canvas.height - 40);
```

# Drawing Paths

- Use a path to draw a complex shape
- Use the **beginPath**, **moveTo**, **lineTo**, and **closePath** functions to define the shape
- Draw the shape by using the **stroke** function
  - Specify the style by using the **strokeStyle** function
- Fill the shape by using the **fill** function
  - Specify the style by using the **fillStyle** function

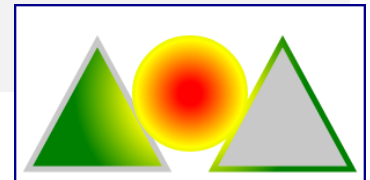
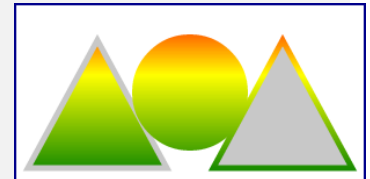


# Using Gradients and Patterns

- To fill a shape with a gradient:

```
var grad = ctx.createLinearGradient(x1, y1, x2, y2);  
var grad = ctx.createRadialGradient(startCx, startYy, startRad,  
                                   endCx, endCy, endRad);
```

```
grad.addColorStop(fraction1, color1);  
grad.addColorStop(fraction2, color2);  
...  
ctx.fillStyle = grad;
```



- To fill a shape with a pattern:

```
var image = document.getElementById("anImageElement");  
var pattern = ctx.createPattern(image, "repeat");  
ctx.fillStyle = pattern;
```



# Transforming Shapes

- To rotate the canvas context:

```
ctx.rotate(clockwiseAngleInRadians);
```

- To translate the canvas context:

```
ctx.translate(deltaX, deltaY);
```

- To scale the canvas context:

```
ctx.scale(xScaleMultiple, yScaleMultiple);
```

- To adjust the current transformation matrix:

```
ctx.transform(scaleX, skewX, scaleY, skewY, translateX, translateY);
```

- To set a new transformation matrix:

```
ctx.setTransform(scaleX, skewX, scaleY, skewY, translateX, translateY);
```

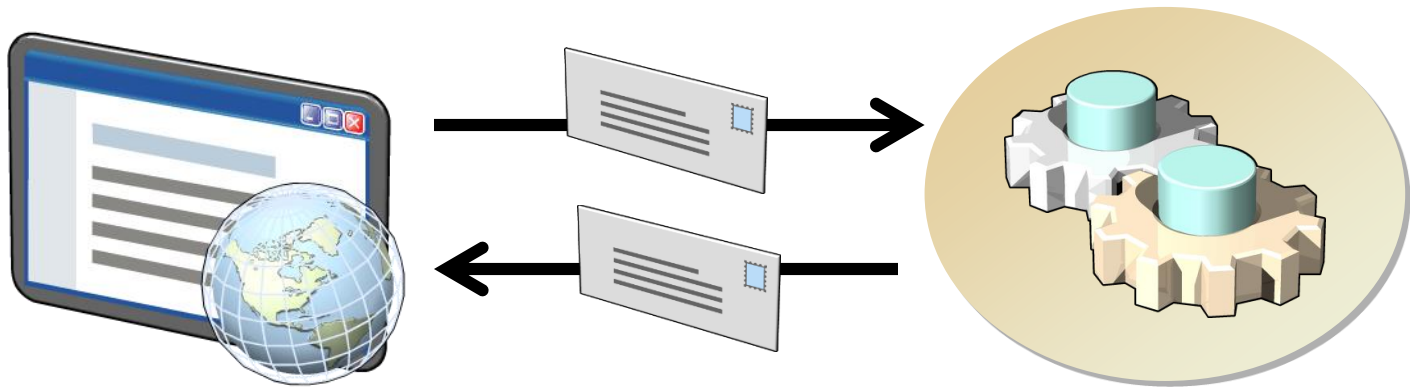
## **Zadanie 53** – *Canvas*

# Topic 23: Understanding Web Workers

- What is a Web Worker?
- Why Use a Web Worker?
- Web Worker Isolation
- Dedicated and Shared Web Workers

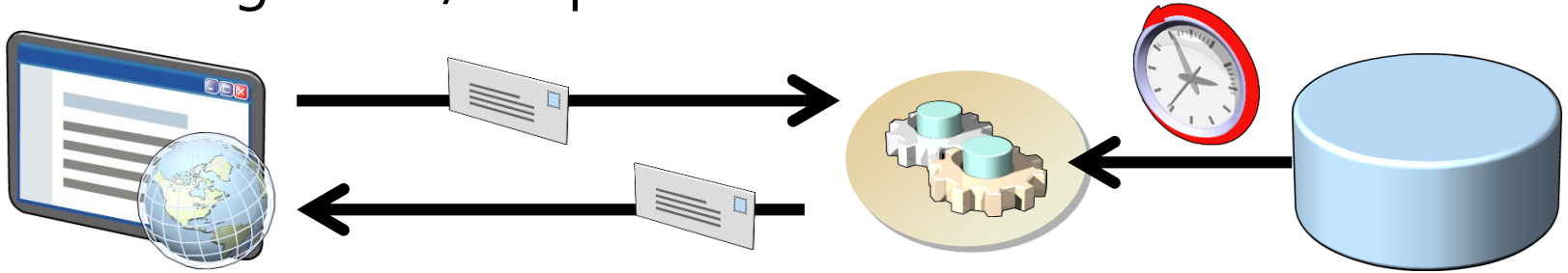
# What is a Web Worker?

- JavaScript code running in a web page is single-threaded
  - Long-running functions can cause the browser to become unresponsive
- Web workers enable a web page to move code to a parallel execution environment, enabling the browser to remain responsive
  - Code in the web page communicates with the web worker by passing messages

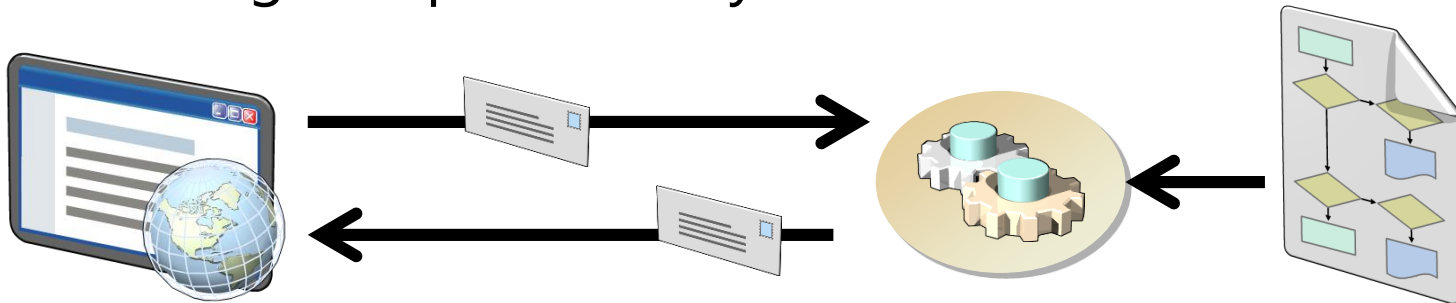


# Why Use a Web Worker?

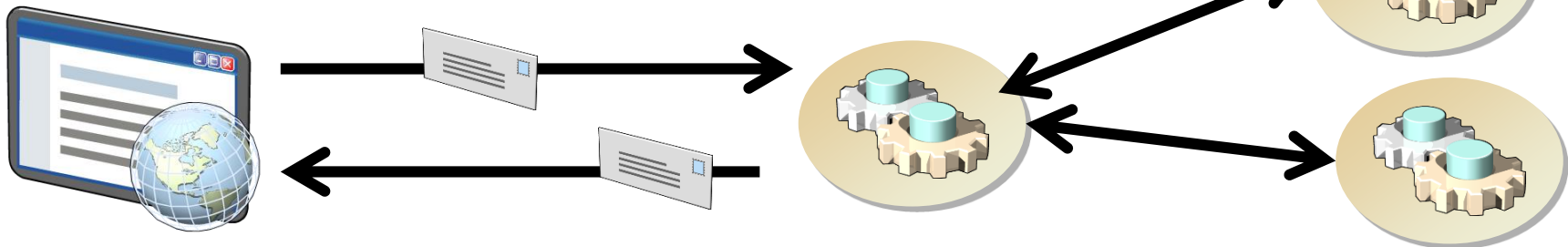
- Performing slow I/O operations:



- Performing computationally-intensive calculations:



- Implementing multitasking:



# Web Worker Isolation

- A web worker runs isolated from the web page and other web workers
  - It cannot access the document of the web page
  - It cannot access data or JavaScript code in the web page
- A web worker has access to a limited subset of JavaScript functionality
- A web page communicates with a web worker by sending and receiving messages:
  - Send messages by using the **postMessage()** function
  - Receive messages by handling the **message** event

# Dedicated and Shared Web Workers

- Dedicated web workers:
  - Belong to a single page
  - Can only communicate with that page
  - Stop when the page is closed
- Shared web workers:
  - Can be accessed by all pages in a web application
  - Can communicate with any page in the web application
  - Stop when the web application finishes



## Lesson 2: Performing Asynchronous Processing by Using Web Workers

- Creating and Terminating a Dedicated Web Worker
- Communicating With A Dedicated Web Worker
- The Structure of a Web Worker
- Creating a Shared Web Worker
- Demonstration: Creating a Web Worker Process

# Creating and Terminating a Dedicated Web Worker

- Starting a web worker:

```
var webWorker;  
if( typeof(Worker) !== "undefined") {  
    webWorker = new Worker("processScript.js");  
}
```

- Terminating a web worker from the web page:

```
webWorker.terminate();
```

- Terminating a web worker from inside the web worker:

```
self.close();
```

# Communicating With A Dedicated Web Worker

- The web page sends messages to the web worker by using the **postMessage()** function

```
webWorker.postMessage("Here is some data");}
```

- The web worker receives messages by handling the **message** event

```
function messageHandler(event) { ... }  
self.addEventListener("message", messageHandler, false);
```

- The web worker sends a reply by using **postMessage**, and the web page receives the reply by catching the **message** event
- If the web worker fails with an error, it sends the **error** event and terminates

# The Structure of a Web Worker

- Web workers often implement a message loop and the Command pattern:

```
function messageHandler(event) {  
    var data = event.data;  
    switch (data.command) {  
        case "DOWORK": // process the DOWORK command  
            ...  
            break;  
        case "DOMOREWORK": // process the DOMOREWORK command  
            ...  
            break;  
        case "FINISH": // tidy up and shut down  
            ...  
            self.postMessage("Shutting down");  
            self.close();  
        }  
    }  
}
```

- Web workers can import scripts and access some global objects and functions

# Creating a Shared Web Worker

- Use the SharedWorker constructor to create a shared web worker
  - Each web page communicates with a shared web worker by using its own port

```
function replyHandler(event) { ... }  
var sharedWebWorker = new SharedWorker("sharedProcessScript.js");  
sharedWebWorker.port.addEventListener("message", replyHandler, false);  
sharedWebWorker.port.start();  
...  
var data = ...;  
sharedWebWorker.port.postMessage(data);
```

- The **connect** event in a shared web worker fires when a new port is opened

## **Zadanie 54** – *Web Workers*

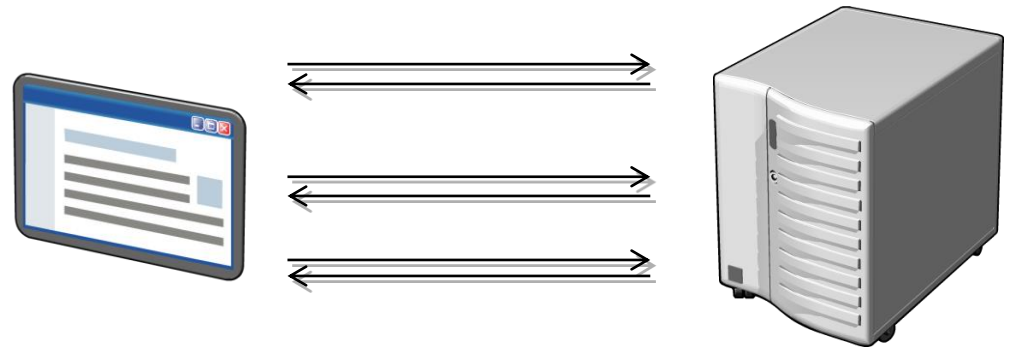
# Topic 24: Introduction to Web Sockets

- The Problem of Web-based Real-time Communications
- What is a Web Socket?
- Connecting to a Server by Using a Web Socket
- Sending Messages to a Web Socket
- Receiving Messages From a Web Socket

# The Problem of Web-based Real-time Communications

- Common techniques for implementing real-time communications include:

- Continuous polling
- Long polling

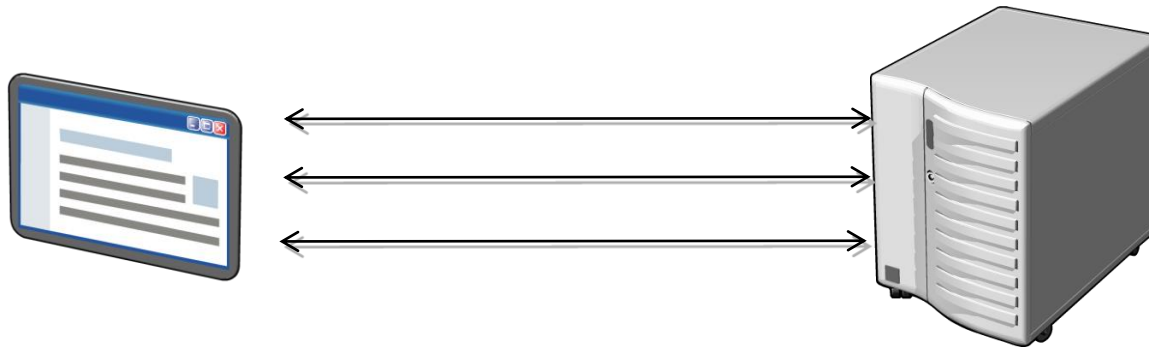


- Real-time communication solutions between web page and server have two main issues
  - Additional HTTP headers increase message size
  - HTTP is not full-duplex



# What is a Web Socket?

- Web sockets are a solution for implementing real-time bidirectional communications on the web



- RFC6455 defines the ws and wss protocols
- The WebSockets API defines a JavaScript API for code running in a browser

# Connecting to a Server by Using a Web Socket

- The **WebSocket** object contains the functionality required to communicate with a server through a web socket

- Establish a connection by creating a new web socket:

```
var socket =  
    new WebSocket('ws://websocketserver.contoso.com/bookings');
```

- Check that the socket was opened successfully before using it:

```
socket.onopen = function() {  
    alert("Connection to server now open!");  
    ...  
};
```

- Use the **close()** function to terminate the connection:

```
socket.close();
```

# Sending Messages to a Web Socket

- Use the **send()** function to send messages to a server

```
var message = ...;  
socket.send(message);
```

- Use the **bufferedAmount** property to determine:
  - If there is a backlog before sending
  - If the message has been sent
- Handle the **error** event to determine whether an error has occurred
- Messages can be text, binary, or array data

# Receiving Messages From a Web Socket

- The **message** event fires when a message is received from a server:
  - Examine **event.type** to determine whether the message is text or binary
  - Read **event.data** to retrieve the message

```
socket.onmessage = function(event) {  
  if (event.type == "Text") {  
    handleTextMessage(event.data);  
  } else {  
    handleBinaryMessage(socket.binaryType, event.data);  
  }  
};
```

- Before receiving a message, set the **binaryType** property of the web socket object to indicate the expected format for binary data

## **Zadanie 55** – *Web sockets*