Univerzitet Singidunum

Web programming

www.singidunum.ac.rs

JavaScript

- Web oriented programming languages are divided to server and client side languages
- Server languages are executing on the web server, and after execution, HTML code is sent to browser
- Server side languages are more complex, and they can operate with database servers

JavaScript

- Client side languages are simpler
- They are executing on the client side, in browser
- After sending request for web page, browser receives HTML code from the web server, and after interpretation it shows it to the user
- Client side code, integrated into HTML page, is visible to the user (Page Source)
- Server side code is never visible to user (we just see added HTML code)

JavaScript

- Script languages in general are not programming languages, as they don’t have their own environment, and usually are integrated with some other programming languages
- Easy to learn
- Understandable syntax

- Examples for web: JavaScript, VisualBasic Script, ASP Script, PHP

JavaScript

- JavaScript can be used with all environments for writing HTML, it is easy to learn, and almost all written JavaScripts on the web are free
- High level of interactivity
- Built in to almost all web browsers, which can read it without installing additional programs
- Procedural and object oriented
- It is not compiled separately, but together with HTML, and just syntax check from the browser
- Syntax is similar to Java, C, or C++
- CASE SENSITIVE
- In HTML code it is separated with <script></script> tags

JavaScript

- JavaScript VS Java?
- They appeared in almost same time
- Completely different approaches to activating web pages
- Different companies (Netscape vs Sun)
- Commonly mistaken by users
- JavaScript is not Java

- JavaScript cannot be executed individually, or in any other environment which is not web browser
- Java applet is the only Java variant which is connected to web

JavaScript

- Java is object oriented programming language
- It is necessary to compile the code
- Code is platform independent
- It uses classes available either inside compiled code, or on the user computer
- Java is fully object oriented

JavaScript

- Differences

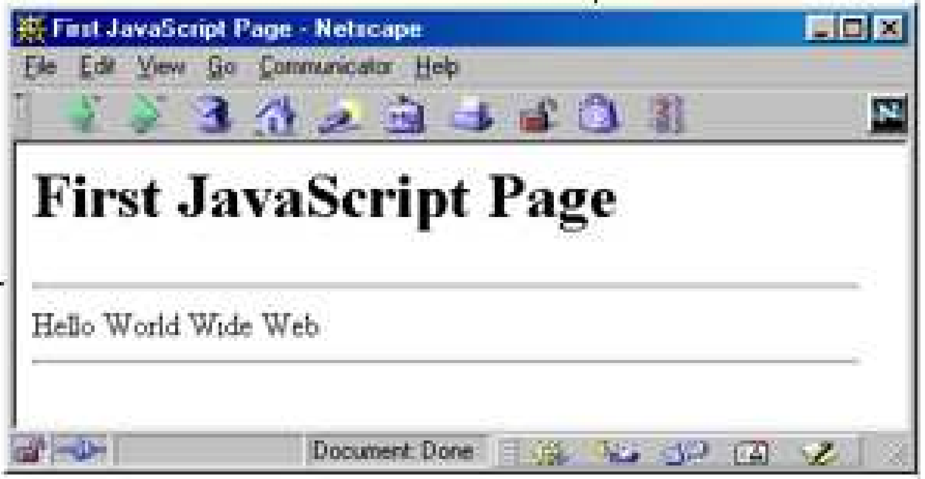
Java	JavaScript
Programming language	Script language
Strong variable definition	Free variable definition
Full object oriented code	Partly object oriented code
Strong type check	No type check
Object execution structure	Vertical execution structure
Java classes	JavaScript objects

What can we do with JS?

- To create interactive user interface in a web page (e.g., menu, pop-up alert, windows, etc.)
- Manipulating web content dynamically
  - Change the content and style of an element
  - Replace images on a page without page reload
  - Hide/Show contents
- Generate HTML contents on the fly
- Form validation
- AJAX (e.g. Google complete)
- etc.

First script

```
<html>
<head><title>First JavaScript Page</title></head>
<body>
<h1>First JavaScript Page</h1>
<script type="text/javascript">
  document.write("<hr>");
  document.write("Hello World Wide Web");
  document.write("<hr>");
</script>
</body>
</html>
```


First JavaScript Page - Netscape
File Edit View Go Communicator Help
Document: Done

First JavaScript Page

Hello World Wide Web

JavaScript

- Code written in JavaScript is inserted inside <script> tag

<script type=“text/javascript”> and </script>

- It tells HTML that the following code should not be displayed as text on the screen, but it is actually executing statements.
- If browser cannot execute JavaScript, display of the written code can be forbidden with operator <!-- //-->
- These symbols are ignored by browsers which support JavaScript
- HTML code is executed in the order it was written
- Script code is also executed in the order it was written

Embedding JavaScript

- The scripts inside an HTML document is interpreted in the order they appear in the document.
  - Scripts in a function is interpreted when the function is called.

- So where you place the <script> tag matters.

JavaScript

<script language="javascript"> was used in very old browsers, and is deprecated.

<script type="text/javascript"> is the HTML 4 standard.

In HTML 5, the type parameter is optional (JavaScript is the default), so you can just do <script>.

External JavaScript

- Use the **`src`** attribute to include JavaScript codes from an external file.
- The included code is inserted in place.

```
<html>
<head><title>First JavaScript Program</title></head>
<body>
<script type="text/javascript"
        src="your_source_file.js"></script>
</body>
</html>
```

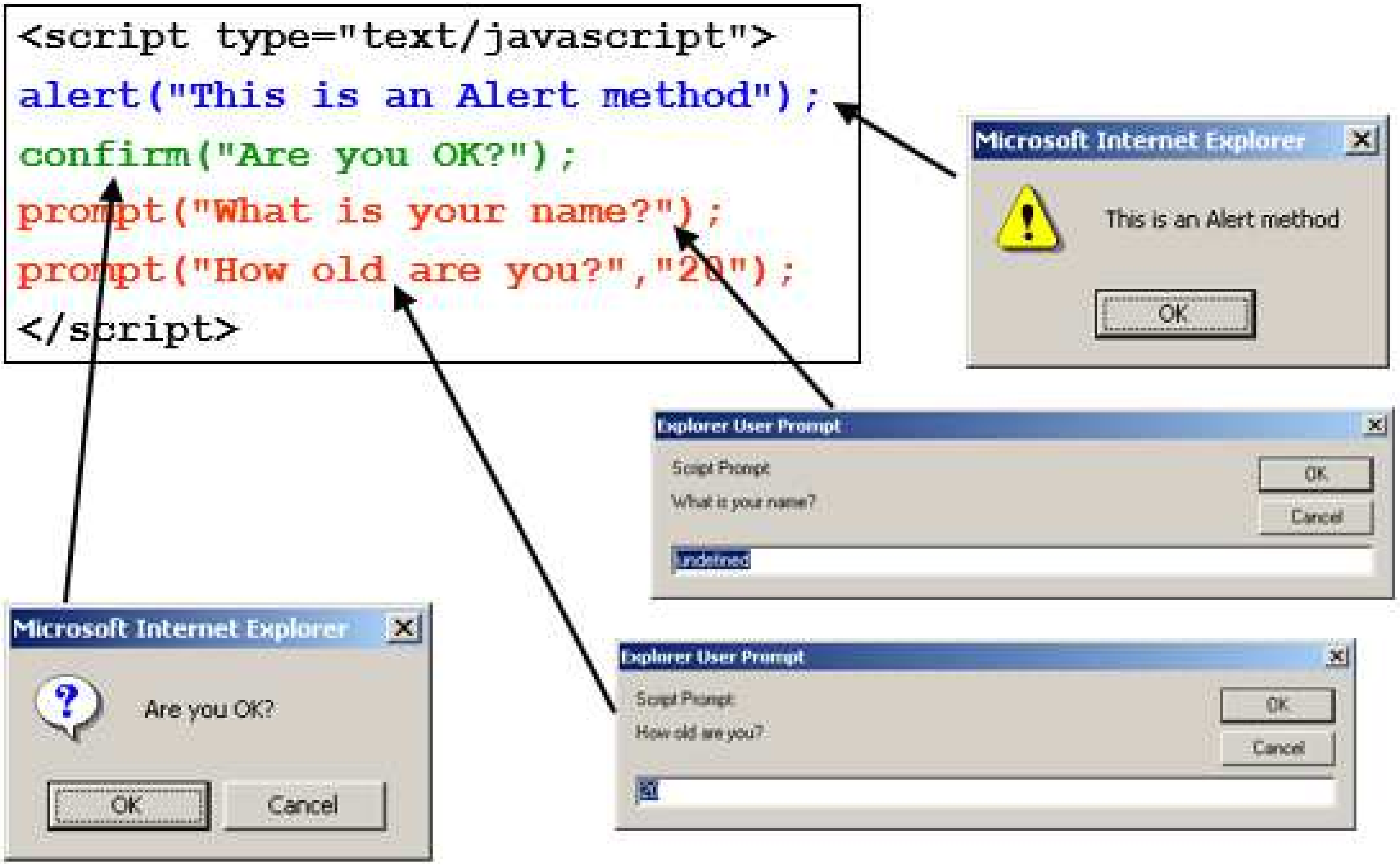
Inside your source file.js

```
document.write("<hr>");
document.write("Hello World Wide Web");
document.write("<hr>");
```

Hide JS from incompatible browsers

```
<script type="text/javascript">
<!-
  document.writeln("Hello, WWW");
// -->
</script>
<noscript>
  Your browser does not support JavaScript.
</noscript>
```

alert(), confirm() and prompt ()



```
<script type="text/javascript">
alert("This is an Alert method");
confirm("Are you OK?");
prompt("What is your name?");
prompt("How old are you?","20");
</script>
```

Microsoft Internet Explorer

This is an Alert method

OK

Explorer User Prompt

Script Prompt

What is your name?

undefined

OK Cancel

Microsoft Internet Explorer

Are you OK?

OK Cancel

Explorer User Prompt

Script Prompt

How old are you?

20

OK Cancel

alert() and confirm()

```
alert("Text to be displayed");
```

- Display a message in a dialog box.
- The dialog box will block the browser.

```
var answer = confirm("Are you sure?");
```

- Display a message in a dialog box with two buttons: "OK" or "Cancel".
- `confirm()` returns `true` if the user click "OK". Otherwise it returns `false`.

prompt()

```
prompt("What is your student id number?");
prompt("What is your name?", "No name");
```

- Display a message and allow the user to enter a value
- The second argument is the "default value" to be displayed in the input textfield.
- Without the default value, "undefined" is shown in the input textfield.

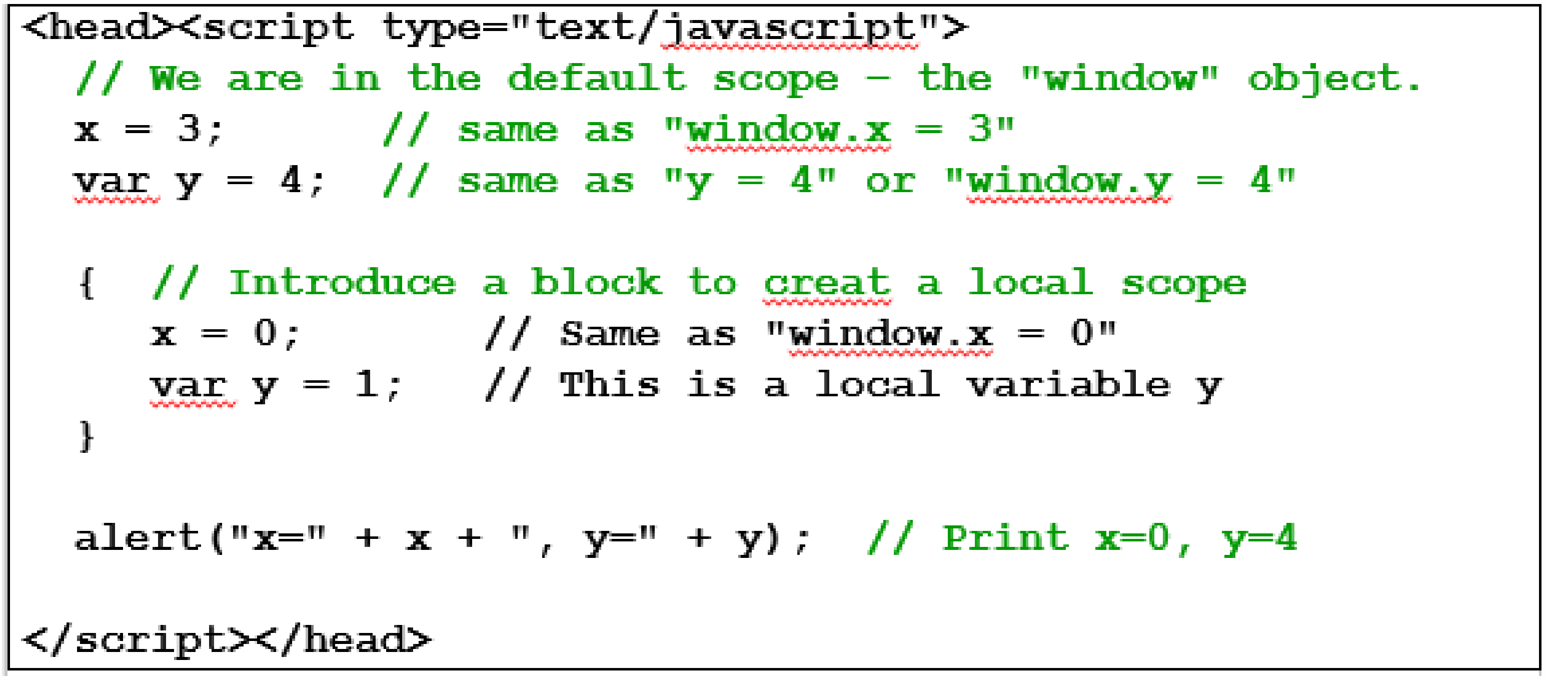
- If the user click the "OK" button, `prompt()` returns the value in the input textfield as a string.
- If the user click the "Cancel" button, `prompt()` returns null.

Identifiers

- Same as Java/C++ except that it allows an additional character – '$'.

- Contains only 'A' – 'Z', 'a' – 'z', '0' – '9', '_', '$'
- First character cannot be a digit
- Case-sensitive
- Cannot be reserved words or keywords

Variables and declaration



```
<head><script type="text/javascript">
  // We are in the default scope - the "window" object.
  x = 3;        // same as "window.x = 3"
  var y = 4;    // same as "y = 4" or "window.y = 4"

  {  // Introduce a block to creat a local scope
     x = 0;          // Same as "window.x = 0"
     var y = 1;      // This is a local variable y
  }

  alert("x=" + x + ", y=" + y);  // Print x=0, y=4

</script></head>
```

- Local variable is declared using the keyword 'var'.
- Dynamic binding – a variable can hold any type of value
- If a variable is used without being declared, the variable is created automatically.
  - If you misspell a variable name, program will still run (but works incorrectly)

Data types in JavaScript

- Primitive data types
  - Number: integer & floating-point numbers
  - Boolean: true or false
  - String: a sequence of alphanumeric characters

- Composite data types (or Complex data types)
  - Object: a named collection of data
  - Array: a sequence of values (an array is actually a predefined object)

- Special data types
  - Null: the only value is "null" – to represent nothing.
  - Undefined: the only value is "undefined" – to represent the value of an non-initialized variable

Strings

- A string variable can store a sequence of alphanumeric characters, spaces and special characters.

- Each character is represented using 16 bit
  - You can store Chinese characters in a string.
- A string can be enclosed by a pair of single quotes (`'`) or double quote (`"`).
- Use escaped character sequence to represent special character (e.g.: `\"`, `\n`, `\t`)

typeof operator

```
var x = "hello", y;
alert("Variable x value is " + typeof x );
alert("Variable y value is " + typeof y );
alert("Variable x value is " + typeof z );
```

- An unary operator that tells the type of its operand.
  - Returns a string which can be "number", "string", "boolean", "object", "function", "undefined", and "null"
  - An array is internally represented as an object.

Object

- An object is a collection of properties.
- Properties can be variables (Fields) or Functions (Methods)
- There is no "Class" in JavaScript.

Array

- An array is represented by the **`Array`** object. To create an array of N elements, you can write

```
var myArray = new Array(N);
```

- Index of array runs from 0 to N-1.
- Can store values of different types
- Property "**`length`**" tells the # of elements in the array.
- Consists of various methods to manipulate its elements. e.g., **`reverse()`**, **`push()`**, **`concat()`**, etc

Array examples

```
var Car = new Array(3);
Car[0] = "Ford";
Car[1] = "Toyota";
Car[2] = "Honda";

// Create an array of three elements with initial
// values
var Car2 = new Array("Ford", "Toyota", "Honda");

// Create an array of three elements with initial
// values
var Car3 = ["Ford", "Toyota", "Honda"];
```

Array examples

```
// An array of 3 elements, each element is undefined
var tmp1 = new Array(3);

// An array of 3 elements with initial values
var tmp2 = new Array(10, 100, -3);

// An array of 3 elements with initial values
// of different types
var tmp3 = new Array(1, "a", true);

// Makes tmp3 an array of 10 elements
tmp3.length = 10; // tmp[3] to tmp[9] are undefined.

// Makes tmp3 an array of 100 elements
tmp3[99] = "Something";
// tmp[3] to tmp[98] are undefined.
```

Type Conversion (to Boolean)

- The following values are treated as false
  - null
  - undefined
  - +0, -0, NaN (numbers)
  - "" (empty string)

Type Conversion

- Converting a value to a number

```
var numberVar = someVariable – 0;
```

- Converting a value to a string

```
var stringVar = someVariable + "";
```

- Converting a value to a boolean

```
var boolVar = !!someVariable;
```

Operators

- Arithmetic operators
  - +, -, *, /, %
- Post/pre increment/decrement
  - ++, --
- Comparison operators
  - ==, !=, >, >=, <, <=
  - ===, !== (Strictly equals and strictly not equals)
    - i.e., Type and value of operand must match / must not match

== VS ===

```
// Type conversion is performed before comparison
var v1 = ("5" == 5);    // true

// No implicit type conversion.
// True if only if both types and values are equal
var v2 = ("5" === 5);   // false

var v3 = (5 === 5.0); // true

var v4 = (true == 1); // true (true is converted to 1)

var v5 = (true == 2); // false (true is converted to 1)

var v6 = (true == "1") // true
```

Univerzitet Singidunum

Logical Operators

- `!` – Logical NOT
- `&&` – Logical AND
  - `OP1 && OP2`
  - If OP1 is true, expression evaluates to the value of OP2. Otherwise the expression evaluates to the value of OP1.
  - Results may not be a boolean value.
- `||` – Logical OR
  - `OP1 || OP2`
  - If OP1 is true, expression evaluates to the value of OP1. Otherwise the expression evaluates to the value of OP2.

Logical operators

```
var tmp1 = null && 1000;        // tmp1 is null

var tmp2 = 1000 && 500;         // tmp2 is 500

var tmp3 = false || 500;        // tmp3 is 500

var tmp4 = "" || null;          // tmp4 is null

var tmp5 = 1000 || false;       // tmp5 is 1000


// If foo is null, undefined, false, zero, NaN,
// or an empty string, then set foo to 100.
foo = foo || 100;
```

Operators

- String concatenation operator
  - `+`
  - If one of the operand is a string, the other operand is automatically converted to its equivalent string value.
- Assignment operators
  - `=, +=, -=, *=, /=, %=`
- Bitwise operators
  - `&, |, ^, >>, <<, >>>`

Conditional Statements

- “if” statement
- “if … else” statement
- "? :" ternary conditional statement
- “switch” statement

- The syntax of these statements are similar to those found in C and Java

Looping statements

- “for” Loops
- “for/in” Loops
- “while” Loops
- “do … while” Loops
- “break” statement
- “continue” statement

- All except "for/in" loop statements have the same syntax as those found in C and Java.

for/in statement

- For each

```
for (var variable in object) {
      statements;
}
```

- To iterate through all the properties in "object".
- "**`variable`**" takes the name of each property in "object"
- Can be used to iterate all the elements in an Array object.

for/in

```
var txt = "";
var person = {fname:"John", lname:"Doe", age:25};
var x;
for (x in person) {
    txt += person[x] + " ";
}
```

The for/in statement loops through the properties of an object.

```
John Doe 25
```

for/in

```
var obj = new Object();  // Creating an object

// Adding three properties to obj
obj.prop1 = 123;
obj.prop2 = "456";
obj["prop3"] = true;    // same as obj.prop3 = true

var keys = "", values = "";
for (var p in obj) {
  keys += p + " ";
  values += obj[p] + " ";
}

alert(keys);
// Show "prop1 prop2 pro3 "

alert(values);
// Show "123 456 true "
```

Functions

- A JavaScript function is a block of code designed to perform a particular task.
- A JavaScript function is executed when "something" invokes it (calls it).
- A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ().
- Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).
- The parentheses may include parameter names separated by commas:
- (parameter1, parameter2, ...)

Functions

- The code to be executed, by the function, is placed inside curly brackets: **{ }**

```
function name(parameter1, parameter2, parameter3) {
    code to be executed
}
```

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Functions

- Function Invocation
- The code inside the function will execute when "something" invokes (calls) the function:
  - When an event occurs (when a user clicks a button)
  - When it is invoked (called) from JavaScript code
  - Automatically (self invoked)

Function

- Function Return
- When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement.
- Functions often compute a return value. The return value is "returned" back to the "caller":

Functions

- Example
- Calculate the product of two numbers, and return the result:

```
var x = myFunction(4, 3);    // Function is called,
                             //return value will end up in x
function myFunction(a, b) {
        return a * b;            // Function returns the
                                 //    product of a and b
}
```

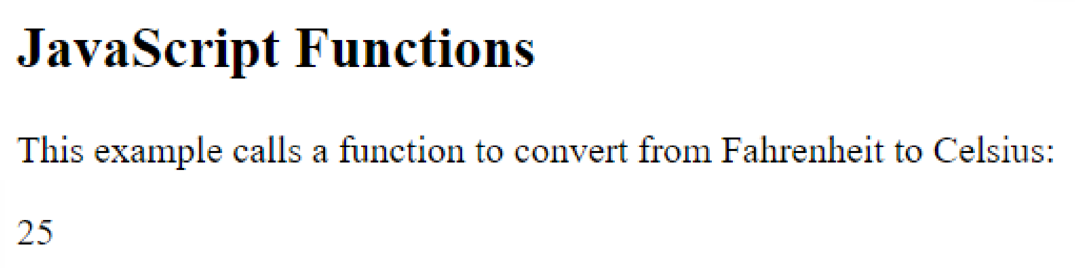
The result in x will be: 12

Functions

- Why Functions?
- You can reuse code: Define the code once, and use it many times.
- You can use the same code many times with different arguments, to produce different results.

Functions

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Functions</h2>

<p>This example calls a function to convert from Fahrenheit to Celsius:</p>
<p id="fhr"></p>

<script>
function toCelsius(f) {
    return (5/9) * (f-32);
}
document.getElementById("fhr").innerHTML = toCelsius(77);
</script>

</body>
</html>
```



JavaScript Functions

This example calls a function to convert from Fahrenheit to Celsius:

25

Univerzitet Singidunum

Functions

```
// A function can return value of any type using the
// keyword "return".

// The same function can possibly return values
// of different types
function foo (p1) {
        if (typeof(p1) == "number")
                return 0;     // Return a number
        else
        if (typeof(p1) == "string")
                return "zero"; // Return a string

        // If no value being explicitly returned
        // "undefined" is returned.
}

foo(1);                 // returns 0
foo("abc");             // returns "zero"
foo();          // returns undefined
```

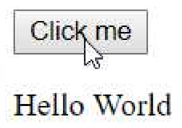
Functions

- Variable Arguments
  - Local variable (an array) available in every function
  - You can access the arguments through parameters or through arguments array

```
function sum ()
{
  var s = 0;
  for (var i = 0; i < arguments.length; i++)
      s += arguments[i];
  return s;
}


sum(1, 2, 3);             // returns 6
sum(1, 2, 3, 4, 5);       // returns 15
sum(1, 2, "3", 4, 5);     // returns ?
```

Functions

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to trigger a function that will output "Hello World" in a p element
with id="demo".</p>

<button onclick="myFunction()">Click me</button>

<p id="demo"></p>

<script>
function myFunction() {
    document.getElementById("demo").innerHTML = "Hello World";
}
</script>

</body>
</html>
```



```
Hello World
```

Built-in Functions

- **`eval(expr)`**
  - evaluates an expression or statement
    - eval("3 + 4");   // Returns 7 (Number)
    - eval("alert('Hello')");   // Calls the function alert('Hello')

- **`isFinite(x)`**
  - Determines if a number is finite

- **`isNaN(x)`**
  - Determines whether a value is “Not a Number”

Univerzitet Singidunum

Built-in Functions

- **`parseInt(s)`**
- **`parseInt(s, radix)`**
  - Converts string literals to integers
  - Parses up to any character that is not part of a valid integer
    - `parseInt("3 chances")             // returns 3`
    - `parseInt("   5 alive")            // returns 5`
    - `parseInt("How are you")           // returns NaN`
    - `parseInt("17", 8)                 // returns 15`

- **`parseFloat(s)`**
  - Finds a floating-point value at the beginning of a string.
    - `parseFloat("3e-1 xyz")            // returns 0.3`
    - `parseFloat("13.5 abc")            // returns 13.5`

Objects

- JavaScript variables are containers for data values.
- This code assigns a **simple value** (Fiat) to a **variable** named car:

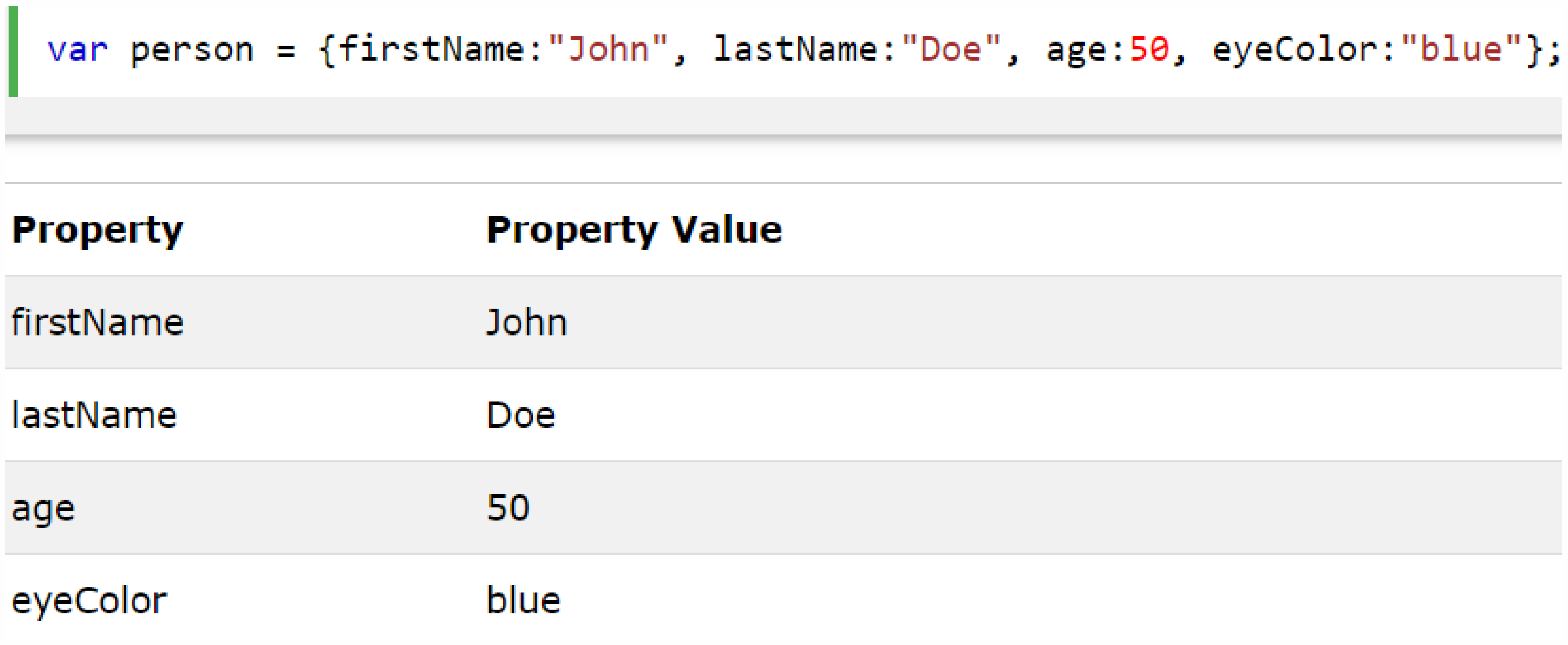
        var car = "Fiat";

- Objects are variables too. But objects can contain many values.
- This code assigns **many values** (Fiat, 500, white) to a **variable** named car:

        var car = {type:"Fiat", model:"500",
color:"white"};

Objects

- Object Properties - The name:values pairs (in JavaScript objects) are called properties.

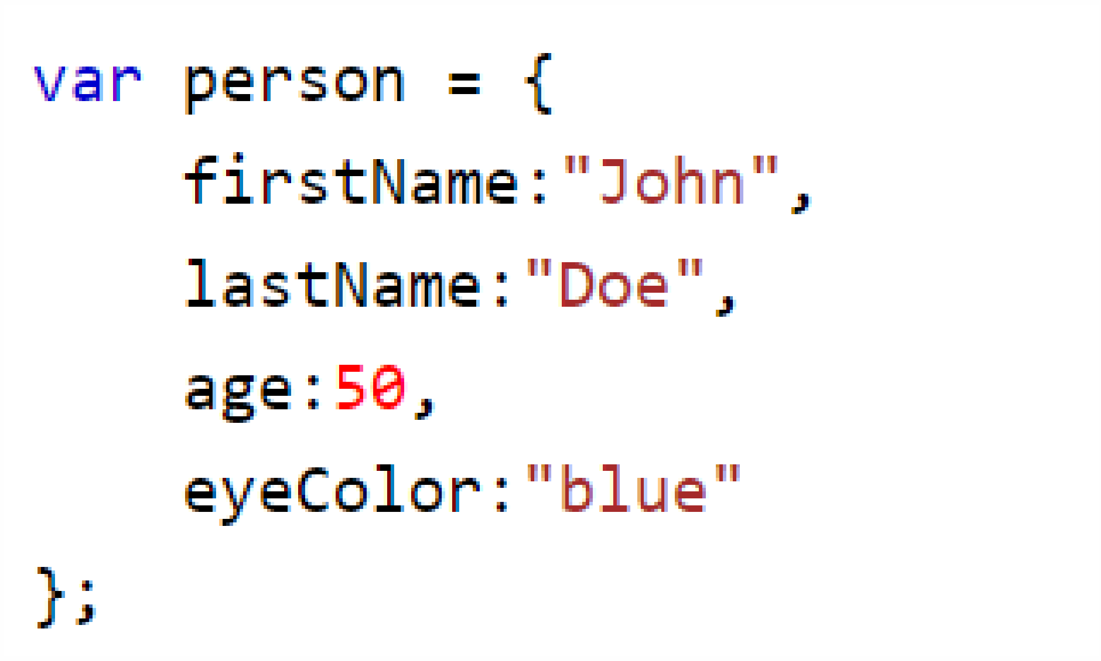


```
var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Property	**Property Value**
firstName	John
lastName	Doe
age	50
eyeColor	blue

Objects

- Definition of object



```
var person = {
    firstName:"John",
    lastName:"Doe",
    age:50,
    eyeColor:"blue"
};
```

Objects

- Adding method to the object

```
var person = {
  // Declare fields
  // (Note: Use comma to separate fields)
  firstName : "John",
  lastName : "Doe",

  // Assign a method to object "person"
  sayHi : function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}

person.sayHi();  // Call the method in "person"
```

Objects

- Nested notation is possible:

```
var triangle = {
  // Declare fields (each as an object of two
fields)
  p1 : { x : 0, y : 3 },
  p2 : { x : 1, y : 4 },
  p3 : { x : 2, y : 5 }
}

alert(triangle.p1.y);   // Show 3
```

Object Construction

```
function Person(fname, lname) {
  // Define and initialize fields
  this.firstName = fname;
  this.lastName = lname;

  // Define a method
  this.sayHi = function() {
    alert("Hi! " + this.firstName + " " +
          this.lastName);
  }
}

var p1 = new Person("John", "Doe");
var p2 = new Person("Jane", "Dow");

p1.sayHi();  // Show "Hi! John Doe"
p2.sayHi();  // Show "Hi! Jane Dow"
```

Events

- An event occurs as a result of some activity
  - e.g.:
    - A user clicks on a link in a page
    - Page finished loaded
    - Mouse cursor enter an area
    - A preset amount of time elapses
    - A form is being submitted

Event Handlers

- Event Handler – a segment of codes (usually a function) to be executed when an event occurs
- We can specify event handlers as attributes in the HTML tags.
- The attribute names typically take the form "`onXXX`" where `XXX` is the event name.
  - e.g.:

```
<a href="…" onClick="alert('Bye')">Other
Website</a>
```

Event Handlers

Event Handlers	Triggered when
onChange	The value of the text field, textarea, or a drop down list is modified
onClick	A link, an image or a form element is clicked once
onDblClick	The element is double-clicked
onMouseDown	The user presses the mouse button
onLoad	A document or an image is loaded
onSubmit	A user submits a form
onReset	The form is reset
onUnLoad	The user closes a document or a frame
onResize	A form is resized by the user

Event Handlers

- Event handlers can be used to handle, and verify, user input, user actions, and browser actions:
- Things that should be done every time a page loads
- Things that should be done when the page is closed
- Action that should be performed when a user clicks a button
- Content that should be verified when a user inputs data
- And more ...

Event Handlers

- Many different methods can be used to let JavaScript work with events:
- HTML event attributes can execute JavaScript code directly
- HTML event attributes can call JavaScript functions
- You can assign your own event handler functions to HTML elements
- You can prevent events from being sent or being handled
- And more ...

Univerzitet Singidunum

onClick Event

```
<html>
<head>
<title>onClick Event Handler Example</title>
<script type="text/javascript">
function warnUser() {
      return confirm("Are you a student?”);
}
</script>
</head>
<body>
<a href="ref.html" onClick="return warnUser()">
<!--
  If onClick event handler returns false, the link
  is not followed.
-->
Students access only</a>
</body>
</html>
```

onLoad Event Handler

```
<html><head>
<title>onLoad and onUnload Event Handler
Example</title>
</head>
<body
  onLoad="alert('Welcome to this page')"
  onUnload="alert('Thanks for visiting this page')"
>
Load and UnLoad event test.
</body>
</html>
```

onMouseOver and onMouseOut

```
<html>
<head>
<title>onMouseOver / onMouseOut Event Handler Demo</title>
</head>
<body>
<a href="https://singidunum.ac.rs"
   onMouseOver="window.status='Singidunum Home'; return
true;"
   onMouseOut="status=''"
>Singidunum</a>
</body>
</html>
```

•When the mouse cursor is over the link, the browser displays the text "CUHK Home" instead of the URL.

•The "return true;" of `onMouseOver` forces browser not to display the URL.

onSubmit EventHandler

```
<html><head>
<title>onSubmit Event Handler Example</title>
<script type="text/javascript">
  function validate() {
    // If everything is ok, return true
    // Otherwise return false
  }
</script>
</head>
<body>
<form action="MessageBoard" method="POST"
 onSubmit="return validate();"
>
…
</form></body></html>
```

•If `onSubmit` event handler returns false, data is not submitted.

•If `onReset` event handler returns false, form is not reset

Built-in JavaScript Objects

Object	Description
Array	Creates new array objects
Boolean	Creates new Boolean objects
Date	Retrieves and manipulates dates and times
Error	Returns run-time error information
Function	Creates new function objects
Math	Contains methods and properties for performing mathematical calculations
Number	Contains methods and properties for manipulating numbers.
String	Contains methods and properties for manipulating text strings

Univerzitet Singidunum

String Object (useful methods)

- length
  - A string property that tells the number of character in the string
- charAt(idx)
  - Returns the character at location "idx"
- toUpperCase(), toLowerCase()
  - Returns the same string with all uppercase/lowercase letters
- substring(beginIdx)
  - Returns a substring started at location "beginIdx"
- substring(beginIdx, endIdx)
  - Returns a substring started at "beginIdx" until "endIdx" (but not including "endIdx"
- indexOf(str)
  - Returns the position where "str" first occurs in the string

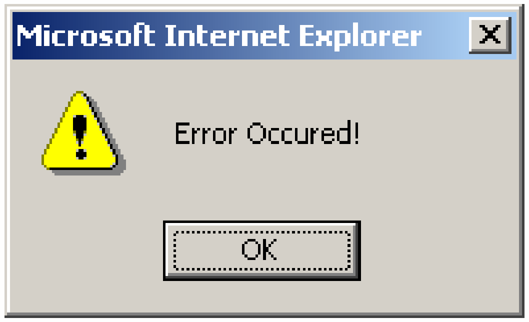
Error and Exception Handling

- Javascript makes no distinction between Error and Exception (Unlike Java)
- Handling Exceptions
  - The onError event handler
    - A method associated with the window object.
    - It is called whenever an exception occurs
  - The try … catch … finally block
    - Similar to Java try … catch … finally block
    - For handling exceptions in a code segment
  - Use throw statement to throw an exception
    - You can throw value of any type
  - The Error object
    - Default object for representing an exception
    - Each Error object has a name and message properties

Univerzitet Singidunum

onError event handler

```
<html>
<head>
<title>onerror event handler example</title>
<script type="text/javascript">
function errorHandler(){
  alert("Error Ourred!");
}
// JavaScript is casesensitive
// Don't write onerror!
window.onError = errorHandler;
</script>
</head>
<body>
<script type="text/javascript">
  document.write("Hello there;
</script>
</body>
</html>
```



Error Occured!

Try catch finally

```
try {
  // Contains normal codes that might throw an exception.

  // If an exception is thrown, immediately go to
  //   catch block.

} catch ( errorVariable ) {
  // Codes here get executed if an exception is thrown
  //   in the try block.

  // The errorVariable is an Error object.

} finally {
  // Executed after the catch or try block finish

  // Codes in finally block are always executed
}
// One or both of catch and finally blocks must accompany
the try block.
```

Try catch finally

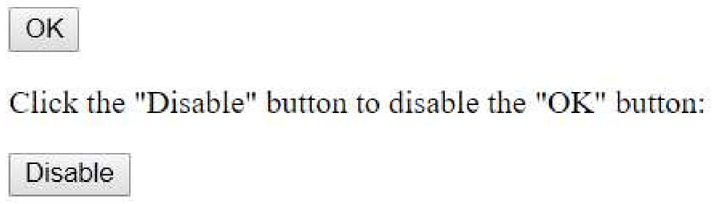
```
<script type="text/javascript">
try{
  document.write("Try block begins<br>");
  // create a syntax error
  eval ("10 + * 5");

} catch( errVar ) {
  document.write("Exception caught<br>");
  // errVar is an Error object
  // All Error objects have a name and message properties
  document.write("Error name: " + errVar.name + "<br>");
  document.write("Error message: " + errVar.message +
                "<br>");
} finally {
  document.write("Finally block reached!");
}
</script>
```

Throwing Exception

```
<script type="text/javascript">
try{
  var num = prompt("Enter a number (1-2):", "1");
  // You can throw exception of any type
  if (num == "1")
    throw "Some Error Message";
  else
  if (num == "2")
    throw 123;
  else
    throw new Error ("Invalid input");
} catch( err ) {
  alert(typeof(errMsg) + "\n" + err);
  // instanceof operator checks if err is an Error object
  if (err instanceof Error)
    alert("Error Message: " + err.message);
}
</script>
```

Working with forms

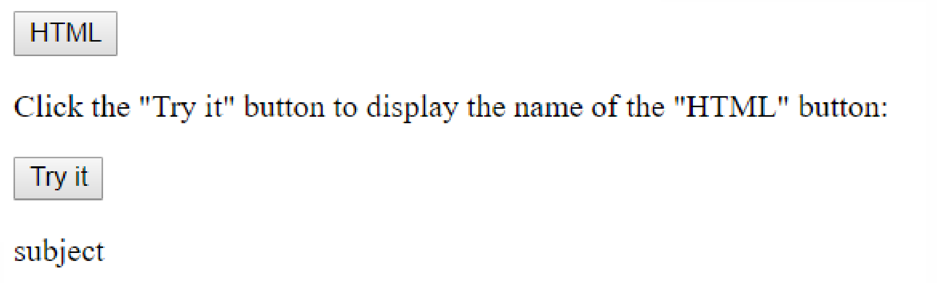
```
<html>
<body>

<form>
<input type="button" id="btn01" value="OK">
</form>

<p>Click the "Disable" button to disable the "OK" button:</p>

<button onclick="disableElement()">Disable</button>

<script>
function disableElement() {
    document.getElementById("btn01").disabled = true;
}
</script>

</body>
</html>
```



Click the "Disable" button to disable the "OK" button:

Get the name of the button

```
<!DOCTYPE html>
<html>
<body>

<form id="frm1" action="/action_page.php">
<button id="btn1" name="subject" type="submit" value="HTML">HTML</button>
</form>

<p>Click the "Try it" button to display the name of the "HTML" button:</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
    var x = document.getElementById("btn1").name;
    document.getElementById("demo").innerHTML = x;
}
</script>

</body>
</html>
```

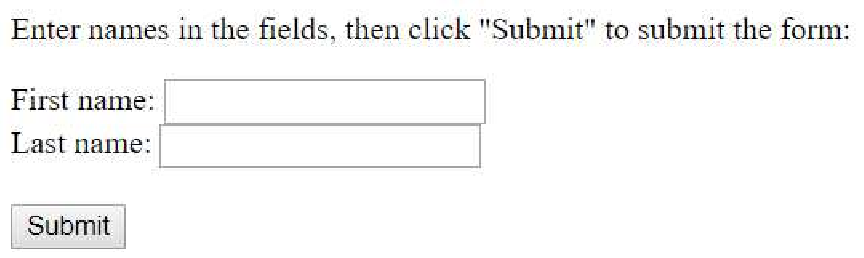


Click the "Try it" button to display the name of the "HTML" button:

subject

Submitting the form

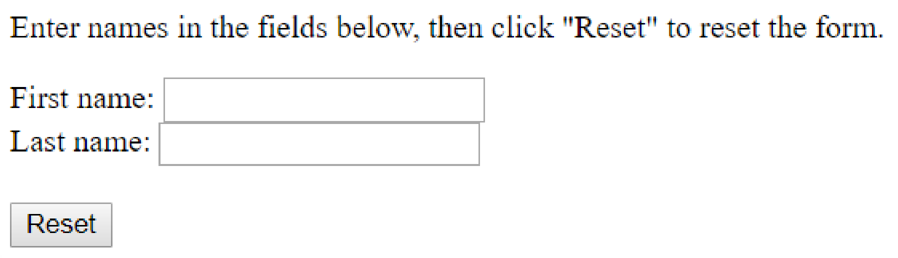
```
<!DOCTYPE html>
<html>
<body>

<p>Enter names in the fields, then click "Submit" to submit the form:</p>

<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br><br>
  <input type="button" onclick="myFunction()" value="Submit">
</form>

<script>
function myFunction() {
    document.getElementById("frm1").submit();
}
</script>

</body>
</html>
```


Enter names in the fields, then click "Submit" to submit the form:

First name:
Last name:

Reset the form with JS

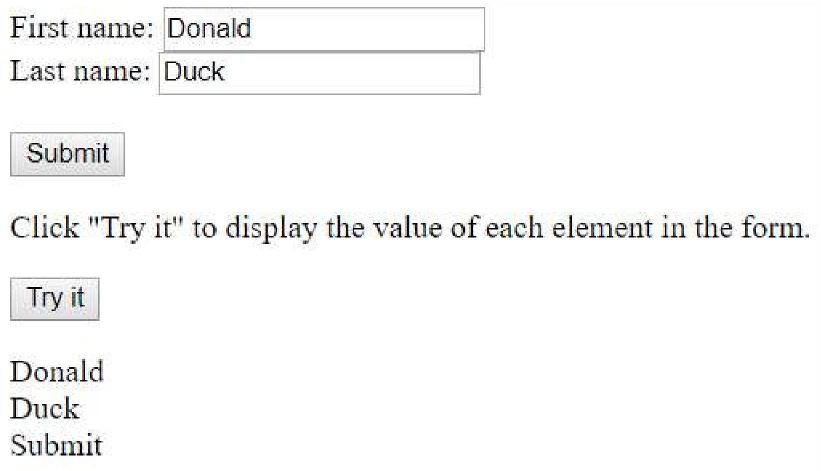
```
<!DOCTYPE html>
<html>
<body>

<p>Enter names in the fields below, then click "Reset" to reset the form.</p>

<form id="frm1">
  First name: <input type="text" name="fname"><br>
  Last name: <input type="text" name="lname"><br><br>
  <input type="button" onclick="myFunction()" value="Reset">
</form>

<script>
function myFunction() {
    document.getElementById("frm1").reset();
}
</script>

</body>
</html>
```


Enter names in the fields below, then click "Reset" to reset the form.

First name:
Last name:

Find the value of each form element

```
<!DOCTYPE html>
<html>
<body>

<form id="frm1" action="/action_page.php">
  First name: <input type="text" name="fname" value="Donald"><br>
  Last name: <input type="text" name="lname" value="Duck"><br><br>
  <input type="submit" value="Submit">
</form>

<p>Click "Try it" to display the value of each element in the form.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
    var x = document.getElementById("frm1");
    var text = "";
    var i;
    for (i = 0; i < x.length ;i++) {
        text += x.elements[i].value + "<br>";
    }
    document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```


First name: Donald
Last name: Duck

Submit

Click "Try it" to display the value of each element in the form.

Try it

Donald
Duck
Submit