The Javascript language

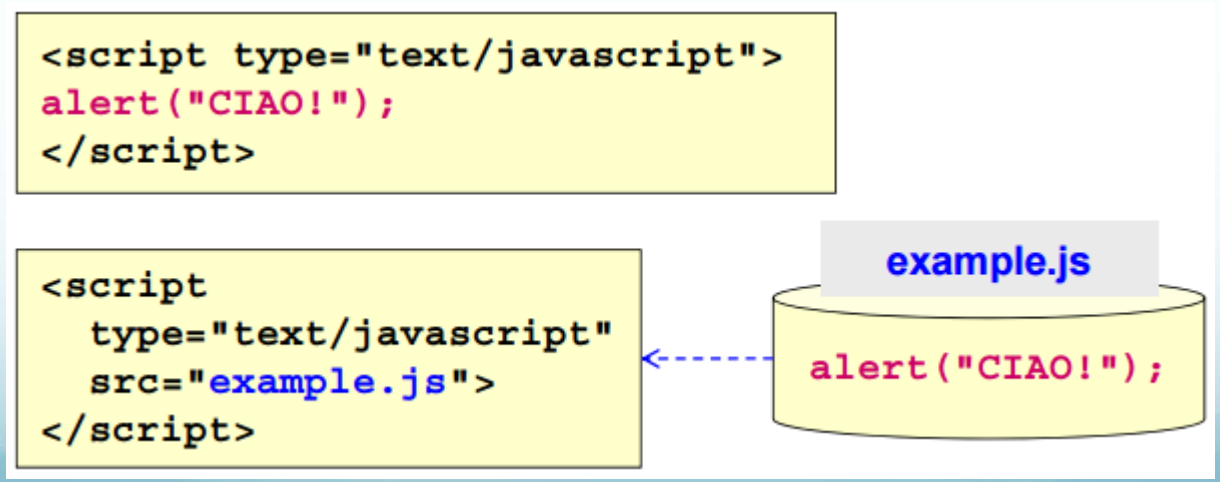
Table of content

Javascript

- Language designed for the web (now used also in other contexts, e.g. Adobe PDF)
- Can be executed on client side or server side
- inserted inside HTML pages via
  - *the <script> tag inside the head or the body*
  - *instructions specified for an event-handle*
- executed when the browser, parsing the HTML code, finds the Javascript code (temporary code, finds the Javascript code (temporary suspension of the HTML interpreter)

The <script> tag

- two main parameters:
  - type="text/javascript"
  - src="URI" (p p) the script is located at the specified URI)
- Examples:



Syntax

- Javascript is case-sensitive
- every instruction is terminated by a semi-column ;
- data typ p g es and corresponding values:
  - decimal numbers (e.g. 14, -7, 3.14, 10.7e-4)
  - Boolean values (true false)
  - character strings, enclosed inside apices double or single
  - Objects
  - the special value null (uninitialized variable)
  - the special value undefined (undefined variable)

HTTP

- C++ style: from "//" to the end of the line
- C style: all the text enclosed between "/*" and "*/"
- Examples:

```
<script type="text/javascript">
// comment spanning a single line in C++ style
/*
comment spanning four lines
in C style
*/
</script>
```

Variables

- Identified by their name:
  - must start with an alphabetic character, $, or _
  - can then contain alp , $, _ hanumeric characters, $, and
  - e.g. cost, $1, student_12345, _2009q1
- not typed (contrary to Java, C, …) but:
  - acquire a type upon initialization
  - can change the type (automatically) to adapt to the context in which they are used context in which they are used

Creating a new variable

- explicit creation by the instruction "var" (with or without an initial value):
  - var total;
  - var total = 0;
  - var greetings = "ciao mum!"
- implicit creation by assigning a value to the variable:
  - total = 0;
- when using a variable before assigning a value:
  - undefined / NaN (if declared with "var")
  - runtime error (if undeclared)

Constants

- Variables with fixed values can be created by the instruction "const":
  - const iva = 0.19;
  - const author = “John“;
- Inside a string you can use:
  - Unicode characters
  - escape sequences
    - \r \n \t \' \" \\
  - attention to the usage context (if the string is used inside HTML, it may require encoding, e.g. `)

Input and output

- JS is a scripting language, designed to be executed on client side (inside a browser) or on server side
- the I/O functions depend on the execution environment
- on the client side, you can use:
  - for input:
    - data from an input pop-up window
    - data from a form (via DOM) data from a form (via DOM)
  - for output:
    - an output pop-up window
    - the HTML page (while it is “constructed" or via DOM)

I/O pop-ups

- window.alert(message)
  - opens a blocking pop-up containing the message and a button to confirm reading (OK)
- window.prompt(prompt [, initial_value])
  - opens a blocking pop-up containing the prompt text, an input field (empty or with the initial value) and two buttons to confirm the inserted response (OK), or to refuse to provide it (Cancel)
  - returns the inserted value, or null in case the user pushes Cancel or closes the pop-up
- formally, these are JS implementations of the window object (DOM level 0) and associated methods

I/O pop-ups: example 1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Pop-up di I/O: example 1</title>
  <script type="text/javascript">
    var n = window.prompt("Name?", “nobody")
    window.alert(n);
  </script>
</head>
<body>
  <p>End of the example.</p>
</body>
</html>
```

I/O pop-ups: example 2

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Pop-up di I/O: example 2</title>
</head>
<body>
  <p>Starting the example.</p>
  <script type="text/javascript">
    var n = window.prompt("Name?", "nobody")
    window.alert("Ciao "+n);
  </script>
  <p>End of the example.</p>
</body>
</html>
```

Output via HTML

- you can use the DOM object "document" with one of the following methods:
  - write(text) – insert the text
  - writeln(text) – insert the text followed by CR LF
- generically known as DHTML:
  - HTML 4.0 or greater
  - CSS (Cascaded Style Sheet)
  - client side scripting languages

Output via HTML: example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Output via HTML: example</title>
</head>
<body>
  <script type="text/javascript">
    var n = window.prompt("Name?", "nobody");
    document.writeln("<p>Ciao "+n+"</p>");
  </script>
</body>
</html>
```

Relational and logical operators

description	*symbol*
equality (value)	`==`
identity, strict equality (value and type)	`===`
inequality (value)	`!=`
strict inequality (value and type)	`!==`
greater than / grater or equal	`>` `>=`
less than / less or equal	`<` `<=`
is part of	`in`
logical AND	`&&`
logical NOT	`!`
logical OR	`||`

Boolean operators and values

- the following values are equivalent to False:
  - false
  - 0
  - NaN
  - the empty string ""
  - null
  - undefined
- all other values are equivalent to True
- thus pay attention to comparisons:
  - (27 == true) returns the value True
  - (27 === true) returns the value False

Arithmetic operators

description	*symbol*
addition	+
unitary increment	++
subtraction	–
unitary decrement	––
multiplication	*
division (floating-point)	/
modulus (rest of integer division)	%

Assignment operators

description	*symbol*	*example*	*equivalence*
assignment	=	a = 5	
sum and assign.	+=	a += 5	a = a + 5
sub and assign.	-=	a -= 5	a = a – 5
mul and assign.	*=	a *= 5	a = a * 5
div and assign.	/=	a /= 5	a = a / 5
mod and assign.	%=	a %= 5	a = a % 5

Character strings

- specially relevant since every input provided by the
- user via the browser is a string
- operators:
  - assignment (=)
  - alphabetical order comparison (== != > >= < <=)
  - concatenation (+ +=)
- attention! if an instruction includes strings,
- numbers and the symbol numbers and the symbol +, everything is treated as string; the use of parentheses is thus necessary

```
ris = "N=" + 5 + 2;     // ris = "N=52"
ris = "N=" + (5 + 2);   // ris = "N=7"
ris = "N=" + 5 - 2;     // ris = NaN
```

Conversion strings – numbers (I)

- Number(object)
  - returns a numerical representation of the object or NaN
- String(object)
  - returns a character string representation of the object or undefined or null

```
n = Number("2");          // n = 2
n = Number("2.3");        // n = 2.3
n = Number("2,3");        // n = NaN
n = Number("2thousands");  // n = NaN
n = Number("2 thousands"); // n = NaN
```

Conversion strings – numbers (II)

- parseInt(string [, base])
  - returns an integer number or NaN
  - can spy ( ) ecify the base (default: 10)
- parseFloat(string)
  - returns a floating-point number or NaN
- parseInt and parseFloat consider only the initial part, stopping at the first invalid character

```
n = parseInt("10",2);     // n = 2
n = parseInt("2.3");      // n = 2
n = parseInt("2,3");      // n = 2
n = parseInt("2 thousands");    // n = 2
n = parseInt("twothousands"); // n = NaN
```

Properties and methods of String object (I)

- lastIndexOf(searchString [, start])
  - position of the last occurrence of the search string (backward starting from start or from the end)
  - -1 if the search string is not found
- slice(begin [, term])
  - creates a new string with the characters enclosed between begin and term (excluded) or the end
  - use negative term to specify positions from the end substring(begin [, term])
- extracts characters from begin to term (or the end)
- substr(begin [, length])
  - extracts length characters starting from begin

Properties and methods of String object (III)

- toLowerCase( )
  - returns the characters converted to lowercase
- toUpp ( ) erCase( )
  - returns the characters converted to uppercase

Testing error values

- comparisons with NaN or other limit values are forbidden, but you can use functions to test these cases
- isFinite(number)
  - true if the number is not equal to +/– infinite or NaN
- isNaN(number)
  - true is the number has the value NaN
- typeof(x)
  - returns a string representing the data type currently associated with X
  - possible responses: Boolean, function, number, object, string, undefined

Flow control

- control structures to alter the normal sequential flow of a JS program
  - if
  - if/else
  - while
  - do/while
  - for
  - for/in

Flow control: "if" / "if-else"

- conditional execution of instructions depending on the value of a Boolean condition
  - if
  - if/else

```
if ( condition )
{
   ... instructions
}
```

```
if ( condition )
{
  ... instructions
}
else
{
  ... instructions
}
```

Example of "if-else" construct

```
<script type="text/javascript">
var t_mis = window.prompt(“Measured temperature?");
if (t_mis <= 0)
  alert(“water is frozen");
else if (t_mis >= 100)
  alert(“water is vapor");
else
  alert(“water is in liquid state");
</script>
```

Multiple selection: the "switch" instruction

- abbreviated form of a cascade of "if-else"
- use "break" to avoid continuing to the next case
- "default" when no explicit case is matched

```
switch ( expression )
{
case value1: ... instructions;
  break;
case value2: ... instructions;
  break;
...
default: ... instructions;
}
```

Example of "switch" construct

```
<script type="text/javascript">
var fruit = window.prompt("Which fruit ?");
switch (fruit) {
case "peach":
  alert ("peaches are 2 Euro/kg"); break;
case "apple":
  alert (“apples are 1.5 Euro/kg"); break;
case "banana":
  alert ("bananas a 1 Euro/kg"); break;
default:
  alert ("sorry, we do not have any "+fruit);
}
</script>
```

Flow control: "while"

- construct to repeat a block of instructions as long as a condition remains true
- therefore the instructions in the cycle may be executed i zero or more times

```
while ( condition )
{
   ... instructions
}
```

Example of "while" cycle

```
<script type="text/javascript">
// countdown
var x = 5;
while (x >= 0)
{
    alert(x);
    x--;
}
</script>
```

Flow control: "do-while"

- construct similar to while, with the difference that the condition is tested at the end of the cycle
- therefore the cycle is always executed at least one time

```
do
{
   ... instructions
} while ( condition );
```

Example of "do-while" cycle

```
<script type="text/javascript">
  var ris;
  do {
    ris = window.prompt(
      "Write 'ciao' or remain blocked");
  } while (ris != "ciao");
</script>
```

Flow control: "for"

- construct to repeat a clock of instructions as long as a conditions remains true
- specifies:
  - an initialization action
  - a condition
  - an action to repeat at the end of every cycle (typically the increment/decrement of the index associated with the cycle)

```
for ( initialisation ; condition; repeated_action )
{
  ... istructions to be repeated
}
```

Example of numerical "for" cycle

```
<script type="text/javascript">
/*
computing the sum of the first 10 naturals
*/
var total = 0;
for (var i=1; i <= 10; i++)
{
  total = total + i;
}
alert("Sum of [1...10] = "+total);
</script>
```

Instructions "break" and "continue"

- the instruction "break" (in addition to be used with switch) terminates the execution of the cycle containing it; the execution continues with the first instruction following the cycle instruction
- the instruction "continue" terminates the execution of the current step and immediately starts the execution of the next step of the cycle containing it

Arrays (vectors)

- in JS they are objects and they must be instantiated (with an initial size that can be changed dynamically)
- can be indexed by an integer or a string (associative arrays)
- have properties and methods to insert, delete and retrieve the elements
- example:

```
var Vector = new Array(10);
for (var i=0; i<10; i++) {
     Vector[i] = "Test " + i;
}
```

Example of array with numeric index

```
<script type="text/javascript">

// array for converting
// from Italian to European grade
var it2eu = new Array(32) // 0 ... 30 30L
for (var i=0; i<18; i++) it2eu[i] = "D"
for (var i=18; i<24; i++) it2eu[i] = "C"
for (var i=24; i<29; i++) it2eu[i] = "B"
for (var i=29; i<=31; i++) it2eu[i] = "A"

var grade = prompt("Italian grade?")
alert("European grade = " + it2eu[grade])

</script>
```

Example of array with string index

```
<script type="text/javascript">

// vocabulary Italian - English
var vocab = new Array()
vocab["giallo"] = "yellow"
vocab["rosso"] = "red"
vocab["verde"] = "green"

var color = prompt("Colore?")
alert( color + " = " + vocab[color] )

</script>
```

Example of array with string index

```
<script type="text/javascript">

// vocabulary Italian - English
var vocab = new Array()
vocab["giallo"] = "yellow"
vocab["rosso"] = "red"
vocab["verde"] = "green"

var color = prompt("Colore?")
if (typeof(vocab[color]) != "undefined")
  alert( color + " = " + vocab[color] )
else
  alert("Sorry, translation not available")
</script>
```

Flow control: "for-in"

- iterates over the elements of an array (without knowing its length)
  - the index takes the numerical values (0 ... length-1)
- iterates over the properties of an object (without knowing the names of the properties)
  - the index takes all the names of the properties, in the order they are declared

```
for (x in array) {
  ... array[x] ...
};
```

```
for (x in object) {
  ... object[x] ...
}
```

Example of "for-in" with arrays

```
var vector = new Array(10);

// explicit cycle on all the elements
for (var i=0; i<vector.length(); i++)
  vector[i] = "test"+i;

// explicit cycle, but more efficient
for (var i=0, n=vector.length(); i<n; i++)
  vector[i] += "!";

// implicit cycle on all the elements
for (var i in vector)
  document.writeln(vector[i]+"<br>");
```

Example of "for-in" with objects

```
var myObject = new Object();
myObject.name = "Antonio";
myObject.age = 24;
myObject.phone = "5551234";

...

// implicit cycle on all the properties
for (var prop in myObject)
{
  document.writeln("<p>myObject." + prop
     + " = " + myObject[prop] + "</p>");
}
```

Functions

- the data passed to functions is called parameters
- the parameters are specified enclosed within parentheses after the name of the function
- among the instructions, you can use "return" to terminate the execution of the function, possibly returning a value via "return(value)"

```
function function_name (par1, par2, ...)
{
   ... instructions ...
}
```

Examples of functions

```
function sum (a, b) { return(a+b); }

document.write(sum(1,2));
```

```
function lessThan (a, b) {
  if (a < b) return (true) else return (false);
}

var a=5;
var b=2;
if (!lessThan(a,b))
  document.write(a + "is not less than " + b);
```

Local and global variables

- variables declared inside a function:
  - accessible only to the instructions of the function itself
  - automatically destroyed at function termination
- variables declared outside any function:
  - accessible to every instruction of the script (including those inside the functions called by the script)
  - automatically destroyed at script termination

Local and global variables: example

```
<script type="text/javascript">
  var i=2; // global variable
  function print_var()
  {
    var j=4; // variable local to print_var()
    alert("print_var(): i="+i);
    alert("print_var(): j="+j);
  }
  print_var();
  alert("i="+i);
  alert("j="+j );
 </script>
```

Functions and parameters

- a function can be called with less parameters than the defined ones
- the missing parameters are undefined, but (only in this case) their usage does not rise an error
    - they have an undefined value which is propagated if used (e.g. generates NaN within an arithmetic computation)
- a function can be called with more parameters than the defined ones the defined ones
    - the exceeding parameters are ignored
- a function can access all its parameters via the array arguments[ ] that contains arguments.length
- distinct values

Functions' variable arguments: example

```
<script type="text/javascript">
  function average()
  // computes the arithmetic average of
  // all the numbers passed as parameters
  {
    var total = 0;
    var n = arguments.length;
    for (var i=0; i<n; i++)
      total += arguments[i];
    return (total / n);
  }
  // usage example (average of three numbers)
  alert( average(11,12,16) );
</script>
```

The Date object

- new Date( )
  - current date & time (on the system running script)
- new Date ("Month day, year [HH:MM:SS]")
  - specified date&time (eg. "March 25, 2009 22:00:07")
- new Date(YYYY, MM, DD [, HH, MM, SS])
  - specified date & time (e.g. 2009, 2, 25, 22, 00, 07)
  - month number: 0=January, 11=December
- if HH, MM or SS are omitted, they are zero
- attention! the string representation of the date
- depends on the O.S. where the script is run
  - thus it is better to individually set the values of the properties (with setDay(), setHour(), ...)

The Date object: methods (I)

- getDay( ) / setDay(week_day)
  - weekday (0=Sunday, 6=Saturday)
- g () (etDate( ) / setDate(month da _ y)
  - getMonth( ) / setMonth (month_num)
- month number: 0=January, 11=December
- getFullYear( ) / setFullYear (year)
- getHours( ) / setHours (hour)
- getMinutes( ) / setMinutes(minutes)
- getSeconds( ) / setSeconds(seconds)

The Date object: methods(II)

- all methods are also available referred to UTC:
  - getUTC...( ) / setUTC...( )
- toString( )
  - converts to a string in the native JS format (i.e.Anglo-Saxon)
- toLocaleString( )
  - converts to a string in the local format configured for the user executing the script

The Math object – properties

- static properties (to be invoked on the Math object):
  - E, Euler's constant (approximately 2.718)
  - LN2, ( ) (pp y) ln(2) (approximately 0.693)
  - LN10, ln(10) (approximately 2.302)
  - LOG2E, log2(e) (approximately 1.442)
  - LOG10E, log10(e) (approximately 0.434)
  - PI, circumference / diameter ratio (approximately 3.14159)
  - SQRT1_2, square root of 1/2 (approximately 0.707)
  - SQRT2, square root of 2 (approximately 1.414)

The Math object– methods

- static methods (to be invoked on the Math object)

method	*definition*
`abs(x)`	absolute value
`asin(x)` `acos(x)` `atan(x)`	
`atan2(y,x)`	atan (y / x)
`sin(x)` `cos(x)` `tan(x)`	
`ceil(x)` `floor(x)`	ceiling, floor
`exp(x)` `pow(x,y)` `sqrt(x)`	e^x x^y $\sqrt{x}$
`log(x)`	ln(x)
`round(x)`	rounding
`max(x,y)` `min(x,y)`	
`random(x)`	random value [0...1[

The Number object – properties

- static properties (to be invoked on the Number object):
  - MAX_VALUE
  - MIN_VALUE
  - NaN
  - NEGATIVE_INFINITY (negative overflow)
  - POSITIVE_INFINITY (positive overflow)

The Number object – methods

- tofixed(num_digits_after_radix_char)
  - converts to fixed-point notation
- toexp (onential(num di _ gits after radix char __ _)
  - converts to exponential notation
- toprecision(num_significative_digits)
  - converts to the specified precision (using the exponential format, if needed)
- all these methods round the result if the original all these methods round the result if the original number contains more digits than what necessary:

```
var num=5.126
alert(num.toprecision(3)) // output is 5.13
alert(num.toprecision(2)) // output is 5.1
```

Thank you