JavaScript

Francesco Marcelloni

and Alessio Vecchio

Dipartimento di Ingegneria dell'Informazione
Università di Pisa
ITALY

JavaScript Standard

- JavaScript was invented by Netscape and was first used in Netscape browsers.
- Now, all the browsers support JavaScript
- ECMAScript, standardized version of JavaScript, is documented in the ECMA-262 specification (several editions released).
- ECMA (European Computer Manufacturers Association). ECMA is an international standards association for information and communication systems.
- The ECMA-262 standard is also approved by the ISO (International Organization for Standards) as ISO-16262.

How to insert a JavaScript into an HTML page?

- Use the <script> tag. Inside the <script> tag use the type attribute to define the scripting language.

```
<html>
<body>
<script type="text/javascript">
...
</script>
</body>
</html>
```

- The script element may appear any number of times in the head or body of an HTML document.

How to insert a JavaScript into an HTML page?

- <script> tag
  - src = uri

    This attribute specifies the location of an external script.
  - type = content-type

    This attribute specifies the scripting language of the element's contents and overrides the default scripting language.
    - The scripting language is specified as a content type (e.g., "text/javascript").
    - In HTML5 javascript is the default type and the attribute **can be omitted**.

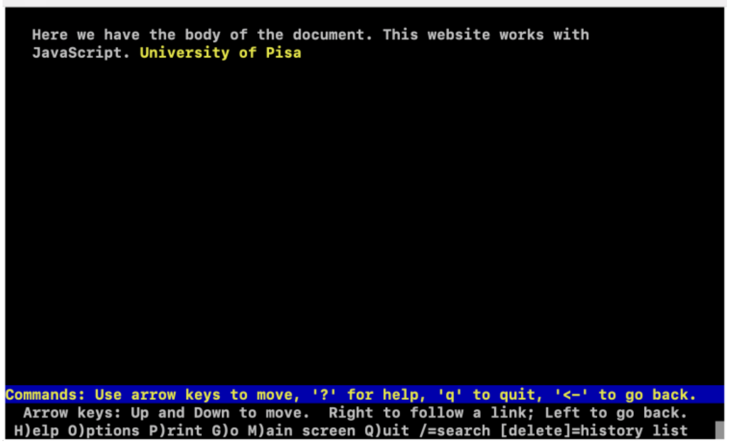
How to insert a JavaScript into an HTML page?

- Example
  - The document.write command is a standard JavaScript command for writing output to a page.
  - By entering the document.write command between the <script> and </script> tags, the browser will recognize it as a JavaScript command and execute the code line.

```
<html>
<body>
<script>
  document.write("Hello World!");
</script>
</body>
</html>
```

How to handle simple browsers?

- Browsers that do not support JavaScript.
- HTML element <noscript> should be used.



```
Here we have the body of the document. This website works with
JavaScript. University of Pisa
```

```
Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.
  Arrow keys: Up and Down to move.  Right to follow a link; Left to go back.
 H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

```
<!DOCTYPE html>
<html lang=en>
  <head>
    <title>Example about no script</title>
    <script>
      alert("JS");
    </script>
  </head>
  <body>
    Here we have the body of the document.
    <noscript>
      This website works with JavaScript.
      <a href="http://www.unipi.it">University of Pisa</a>
    </noscript>
  </body>
</html>
```

The noscript element

- The **noscript** element allows authors to provide alternate content when a script is not executed.
- Its content should only be rendered if:
  - The user agent is configured not to evaluate scripts.
  - The user agent does not support the scripting language.
- Example:

```
<noscript>
      <meta http-equiv="Refresh" content="2";
url="paginasenzaJavaScript.html">
</noscript>
```

- If the noscript element is rendered, the user is re-addressed after 2 seconds to the url specified in the noscript element

Where to insert the JavaScript Code

Where to put the JavaScript?
Head solution

- When in the head, JavaScripts in a page will be executed immediately while the page loads into the browser.
- To prevent the automatic execution, code has to be inserted into a function.
- Functions can be put in the head section, thus they do not interfere with page content

```
<html>
<head>
<script>
function message() {
  alert("This alert box was called   with the onload event");
}
</script>
</head>
<body onload="message()">
</body>
</html>
```

Where to put the JavaScript?
Body Solution (not recommended)

- You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head section.

```
<html>
<head>
</head>
<body>
<script>
document.write("This message is written by JavaScript");
</script>
</body>
</html>
```

Where to Put the JavaScript? External File

- If you want to run the same JavaScript on several pages, without having to write the same script on every page, you can write a JavaScript in an external file.
- Save the external JavaScript file with a .js file extension.
- Note: The external script cannot contain any HTML tags (in particular, <script></script> tags)

To use the external script, point to the .js file in the "src" attribute of the <script> tag:

```
<html>
<head>
  <script src=”myscript.js">
  </script>
</head>
  <body>
  </body>
</html>
```

When is a script executed?

- Global code (not in the body of functions):
  - is executed when it is met during the rendering of the page.
  - The global code can be in the HTML page or in an external file.
- Code in functions
  - is executed only if the function is called
- Event *onload*
  - The code corresponding to the event is executed when the page is loaded on the browser and after the execution of the code external to each function

When is a script executed?

Modern browsers:

- once they encounter a JavaScript file they pause the rendering of HTML
- run though the entire JavaScript file before they resume the HTML rendering
- in this example, **document.write** does run first, but we do not see the result before execution is completed
- the alert dialog pauses that processing.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
<title>Execution order</title>
<script>
```

When is a script executed?

```
    window.alert("Not in function");
    document.write("<h1> This is a heading. </h1>");
    document.write("<p> This is a paragraph. </p>");
    document.write("<p> This is another paragraph. </p>");
    function loading() { window.alert("onLoad event");}
</script>
</head>
<body onload="loading()" >
<p>Text in the body<\p>
<script>
document.write("<p>Paragraph written in the body.</p>");
window.alert("Code in Body"); </script>
</body>
</html>
```

JavaScript Statements

JavaScript Code

- JavaScript code is a sequence of Javascript statements
- The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement.
  - Note: Using semicolons makes it possible to write multiple statements on one line.
- JavaScript is case sensitive

JavaScript Statements

- Statements are executed by the browser in the sequence they are written.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
<title>Example</title>
<script>
  document.write("<h1>Introduction</h1>");
  document.write("<p>In this course we will introduce" +
 "<a href=\"http://https://developer.mozilla.org/en-US/docs/Web/JavaScript\">"
+ " Javascript</a></p>");
  document.write("<h2>Javascript Statements</h2>");
  document.write("<p>A Javascript statement is ... </p>");
</script>
</head>
<body>
<p>Example</p>
</body>
```

JavaScript Blocks

- JavaScript statements can be grouped together in blocks.
  - Blocks start with a left curly bracket { and ends with a right curly bracket }.
  - The purpose of a block is to execute the sequence of statements together.

Example

```
<script>
{  document.write("<h1>This is a heading</h1>");
   document.write("<p>This is a paragraph.</p>");
   document.write("<p>This is another paragraph.</p>");
}
</script>
```

JavaScript Comments

- Comments can be added to explain the JavaScript, or to make the code more readable.
  - Single line comments start with //
  - Multi line comments start with /* and end with */

```
<script>
/*
The code below will write one heading and two paragraphs
*/
document.write("<h1>This is a heading</h1>");
// Write a heading
document.write("<p>This is a paragraph.</p>"); //paragraph
document.write("<p>This is another paragraph.</p>");
</script>
```

JavaScript Variables

- Rules for JavaScript variable names:
  - Variable names are case sensitive (y and Y are two different variables)
  - The first character in the name must be a letter (a-z or A-Z) or an underscore (_).
  - The rest of the name can be made up of letters (a-z or A-Z), numbers (0-9), or underscores (_).
  - Names should describe what variables are.
- The type of the variable is not specified

JavaScript Variables

- JavaScript allows declaring variables by simply using them
- Anyway, declaring variables helps to ensure that programs are well organized and helps to keep track of the scope of variables
- To declare JavaScript variables you can use the **var** statement

```
var x;
var username;
```

After the declaration, the variables are empty (they have no value yet)

JavaScript Variables

Since **ES6**, there are three kinds of variable declarations:

- var declares a variable, intialization is optional
- let declares a block-scoped, local varible
- const declares a block-scoped constant (read only)

- **let** and **const** have been introduced to solve some problems caused by **var**

let vs var

```
let x = 10;

if (x === 10) {
  let x = 20;

  console.log(x);
  // prints 20
}

console.log(x);
// prints 10
```

```
function varExample() {
  var x = 10;
  {
    var x = 20;  // it is the same variable
    console.log(x);  // prints 20
  }
  console.log(x);  // prints 20
}

function letExample() {
  let x = 10;
  {
    let x = 20;  //it is a different variable
    console.log(x);  // prints 20
  }
  console.log(x);  // prints 10
}
```

the scope of a **var** is the whole function where it is declared

let vs var

- When used outside functions, var creates a property in the global object

```
var a = 'ABC';
let b = 'XYZ';
console.log(this.a); // "ABC"
console.log(this.b); // undefined
```

```
var a = 1;
var b = 2;
if (a === 1) {
  var a = 10; // the scope is global because of the other a
  let b = 20; // the scope is this block
  console.log(a); // prints 10
  console.log(b); // prints 20
}
console.log(a); // prints 10
console.log(b); // prints 2
```

```
if (x) {
  let v;
  let v; // error, the same
         // variable defined twice
}
```

const

- scope works similarly to let, must be initialized

MANY EXAMPLES SHOULD USE LET AND CONST

```
const PI = 3.14;

// this will throw an error - Uncaught
// TypeError: Assignment to constant variable.
PI = 5;

console.log('The value of PI is ' + PI);

// trying to redeclare a constant throws an error
// Uncaught SyntaxError: Identifier 'PI' has already
// been declared
const PI = 9;

// Error: the name PI is reserved
var PI = 20;

// this throws an error since it is already defined
let PI = 20;

// Error, must be initialized, it is a const
const DIM;
```

JavaScript Variables

- Variables can be initialized

```
var x=5;
var carname="Volvo";
```

- If you assign values to variables that have not been declared yet, the variables will automatically be declared.
  - and they will be global variables... (error-prone)
- If you redeclare a JavaScript variable, it will not lose its original value.

```
var x=5;
var x;
```

- NOTE: the variable x will still have the value of 5. The value of x is not reset when you redeclare it.

Use of undeclared variables

- Esempio relativo a all'uso (sbagliato) di variabili non dichiarate.
- Vengono dichiarate automaticamente e sono global.

```
<!DOCTYPE html>
<html>
<head>
<script>
var abc = 10;
if (abc===10) {
  def = 20;
}
console.log(window.abc); // Stampa 10
console.log(window.def); // Stampa 20
// def automaticamente creata e globale
</script>
</head>
<body>
Esempio relativo a all'uso (sbagliato) di variabili non dichiarate.
Vengono dichiarate automaticamente e sono global.
</body>
</html>
```

Loosely typed

- JavaScript is what is called a loosely typed programming language:
  - the type of a variable is not defined when a variable is created and can, at times, change based on the context.

```
var text1 = "19";
var num1 = 96;
num1 = text1 + num1;
```

The variable num1 contains "1996".

```
let x = 5;
x = "hello";
```

Types of Values

- JavaScript recognizes the following types of values:
  - Number
  - String
  - Boolean
  - null – a special keyword denoting a null value
    - null is also a primitive value
  - undefined - a top-level property whose value is undefined
    - undefined is also a primitive value

Numeric Values

- Integer

NUMBER SYSTEM	NOTATION
Decimal (base 10)	A normal integer without a leading 0 (zero) (ie, 752)
Octal (base 8)	An integer with a leading 0 (zero) (ie, 056)
Hexadecimal (base 16)	An integer with a leading 0x or 0X (ie, 0x5F or 0XC72)

- Floating Point Values
  - 2.3e-3
  - 2.3E-3

String Values

- String

  contains zero or more characters enclosed in single or double quotes

  - NOTE: the empty string is distinct from the null value
  - The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string

    - \'
    - \"
    - \\

String type

Escape characters

Character	Description
\n	new line
\t	tab
\r	carriage return
\f	form feed
\b	backspace

NOTE: when output to document, the escape characters only work in the following situations:

- within <pre> tags
- alert(), confirm() and prompt()
- within <textarea> tags

Boolean and null Values

- Boolean
    - Note: Values of 1 and 0 are not considered Boolean values in JavaScript
- null Value
    - Represents Nothing
- NaN – Not a Number (returned by some functions like parseInt() and parseFloat())

Variable Scope

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
<title>Scope</title>
<script>
var cc = 0 ;          //global
var dd = scr();    // global
document.writeln("global: " + cc); // print value of cc
document.writeln("local: " + dd);  // print value of dd
function scr() {
    var cc = 3;    //local variable hides the global variable cc
    // without var, it would be an assignment to global variable cc
    return cc;
}
</script>
</head>

<body>
<p>Scope</p>
</body>
</html>
```

JavaScript Operators
Arithmetic Operators

- Let us assume that y=5

Operator	Description	Example	Result
+	Addition	x=y+2	x=7
-	Subtraction	x=y-2	x=3
*	Multiplication	x=y*2	x=10
/	Division	x=y/2	x=2.5
%	Modulus (division remainder)	x=y%2	x=1

++	Increment	x=++y
		x=y++
			x=6
			x=5

--	Decrement	x=--y
		x=y--
			x=4
			x=5

JavaScript Operators
Assignment Operators

Operator	Example	Same As	Result
=	x=y		x=5
+=	x+=y	x=x+y	x=10
-=	x-=y	x=x-y	x=5
*=	x*=y	x=x*y	x=25
/=	x/=y	x=x/y	x=5
%=	x%=y	x=x%y	x=0

JavaScript Operators
The + Operator used on Strings

- The + operator can also be used to add string variables or text values together

```
txt1 = "This is a very";
txt2 = "nice day";
txt3 = txt1 + " " + txt2;
```

- Note: if you add a number and a string, the result will be a string

```
<script>
x = "5" + "5";
document.write(x);
document.write("<br>");                //55
x = 5 + "5";
document.write(x);
document.write("<br>");                //55
</script>
```

JavaScript Operators
Comparison Operators

- Given x=5, the table below explains the comparison operators

Operator	Description	Example
==	is equal to	x==8 is false x==’5’ is true
===	is exactly equal to (value and type)	x===5 is true x===’5’ is false
!=	is not equal (it attempts conversion)	x!=8 is true x!=5 is false
!==	is not equal and/or not of the same type	x!==’5’ is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

JavaScript Operators
Comparison Operators

- Comparison operators
    - If either or both values are **NaN**, then they are **not** equal.
    - Objects, arrays, and functions are compared by reference. This means that two variables are equal only if they refer to the same object.
    - If both are **null**, or both **undefined**, they are equal.
    - If one value is **null** and one **undefined**, they are equal.

- Two separate arrays are never equal by the definition of the == operator, even if they contain identical elements.

JavaScript Operators
Logical Operators

- Given x=6 and y=3, the table below explains the logical operators

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x==5 || y==5) is false
!	not	!(x==y) is true

JavaScript Operators
Bitwise Operators

Operator	Description	Example
&	and	a & b
|	or	a | b
!	xor	a ^ b
~	not	~a
<<	Left shift	a<<b
>>	Sign-propagating right shift	a>>b
>>>	Zero-fill right shift	a>>>b

JavaScript Operators

- If the types of the two values differ, attempt to convert them into the same type so they can be compared:
  - If one value is a number and the other is a string
    - convert the string to a number and try the comparison again, using the converted value.
  - If either value is true
    - convert it to 1 and try the comparison again.
  - If either value is false
    - convert it to 0 and try the comparison again.
  - If one value is an object and the other is a number or string
    - convert the object to a primitive value by either its toString() method or its valueOf() method. Native JavaScript classes attempt valueOf() conversions before toString() conversion.
  - Any other combinations of types are not equal.

JavaScript Operators

- Conditional Operator

  (condition) ? **val1** : **val2**

  Example:

```
var username = prompt("Please enter your name", "");
var greeting = "Hello ";
greeting += ((username != null) ? username : "guy");
```

JavaScript Operators
typeof operator

- **typeof** operator

  Two ways:

  1. typeof operand
  2. typeof (operand)

- The typeof operator returns a string indicating the type of the operand. The parentheses are optional.

```
var num1 = 3; var num2 = 3.0;
var b=true; var shape="round";
typeof num1;        // returns 'number'
typeof num2;        // returns 'number'
typeof b;           // returns 'boolean'
typeof shape;       // returns 'string'
```

JavaScript Operators
void operator

- void operator

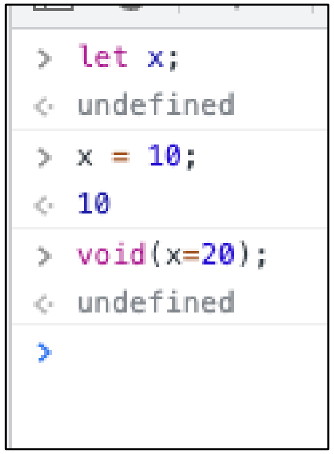
  Two ways:

  1. void (expression)
  2. void expression

- The void operator specifies a JavaScript expression to be evaluated without returning a value. The parentheses surrounding the expression are optional, but it is good style to use them.

- The following code creates a hypertext link that changes the background color

  <a
    href="javascript:void(document.body.style.backgroundColor=red')">
  Change background to red</a>



```
> let x;
< undefined
> x = 10;
< 10
> void(x=20);
< undefined
>
```

Conditional Statements
if statement

```
if (condition) {
  code to be executed if condition is true
}
```

Conditional Statements
if … else statement

```
if (condition) {
  code to be executed if condition is true
} else {
  code to be executed if condition is not true
}
```

Conditional Statements
if … else if … else statement

```
if (condition1)
 {
 code to be executed if condition1 is true
 }
else if (condition2)
 {
 code to be executed if condition2 is true
 }
...
else
 {
 code to be executed if condition1 and condition2 are not true
 }
```

Conditional Statements
if … else if … else statement

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Statement if</title>
    <script>
        var d = new Date();
        var time = d.getHours();
        if (time<12) {
            document.write("<em>Good morning</em>");
        } else if (time>=12 && time<17) {
            document.write("<em>Good afternoon</em>");
        } else if (time>=17 && time<20) {
            document.write("<em>Good evening</em>");
        } else {
            document.write("<em>Good night!</em>");
        }
    </script>
</head>

<body>
    <p>Example about if-else<p>
</body>
</html>
```

Conditional Statements
switch statement

```
switch(n)
{
case 1:
  execute code block 1
  break;
case 2:
  execute code block 2
  break;
default:
  code to be executed if n is different from case 1 and 2
}
```

Conditional Statements
switch statement

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
<title>Example</title>
<style>
p {color: white; background-color: grey; }
p.sat {color: red; background-color: black; }
p.sun {color: green; background-color: red; }
</style>
<script>
var d=new Date();
theDay=d.getDay();
switch (theDay) {
  case 6:
    document.write("<p class='sat'>Super Saturday</p>");
    break;
  case 0:
    document.write("<p class='sun'>Sleepy Sunday</p>");
    break;
  default:
  document.write("<p>I'm looking forward to this weekend!</p>");
}
</script>
</head>
<body>
<p>Example about switch</p>
</body>
</html>
```

Loop statements
for

for (var=startvalue;var OP endvalue;var=var+increment)

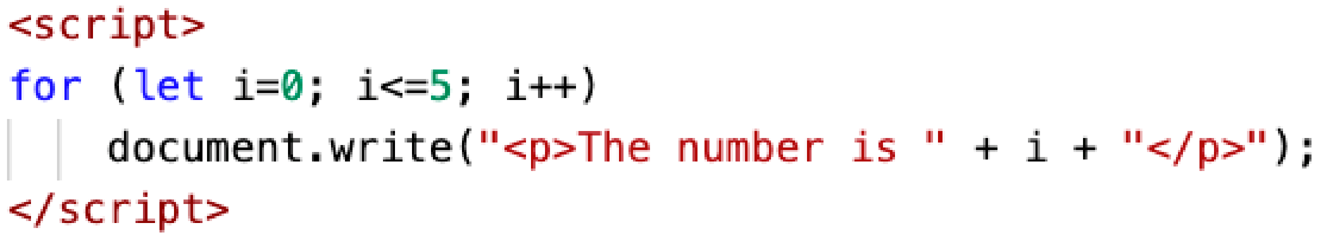
{

code to be executed

}

where OP is any comparison operator.

Example



```
<script>
for (let i=0; i<=5; i++)
    document.write("<p>The number is " + i + "</p>");
</script>
```

Loop statements
while

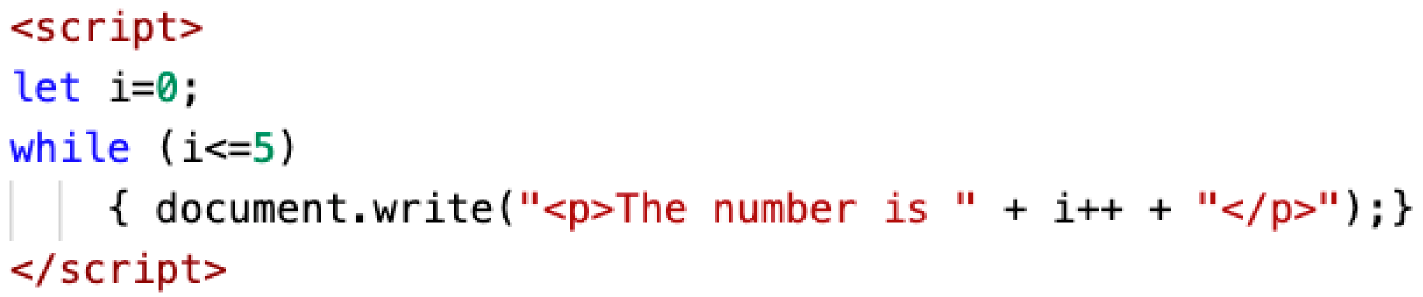
while (var OP endvalue)

{

code to be executed

}

where OP is any comparison operator.



```
<script>
let i=0;
while (i<=5)
        { document.write("<p>The number is " + i++ + "</p>");}
</script>
```

Loop statements
do…while

do

 {

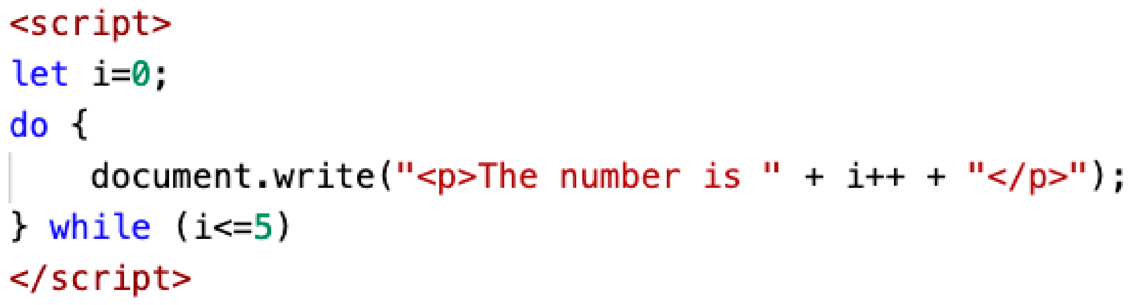
code to be executed

}

while (var OP endvalue);

where OP is any comparison operator.

Example



```
<script>
let i=0;
do {
    document.write("<p>The number is " + i++ + "</p>");
} while (i<=5)
</script>
```

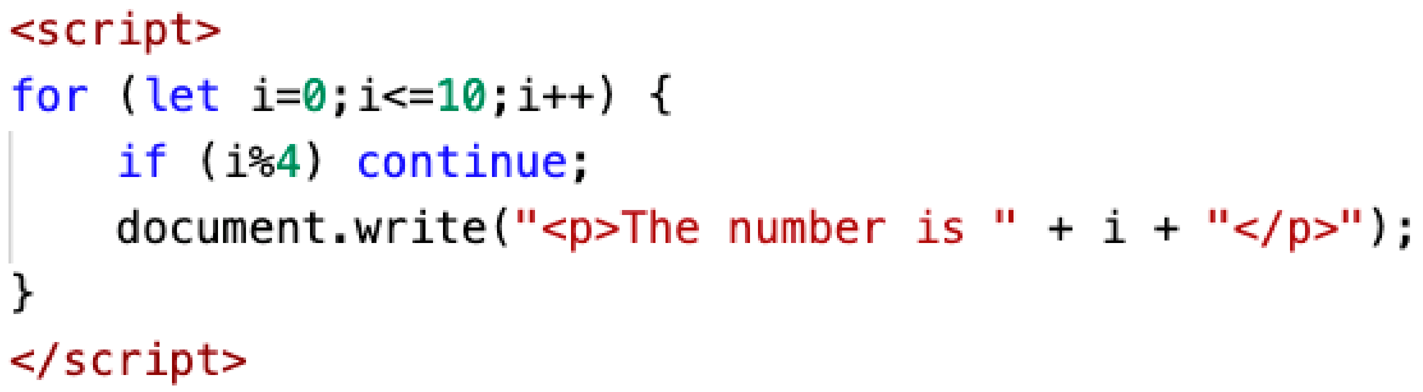
Break statement

- The break statement will break the loop and continue executing the code that follows after the loop (if any).

```
<script>
let i=0;
while (i<=5){
    document.write("<p>The number is " + i++ + "</p>");
    if (i==4) break;
}
</script>
```

Continue statement

- The continue statement will break the current loop and continue with the next value



```
<script>
for (let i=0;i<=10;i++) {
    if (i%4) continue;
    document.write("<p>The number is " + i + "</p>");
}
</script>
```

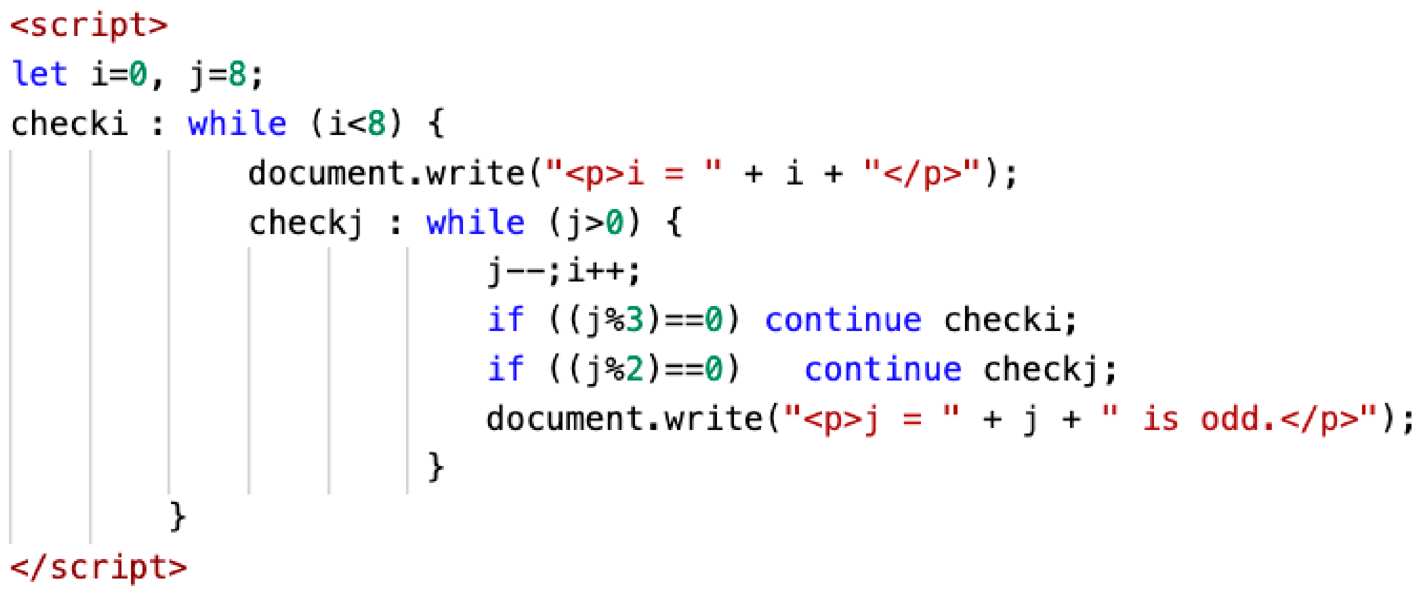
Label statement

- A label provides a statement with an identifier that lets you refer to it elsewhere in your program.

  label : statement

- The value of label may be any JavaScript identifier that is not a reserved word.
- On using label with break and continue
  - break [*label*] - terminates the specified enclosing label statement
  - continue [*label*] - restarts a label statement or continues execution of a labelled loop with the next iteration

continue statement (example)



```
<script>
let i=0, j=8;
checki : while (i<8) {
            document.write("<p>i = " + i + "</p>");
            checkj : while (j>0) {
                        j--;i++;
                        if ((j%3)==0) continue checki;
                        if ((j%2)==0)   continue checkj;
                        document.write("<p>j = " + j + " is odd.</p>");
                     }
         }
</script>
```

with statement

- The with statement establishes the default object for a set of statements.
- JavaScript looks up any unqualified names within the set of statements to determine if the names are properties of the default object. If an unqualified name matches a property, then the property is used in the statement; otherwise, a local or global variable is used.
- A with statement looks as follows:

```
with (object){
statements
}
```

- DEPRECATED

```
var a, x, y;
var r=10;
with (Math) {
a = PI * r * r;
x = r * cos(PI);
y = r * sin(PI/2);
}
```

Functions

```
function name(arg1, arg2, ..., argN)
{  var x;
   some code
   return x;
}
```

- You may call a function from anywhere within a page (or even from other pages if the function is embedded in an external .js file).
- Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that a function is read/loaded by the browser before it is called, it could be wise to put functions in the <head> section.

Functions

- All parameters are passed to functions by value; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.
- Objects are passed by reference: if the function changes the object's properties, that change is visible outside the function.
- A function can be recursive, that is, it can call itself.

Functions

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Example</title>
<style>
p {color: white; background-color: red; }
</style>
<script>
let x = 10;

function fun(s) { // function definition
  const a = 7;
  let c = 5 * s;
  s = a;
  document.write("<p>This is the result: " + c + "</p>");
  // print value of c on the HTML page
}
</script>
</head>

<body>
<script>
  document.write("Before calling function, x=" + x);
  fun(x);
  document.write("After calling function, x=" + x);
</script>
</body>
</html>
```

Functions

- The arguments of a function are maintained in the array "arguments".
- Within a function, you can address the parameters passed to it by:

  arguments[i]

  functionName.arguments[i]

  where i is the ordinal number of the argument, starting at zero.

  arguments[0] -> the first argument passed to a function.

- The total number of arguments is indicated by arguments.length.

Functions

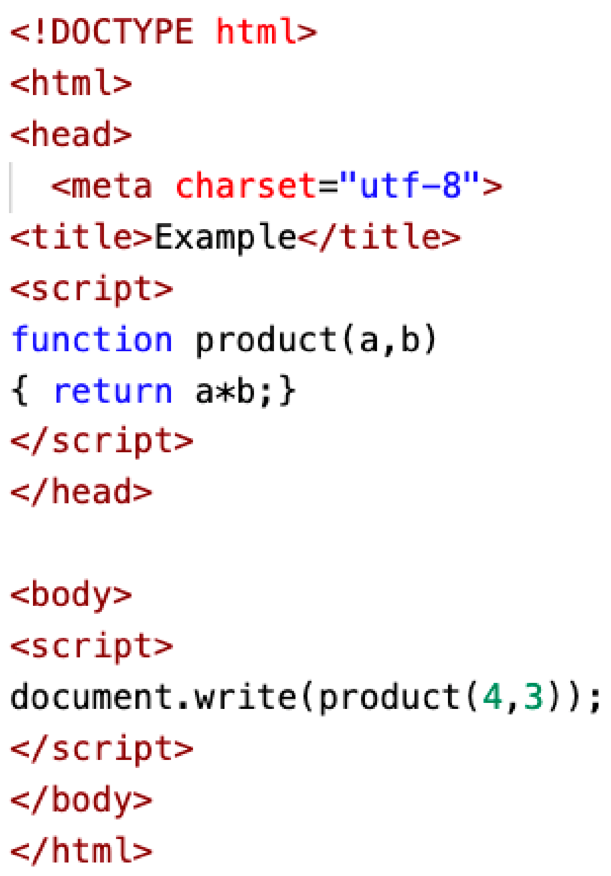
- Using the arguments array, you can call a function with more arguments than it is formally declared to accept.
  - This is often useful if you do not know in advance how many arguments will be passed to the function.
  - You can use arguments.length to determine the number of arguments actually passed to the function, and then treat each argument using the arguments array.

Functions

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function myConcat(separator) {
  let result=""; // initialize list
  // iterate through arguments
  for (let i=1; i<arguments.length; i++) {
    result += arguments[i] + separator;
  }
  result += "<br>";
  return result
}
</script>
</head>

<body>
<script>
// returns "red, orange, blue, "
document.write(myConcat(", ","red","orange","blue"));
// returns "elephant; giraffe; lion; cheetah;"
document.write(myConcat("; ","elephant","giraffe","lion", "cheetah"));
// returns "sage. basil. oregano. pepper. parsley. "
document.write(myConcat(". ","sage","basil","oregano", "pepper", "parsley"));
</script>
</body>
</html>
```

The return statement



```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
<title>Example</title>
<script>
function product(a,b)
{ return a*b;}
</script>
</head>

<body>
<script>
document.write(product(4,3));
</script>
</body>
</html>
```

Predefined functions

- JavaScript has several top-level predefined functions:
  - eval(string) - Evaluates a string and executes it as if it was script code
    eval("x=10;y=20;document.write(x*y)"); //200
  - isFinite - Determines whether a value is a finite, legal number
    document.write(isFinite(123)+ "<br>"); //true
    document.write(isFinite("2005/12/12")+ "<br>"); //false
  - isNaN -The isNaN() function determines whether a value is an illegal number (Not-a-Number).
    This function returns true if the value is NaN, and false if not.
    - document.write(isNaN(123)+ "<br>"); //false
    - document.write(isNaN("2005/12/12")+ "<br>"); //true

Predefined functions

- parseInt(string,radix)

  Parses a string and returns an integer of the specified radix (base).

  radix - a number that represents the numeral system to be used
- parseFloat(string)

  Parse a string and returns a float number.

  If the first character cannot be converted to a number, the two functions return NaN.
- Number(object) and String(object)

  Converts the object argument to a number or to a string that represent the object's value.

  If the value cannot be converted to a legal number, NaN is returned.

Predefined functions

```
<script>
eval("x=10;y=20;document.write(x*y)");
document.write("<br>" + isFinite(123)+ "<br>");
document.write(isFinite("2005/12/12")+ "<br>");
document.write(isNaN(123)+ "<br>");
document.write(isNaN("2005/12/12")+ "<br>");
document.write(parseInt("His age is 40 years")+ "<br>");
document.write(parseInt("40 years")+ "<br>");
</script>
```

Predefined functions

- escape(string) and unescape(string)

  The escape() function encodes a string. This function makes a string portable, so it can be transmitted across any network to any computer that supports ASCII characters. This function encodes special characters, with the exception of: * @ - _ + . /

  The unescape() function decodes an encoded string.

```
<script>
let str = "Che facciamo? Un esempio su encode! con * e è";
let str_escaped = escape(str);
document.write(str_escaped + "<br>");
document.write(unescape(str_escaped));
</script>
```

document.write() and document.writeln() methods

- document.write(exp1, exp2, exp3, ...)

  The write() method writes HTML expressions or JavaScript code to a document.

  Multiple arguments can be listed and they will be appended to the document in order of occurrence
- document.writeln(exp1, exp2, exp3, ...)

  The writeln() method is identical to the write() method, with the addition of writing a newline character after each statement.

Since HTML ignores the newline characters, the effects of the methods on the HTML pages are equal except when used within <pre> tags

The document.write() and document.writeln() methods

```
<body>
<pre>
<script>
document.write("Hello World!");
document.write("Have a nice day!");
</script>
</pre>
<pre>
<script>
document.writeln("Hello World!");
document.writeln("Have a nice day!");
</script>
</pre>

</body>
```

Popup Boxes
Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.

```
alert("sometext");
```

Popup Boxes
Alert Box

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function showAlert()
{ alert("I am an alert box!");}
</script>
</head>

<body>
<p><input type="button" onclick="showAlert()" value="Show an alert
box">
</p>
</body>
</html>
```

Popup Boxes
Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

```
confirm("sometext");
```

Popup Boxes
Confirm Box

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function showConfirm() {
  let r = confirm("Press a button!");
  if (r) {
    alert("You pressed OK!");
  } else {
    alert("You pressed Cancel!");
  }
}
</script>
</head>
<body>
<p><input type="button" onclick="showConfirm()" value="Show confirm box">
</p>
</body>
```

Popup Boxes
Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

```
prompt("sometext","default value");
```

Popup Boxes
Prompt Box

```
<head>
<meta charset="utf-8">
<title>Example</title>
<script>
function showPrompt() {
  let name = prompt("Please enter your name", "Harry Potter");
  if (name != null && name != "") {
    document.write("Hello " + name + "! How are you today?");
  }
}
</script>
</head>
<body>
<p>
<input type="button" onclick="showPrompt()" value="Show prompt box">
</p>
</body>
```

Catching Errors try...catch statement

```
try
 {
 //Run some code here
 }
catch(err if expression)
 {
 //Handle errors here
 }
```

- The try...catch statement allows you to test a block of code for errors.
- The try block contains the code to be run, and the catch block contains the code to be executed if an error occurs.
- *err* is initialized with the exception object
- *expression* is a test expression

Catching Errors try…catch statement

```
try {
   myroutine();   // may throw three exceptions
}
catch (e if e instanceof TypeError) {
   // statements to handle TypeError exceptions
}
catch (e if e instanceof RangeError) {
   // statements to handle RangeError exceptions
}
catch (e if e instanceof EvalError) {
   // statements to handle EvalError exceptions
}
catch (e){
   // statements to handle any unspecified exceptions
   logMyErrors(e) // pass exception object to error handler
}
```

Catching Errors
try…catch statement

```
<script>
function message() {
  try {
    addlert("Welcome guest!");
  } catch(err) {
    let txt="There was an error on this page.\n\n";
    txt += err;
    txt += "\n\nClick OK to continue viewing this page,\n";
    txt += "or Cancel to return to the home page.\n";
    if(!confirm(txt)) {
      document.location.href= "http://www.example.com";
    }
  }
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()">
</body>
```

Catching Errors
Throw statement

- The throw statement allows you to create an exception.

  throw(exception)

- The exception can be a string, integer, Boolean or an object.

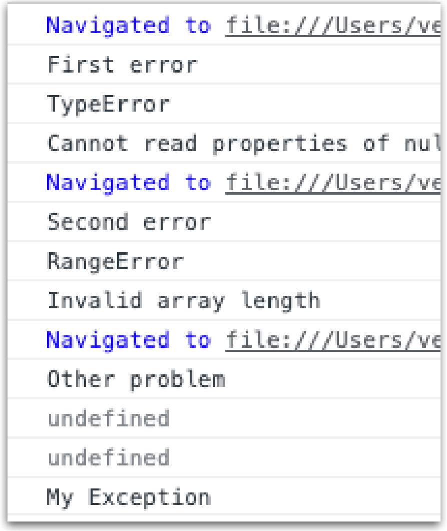
Catching Errors
Throw statement

```
<body>
<script>
let x = prompt("Enter a number between 0 and 10:","");
try {
  if(x>10) { throw "Err1"; }
  else if(x<0) { throw "Err2"; }
  else if(isNaN(x)) { throw "Err3"; }
  // Other statements
  console.log("Here!");
} catch(er) {
  if(er=="Err1") {
    alert("Error! The value is too high");
  }
  if(er=="Err2") {
    alert("Error! The value is too low");
  }
  if(er=="Err3") {
    alert("Error! The value is not a number");
  }
}
</script>
</body>
```

Conditional catch blocks are not standard

```
<script>
try {
 let i = prompt("Insert 1, 2 , or 3", "1");
 switch(i) {
  case "1":
   let s = null; s.funz(); break;
  case "2":
   let a = new Array(-1); break;
  case "3":
   throw 'My Exception'; break;
 }
 console.log("End of try block");
} catch(e) {
 if (e instanceof TypeError) {
  console.log("First error"); console.log(e.name);
  console.log(e.message);
 } else if (e instanceof RangeError) {
  console.log("Second error"); console.log(e.name);
  console.log(e.message);
 } else {
  console.log("Other problem");
  console.log(e.name); console.log(e.message); console.log(e);
 }
}
</script>
```

- The conditional catch blocks (few slides before) are non-standard
- Use if-else or switch in catch block



```
Navigated to file:///Users/ve
First error
TypeError
Cannot read properties of nul
Navigated to file:///Users/ve
Second error
RangeError
Invalid array length
Navigated to file:///Users/ve
Other problem
undefined
undefined
My Exception
```