Javascript

Roberta Calegari

roberta.calegari@unibo.it

Alma AI - Alma Mater Research Institute for Human-Centered Artificial Intelligence—Università di Bologna

Master in diritto delle nuove tecnologie e informatica giuridica
Academic Year 2020/2021

Next in Line...

Focus on...

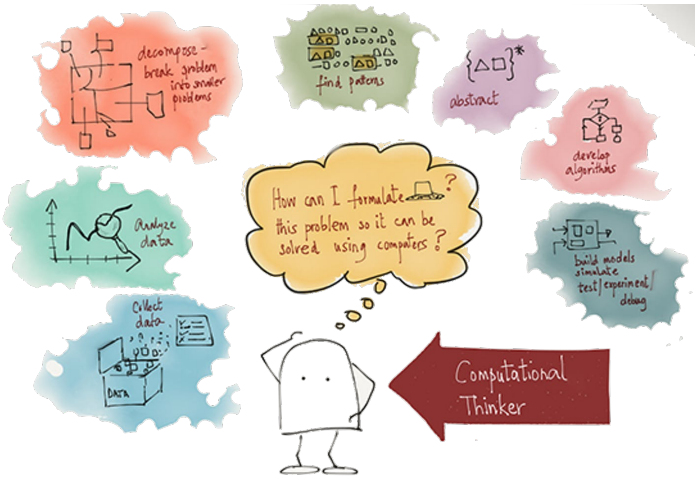
Computational Thinking & Coding

Computational thinking

Computational thinking refers to the thought processes involved in formulating problems and their solutions as computational steps or algorithms that can be carried out by a software agent (e.g. a computer) [Wing, 2011, Aho, 2011].

Coding

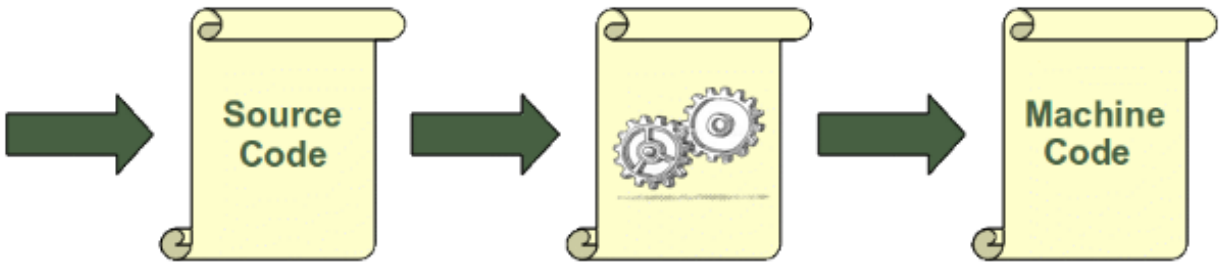
Coding is arguably a setting for developing capacity for computational thinking [Brennan and Resnick, 2012].



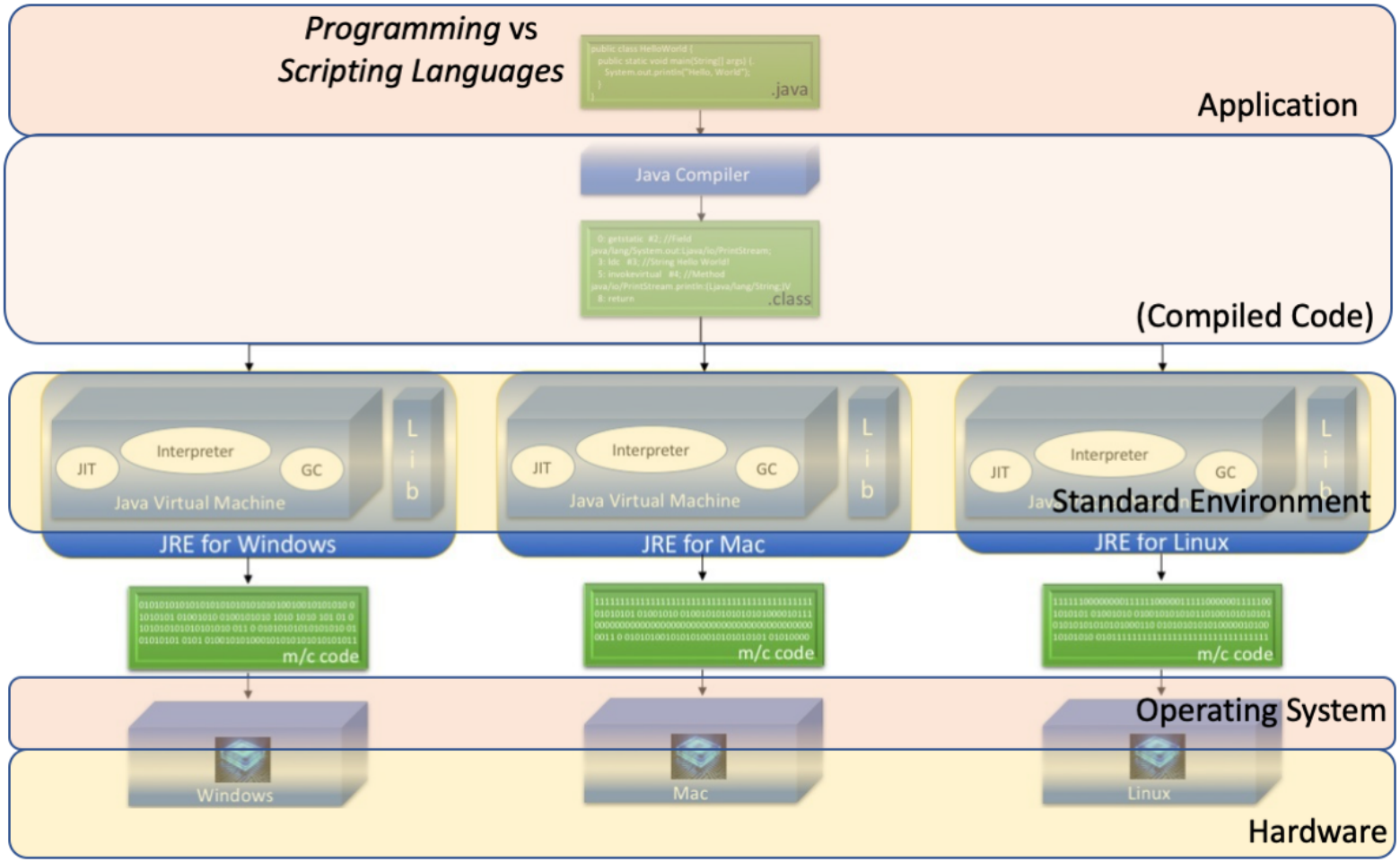
Introduction to Programming Languages

A programming language is a vocabulary and set of grammatical rules for instructing a computer to perform specific tasks.

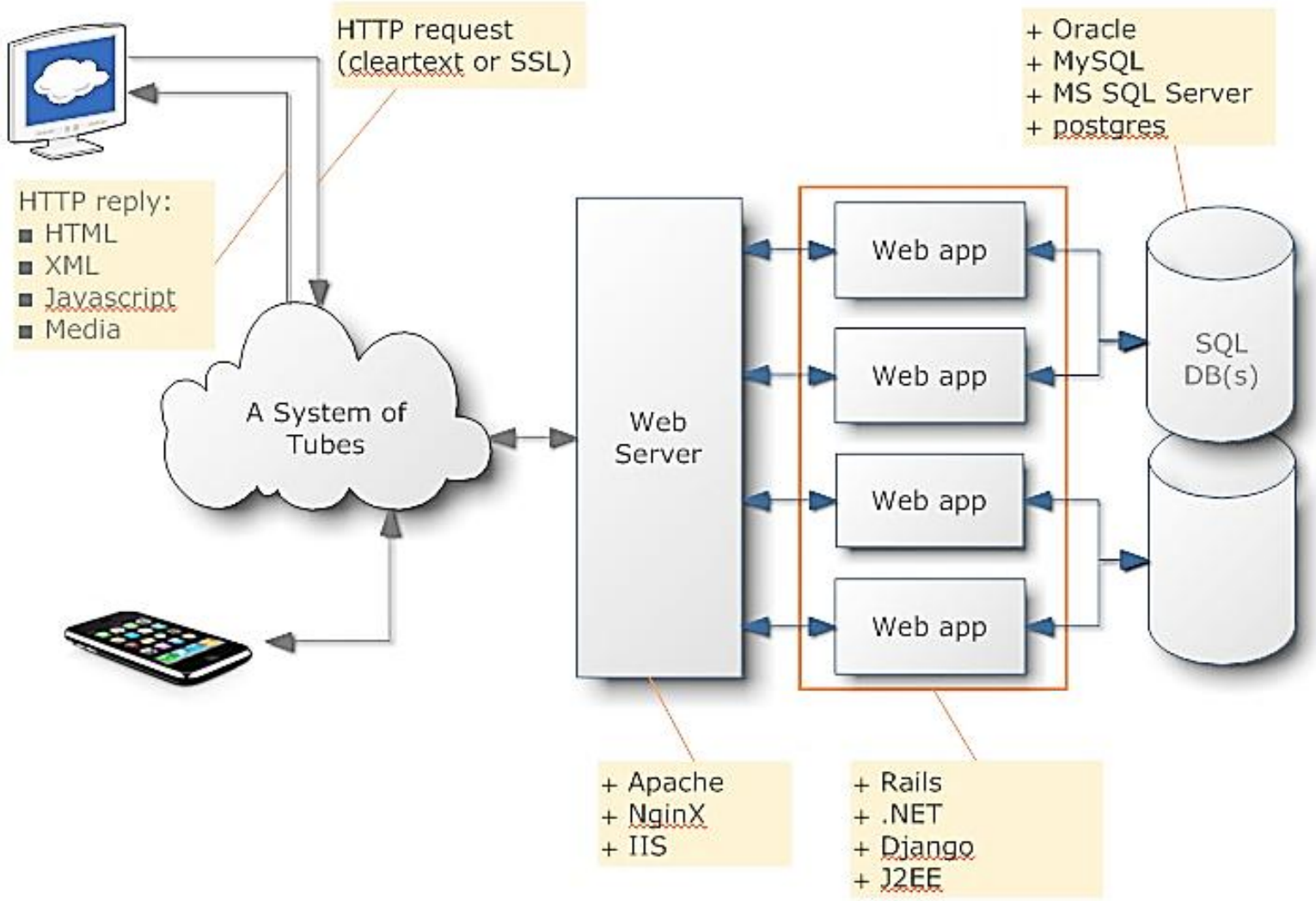
A language used to write instructions for the computer. It lets the programmer express data processing in a symbolic manner without regard to machine-specific details.



From Programming Language to Infrastructures I



From Programming Language to Infrastructures II



Focus on...

Many Programming Languages, Many Paradigms I

- Imperative: programmers instruct the machine how to change its state

→ e.g. C++, C#, Java, *Javascript* (can be exploited also as functional)

- Declarative: programmers just declare properties of the desired result, but not how to compute it

→ e.g. *Prolog*

Many Programming Languages, Many Paradigms II

Imperative approach $\rightarrow$ specify an operation typically means say *how to do* it *instruction by instruction*

- sub-operations are performed immediately
- order of the sub-operations is that in which they are written
- hard to parallelize it without remaking it from scratch

Imperative Paradigm

- a program is precisely a sequence of orders
- control every single detail of every single operation (no delegation)
- result: in some situation you are overwhelmed by so much control —difficulty in abstracting, impossibility to predict & plan everything

Many Programming Languages, Many Paradigms III

There are other paradigms, not oriented to control, in which the focus is on *what you want*, leaving in the background how to do it

How does computation works in such paradigms?

Declarative

- axioms and rules that express true relationships in the domain
- answer questions (queries) through a process of logical-deductive inference

Functional Paradigm

- transformation of data in terms of functions that act gradually on them
- provide the initial data and taking the output data from the last function

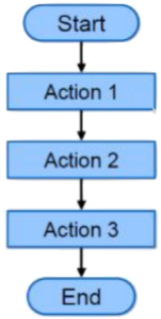
Next in Line...

Focus on...

Flow control sequence

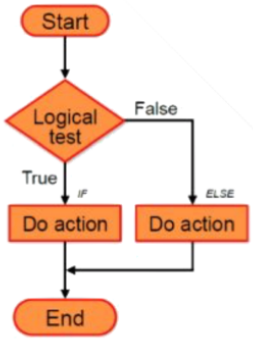
- Sequence: the order of commands

```
Go to the shops
Buy all items on list
```



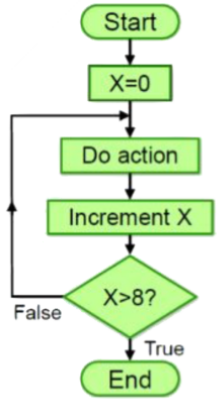
- Condition: if/then commands

```
If donuts are less than $1:
  Buy 3 donuts
```



- Loop: commands that are repeated

```
For every item on list
  Compare price
  Buy the cheaper item
```



Focus on...

Main abstraction I

Variable:

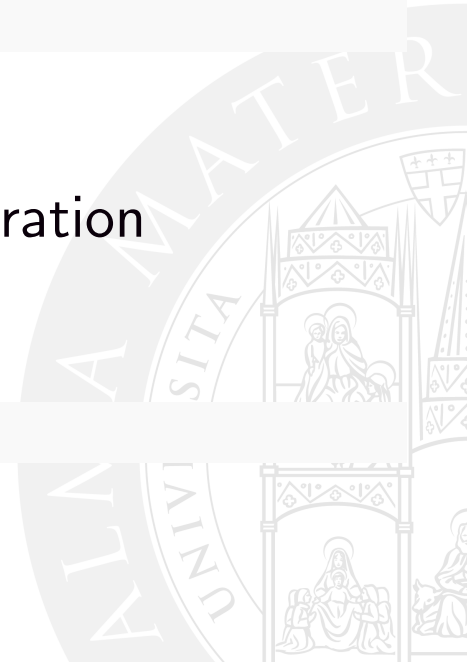
- container of data
- stored in memory

```
price_item_1 = 4
price_item_2 = 6
```

Operator:

- symbol to perform specific math, relation or logic operation
- produce final result

```
total = price_item_1 + price_item_2
```



Main abstraction II

Function:

- abstraction of a specific task (a.k.a. procedure)
- set of instructions
- useful for repeating tasks (no code duplication)

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

result = sum(price_item_1, price_item_2)
```

Next in Line...

1. Context & Definition
2. Coding
3. Javascript

Focus on...

Javascript in a nutshell

- A scripting language: interpreted, not compiled
- History
  - Originally defined by Netscape (LiveScript) - Name modified in JavaScript after an agreement with Sun in 1995
  - Microsoft calls it JScript (minimal differences)
  - Reference: standard ECMAScript 262
- Object based (but not object oriented)
- JavaScript programs are directly inserted in the HTML source of web pages

Javascript: where & how

Javascript engines can be found

- in browsers
- as a stand alone virtual machine
- as API embedded in other platforms / languages

The most popular

- V8 (Google)
- JScript (Microsoft)
- GraalVM (Java 11+) interoperable with Java
- Nashorn (Java 8-11) integrated with Java
- Rhino (Java < 8)
- SpiderMonkey (Mozilla)

JavaScript in HTML: what can we do?

- Document content and presentation control
  - The document object
  - DOM
- Browser control
  - The window object
- Form management
  - The Form, Button, ... objects
- User interaction
  - Event management
  - Interaction state management

The Web Page

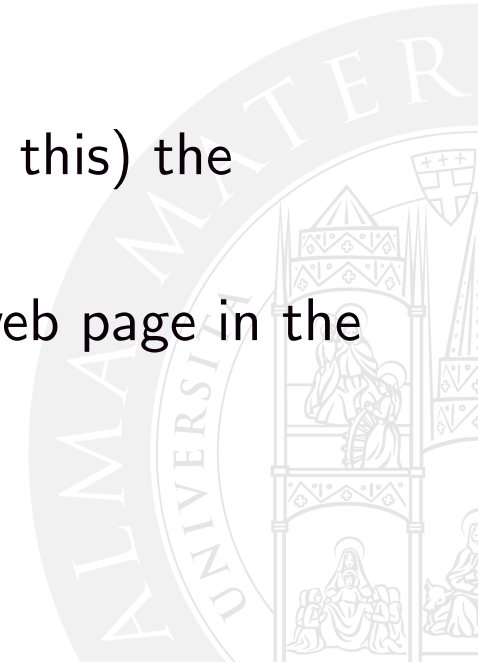
```
<html>
  <head><title>...</title></head>
  <body>
   ...
  <script language="JavaScript">
  <!-- HTML comment to avoid puzzling old browsers

   ... put here your JavaScript program ...

  // JavaScript comment to avoid puzzling old browsers -->
  </script>
  </body>
</html>
```

An HTML page may contain multiple `<script>` tags

Focus on...

Document Object Model

- JavaScript as a language references the Document Object Model (DOM)
- Following that model, every document has the following structure
  - `window`
    - `document`
    - ...
- The window object represents the current object (i.e. this) the current browser window (visualising entity)
- The document object represents the content of the web page in the current browser window (visualised entity)



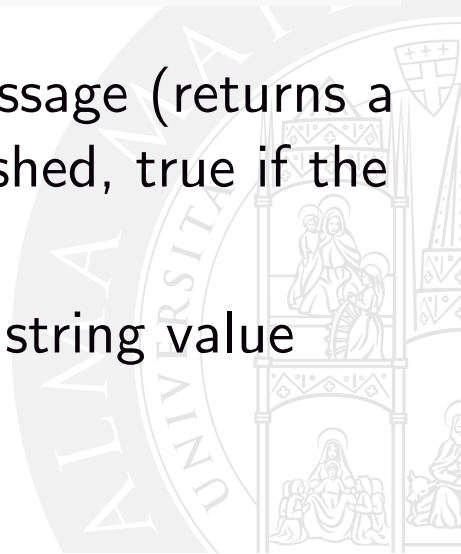
The window object

The window object is the root of the DOM hierarchy and represents the browser window

- alert, which makes an alert window appear, displaying the give message

```
x =-1.55; y = 31.85; sum = x + y;
msg = "Sum of "+ x + " and " + y + " is " + sum;
alert(msg); /* returns undefined */
```

- confirm to display a dialog to confirm or dismiss a message (returns a boolean value: false if the Cancel button has been pushed, true if the OK button has been pushed)
- prompt to display a dialog to input a value (returns a string value containing the input)



The document object

Document object → current web page (not current browser window!)

- Main components (all arrays): forms, anchors, links, images, applets
- Document element referred to by the value of its `id` attribute
  - e.g. for an image identified as image0

```
document.getElementById("image0")
document.images["image0"]
```

- Possible to invoke many different methods on it
  - e.g. write method to prints values on the page

```
document.write("Scrooge McDuck")
document.write(2020-1947)
document.write('<img src="image.jpeg">')
```

- The `this` reference to the window object can be omitted:

```
document.write       // this statements are
this.document.write  // equivalent
```

Focus on...

Strings

String type denotes strings composed by Unicode character. Strings are immutable objects with properties (length) and methods (substring, ...)

- strings can be delimited by using single or double quotes
- constants are delimited by quotes, nest different kind of quotes, alternate them

```
e.g. document.write('<img src="img.gif">')
e.g. document.write("<img src='img.gif'>")
```

- Use + to concatenate strings (attention to numbers!!)

```
document.write('donald' + 'duck')
document.write(beta + alfa + 2)
document.write(beta + (alfa + 2))
```

Numbers, Constants and comments

Number type denotes a 64-bit real → division is always between reals, even with integer operands

- `Numeric constants` are sequences of numeric characters not enclosed between quotes - their type is number
- `NaN` constant represents the "result" of impossible mathematical operations (it is not equal to anything)
- `Infinity` constant represents a value greater than the maximum positive real that can be represented ($1.79 * 10^{+308}$)
- Other constants are `null`, `undefined` (returned by functions that return nothing, initially attributed to uninitialized variables)
- `Boolean constants` are true and false - their type is boolean
- `Comments` can be

```
// on a single line
/* multi line */
```

Expressions

These are legal expressions in JavaScript

- numeric expressions, with operators like + - * / % ...
- conditional expressions, using the ?: ternary operator
- string expressions, concatenating with the + operator
- assignment expressions, using =

Some examples

```
window.alert(18/4)
window.alert(3>5 ? 'yes' : "no")
window.alert("donald" + 'duck')
```

Variables I

Variables in JavaScript are loosely typed: values of different types to the same variable at different times

```
a=19; b= 'bye'; a='world'; // different types!
```

- Legal operators include increment (++), decrement (- -), extended assignment (e.g. +=)
- declaration can be implicit or explicit (with the keyword var)

```
num = 18
var foo = 19
```

Variables II

- Variable scope is
  - global variables defined outside functions
  - local variables defined inside functions (input parameters included)
- Warning: a block does not define a scope!!

```
x = '32'  // the string '32'
{
  { x = 5 } // internal block
  y = x + 3 // here x is 5, not '32'
}
```

Dynamic types

The typeof operator is used to retrieve the (dynamic) type of an expression or a variable

```
typeof(18/4) returns number
typeof "aaa" returns string
typeof false returns boolean
typeof document returns object
typeof document.write returns function
```

When used with variables, the value returned by typeof is the current type of the variable

```
a = 18; typeof a // returns number
a = 'hi'; typeof a // returns string
```

Concatenation between strings and numbers leads to an automatic conversion of the number value into a string value (be careful...)

```
window.alert(bravo + alpha + 2)
window.alert(bravo + (alpha + 2))
```

Instructions

Instructions must be separated by an end-of-line character or by a semicolon

```
alpha = 19 // end-of-line
bravo = 'donald duck';
charlie = true
window.alert(bravo + alpha)
```

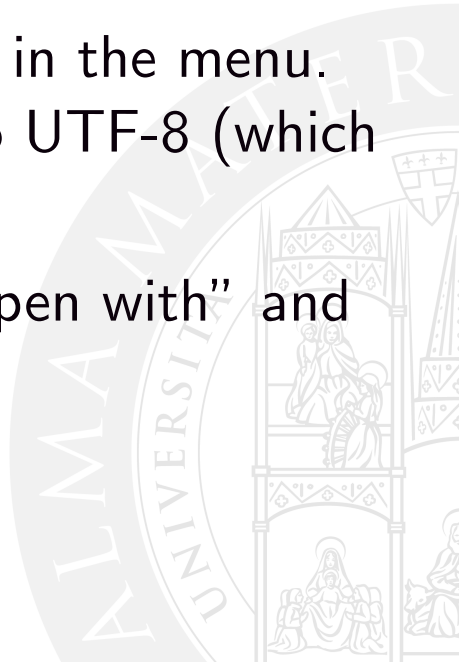
ATTENTION: risk of semicolon insertion

```
return 18; // returns 18
return // returns undefined
18; // (unreachable statement)
```

Focus on...

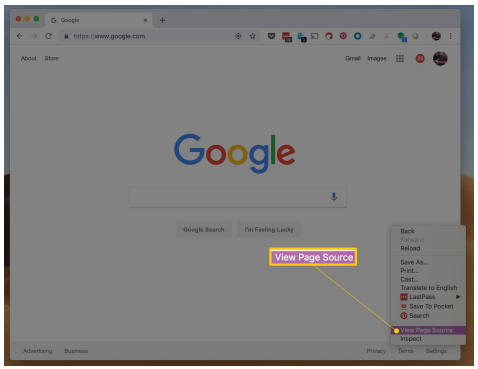
Set up the environment I

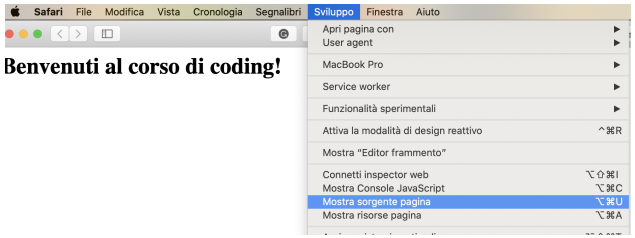
- Web pages can be created and modified by using professional HTML editors
- For learning HTML $\rightarrow$ a simple text editor like Notepad (Win) or TextEdit (Mac).
- Save the file on your computer. Select File $>$ Save as in the menu. Name the file "myPage.html" and set the encoding to UTF-8 (which is the preferred encoding for HTML files).
- double click on the file – or right-click and choose "Open with" and select your browser

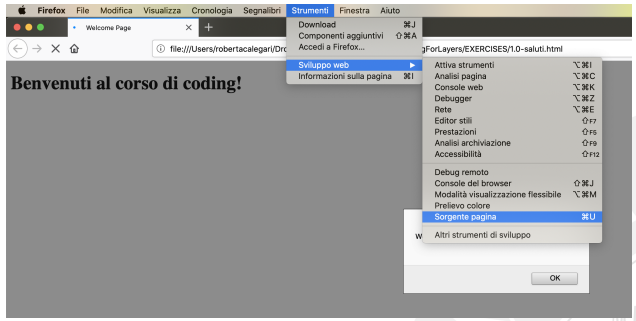


Set up the environment II

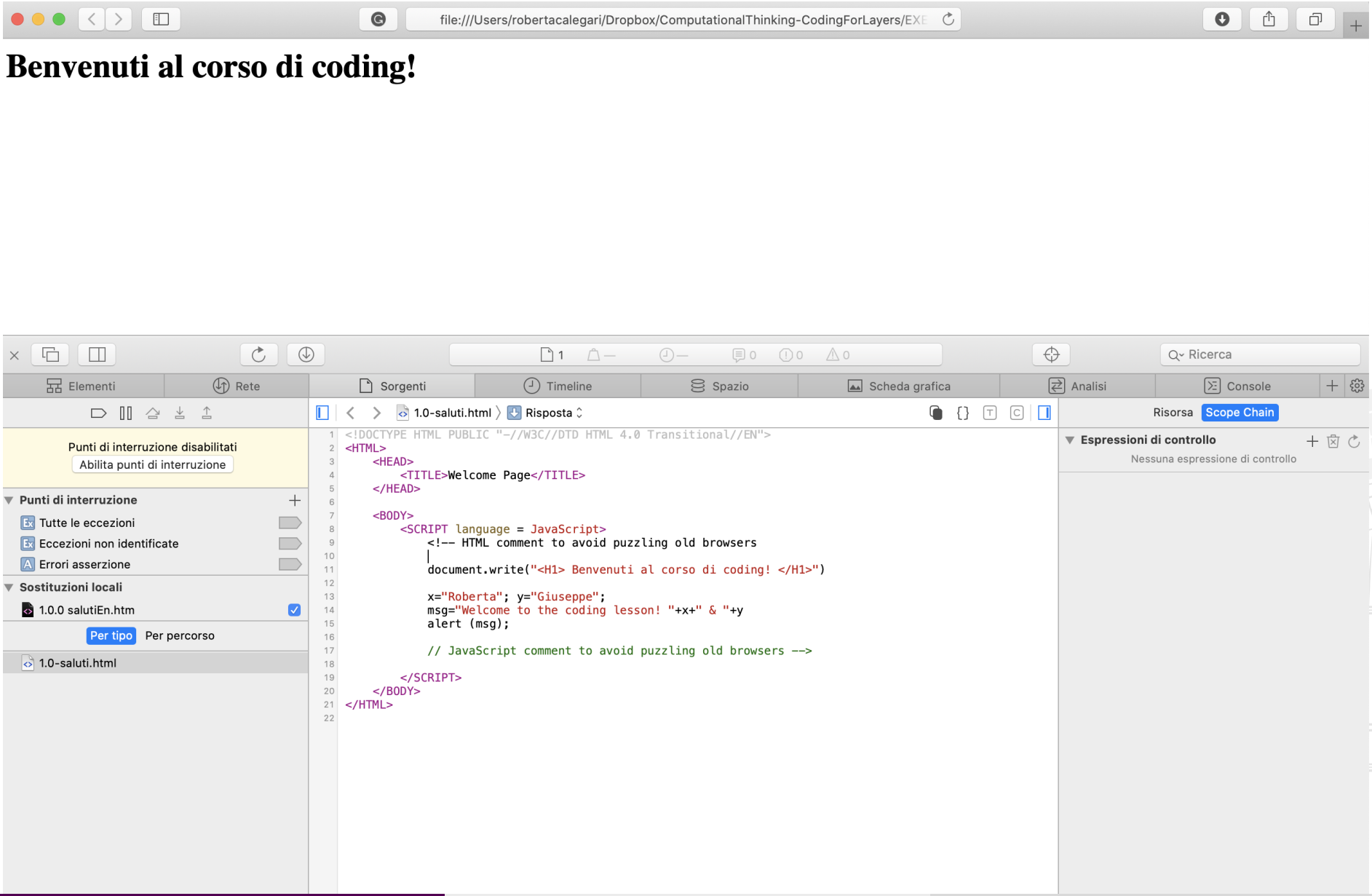
- From browser, in "Developer" Mode you can see and modify source:







Set up the environment III



HTML Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD>
    <TITLE>My HTML Page</TITLE>
  </HEAD>
  <BODY>
    <h1>I miei motori di ricerca</h1>

    <p><big> Yahoo: <a href="http://www.yahoo.com"> yahoo</a></big></p>

    <p><big>Google: <a href="http://www.google.com"> google</a></big></p>

    <p><big>Bing: <a href="https://www.bing.com/"> bing</a></big></p>
  </BODY>
</HTML>
```

Exercise 1: Welcome Page

- Welcome Page
- Alert message composed by String concatenation

Exercise 2: String Parsing

- Read name and surname by user (`window.prompt`)
- Concatenate them
- Display the result (`document.write`)

Exercise 3: Integer Sum

- Read numbers by user as strings (`window.prompt`)
- Convert strings into integers (`parseInt`)
- Add numbers
- Display the result (`document.write`)

Exercise 4: Salary & Taxes

- Read salary and tax (`window.prompt`)
- Display the net salary (`document.write`)

Exercise 5: Area

- Write a JavaScript program to find the area of a triangle where lengths of the three of its sides are 5, 6, 7
- Exploit `Math.sqrt`
- Display the area (`window.alert`)

Exercise 6: Form Divide/Multiply

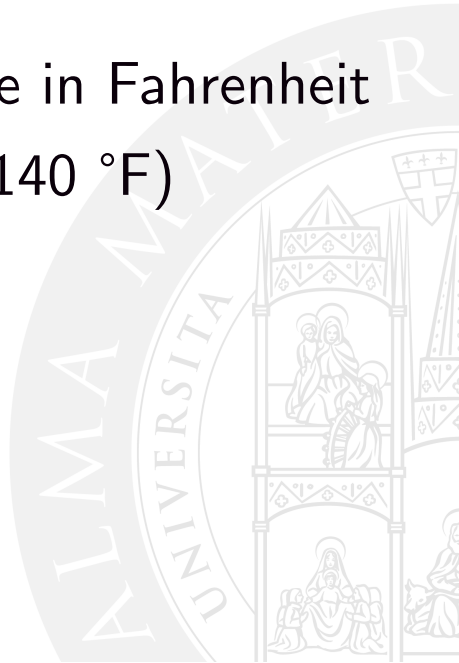
- Write a JavaScript program to calculate multiplication and division of two numbers (input from user).

Exercise 7: Temperature conversion

- Write a JavaScript program to convert temperatures to and from Celsius, Fahrenheit.

$$Formula : c/5 = (f - 32)/9$$

- where c = temperature in Celsius and f = temperature in Fahrenheit
- convert in Fahrenheit 60° (Expected Output: 60°C is 140 °F)
- convert in Celsius 45°F (Expected Output: 45°F is 7.22222222222222°C)



Focus on...

Comparison operators

Boolean conditions

→ relational operators (==, !=, >, <, >=, <=)
→ logic operators AND (&&), OR(||), NOT (!).

Problem 1:

- not only the false value is considered false, but every falsy value (`null`, `undefined`, empty string (`' '`), `0` and `NaN`)
- any other object, including the string 'false', is true .

Problem 2:

- The evil brothers == and != apply type coercion according to unnatural rules, with results sometimes incomprehensible
- The good brothers: new operators ===,!== offer a more sensible alternative to the questionable behavior of the previous two

Control structures

Control structures:

- condition: if, switch
- loop: for, while, do/while
- special structures used to work on objects: for/in, for/of and with

Conditional Statements I

`if` statement to specify a block of JavaScript code to be executed if a condition is true

```
if (condition1) {
    /* code to be executed if condition1 is true */
} else if (condition2) {
    /* code to be executed if the condition1 is false
    and condition2 is true */
} else {
    /* code to be executed if the condition1 is false
    and condition2 is false */
}
```

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

Conditional Statements II

`switch` statement to select one of many code blocks to be executed

```
switch(expression) {

    case x:
        /* code block */
        break;
    case y:
        /* code block */
        break;
    default:
        /* code block */

}
```

```
var x = 0;
switch (x) {
  case 0:
    text = "Off";
    break;
  case 1:
    text = "On";
    break;
  default:
    text = "No value found";

}
```

Loops I

Loops are handy, if you want to run the same code over and over again, each time with a different value

```
while (condition) {
  /* code block to be executed*/
}
```

```
i=1;
while (i < 10) {
  text += "The number is " + i;
  i++;
}
```

Loops II

```
for (statement 1; statement 2; statement 3) {
  /*code block to be executed*/
}
```

- Statement 1 is executed (one time) before the execution of the code block.
- Statement 2 defines the condition for executing the code block.
- Statement 3 is executed (every time) after the code block has been executed.

```
v = [13, 24, 37];
for (i=0; i < v.length; i++)
  document.write(v[i]+", ");
/* stampa 13, 24, 37 */
```

Functions definition

Functions are introduced by the keyword `function` and their body is enclosed in a block

- They can be either procedures (no return) or proper functions
- Formal parameters are written without their type declaration

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

function printSum(a,b) {
  window.alert(a+b)
}
```

Function parameters

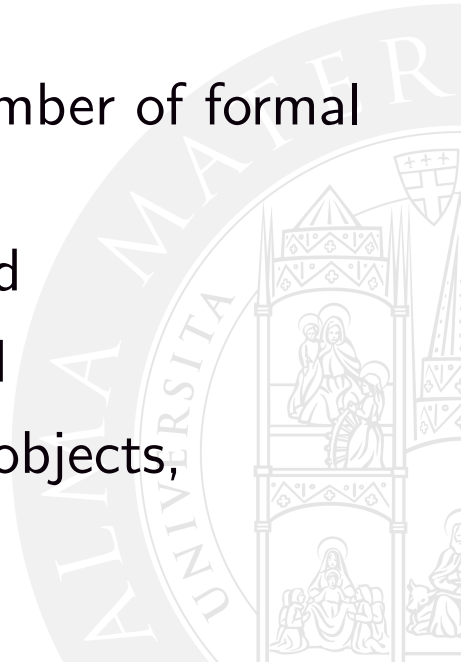
The functions are invoked through the classic call operator (), providing the current parameters:

```
result = sum (18, -3);
```

If the types of the current parameters do not make sense → runtime error

Number of actual parameters can be different from the number of formal ones

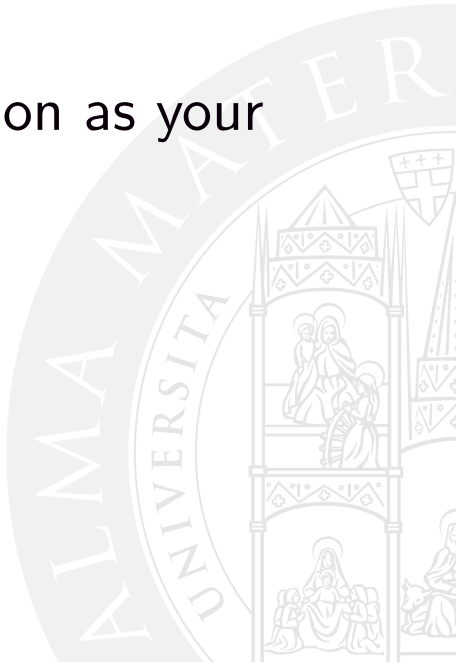
- If more than necessary → extra parameters are ignored
- If less than necessary → missing parameters undefined
- Parameters are always passed by value (working with objects, references are copied)



Focus on...

Exercise 1: Revenue

- Read the revenue (`window.prompt`)
- `if` revenue $<$ 10000 Euro display the message "You must present declaration as your revenue is above 10000 Euro"
- otherwise display "You need not present your declaration as your revenue is below 10000 Euro"

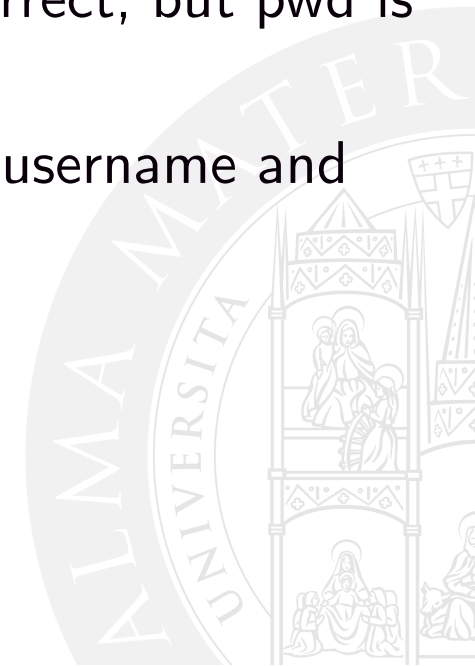


Exercise 2: Date

- Write a JavaScript program to get the current date. (object `Date`)
- Format Output as: mm-dd-yyyy, mm/dd/yyyy or dd-mm-yyyy, dd/mm/yyyy

Exercise 3: Password

- Read Username and Password (`window.prompt`)
- Authorize the user if credential are correct
- Write message "incorrect password" if username is correct, but pwd is wrong
- Write message "incorrect username and password" if username and pwd are both wrong



Exercise 4: Math I

- Write a JavaScript program to get the difference between a given number and 13, if the number is greater than 13 return double the absolute difference.

Exercise 4: Math II

- Write a JavaScript program to compute the sum of the two given integers. If the two values are same, then returns triple their sum.

Exercise 5: Function & Doc Properties

- Write a function `propertyList` that for each object in `document`
- Print its value IF its type is String

```
function propertyList()
{
  for (prop in document )
  {
      if (typeof(document[prop]) === "string")
        document.writeln("Property: " + prop + " value: " + document[prop] + "<br>")
  }
}
```

- Write a function `colorChange` to set background and frontground colors according to user parameters

```
function colorChange()
{
  color = window.prompt("Insert background color (cyan, red, magenta, white, black)");
  document.bgColor=color;
  color = window.prompt("Insert frontground color (cyan, red, magenta, white, black)");
  document.fgColor=color;
}
```

Exercise 6: While

- Generate the multiplication table of a number
- Until a given limit

Exercise 7: Avarage

- Class average program to calculate average of grades
- Use value -1 to stop
- Print the result

Exercise 8: Nested Loops

- Write a JavaScript program to construct the following pattern, using a nested for loop.

```
*
* *
* * *
* * * *
* * * * *
```

Exercise 9: Loop and conditions

- Write a JavaScript for loop that will iterate from 0 to 15
- For each iteration, it will check if the current number is odd or even, and display a message to the screen
- Print the result

Sample Output:

"0 is even"
"1 is odd"
"2 is even"

JavaScript in a few hours?

- Tutorial on the Internet (http://www.google.it, search: JavaScript Tutorial)
- Books: "JavaScript: The Definitive Guide" (David Flanagan, O'Reilly/Apogeo)
- or anything you like...

References I

Aho, A. V. (2011).
Ubiquity symposium: Computation and computational thinking.
Ubiquity, 2011(January).

Brennan, K. and Resnick, M. (2012).
New frameworks for studying and assessing the development of computational thinking.
In *Proceedings of the 2012 annual meeting of the American educational research association, Vancouver, Canada*, volume 1, page 25.

Wing, J. (2011).
Research notebook: Computational thinking—what and why.
The link magazine, 6.

Javascript

Roberta Calegari

roberta.calegari@unibo.it

Alma AI - Alma Mater Research Institute for Human-Centered Artificial Intelligence—Università di Bologna

Master in diritto delle nuove tecnologie e informatica giuridica
Academic Year 2020/2021