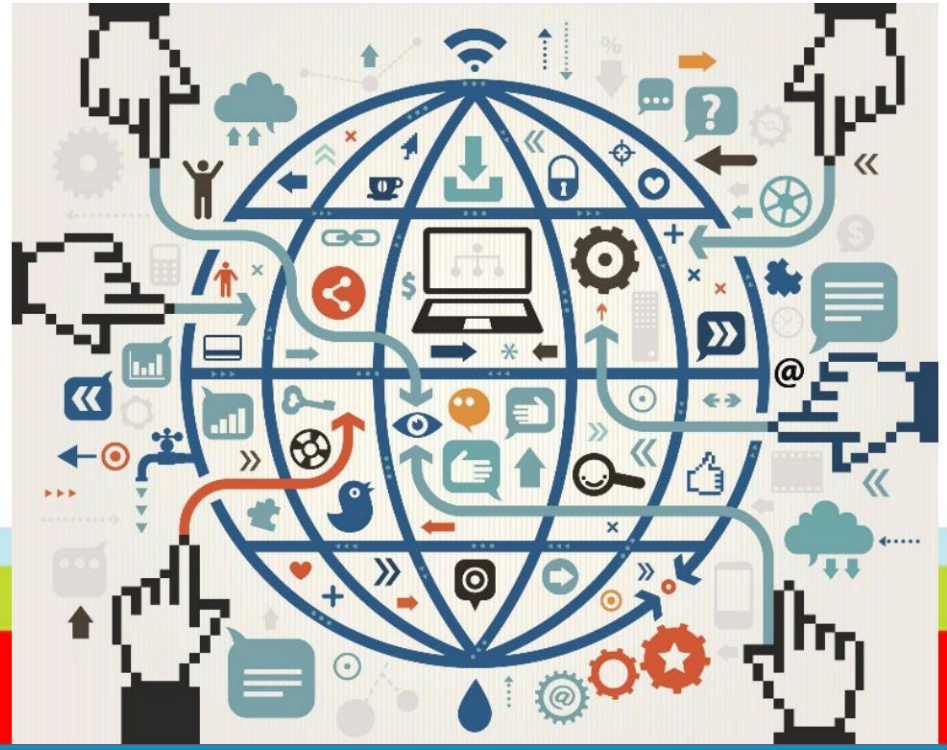
CEU | Universidad San Pablo



JavaScript

Web Systems I

Degree in Information Systems Engineering Álvaro Sánchez Picot

alvaro.sanchezpicot@ceu.es

v20231010

Introduction

- JavaScript (JS)
- Originally created to make websites live
- Not related to Java
- Programs are scripts •

Based on the ECMAScript (ES) specification •

You need a JavaScript engine to run it:

- V8 in Chrome and Opera
- SpiderMonkey in Firefox
- Chackra/ChackraCore in IE/Microsoft Edge
- Nitro and SquirrelFish in Safari

ECMAScript

- Standard for scripting languages •

Describes only the syntax and semantics of the core

  – Each implementation adds other functionalities such as I/O or file management

- Implementations: JavaScript, ActionScript, JScript... •

Important versions:

  – ES6/ES2015: declaration of classes, modules, arrow functions, promises, let, const…
  – ES8/ES2017: async/

await • Latest specification (ES14/ES2023)

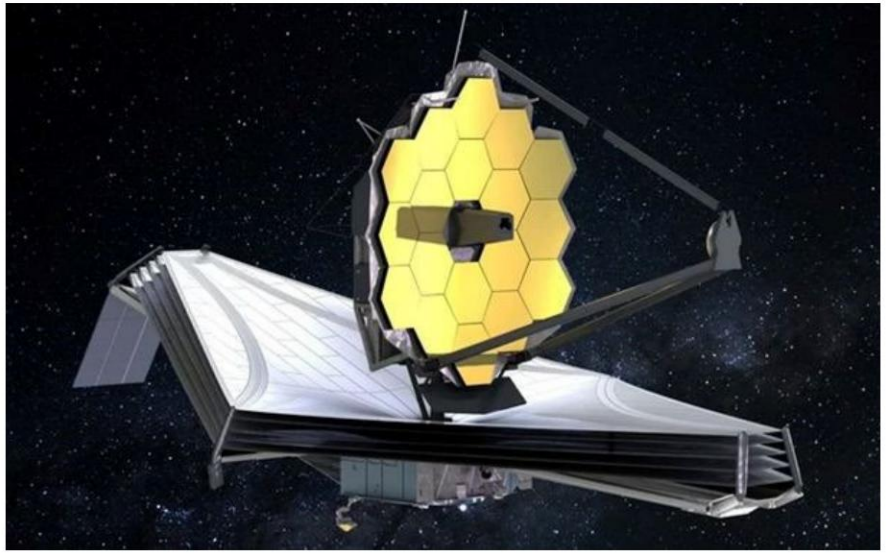
- More information about versions

Introduction

Key Features: • Cross-platform • Object-oriented • Full integration with HTML/CSS • Simple things are easy • Support of almost all browsers • Enabled by default in browsers • Weak typing • Dynamic typing

Introduction

- Used in:
  - Web development
  - – On the server: node.js
  - – On the James Webb Space Telescope (info.)
  - – Other projects such as drone programming



The primary command source in normal operations is the Script Processor Task (SP), which runs scripts written in JavaScript upon receiving a command to do so. The script execution is performed by a JavaScript engine running as separate task that supports ten concurrent JavaScripts running independently of each other. A set of extensions to the JavaScript language have been implemented that provide the interface to SP, which in turn can access ISIM FSW services through the standard task interface ports. Also, to provide communication between independently running JavaScripts, there are extensions that can set and retrieve the values of shared parameters.

Extract from ISIMAnuscript (link)

Interpreted or compiled?

Introduction

- It is an interpreted language
- Modern browsers use Just-In-Time (JIT) compilation:
  - There is a monitor (profiler) that checks the execution of the code
  - Parts of the code that run a lot (warm) are converted to bytecode
  - The compiler performs some optimizations
  - More information

Introduction

- JS in the browser does not have access to OS functions
  - File writing is very limited
- Different tabs/windows generally don't know each other
  - “Same Origin Policy”
- A website can communicate with the server it came from
  - Reception of data from other domains is very limited

Introduction

Transpiled languages •

Languages that have another syntax and are converted to JS •

Examples: –

CoffeScript: Shortest syntax –

TypeScript: strict data types. Developed by Microsoft – Flow: type of data. Powered by Facebook – Dart: standalone language. Developed by Google.

–Brython : Python 3 transpiler for JS.

Introduction

Compatibility: •

Not all features are supported by all browsers • https://caniuse.com/

Introduction

- IDE
  - Visual Studio Code
  - Webstorm
- Lightweight editors
  - Atom
  - Sublime Text
  - Notepad++

Introduction

Development console:

- Google Chrome (Ctrl + Shift + I)
- Firefox, Edge (F12)
- Safari: Within options > Advanced, enable the development menu.

  Cmd + Opt + C

Introduction

- Code embedded in the HTML

```
<script>
   alert('Hello, world!'); </
script>
```

- External script:

```
<script src="/path/to/script.js"></script>
```

Introduction

• Previously it was convenient to place the scripts at the end of body • So that all the previous <body> components have been loaded

...

<script src="/path/to/script.js"></script> </body>

Introduction

- Now it is convenient to add the scripts in <head>
- Use the **defer** attribute

```
<head> …
    <script src="/path/to/script.js" defer></script>
</head>
```

- Or use the **async** attribute

```
<head> …
    <script src="/path/to/script.js" async></script>
</head>
```

Introduction

defer

• It is loaded in parallel with other elements • It

is not executed until the html has been completely parsed

async

• Loaded in parallel with other scripts •

Executed as soon as available

Comments

- From a line with //

```
//This is a comment
```

- Multiline with /* */ (Cannot be nested)

```
/*
Multiline comment
*/
```

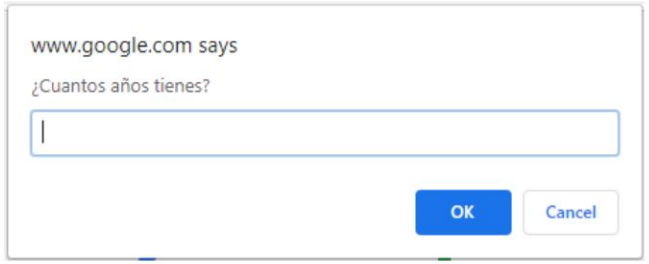
Interaction

- alert
  - Shows a message to the user

```
alert("Hello");
```

- prompt
  - Prompts the user to enter information //

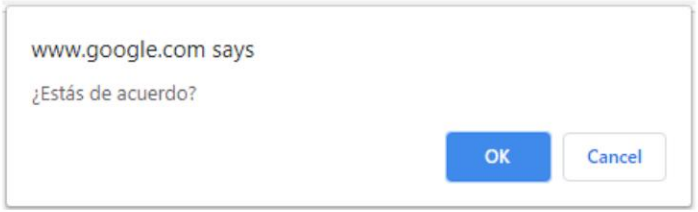
```
prompt(title, [default]); let result =
prompt("How old are you?",""); console.log(result);
console.log(typeof(result)); //
string
```



Interaction

- confirm
  - Asks the user to confirm

```
//result = confirm(question);
let str = confirm("Do you agree?");
console.log(str) //true or false •
```



console.log –
  Print a message to the browser console //

```
console.log(message);
console.log("This is printed to the console");
```

Interaction

• It is advisable not to use alert, prompt or confirm •

They are very intrusive

• They completely block the page

• It is advisable to use more usable alternatives

– Ex.: https://www.toptal.com/designers/ux/notification-design • We will mostly use console.log() for testing

Statements

- Statements
- Separated by ';' although it can be omitted in a line break, sometimes (not recommendable)

```
alert('Hello'); alert('World');
```

```
alert('Hello')
alert('World')
```

```
alert('Hello');
alert('World');
```

Statements

- Statements

```
//Works correctly
console.log(3 +
1
+ 2);

//TypeError: Cannot read property '2' of undefined
alert("There will be an error") [1,
2].forEach(alert)
```

- ASI rules (Automatic Semicolon Insertion)

Variables

- Named data storage

```
letmessage;
message = 'Hello!';
console.log(message);
```

- Multiple declaration:

```
let user = 'John',
    age = 25,
    message = 'Hello';
```

Variables

- You can change the data type, although be careful:

```
letmessage;
console.log(typeof(message)) //undefined message
= 'Hello!';
console.log(typeof(message)) //string
message = 1234
console.log(typeof(message)) //number
```

Variables

Syntax:

- They can only contain letters, digits or $ or • _

The first character cannot be a digit • It is

recommended to use lower camelCase for multiple words

aVariableWithALongName

- List of reserved words

Constants

- Variable that does not change its value

```
const myBirthday = '18.04.1982';
```

- Its value cannot be reassigned

```
const myBirthday = '05/23/1983';
//SyntaxError: Identifier 'myBirthday' has already been declared
```

- A value must be assigned when declaring them

```
const errorOnDeclaration;
//SyntaxError: Missing initializer in const declaration
```

Constants – Conventions

- For known constants, it is recommended that they only contain capital letters and words separated by _

```
const COLOR_RED = "#F00";
```

- For other constants use lower camelCase

```
const pageLoadTime = /* time taken by a webpage to load */;
```

let vs var

- In general do not use

var • var does not have block

scope if (true) {

```
    var test = true;


} alert(test); // true, the variable lives after if
```

```
if (true) {
    let test = true;


} alert(test); // ReferenceError: test is not defined
```

let vs var

- var has function scope

```
function sayHi() { if
    (true) {
        var phrase = "Hello";

    } console.log(phrase); // works

} sayHi();
console.log(phrase); // ReferenceError: phrase is not defined
```

let vs var

- var can be redeclared, let not

```
let user;
let user;
// SyntaxError: Identifier 'user' has already been declared
----
var user = "Pete";
var user = "John";
console.log(user); // John
```

let vs var

- With var it is not necessary to put var when declaring the variable

```
user = "Pete";
console.log(user); //Pete
```

let vs var

- Var variables can be declared after use (hoisting)

```
// Valid in normal mode
number = 1234;
console.log(number);
----
//Invalid in strict mode
 'use strict'
number2 = 1234;
console.log(number2);
//ReferenceError: number2 is not defined
```

let vs var

- Var variables can be declared after use (hoisting) (cont.)

```
// Valid in strict mode
 'use strict'
number3 = 1234;
console.log(number3); //1234
var number3;
-----
number4 = 1234;
console.log(number4); let
number4;
//ReferenceError: Cannot access 'numero4' before initialization
```

Data Types – Number

- For both integers and floating point •

Integer values between –(253-1) and (253-1)

  – BigInt for values outside that range (ending in n)

  const bigInt = 1234567890123456789012345678901234567890n;

- Special numeric values

  – Infinity, -Infinity

  console.log(1/0); //Infinity

  console.log(-1/0); //-Infinity let a =

  Infinity – NaN (Not

  a number)

  console.log("text" / 2); //NaN

Data types – String

- Statement

let str = "Hello"; let str2
= 'Single quotes are ok too';

- Template literals (not supported in Internet Explorer) let
str = `Hello`; let
phrase = `can embed another ${str}`; let
text = `the result is ${1 + 2}`

- There is no character type (char)

Data types – String

• "string".length

console.log("Test text".length) // 15 • "string".charAt(index)

console.log("Test

text".charAt(2)) // x • " string".split(pattern) const result =

"Test text".split(" "))

console.log(result) // ['Text', 'of', 'test'] const [p1, p2, p3]

= "Test text".split(" ")

Data types – String

• "string".slice(start, [end])

console.log("Test text".slice(6)) // test console.log("Test text".slice(6,8)) / / of • More operations

Data types – boolean

- Statement

```
let nameFieldChecked = true; let
ageFieldChecked = false;
```

- They can be the result of a comparison

```
let isGreater = 4 > 1;
```

Data types – null

- Statement

let age = null; •

Not a reference to an object that does not exist •

Special value that represents nothing, empty, unknown value •

typeof returns object for historical compatibility, even if typeof(null); // object

Data types – undefined

- Represents that no value has been assigned
- Statement

```
let age;
console.log(age); //undefined let
name = undefined; //We shouldn't do this
```

Data Types – Conversions

- Converting to String using String()

```
console.log(String("23")); //'23'
console.log(String(true)); //'true'
console.log(String(false)); //'false'
console.log(String(undefined)); //'undefined'
console.log(String(null)); //'null'
```

Data Types – Conversions

- Converting to Number using Number()

```
console.log(Number("23") + 1); //24
console.log(Number(" 23 ")); //23
console.log(Number("")); //0
console.log(Number("thirteen")); //
NaN console.log(Number(true)); //
1 console.log(Number(false)); //
0 console.log(Number(undefined)); //
NaN console.log(Number(null)); //0
```

Data Types – Conversions

- Converting to Number using +

```
let apples = "2";
let oranges = "3";
console.log(+apples + +oranges); //5
```

Data Types – Conversions

- Converting to boolean using Boolean()

```
console.log(Boolean(0)); //false
console.log(Boolean(1)); //true
console.log(Boolean(11234)); //true
console.log(Boolean("")); //false
console.log(Boolean("Hello")); //true
console.log(Boolean("0")); //true
console.log(Boolean(" ")); //true
console.log(Boolean(null)); //false
console.log(Boolean(undefined)); //false
```

Data Types – Conversions

- Explicit conversions

```
console.log("2" + 2); //22
console.log(4 + 5 + "px"); //9px
console.log("$" + 4 + 5); //$45
console.log("2" / "5"); //0.4
console.log(6 – "2"); //4
console.log(false + 34); //34
console.log(true + 34); //35
console.log(true + "34"); //true34
```

Operators

- Sum +
- Subtraction -
- Multiplication *
- Division /
- Rest %

```
console.log(5 % 2); //1
```

- Exponent **

```
console.log(2 ** 3); //8
```

- Increment / decrement ++/--

Operators

- Bit operators
  - AND &
  - OR |
  - XOR ^
  - NOT~
  - LEFT SHIFT <<
  - RIGHT SHIFT >>
  - ZERO-FILL RIGHT SHIFT >>>
- Priority order

Comparisons

- Greater than >
- Less than < •

Greater than or equal to

>= • Less than or equal to

<= • Equal ==

- Different !=

Comparisons

- Strings
  - Letter by letter
  - According to Unicode order

```
console.log('Z' > 'A'); // true console.log('Glow' >
'Glee'); // true console.log('Bee' > 'Be'); // true console.log('a'
> 'A'); // true console.log('Á' > 'a'); // true
```

Comparisons

- Different types
  - Converted to numbers

```
console.log('2' > 1); // true
console.log('01' == 1); // true
console.log(true == 1); // true
console.log(false == 0); // true
console.log('0' == 0); // true
console.log(Boolean('0') == Boolean(0)); // false console.log(null
== undefined); //true
```

Comparisons

- Different types
  - If it cannot be converted to a number, it returns false

```
console.log('Hello' > 34); // false
console.log('Hello' < 34); // false
console.log('Hello' == 34); // false
```

Comparisons

- Different types
  - – So that the conversion is not done, use strict equality === – If the types are different, false is returned with ===

```
console.log('01' === 1); // false
console.log(true === 1); // false
console.log(false === 0); // false
console.log('0' === 0); // false
console.log(null === undefined) //false
```

Comparisons

- Different types
  - Strange cases with null and undefined compared to 0
  - Treat these cases with care

```
console.log(null > 0); // false
console.log(null == 0); // false
console.log(null >= 0); // true
console.log(undefined > 0); // false
console.log(undefined < 0); // false
console.log(undefined == 0); // false
```

Conditionals – if

- The expression is evaluated and converted to boolean if (year == 2015) console.log('You are right!');

```
if (year == 2015)
    { console.log('That's correct!');
    console.log('You're so smart!');
} else if (year == 2016)
    { console.log(); }
else
    { console.log();
}
```

Conditionals – if

- …?…:… (conditional ternary operator)

```
let accessAllowed = (age > 18) ? true : false;

let accessAllowed2 = age > 18 ? true : false; //Not recommended

let message = (age < 3) ? 'Hi baby!' : (age < 18) ?
    'Hello!' : (age < 100) ?
    'Greetings!' : 'What an unusual age!';


(company == 'Netscape') ?
     alert('Right!') : alert('Wrong.'); // Not recommended
```

Conditionals – Logical Operators

- OR ||
  - – Evaluated from left to right
  - – Returns the value of the first operand that evaluates to true
  - – If the end is reached, the value of the last operand is returned

```
let firstName = ""; let
lastName = ""; let
nickName = "SuperCoder";
console.log( firstName || lastName || nickName || "Anonymous"); //
SuperCoder
```

  - – Short circuit

```
true || alert("not printed"); false ||
alert("printed");
```

Conditionals – Logical Operators

- AND &&
  - – Evaluated from left to right
  - – Returns the value of the first operand that evaluates to false
  - – If the end is reached, the value of the last operand is returned

```
alert( 1 && 0 ); // 0 alert( 1
&& 5 ); // 5 alert( null &&
5 ); // null alert( 0 && "no matter
what" ); // 0
```

  - – Short circuit

```
true && alert("printed"); false
&& alert("not printed");
```

Conditionals – Logical Operators

- NOT !
  - Convert the operand to boolean
  - Returns the inverse value

```
alert( !true ); // false alert( !
0 ); // true
```

  - Sometimes double is used to convert to boolean instead of Boolean()

```
let test = "non-empty string";
let test2 = "";
alert( !!test ); // true alert( !!
test2 ); // false alert( !!null ); //
false
```

Conditionals – Logical Operators

- Nullish coalescing ??

  - to ?? b ÿ returns *a* if it is defined (it is not null or undefined) and if not *b* –

  Provides a default value to a variable – Not

  supported by all browsers (check).

```
result = (a !== null && a !== undefined) ? a:b;
result = a ?? b
```

  – For security reasons it cannot be used together with && or || without parentheses

```
let x = 1 && 2 ?? 3; // SyntaxError: Unexpected token '??'
```

- More information about operators

Conditionals – switch

• Replaces multiple ifs with strict equality === • Starts executing from the first condition true • break to end the execution of the switch • default equivalent to else

```
switch (a) {
    case 4:
        console.log( 'Only 4 is executed' ); break;

    case 5:
        console.log( '5 and default are executed' );
        break;
    default:
        console.log( "Default" );
}
```

Conditionals – Exercise

- What is the result for each one?

```
alert( 1 && null && 2 );
alert( null || 2 && 3 || 4 ); if (-1 ||
0) alert( 'first' ); if (-1 && 0)
alert( 'second' ); if (null || -1 && 1)
alert( 'third' ); alert( alert(1) && alert(2) );
```

Loops

• while is executed as long as the condition is true let i =
3; while (i)
{ alert(i);
    Yo--;

}
---

let i = 3;
while (i) alert(i--);

Loops

- do..while is executed at least once

```
let i = 0;
do
    { console.log( i ); i+
    +; }
while (i < 3);
```

Loops

- for

```
for (let i = 0; i < 3; i++) {
    console.log(i);

} console.log(i) //ReferenceError: i is not defined
----
let i = 0; for
(; i < 3;) { alert( i++ );

}
```

Loops

- break
  - Force loop termination

```
let sum = 0; while
(true) {
    let value = +prompt("Enter a number", ''); if (!value) break;

    sum += value;

} console.log( 'Sum: '                + sum );
```

Loops

- continue
  - Forces the loop to continue with the next iteration

```
for (let i = 0; i < 10; i++) { if (i % 2 ==
    0) continue; console.log(i); // 1,
    then 3, 5, 7, 9
}
```

Loops

- labels
  - Identify the loop
  - References for break / continue
  - They cannot be used to jump anywhere

```
outer:
for (let i = 0; i < 3; i++) {
    for (let j = 0; j < 3; j++) { let input =
        prompt(`Value at coords (${i},${j})`, ''); if (!input) break outer; // do
        something with the value...

    }

} alert('Done!');
```

Features

- Main building blocks • Code reuse

- Variables declared inside are not accessible outside

```
function showMessage() { let
    message = 'Hello everyone!';
    console.log(message);

} showMessage();
showMessage();
console.log(message); //ReferenceError: message is not defined
```

Features

- Exterior variables are accessible and modifiable inside let

```
userName = 'John';
function showMessage() {
    userName = "Bob";
    let message = 'Hello, '              + userName;
    console.log(message);

} console.log( userName ); //John
showMessage();
console.log( userName ); // Bob
```

Features

- If the same variable is declared inside and outside, the inside shadows the outside
- The same thing happens in conditionals and loops

```
let userName = 'John';
function showMessage() { let
    userName = "Bob"; let
    message = 'Hello, '                + userName;
    console.log(message);

} console.log( userName ); // John
showMessage();
console.log( userName ); // John
```

Functions – Parameters

- The function can be called with fewer parameters and then it is considered undefined

```
function showMessage(from, text) {
    console.log(from + ': '              +text);
}
showMessage('Ann', 'Hello!'); //Ann: Hello!
showMessage('Ann'); //Ann: undefined
```

Functions – Parameters

• A default value can be included function

```
showMessage(from, text = 'no text') { + text );
    alert( from + ': '

} showMessage('Ann', 'Hello!'); //Ann: Hello!
showMessage('Ann'); // Ann: no text
```

Functions – Parameters

- The default can be a call to another function function

```
showMessage(from, text = anotherFunction()) { //
    anotherFunction() only executed if no text given
    // its result becomes the value of text
}
```

Functions – Parameters

- Variable number of parameters

```
function foo() {
    for (let i = 0; i < arguments.length; i++) {
        console.log(arguments[i]);
    }

} foo('Hello');
foo(1,2,3,4,5,6,7,8,9);
```

Functions – Parameters

- Variable number of parameters

```
function my_log(x, ...args) {
        console.log(x, args, ...args);

} my_log('Hello', 'what', 'so');
//Hello ['what', 'so'] how are you
```

Functions – return

- To return a result in the function
- If it does not exist, the function returns

undefined • Empty (return;) can be used to terminate the execution of the function function

```
    checkAge(age) { if (age >= 18) {
        return true;
    }
        return confirm('Do you have permission?');

} let age = prompt('How old are you?', 18); let valid
= checkAge(age);
```

Functions – Expressions

• You can assign a function to a variable • Add the ; at the end let

```
sayHi = function()
   { console.log( "Hello" ); };

console.log(sayHi); //the code of the function is printed
console.log(sayHi());// Hello\nundefined
```

Functions – Expressions

• A function declaration can be used before, an expression, not

```
sayHi("John"); // Hello, John
greet("John"); // ReferenceError: greet is not defined function
sayHi(name)
    { alert( `Hello, ${name}` );

} let greet = function(name)
    { alert( `Hello, ${name}` ); };
```

Functions – Arrows

- Equivalent to lambda functions
- Anonymous functions
- Concise way to declare a function

let sum = (a, b) => a + b; • Code

almost equivalent to let sum =

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

Functions – Arrows

- You can use multiple lines with {}, in that case you need to return

```
let sum = (a, b) => {
    let result = a + b;
    return result;
};
console.log( sum(1, 2) ); // 3
```

Arrays

• Create arrays: let fruits = ['Apple', 'Banana']; • Access an element

```
console.log(fruits[0]);
```

• Length

```
console.log(fruits.length);
```

• Go through it

```
fruits.forEach(function(item, index, array) {
    console.log(item, index); });
```

Arrays

• Add an element

let newLength = fruits.push('Orange'); • Delete the last element

let last = fruits.pop();

• Find an index of an element

let pos = fruits.indexOf('Banana');

• Delete an item

let removedItem = fruits.splice(pos, 1); • More information

Objects

• Store collections of various data

let user = new Object(); // "object constructor" syntax let user = {}; //
"object literal" syntax • Can be initialized with data

using key: value separated by comma

let user =

{ name: "John",

age:

30 }; • To access information

console.log(user.name);

console.log(user["name"]);

Objects

- You can add new properties let user =

```
{
    name: "John",
    age: 30
};
user.isAdmin = true;
```

- Properties can be deleted with delete

```
delete user.age;
```

Objects

- A property can have more than one word, but then it must use quotes

```
let user = {
    name: "John",
    age: 30,
    "likes birds": true
};
```

- These properties must be accessed with []

```
console.log(user["likes birds"]);
console.log(user["age"]);
```

Objects

- Check if a property exists let user =
{};
console.log(user.noSuchProperty === undefined); // true

- You can use the *in* operator which returns true if it exists
console.log("noSuchProperty" in user); // false let
key = "age";
console.log(key in user); // false

Objects

- Loop through

objects let user = {

```
    name: "John",
    age: 30,
    isAdmin: true
};
for (let key in user) {
    alert(key); // name, age, isAdmin
    alert(user[key]); // John, 30, true
}
```

Objects

- Objects can be nested let user

```
= {
    name: "John",
    birthday: {
        year: 1990,
        month: "November"


} }; console.log(user.birthday.year);
console.log(user["birthday"]["month"]);
```

API

- There are a lot of predefined interfaces and APIs (list) • Window object
  - Global variable available in the browser
  - Offers many methods and objects, accessible without using the window object
  - document: the HTML document
  - alert(), prompt()…
  - Information about the window: screenX, screenY, scrollX, scrollY…
  - More information

API

- There are a lot of predefined interfaces and APIs (list) •

onload property

  – By when the resource has been loaded

```
window.addEventListener('load', (event) => {
    init(); }); //
```

Alternatively but not recommended window.onload = function() { init(); };

API

- There are a lot of predefined interfaces and APIs (list)
  - onclick property

```
<button id="button">Click me</button>
----

let element = document.getElementById("button");
element.addEventListener("click", myScript); //Alternatively
but not recommended element.onclick = function()
{myScript};
```

API

- Access the document

```
let markup = document.documentElement.innerHTML;
```

- Access elements

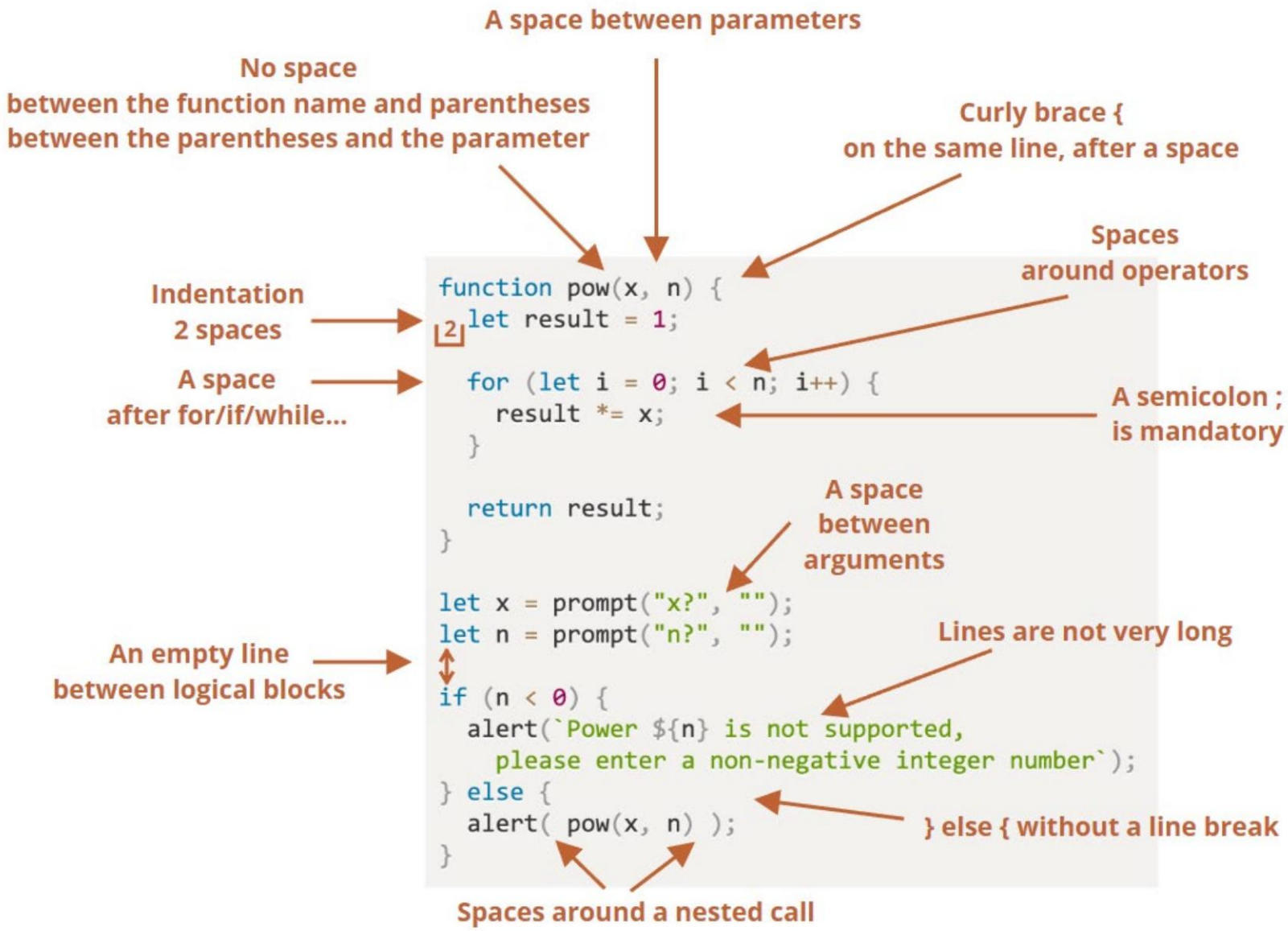
```
let myElement = document.getElementById("enter"); let x =
document.getElementsByTagName("p"); let x =
document.getElementsByClassName("enter"); let x =
document.querySelectorAll("p.intro");
```

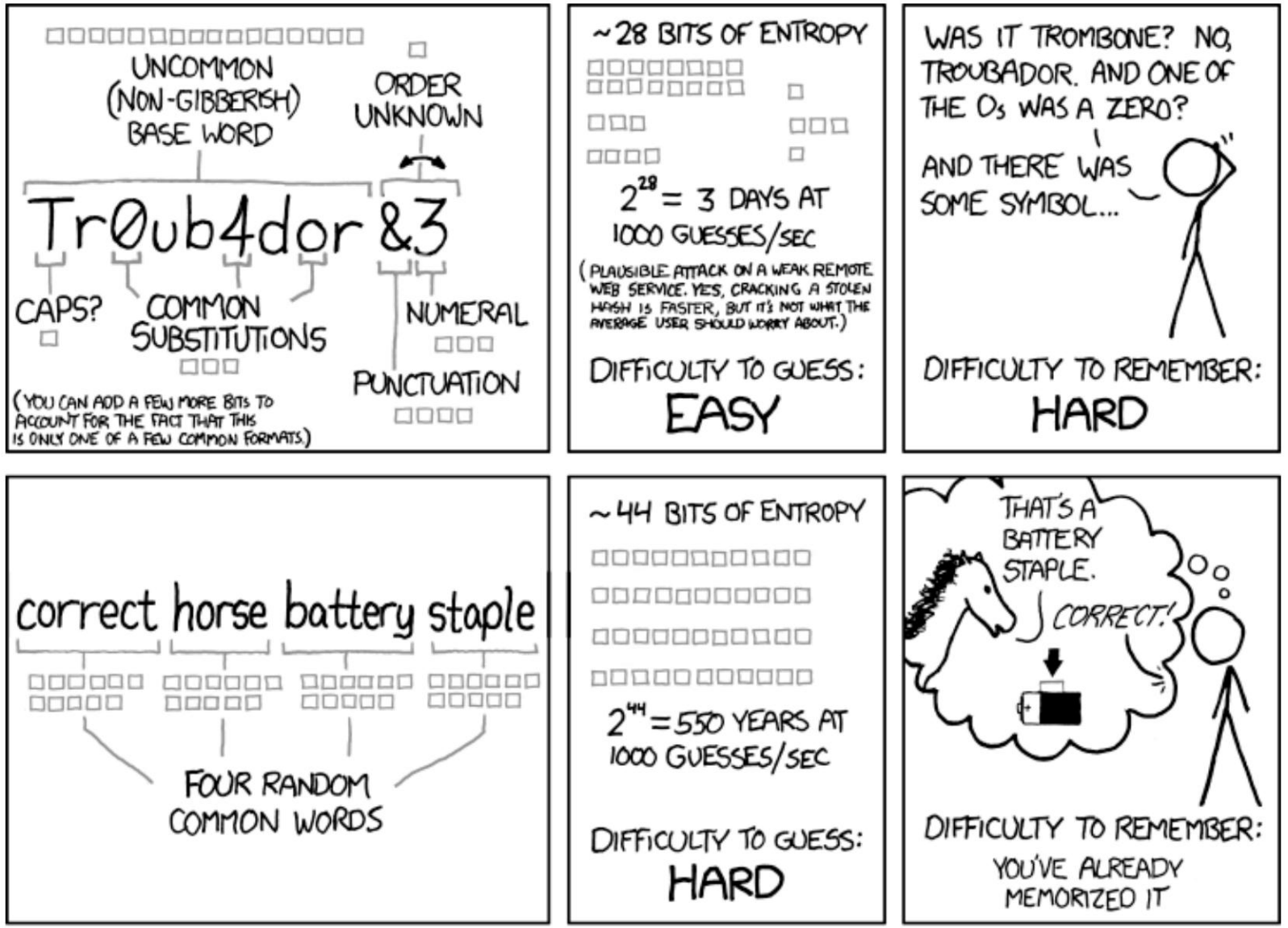
- Add new elements

```
let element = document.getElementById("new");
element.appendChild(tag);
```

Style

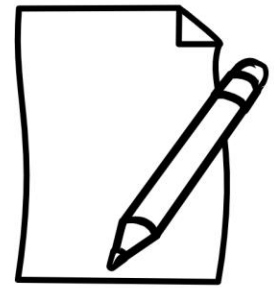
Fountain





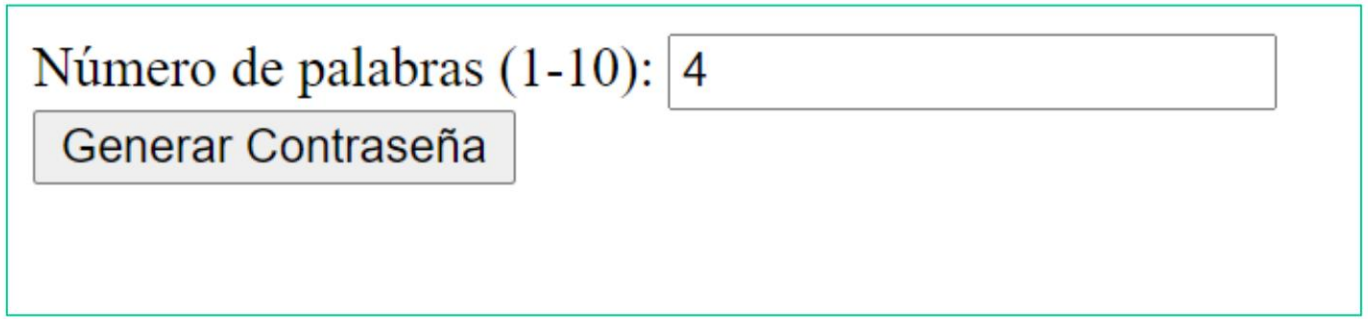
THROUGH 20 YEARS OF EFFORT, WE'VE SUCCESSFULLY TRAINED EVERYONE TO USE PASSWORDS THAT ARE HARD FOR HUMANS TO REMEMBER, BUT EASY FOR COMPUTERS TO GUESS.

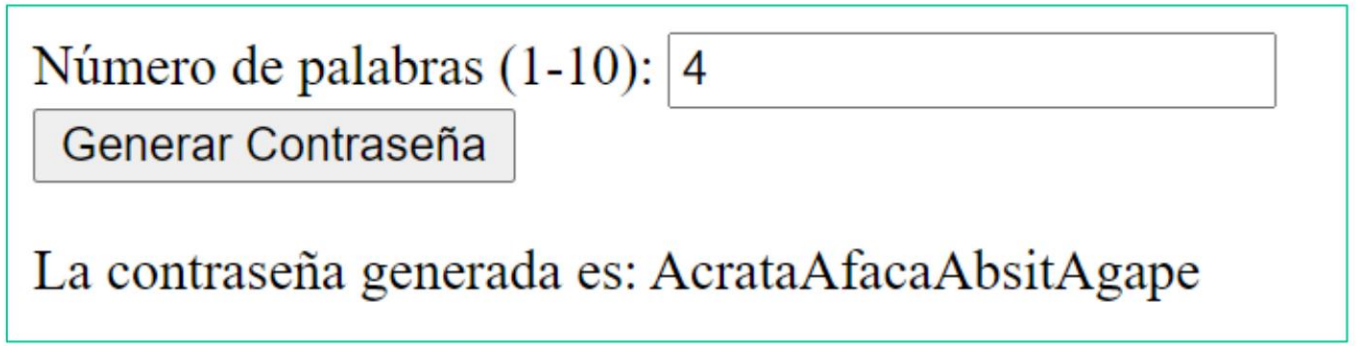
JS – Exercise 1



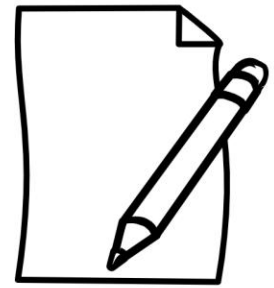
- Design a website that generates a password
- Look up a dictionary and save the information in a variable
- Use the dictionary to generate the password with multiple words random
- Add some configuration option:
  - Number of words
  - Start each word in capital letters
  - Do not repeat words
  - ...
- Optional: Save the dictionary in a file

JS – Exercise 1





JS – Exercise 2



This web page prevents us from selecting the text

• What is happening? •

How do we avoid it? •

Why doesn't the select text cursor appear?