CSE104 Web Programming

JavaScript

Introduction to JavaScript

Why JavaScript

So far, we use HTML and CSS to describe webpages

HTML/CSS have limitations:

- No loops

  ex. Drawing a set of images stored in a directory

- No conditionals

  ex. Specific action when the user click on a button

JavaScript $\Rightarrow$ A real programming language allowing to add dynamic responses to webpages

Example of JS use

HTML

```
<h1>Click on the Balloon ! </h1>
<img class="ballon1" src="balloon.jpg">
<img class="ballon2" src="balloon.jpg">
<img class="ballon3" src="balloon.jpg">
```

Click on the Balloon !

JavaScript

```
document.querySelector(".ballon1").addEventListener('click', actionExplode);
document.querySelector(".ballon2").addEventListener('click', actionExplode);
document.querySelector(".ballon3").addEventListener('click', actionExplode);

function actionExplode(event) {
  const element = event.currentTarget;
  element.src = "bang.jpg";
}
```

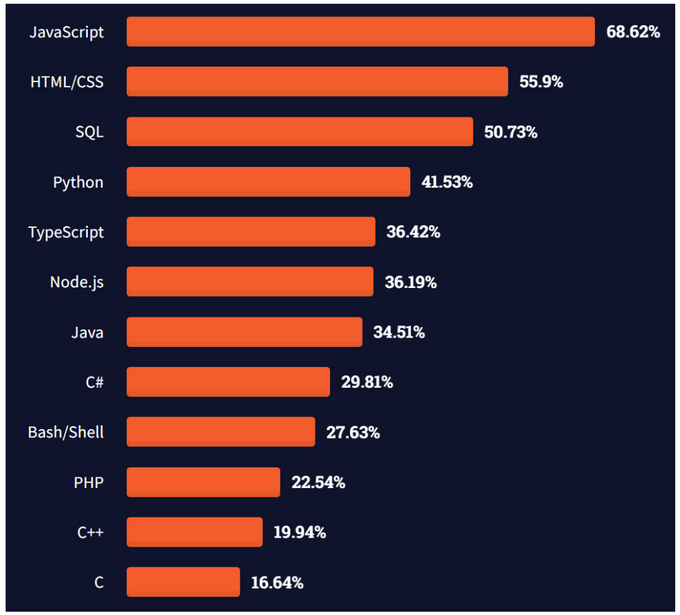
Quick history

- Created in 1995
- A scripting language to add dynamic behavior in the Netscape Navigator.
- Specification written in 10 days by Brendan Eich (co-founder of the Mozilla project)
- Initial objective: Quick and dirty prototype oriented scripting language
- "Unexpected" side effect:
  - Has been implemented in all web browser.
  - Wide diffusion $\Rightarrow$ Today the fundamental language for web programming.
  - Became integrant part of the Web, like HTML and CSS.

Current state

N.1 used programming language

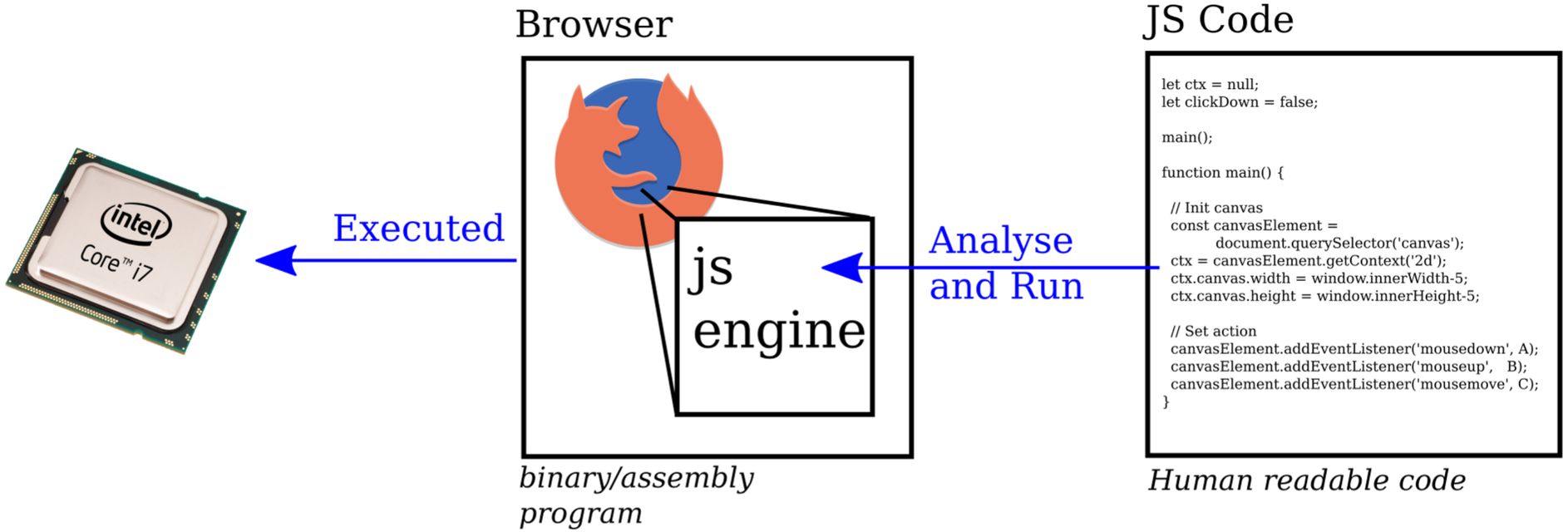
Cannot avoid JS for web application



Notes

- Do not get confused b/w JavaScript and Java
  - Two different languages
  - Mostly marketing reasons (Java was very popular in 1995, including "Java" in the name allowed to gain popularity).
- JavaScript is not a well-thought language for large scale programs.
  - Remember its main purpose was: Prototype scripting for dynamic web.
  - Frameworks can help to design larger softwares.

JavaScript: An Interpreted Language

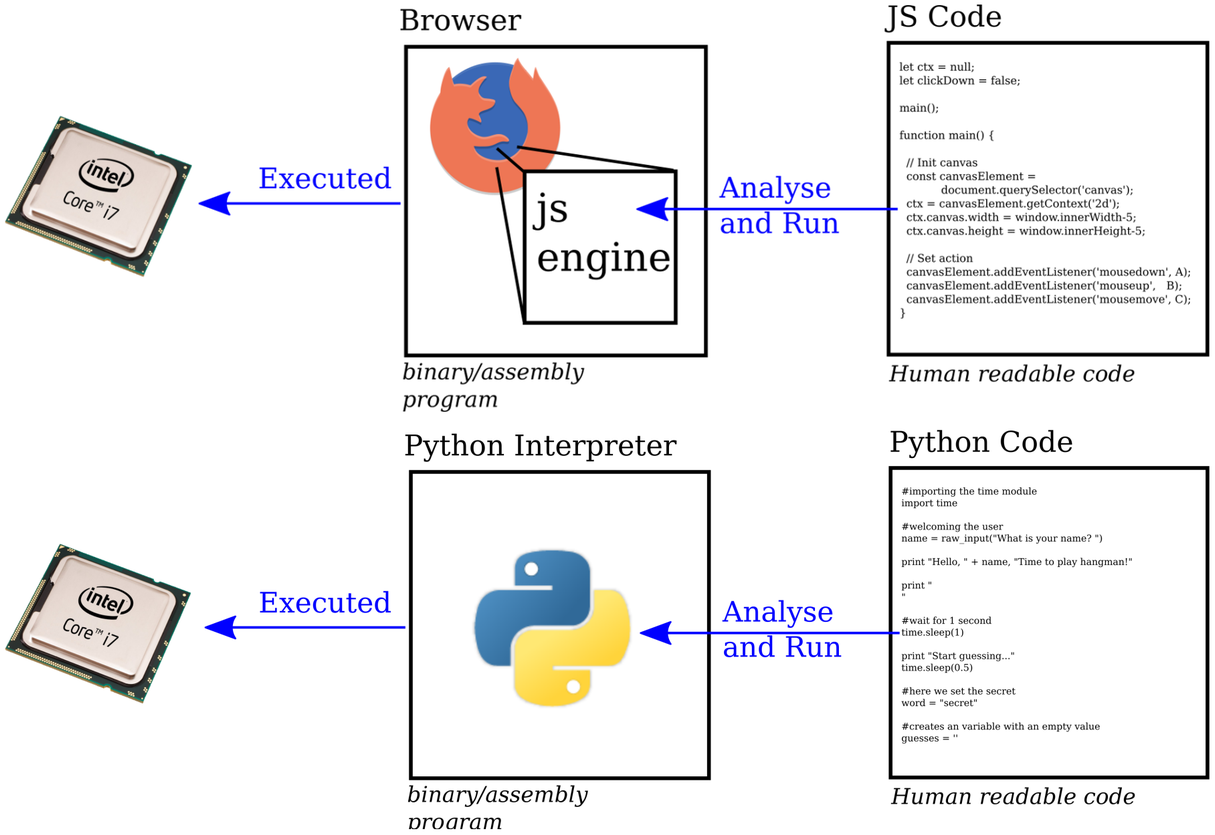


The browser (the JS engine) is parsing, analysing, and running the statements of the js code

Examples of JS engine: SpiderMonkey (Firefox), V8 (Google Chrome)

JavaScript: An Interpreted Language

Similar to Python

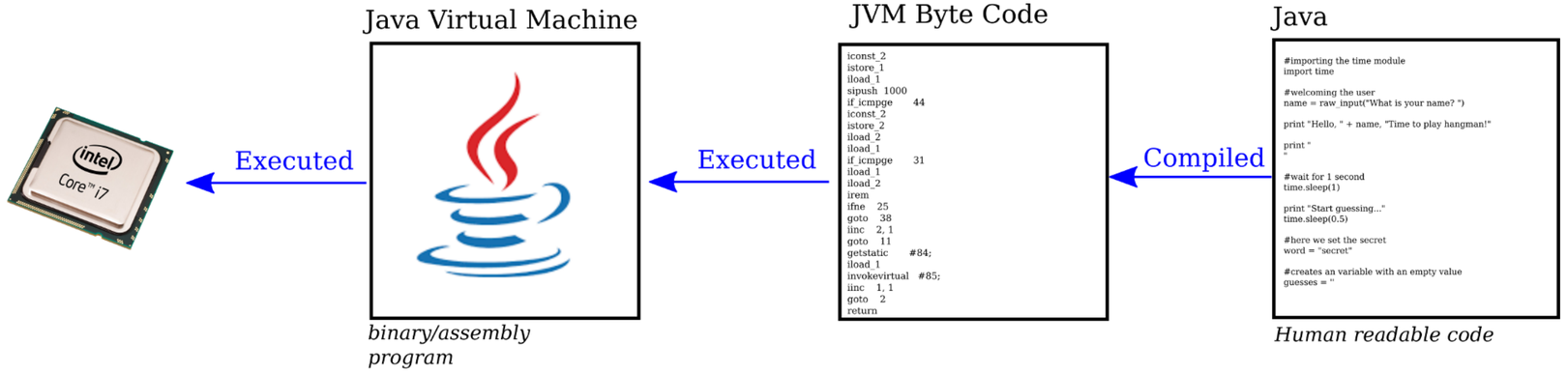


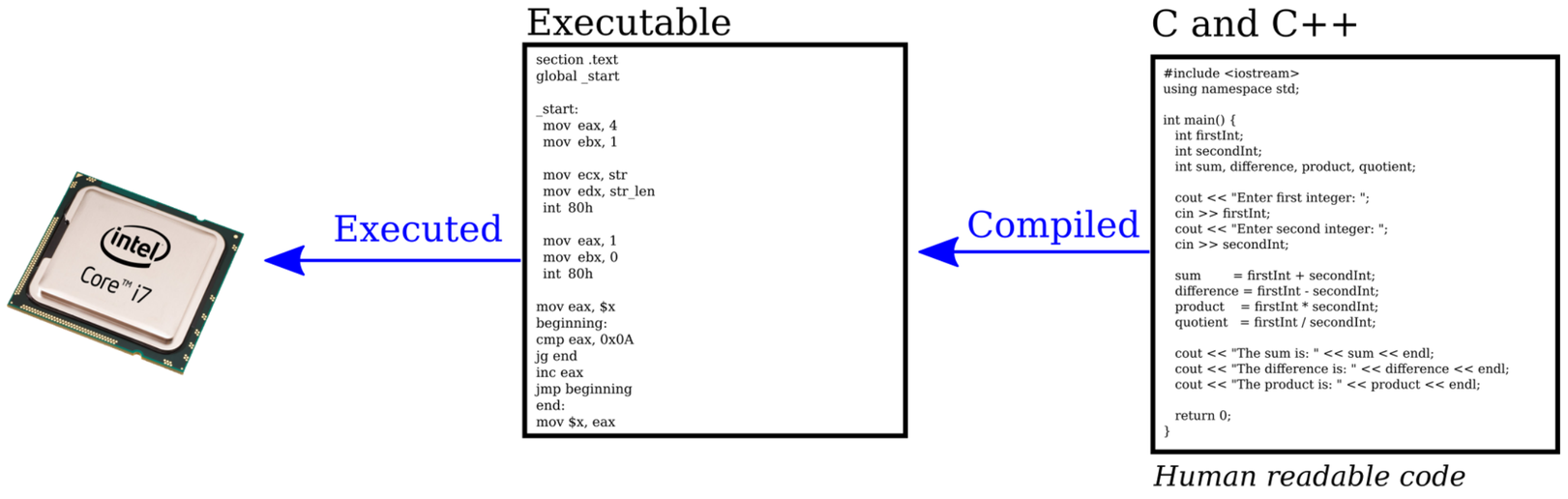
(+) Very dynamic

(-) Slow, no static validity check

JavaScript: An Interpreted Language

Comparison with Java and C





(+) Faster, static check

(-) Lower level, less dynamic

First steps in JavaScript

Running JavaScript code

Need a third file: *script*.**js**

index.html

```
<!DOCTYPE html>
<html>

<head>
        <title> CSE104 </title>
        <script src="script.js" defer></script>
</head>

<body>
        <h1>Content of Webpage</h1>
</body>

</html>
```

script.js

```
console.log("Hello!");
```

Similar to CSS: include link within HTML head

- < script > tag to include code
- src=" ... " the path to the JavaScript file
- defer: execute the script when the HTML page is parsed

Avoid other inline JS inclusion

Complete header template

```
<!DOCTYPE html>

<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="stylesheet" type="text/css" href="style.css">
    <script src="script.js" defer></script>
    <title> My Webpage </title>
</head>

<body> </body>

</html>
```

Include both

- CSS file *style*.**css**
- JavaScript file *script*.**js**

Display JavaScript output

HTML

```
<h1>Content of Webpage</h1>
```

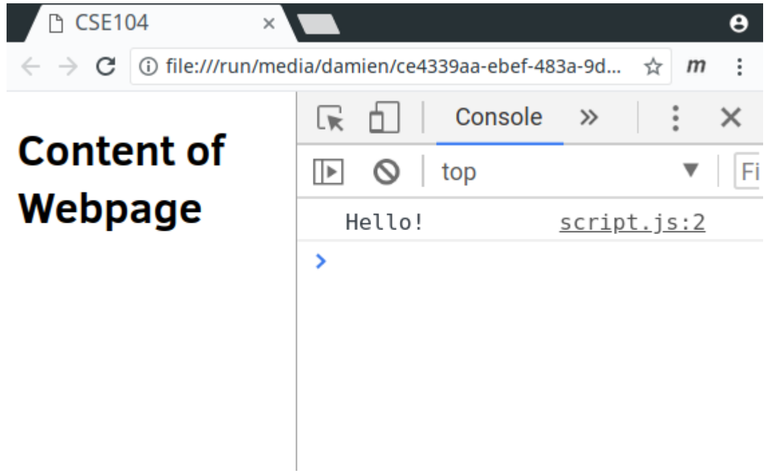
- No visual output of JavaScript
- Print on the browser' console

  console.log('...')

  equivalent to print('...') in Python

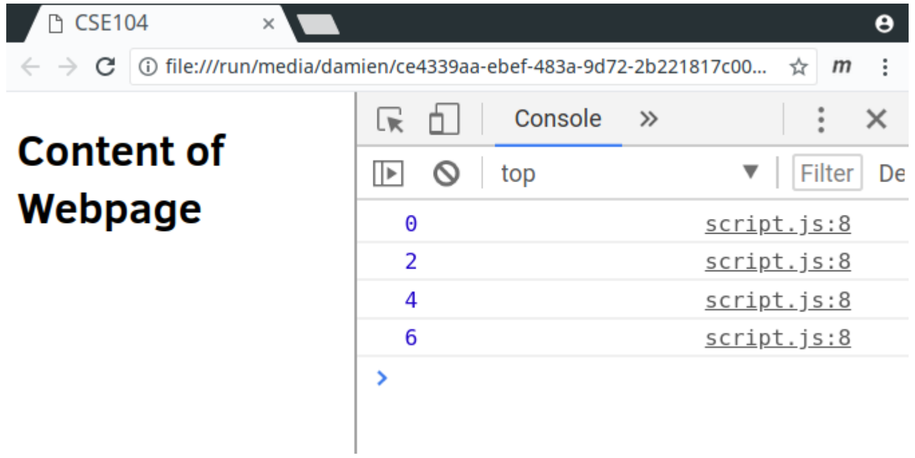
JS

```
console.log("Hello");
```



Loops, conditionals

```
const N=7;
let k=0;
for (k = 0; k < N; k++) {
  if( k%2 === 0 ) {
    console.log( k );
  }
}
```



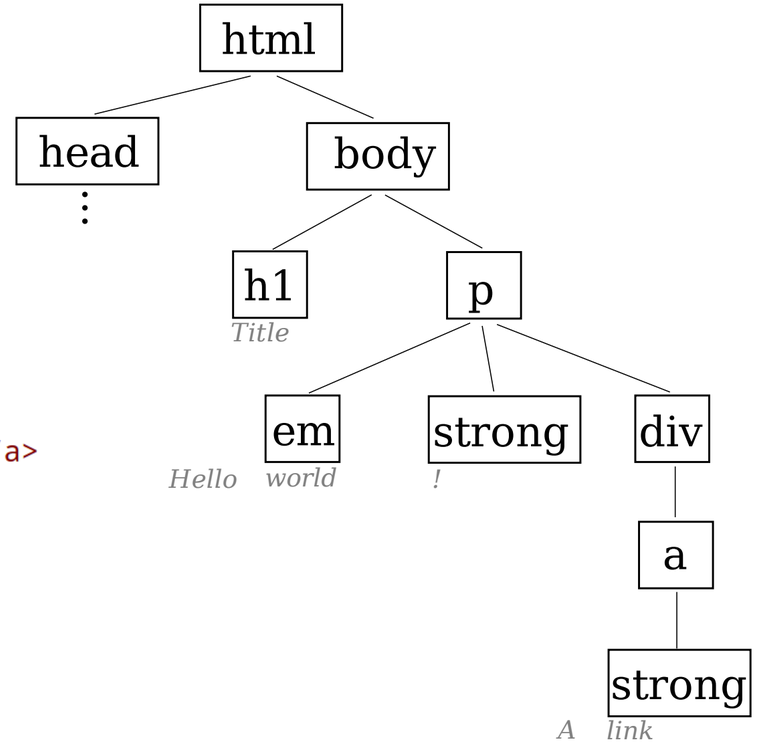
- **let**: define a variable (can be re-assigned)
- **const**: define a non re-assignable variable

Avoid *var*: outdated JavaScript, bad scoping

Modifying webpage through DOM in JS

DOM: Document Object Model

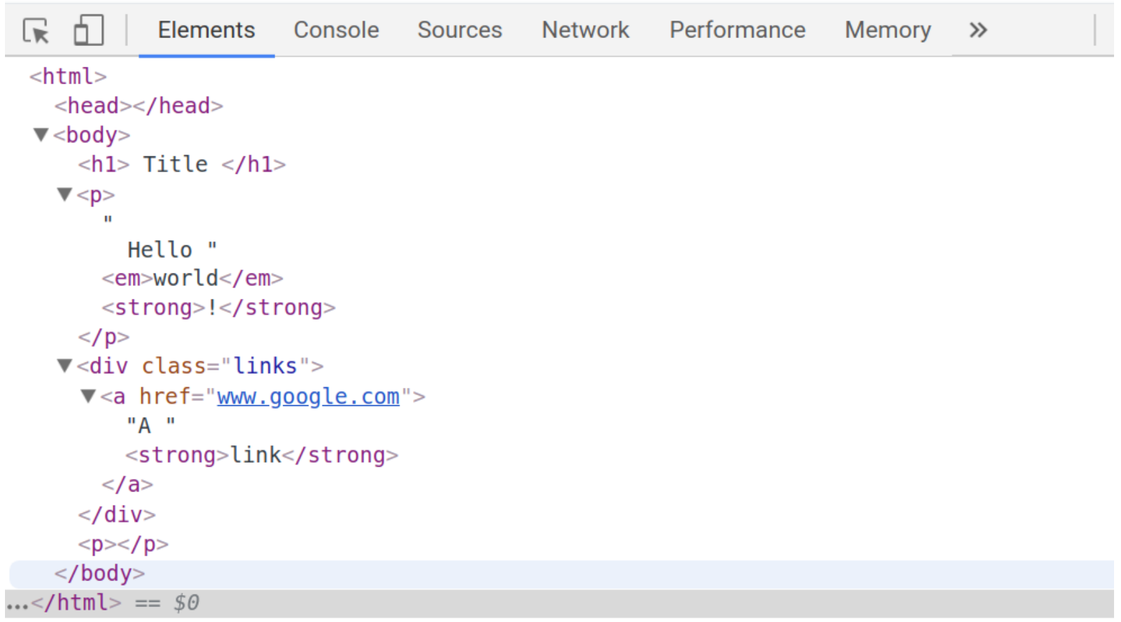
```
<html>

<head></head>

<body>
<h1> Title </h1>

<p>
  Hello <em>world</em> <strong>!</strong>
  <div class="links">
    <a href="www.google.com">A <strong>link</strong> </a>
  </div>
</p>
</body>

</html>
```



- HTML tags are stored as a tree structure
- Hierarchical organization called DOM (similar to XML)

DOM is visible in the developer mode

```
<html>

<head></head>

<body>
<h1> Title </h1>

<p>
  Hello <em>world</em> <strong>!</strong>
  <div class="links">
    <a href="www.google.com">A <strong>link</strong> </a>
  </div>
</p>
</body>

</html>
```



DOM access from JavaScript

With JavaScript, you can: Access, modify, naviguate through DOM elements

HTML

```html
<h1> My Title </h1>

<p>
  Hello <em>world</em> <strong>!</strong>
  </p><div class="links">
    <a href="https://google.com">A
      <strong>link</strong> </a>
  </div>
<p></p>
```

JS

```js
const element = document.querySelector("h1");

console.log(element.textContent);
```

Display on the console:

```
> My Title
```

document: is a global build-in variable. Represent the loaded webpage

querySelector: Select the (first) element matching the parameter

follows CSS selector syntax.

Modifying a webpage

```html
<h1> Title </h1>

<p>
  Hello <em>world</em> <strong>!</strong>
  </p><div class="links">
    <a href="">A
      <strong>link</strong> </a>
  </div>
<p></p>
```

```javascript
const element = document.querySelector("h1");
element.textContent = "Modified in JavaScript !"
```

Modified in JavaScript !

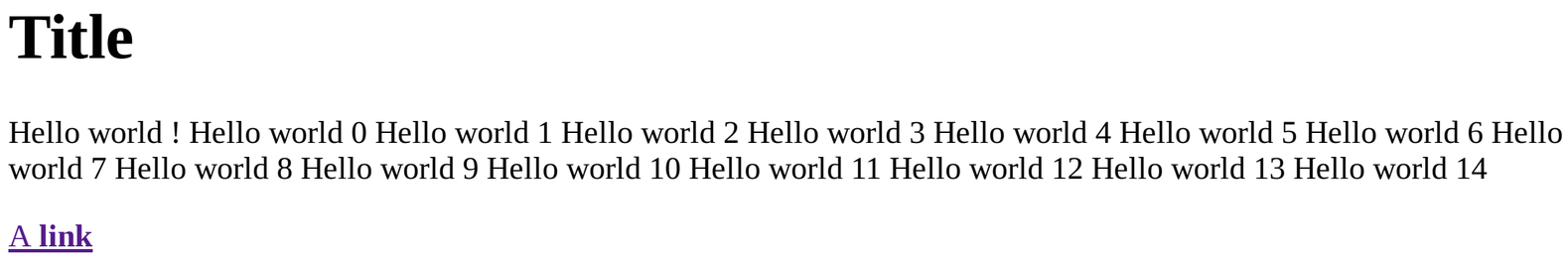
Hello *world* !

A **link**

Modifying a webpage with a loop

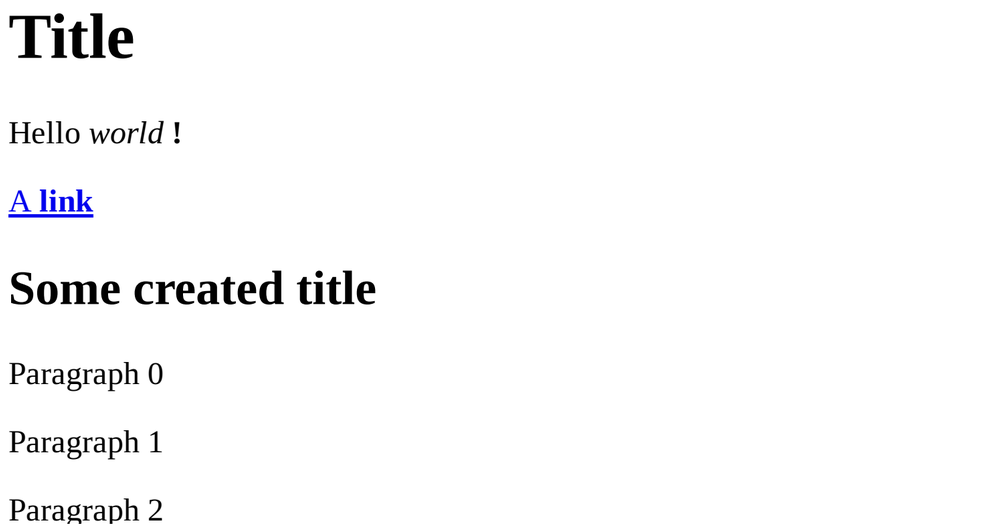
```
<h1> Title </h1>

<p>
  Hello <em>world</em> <strong>!</strong>
  </p><div class="links">
    <a href="">A
      <strong>link</strong> </a>
  </div>
<p></p>
```

```
const element = document.querySelector("p");
for(let k = 0; k < 15; k++) {
  element.textContent += "Hello world "+k+" ";
}
```



Adding elements

```
const newElement = document.createElement("h2");
newElement.textContent = "Some created title";

const parentElement = document.querySelector("body");
parentElement.appendChild(newElement);

for(let k=0; k<10; k++) {
  const paragraphElement = document.createElement("p");
  paragraphElement.textContent = "Paragraph "+k;
  document.querySelector("body").appendChild(paragraphElement);
}
```



Changing style

```
const titleElement = document.querySelector("h1");
titleElement.style = "background-color:lightblue;"

const emElement = document.querySelector("em");
emElement.style = "color:red;"
```

Title

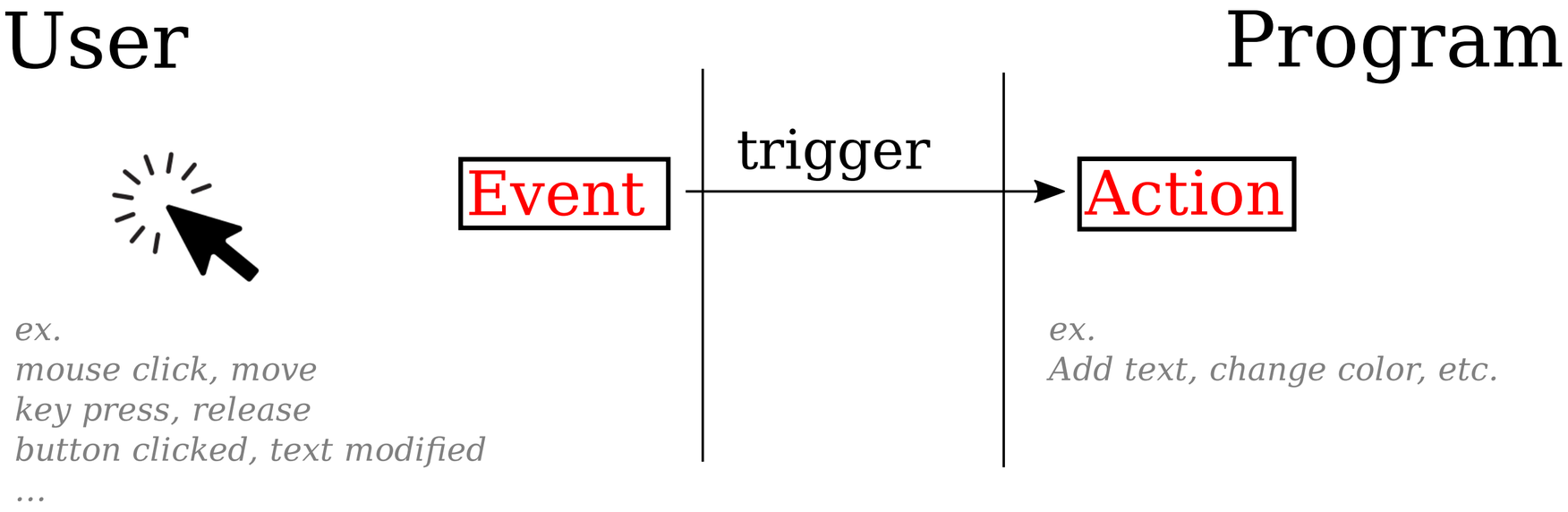
Hello *world* !

A link

User events

Event based programming

Event based programming = Programs that reacts under the actions of the user



User actions are not following standard procedural programming (pre-defined order, and number of times)

$\Rightarrow$ They can act in any order, at any time, and arbitrary number of times.

Event based programming: Listener

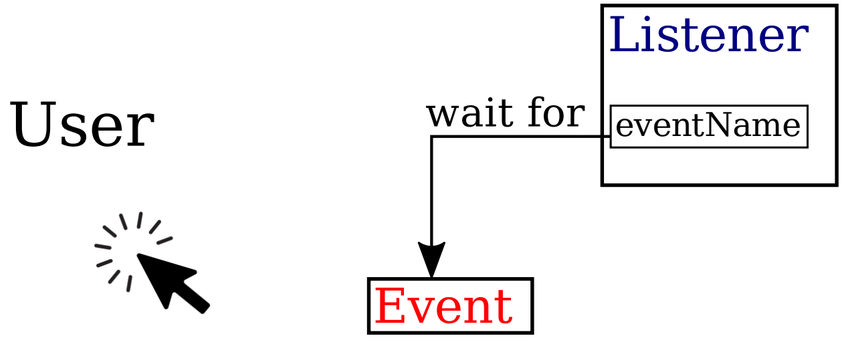
User events must be Listened.

Warning Cannot use standard 'while loops', otherwise the system is blocked waiting one event.

```
// Doesn't work: The program is blocked in the loop !
while( event===false ) {
  ... check(event);
}
```

Use of built-in **Listeners**

- Non blocking functions waiting for an event to happen
- Can be implemented as stack of events, asynchronous threads, etc.

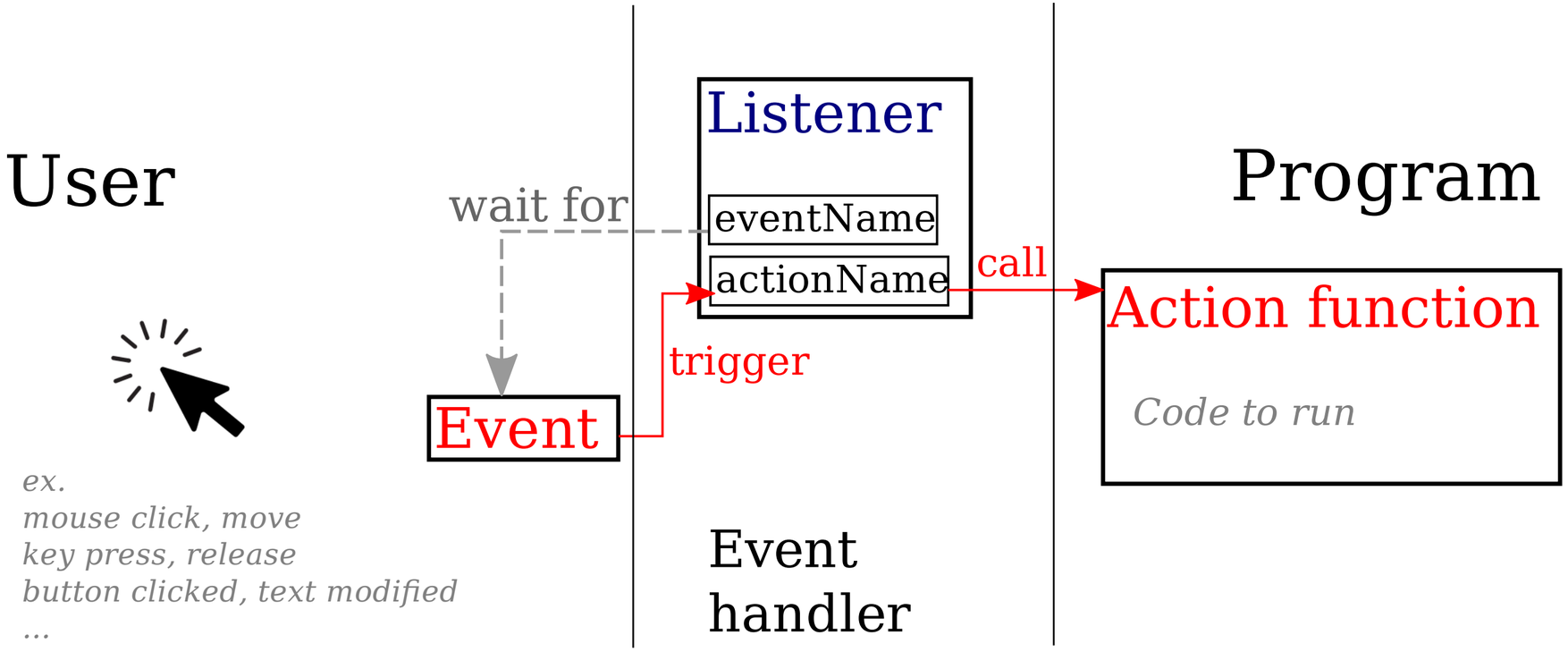


ex.
mouse click, move
key press, release
button clicked, text modified
...

In JS: **[x].addEventListener(eventName, [x]);**

Event based programming: Callback

Once the event happen, the listener need to call an *Action function*



The action function is called *Callback Function*

Function triggered by another one.

In js: **[x].addEventListener(eventName, actionName);**

Event in practice: Mouse click

HTML

```
<h1> My webpage </h1>
```

JS

```
document.addEventListener('click', actionClick );

function actionClick(event) {
  console.log('Mouse clicked !');
}
```

$\Rightarrow$ link

Syntax: eventTarget . addEventListener (eventName , actionCallback);

Webpage modification in the callback function

```
document.addEventListener('click', actionClick );

function actionClick(event) {
  const newElement = document.createElement('p');
  newElement.textContent = "Mouse click !";
  document.querySelector("body").appendChild(newElement);
}
```

My webpage

Callback parameters (event)

The callback function automatically receives a parameter with some information on the source event.

```
document.addEventListener('click', actionClick );

function actionClick(event) {

  const button = event.button;
  const x = event.clientX;
  const y = event.clientY;

  const newElement = document.createElement('p');
  newElement.textContent = `Click ${button} at position (${x},${y})`;
  document.querySelector("body").appendChild(newElement);
}
```

My webpage

Event Target

Events can be trigered by different elements.

```
const h1Element = document.querySelector("h1");
const boxElement = document.querySelector(".box");
h1Element.addEventListener('click', actionTitleClick );
boxElement.addEventListener('click', actionBoxClick );

function actionTitleClick(event) {
  h1Element.textContent = "Title clicked !";
}

function actionBoxClick(event) {
  boxElement.style = "background-color: blue;";
}
```

My webpage

Example

HTML

```
<h1>Click on the Balloon ! </h1>
<img class="ballon1" src="balloon.jpg">
<img class="ballon2" src="balloon.jpg">
<img class="ballon3" src="balloon.jpg">
```

Click on the Balloon !

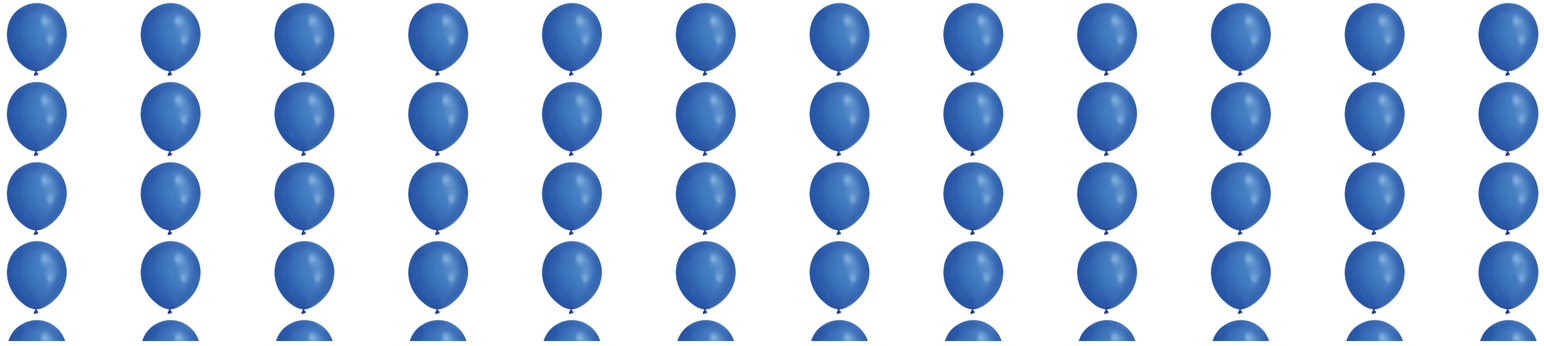
JavaScript

```
document.querySelector(".ballon1").addEventListener('click', actionExplode);
document.querySelector(".ballon2").addEventListener('click', actionExplode);
document.querySelector(".ballon3").addEventListener('click', actionExplode);

function actionExplode(event) {
  const element = event.currentTarget;
  element.src = "bang.jpg";
}
```

Example fully defined in JS

```
const NBalloon = 100;
const bodyElement = document.querySelector('body');
for(let k = 0; k < NBalloon; k++) {
  const newImage = document.createElement('img');
  newImage.src = 'balloon.jpg';
  bodyElement.appendChild(newImage);
  newImage.addEventListener('click', actionExplode);
}

function actionExplode(event) {
  event.currentTarget.src = 'bang.jpg';
}
```

Click on the Balloon !



Events type

JS can handle several buit-in events ex.

- Mouse events (click, dbclick, mousemove, wheel, etc)
- Keyboard events (keydown, keyup, etc)
- Window events (resize, scroll, etc)
- Input/forms (input, change, etc)
- ...

JavaScript language

Good practices

Code you JavaScript code in a separated file (ex. *script.js*)

Include the code in your HTML file using defer

```
<script src="script.js" defer=""></script>
```

(+) Separation of concerns: HTML=document structure; JS=logic

(+) defer keyword: non blocking JS loading. Execution while HTML DOM is loaded.

Avoid inline JS within HTML file

```
<!-- Avoid -->
<script>Some inline JS code here</script>
```

Use the current standard

Language specification standardized by ECMAScript.

Conform to the current ECMAScript v6.

$\Rightarrow$ Add in the first line of your script

```
"use strict;"
```

Declaring variables in JS

Three possible declaration (const, let, var)

```
const a = 3;
let b = 2;
var c = 1;
```

const : cannot be re-assigned, bloc scope.

let : can be re-assigned, bloc scope.

var : can be re-assigned, function scope.

Re-assignment

```
const a = 3;
let b = 2;
var c = 1;

// a = 4; error a cannot be re-assigned
b = 3; //OK
c = 4; //OK
```

Bloc scope VS function scope

```
function someFunction() {
 if( someCondition ) {
   const a = 3;
   let b = 2;
   var c = 1;
 } // a and b are deleted

  // console.log(a); error a doesn't exist
  // console.log(b); error b doesn't exist
  console.log(c); // OK, c exists within the
  // entire function
}
```

Good practices to declare variables

General rule: Prefer non-reassignable, local scope, variables

True for every language

1. Use by default **const** if possible
2. Use **let** if you need to re-assigned the variable
3. Avoid **var** (outdated, too large scope)

   Use only if you really need function scope

Rem.

- **const** and **let** introduced in EC6
- Many internet code still use the outdated *var* from previous JS

Rem on const

You can modifiy the content of a non-reassignable variable (const)

```
const a = [7,4,3];
a[0] = 8; // OK
a[1] = a[2]+4; //OK
a[12] = -4; //OK - array are dynamically reshaped
// a = [4,2,3] Incorrect (re-assignement)
```

Typing

JS is dynamically typed

The program find the type itself, not indicated by the user.

Similar to Python.

```
const a = 5; //a is a number
const b = 4.12; // b is a number
const c = "some string"; //string
const d = [7,4,5,2]; // array
```

Types are adapted dynamically

```
let a = 5; //a is a number
a = "bear"; // now a is a string

const b = [4,"a string",[7,8]];
```

The same variable can contain various types at different part of the code

Good practice

- Avoid it! Hard to read.
- One variable = one type in only one context.

Data structure

Every variable in JS is a dictionary

set of property/value pair

```
let p = {
  x:1;
  y:3;
}
p.x = 4;
console.log(p.y);
```

Functions

Dynamic typing, optional return value (similar to Python)

```
// One parameter, one return value
function discArea(r) {
  return Math.PI * r*r;
}
// Two parameters, no return value
function printMin(a,b) {
  if( a < b ) {
    console.log(a);
  }
  else {
    console.log(b);
  }
}
```

Function are simple JS variables

```
const F1 = function(x) { return 2*x*x+4*x+1; }
const F2 = function(x) { return -3*x*x+7*x-2; }
const currentFunction = F2;
console.log( currentFunction(8) );
```

Function can be defined inline with short syntax

```
const F = (x) => 2*x*x + 4*x +1;
console.log( F(4) );
```

Equality

"==" test equality with potential change of type

"===" test equality without change of type

```
if( 5=="5" ) { // Return true
  console.log("Re-typing");
}
if( 5==="5" ) { // Return false
  console.log("Don't pass here");
}
```

Good practice: Prefer "===" than "=="