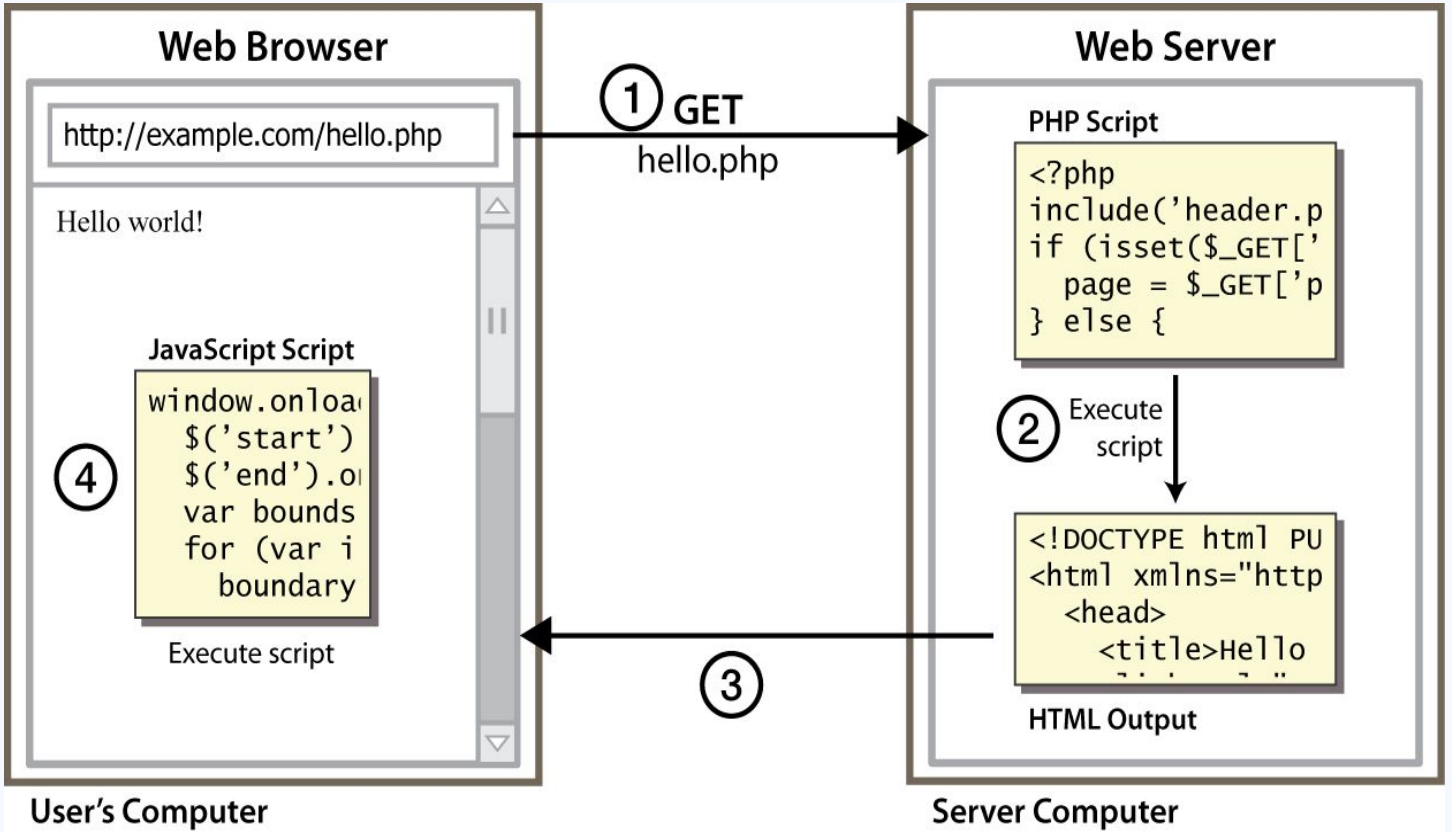
Trường Đại học Công nghệ Thông tin – Đại học Quốc gia Hồ Chí Minh
Bộ môn An toàn thông tin

Javascript Basic – Advanced

Ths. Trần Tuấn Dũng – dungtran@uit.edu.vn

Client Side Rendering



Why use client-side programming?

Backend languages already allows us to create dynamic web pages. Why also use client-side rendering?

- **Usability**: can modify a page without having to post back to the server (faster UI)
- **Efficiency**: can make small, quick changes to page without waiting for server
- **Event-driven**: can respond to user actions like clicks and key presses
- **Better UX:** user don’t have to load new page or reload current page to get new content

What is Javascript?

A lightweight programming language ("scripting language")

- Used to make web pages interactive
- Insert dynamic text into HTML (ex: user name)
- React to events (ex: page load user click)
- Get information about a user's computer (ex: browser type)
- Perform calculations on user's computer (ex: form validation)

A web standard (but not supported identically by all browsers)

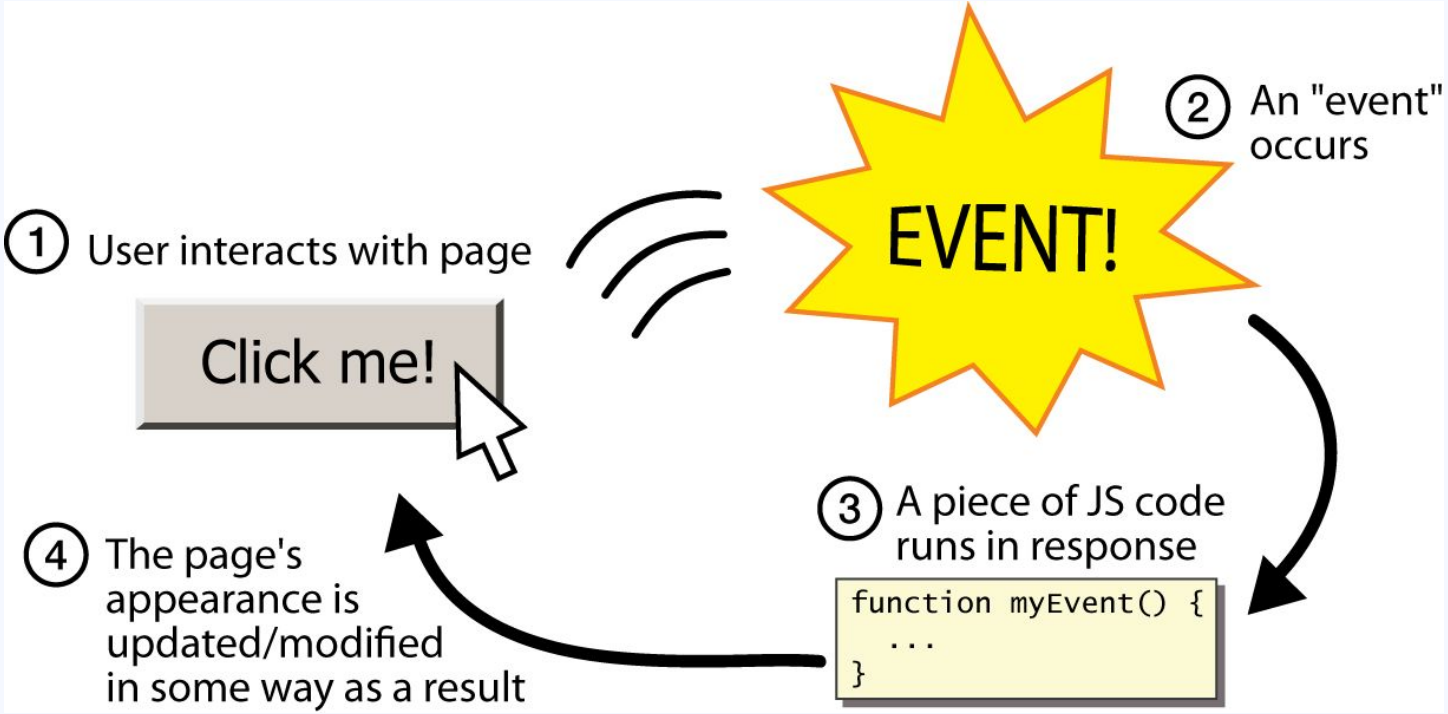
NOT related to Java other than by name and some syntactic similarities

Linking to a JavaScript script

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

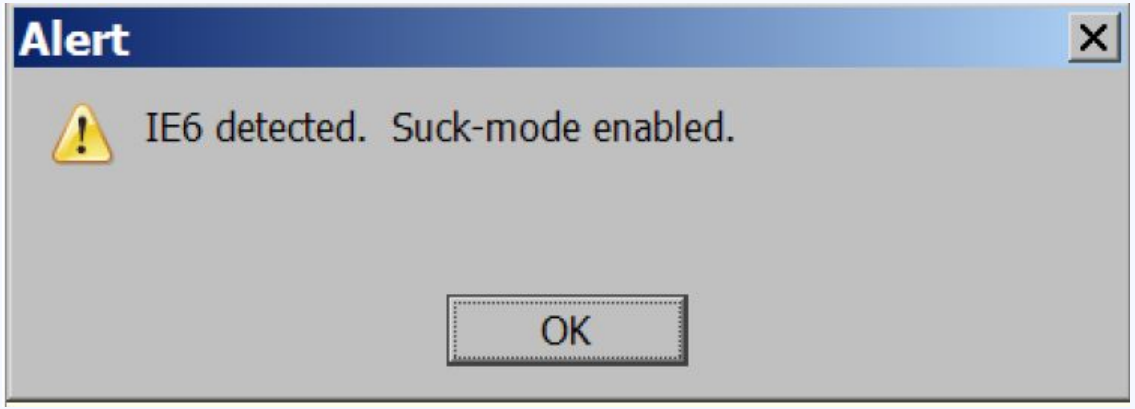
- <script> tag should be placed above closing tag of <body> (not always)
- Script code is stored in a separate .js file
- JS code can be placed directly in the HTML file's body or head (like CSS) using <script> {…js code} </script>

Event-driven programming



A JavaScript event example

```
alert("IE6 detected. Suck-mode enabled.");
```



A JS command that pops up a dialog box with a message

Event-driven programming

- JavaScript programs instead wait for user actions called *events* and respond to them
- **Event-driven programming**: writing programs react to user events

Simple event handler

```
<button>Click me!</button>
```

We have a button inside a HTML file

To make it respond to user action:

1. Choose the control (e.g. button) and event (e.g. mouse click) of interest
2. Write a JavaScript function to run when the event occurs
3. Attach the function to the event on the control

JavaScript functions

```
function myFunction() {
    alert("Hello!");
    alert("How are you?");
}
```
JS

- Similar to others languages function, except no return type

Event handlers

```
<button onclick="myFunction();">Click me!</button>
```

JavaScript functions can be set as event handlers

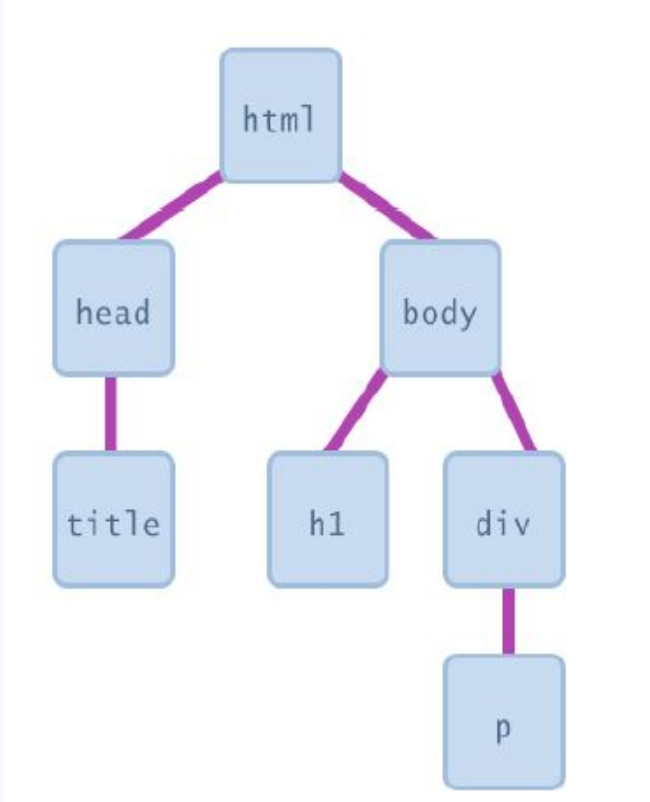
- when you interact with the element, the function will be executed

- **onclick** is just one of many event HTML attributes we'll use

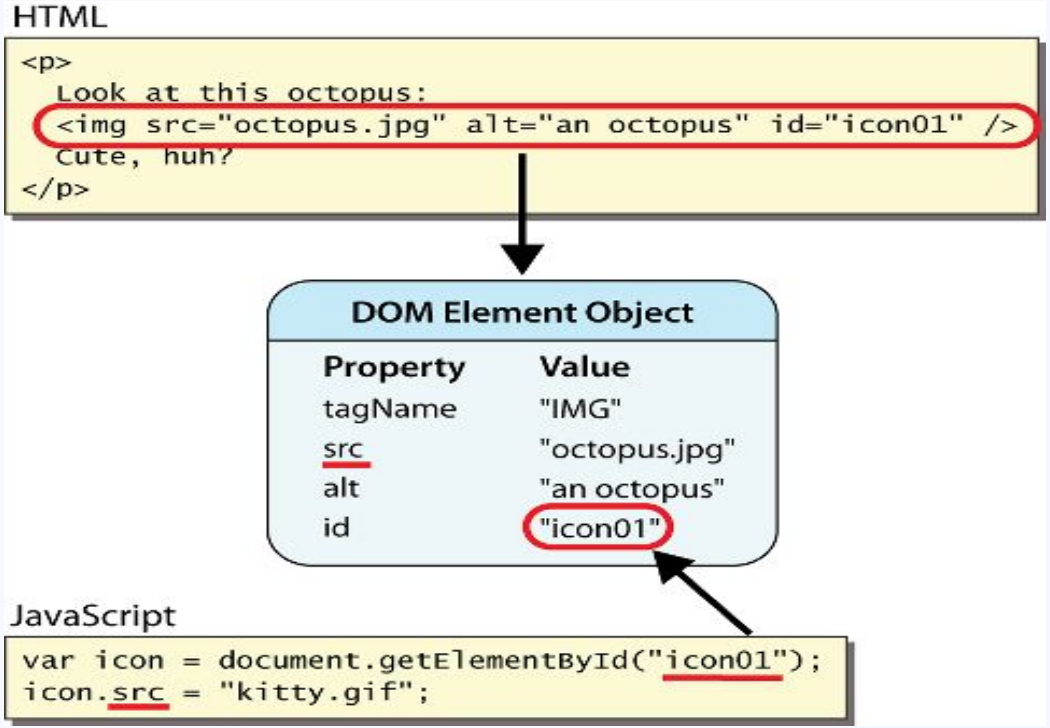
Document Object Model (DOM)

Allow JS code to manipulate the website

- we can examine elements' state
  - e.g. see whether a box is checked
- we can change state
  - e.g. insert some new text into a div
- we can change styles
  - e.g. make a paragraph red



DOM element objects



DOM Manipulation

- To get a reference to an HTML element: **querySelector**
- Use CSS Selectors as param

```
var name = document.querySelector("#id");
```

```
<button onclick="changeText();">Click me!</button>
<span id="output">replace me</span>
<input id="textbox" type="text" />
```

HTML

```
function changeText() {
    var span = document.querySelector("#output");
    var textBox = document.querySelector("#textbox");
    textbox.style.color = "red";
}
```

DOM Manipulation

- Adding event handler directly to HTML is not optimal since we want to have the HTML as clean as possible
- For example, we need to add event handler for click, submit, hover, .etc
- We will need to use a lot of attribute, make the HTML unreadable
- Using element.addEventListener('event', handler)

```
//Event Listeners
form.addEventListener('submit',function(e) {
    //…
});
```

DOM Manipulation

- Some HTML have default event (like form)
- Using event.preventDefault() to stop it from perform default event

```
//Event Listeners
form.addEventListener('submit',function(e) {
    e.preventDefault();
});
```

Variables


```
var name = expression;
```

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;
```

Types are not specified, but JS does have types ("loosely typed")

Number, Boolean, String, Array, Object, Function, Null, Undefined, ...

We can find out a variable's type by using typeof <variables>

Variables

Javascript has 3 ways of variables declaration: let, var and const

	var	let	const
Scope	The scope of a var variable is functional or global scope.	The scope of a let variable is block scope.	The scope of a const variable is block scope.
Re-declare	It can be updated and re-declared in the same scope.	It can be updated but cannot be re-declared in the same scope.	It can neither be updated or re-declared in any scope.
Initialization	It can be declared without initialization.	It can be declared without initialization.	It cannot be declared without initialization.
Access without declare	It can be accessed without initialization as its default value is "undefined".	It cannot be accessed without initialization otherwise it will give 'referenceError'.	It cannot be accessed without initialization, as it cannot be declared without initialization.
Hoisting	These variables are hoisted.	These variables are hoisted but stay in the temporal dead zone untill the initialization.	These variables are hoisted but stays in the temporal dead zone until the initialization.

Hoisting

Javascript allow we to access variables before declaration

Generally, this type of code will throw an Reference Error in most languages

But with Javascript Hoisting, the variable is still accessible

```
console.log(hoist); // will log undefined
var hoist = 500;
```

The code above is the same as

```
Var hoist;
console.log(hoist); // will log undefined
var hoist = 500;
```

Data Type

Javascript have 2 different type of variables:

- Primitive Type: Number, String, Boolean, Undefined, Null, Symbol
- Reference Type: Object, Array, Function, Date

Primitive Type are variables that has **value** store in a memory cell

Reference Type are variables that has **address** store a memory cell (like array in C++)

With **Primitive Type**, when you copy a variable, you create a new variable, meaning changing a variable **will not affect** the other

With Reference Type, things are so different, from the we have **Deep Copy** and **Shallow Copy**

How to differentiate between deep and shallow copies in JavaScript (freecodecamp.org)

Comments

```
// single-line comment
/* multi-line comment */
```

null and undefined

```
var ned = null;
var benson = 9;
// at this point in the code,
// ned is null
// benson's 9
// caroline is undefined
```
JS

- **undefined**: has not been declared, does not exist
- **null**: exists, but was specifically assigned an empty or null value

Logical operators

- > < >= <= && || ! == != === !==
- most logical operators automatically convert types:
  - 5 < "7" is true
  - 42 == 42.0 is true
  - "5.0" == 5 is true
- === and !== are strict equality tests; checks both type and value
  - "5.0" === 5 is false

if/else


```
if (condition) {
    statements;
} else if (condition) {
    statements;
} else {
    statements;
}
```

- identical structure to other languages if/else statement
- JavaScript allows almost anything as a condition

Truthy/Falsy


```
var iLike190M = true;
var ieIsGood = "IE6" > 0; // false
if ("web devevelopment is great") { /* true */ }
if (0) { /* false */ }
```
JS

- any value can be used as a Boolean
  - "falsey" values: 0, 0.0, NaN, "", null, and undefined
  - "truthy" values: anything else
- converting a value into a Boolean explicitly:
  - var boolValue = Boolean(otherValue);
  - var boolValue = !!(otherValue);

for loop

```
var sum = 0;
for (var i = 0; i < 100; i++) {
    sum = sum + i;
}
```

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
    s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheellllloo"
```

JS

while loop


```
while (condition) {
     statements;
}
```

```
do {
    statements;
} while (condition);
```

break and continue keywords also behave the same

Arrays


```
var name = []; // empty array
var name = [value, value, ..., value]; // pre-filled
name[index] = value; // store element
```

```
var ducks = ["Huey", "Dewey", "Louie"];
var stooges = []; // stooges.length is 0
stooges[0] = "Larry"; // stooges.length is 1
stooges[1] = "Moe"; // stooges.length is 2
stooges[4] = "Curly"; // stooges.length is 5
stooges[4] = "Shemp"; // stooges.length is 5
```

Array methods

```
var a = ["Stef", "Jason"]; // Stef, Jason
a.push("Brian"); // Stef, Jason, Brian
a.unshift("Kelly"); // Kelly, Stef, Jason, Brian
a.pop(); // Kelly, Stef, Jason
a.shift(); // Stef, Jason
a.sort(); // Jason, Stef
```

- array serves as many data structures: list, queue, stack, ...
- methods: concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift

Object

```
var person = {}; // empty object
var person = {age: 23, name:”John”}// pre-filled
```

```
person.age //return 23
person[‘name’] // return John
```

Ternary Operator


```
Let hello = isFriend ? "What's up" : "Good Morning"
```

- Ternary operator should always be considered over if/else with short condition and have 3 cases max
- More readable then normal if/else statement, even switch/case

Others Javascript Features

- Web Storage (localStorage, sessionStorage)
- Web Workers: Parallel task on Web
- Geolocation: User Location
- Intersection Observer: Observe if specific element is in observation zone
- Web socket: Real time communication
- Fetch API: Call data from server
- And So on

ES6 Syntax

ES6 Version of Javascript introduces a lot of new syntax, but we only commonly use some:

- Template String: let hello = `hello ${name}` (using backtick)
- ES6 Classes
- Promises
- async/await: async function name() {}; let res = await name()
- Destructuring: let {prop1, prop2} = obj
- Spread: let obj2 = {…obj1}
- Rest: let {prop1, …rest} = obj
- Modules: import ''
- Arrow function

JavaScript Arrow Function (w3schools.com)

ES6 Cheat Sheet & Quick Reference

Event Loop

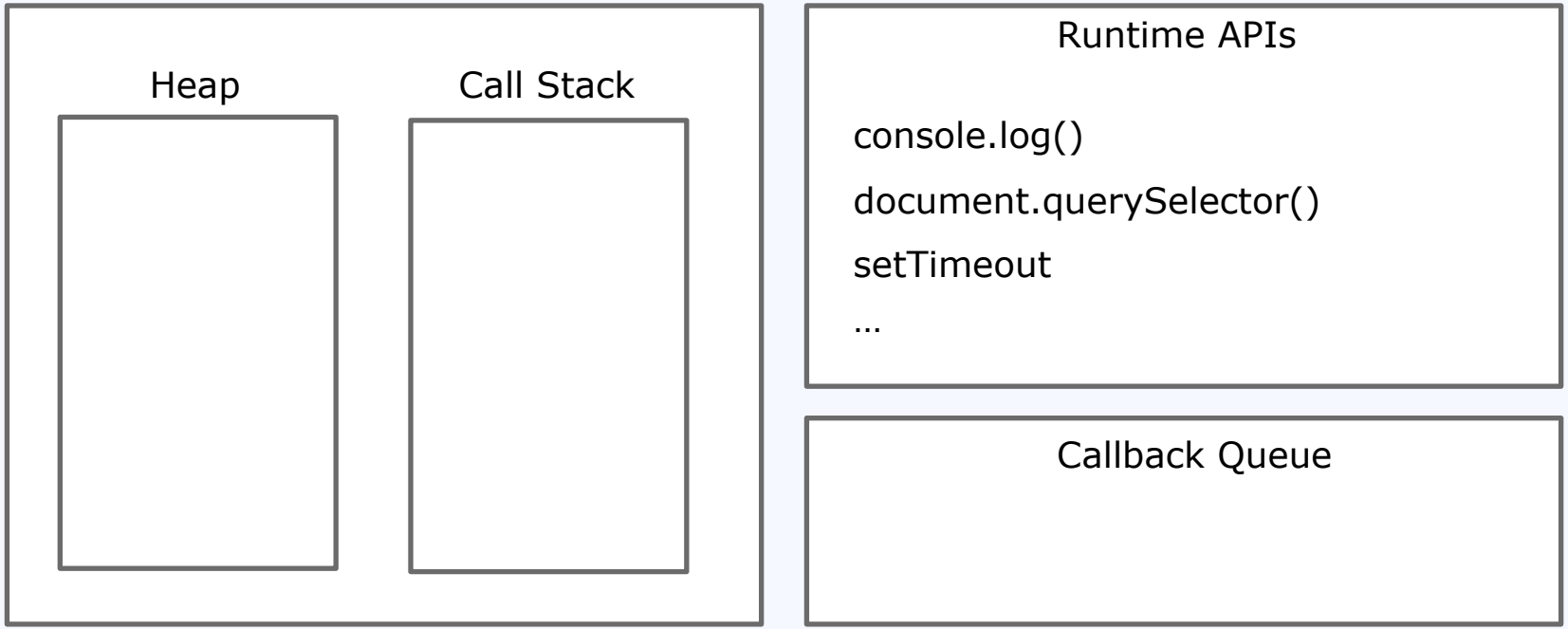
JavaScript has a runtime model based on an event loop, which is responsible for executing the code, collecting and processing events, and executing queued sub-tasks

JavaScript is **single-thread**, but it's capable of running asynchronous task

Event Loop is a mechanism allow Javascript to schedule task, so it look like tasks are running parallelly

Generally, Javascript Runtime comprise: **JS Engine, Runtime APIs (Browser/Node), Callback Queue**

Event Loop


Heap
Call Stack
Runtime APIs
console.log()
document.querySelector()
setTimeout
...
Callback Queue

Event loop example

```
console.log(1)
setTimeout(()
=> {
  console.log(2)
},2000)
console.log(3)
console.log(1)
```

Log:

1

3

2

Call Stack

Global Execution Context

Runtime APIs

Callback Queue

console.log(2)

Modern CSS Library

Most well-known

Modern CSS Library - Backend

Javascript – Lightweight – Beginner Friendly

Typescript – Scalable – Reliable – Typesafe

Exercise

1. Create a button that will randomly open an alert dialog then show user browser agent

2. Create a form allow user to submit random number then show it on screen (numbers will be sort)

Exercise – Homework

Using pure HTML and JS, no frameworks, no libraries. Perform form validation with a form that has:

- Username: at least 6 character and 15 max
- Password: at least 8 and 15 max
- Email
- Confirm password