

# Asincronismo

Javascript

# ¿Qué es el asincronismo?

---

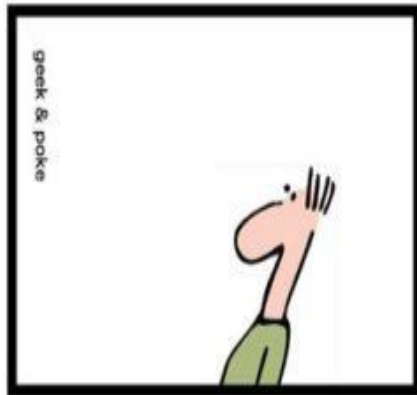
Asincronismo hace referencia al suceso que no tiene lugar en total correspondencia temporal con otro suceso.



**WIKIPEDIA**  
The Free Encyclopedia

# SIMPLY EXPLAINED

REPASO



NO AJAX



AJAX

# Ejemplos cotidianos

---

- [TBC]

Asincronico

Mirar TV y hacer la comida

Cargar Telefono mientras se hace otra cosa

Compra en internet

Chat

Sincronico

Ver partido completo

Compra presencial

Llamada telefonica

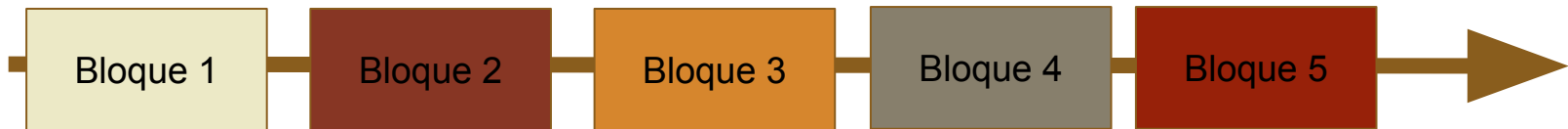
# Ejemplos cotidianos

---

- Asincrónico
  - Mandar un mail, lo envío pero no tengo respuesta instantánea.
  - Subo un video a YouTube, no está disponible ni bien lo subo, se procesa (asincrónicamente) y se pone disponible después.
  - Si ven esta clase en YouTube es asincrónica, paso en algún momento pero la ven ahora.
- Sincrónico
  - Clase de hoy, es sincrónica.

# Programación Sincrónica

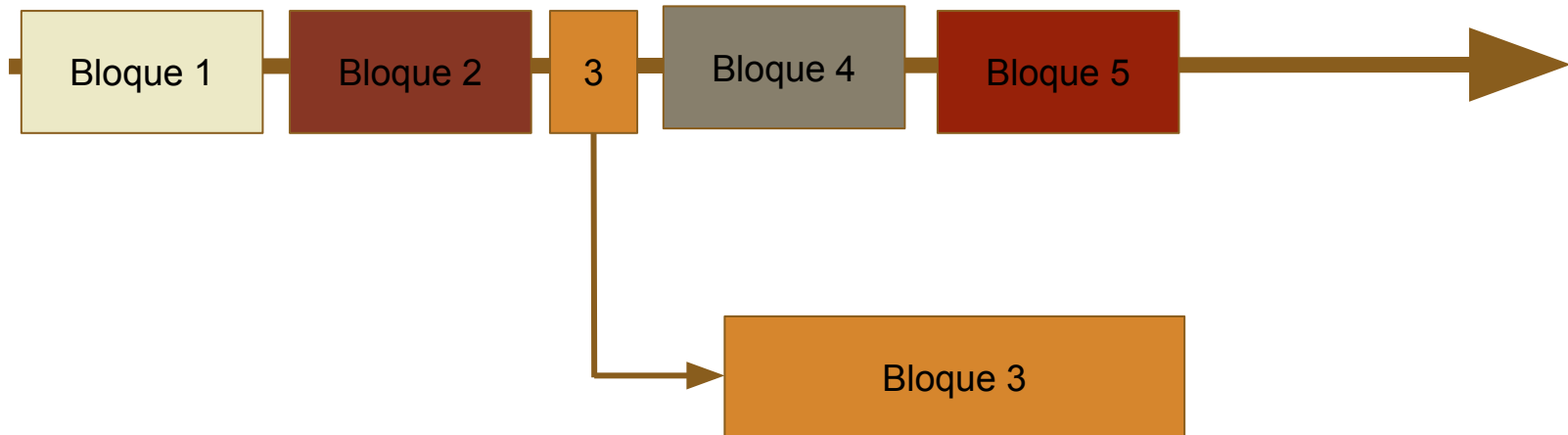
- Es la forma en que aprendieron a programar.
- Cada operación es **bloqueante** (tiene que terminar para que empiece la siguiente)
- Siempre respeta el orden.
- Fácil de seguir



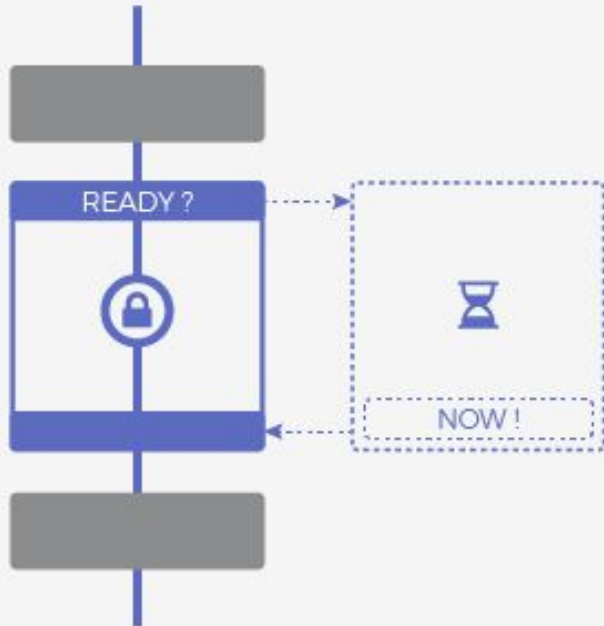
# Programación Asíncrona

- Nos permite devolver el control al programa antes de haber terminado una tarea, mientras sigue operando en otro plano.
- Llamadas no bloqueantes
- Difícil de seguir
- Aumento de la escalabilidad (mismo tiempo, más operaciones)

3

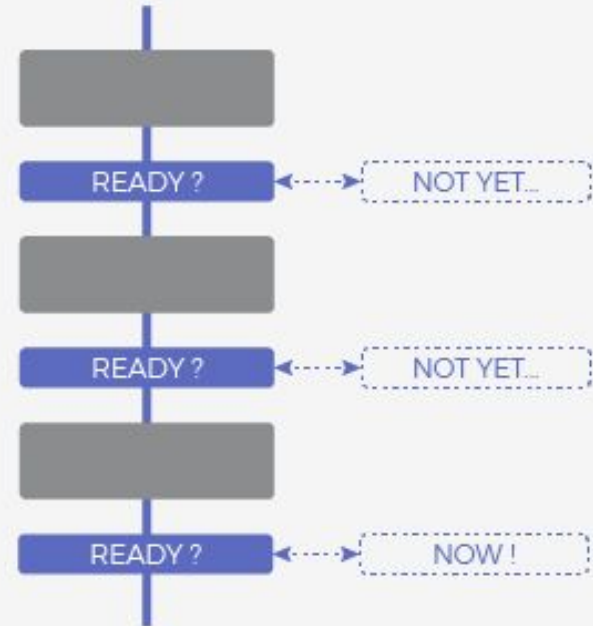


# Bloqueante vs No Bloqueante



## BLOQUEANTE

El control no es devuelto a la aplicación hasta que la llamada bloqueante termine.



## NO BLOQUEANTE

La llamada es devuelta con independencia de su resultado. Se utiliza polling para completar el trabajo.

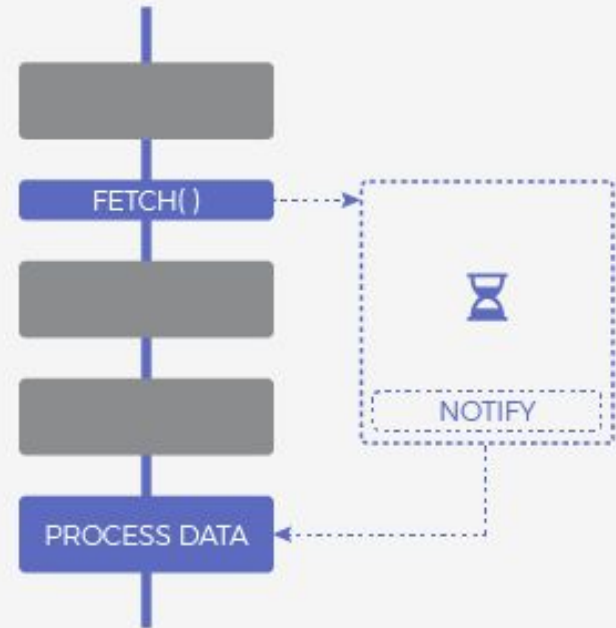


# Sincrónico vs Asíncrono



## SÍNCRONO

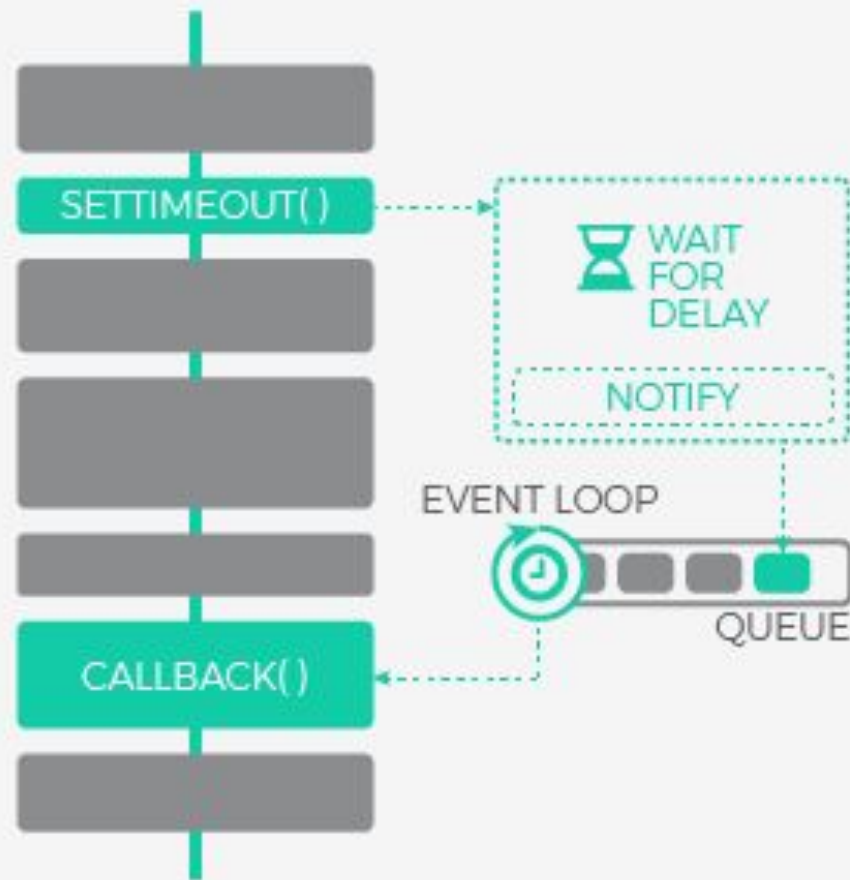
Ejecución secuencial. Retorna cuando la operación ha sido completada en su totalidad.



## ASÍNCRONO

La finalización de la operación es notificada al programa principal. El procesamiento de la respuesta se hará en algún momento futuro.

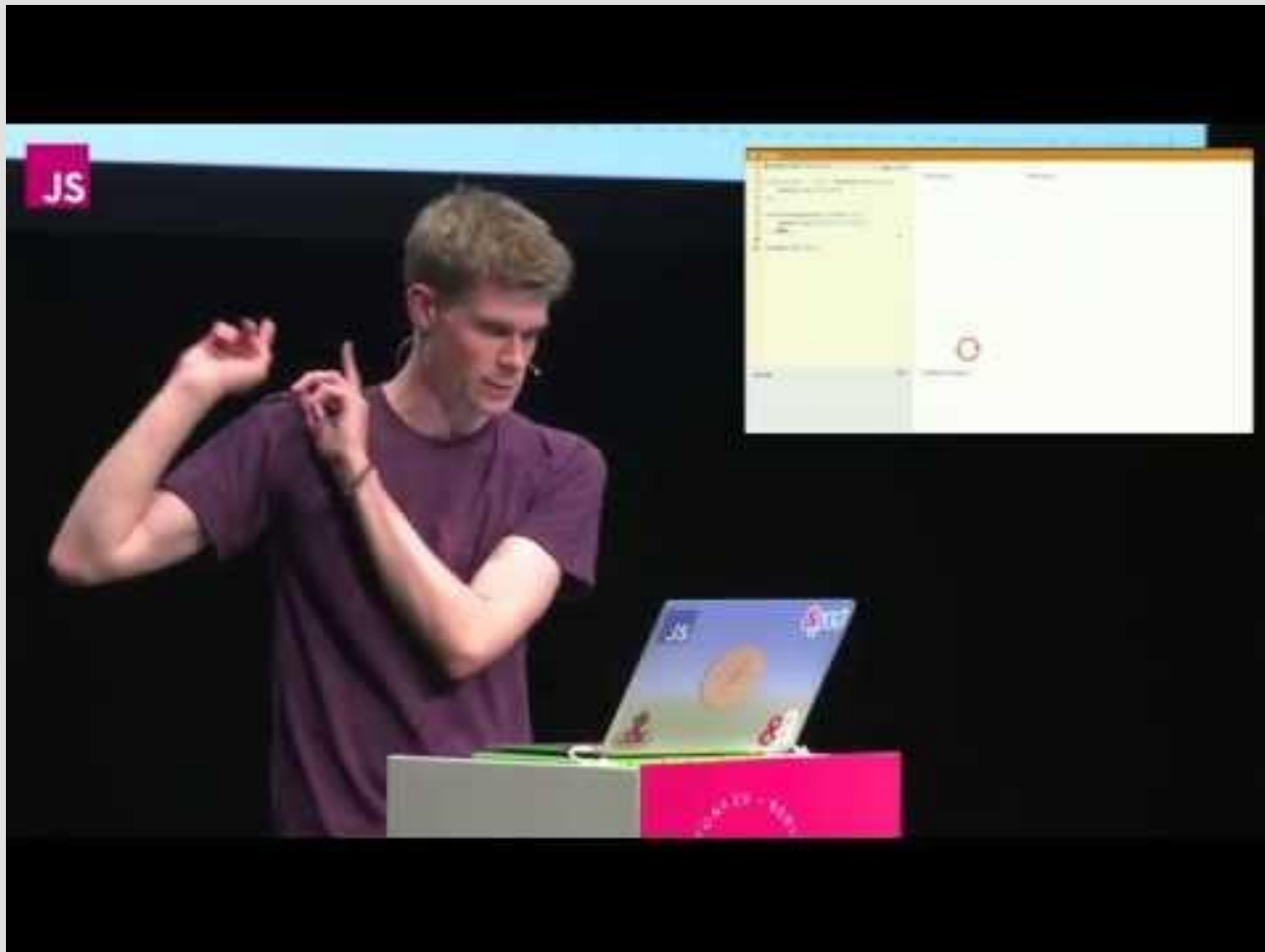
# Javascript - Event Loop



## EL MODELO DE JAVASCRIPT

Javascript emplea un modelo asíncrono no bloqueante con loop de eventos de thread único para sus interfaces de entrada/salida.

**Recomendado: todo el video**  
**Este tema específico: min 12:50**



**Pruebenlo!**

<http://latentflip.com/loupe/>

# Ejemplo - Sincrónico

Tengo este código en mi página:

```
foto = bajarFoto('http://fotos.com/gato.jpg');  
subirFoto(foto, 'http://miweb.com/gatos');
```

- Bajar foto es una llamada bloqueante:
  - No puedo hacer nada hasta que la foto se baje.
- No podría hacer click en ningún otro lugar de la web, estaría freezada.



# Ejemplo - Asíncrono

Tengo el mismo pseudo-código en mi página, pero escrito diferente.

```
bajarFoto( 'http://fotos.com/gato.jpg' , subirFoto );
```

- Ahora la llamada **bajarFoto** se ejecuta y el flujo del programa sigue.
- Cuando termine de bajar la foto, se va a llamar el subirFoto.



**¿Cómo hacemos para llamar una función cuando otra termina?**



# Callback

---

Los callback se ejecutan cuando termina una función asincrónica.

**En Javascript puedo pasar una funcion como parametro :)**

```
function bajarFoto (url, callback);
```

**URL** es un String

**Callback**, es una función :)

El código de bajar foto va a ser algo como

```
function bajarFoto(url, callback){  
  //bajar la foto  
  callback(foto); //se encarga de llamar a su callback cuando termine  
}
```



## Ejemplo: Then

---

La función que le pasamos a un then es el callback para cuando la promesa se haya cumplido.

```
fetch(url)
```

```
//callback para cuando se haya terminado de bajado el archivo
```

```
.then(function(){ .. })
```

```
//callback para cuando se haya terminado generar el json
```

```
.then(function(){ .. });
```

# Notación para asociar callbacks

La forma en que se ve un llamado depende de cómo escribimos la función:  
(son todas equivalentes)

```
function miFuncion() { .. }           //Id
```

```
let miFuncion = function() { .. };    //variable
```

```
bajarFoto (url, miFuncion);
```

```
bajarFoto (url, function(){ .. });    //Función inline anónima
```

```
bajarFoto (url, () => { .. });         //Función arrow
```

**CALLBACK**

# Programación dirigida por eventos

REPASO

Un programa dirigido por eventos sigue los siguientes pasos:

- Comienza la ejecución del programa
- Se llevan a cabo las inicializaciones y demás código inicial
- El programa queda bloqueado “Escuchando” hasta que se produzca algún evento

Se definen:

- Eventos (Click, Drag, Hover, Load, etc.)
- Funciones que se ejecutan en esos eventos
- Se llama el “controlador de eventos”



# Programación dirigida por eventos

REPASO

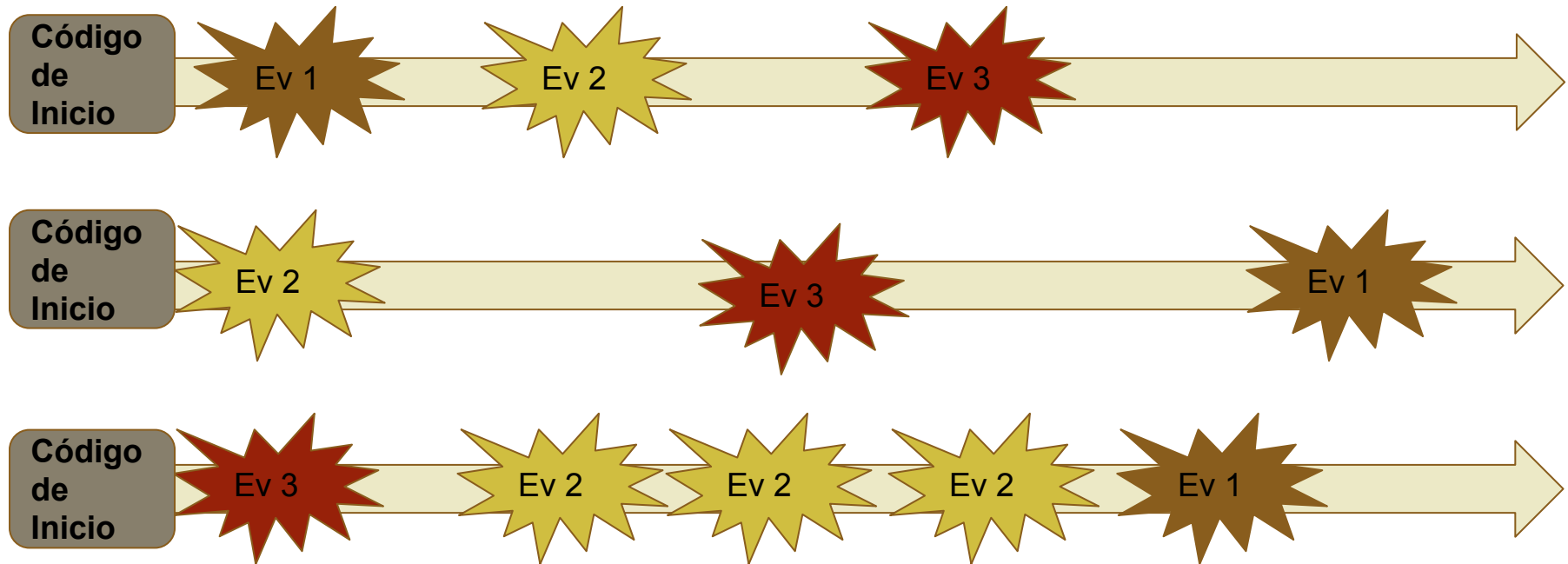
## Programación secuencial

- Sabemos el flujo de la ejecución

## Programamos dirigida por eventos

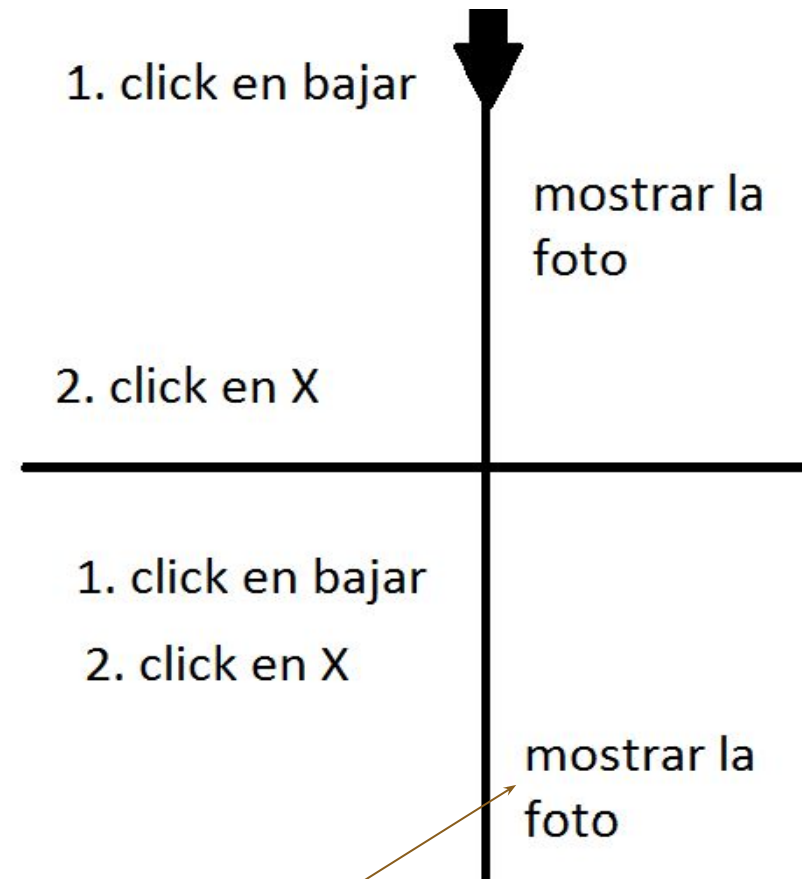
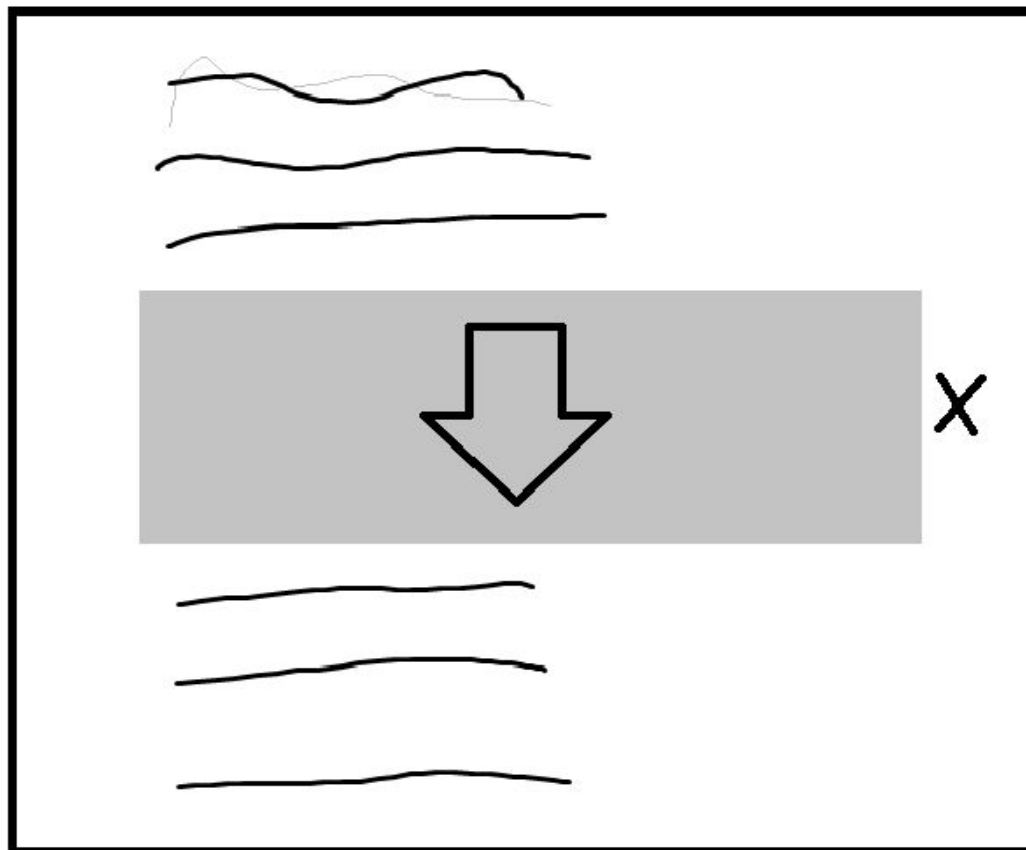
- No sabemos la secuencia exacta de ejecución
- Se disparan diferentes códigos con diferentes acciones

## Ejemplo de tres ejecuciones diferentes:



# Ejemplo

Posibilidad de cerrar la imagen antes de que termine de descargarse. Hay que tener en cuenta que son eventos asincronicos.



La foto ya no tiene donde mostrarse!

**¿Y si sabemos que el callback recibe un parámetro?**



# Notación con parámetros

Cuando sabemos que al callback se le enviarán parámetros

```
function miFuncion(data,...) { .. }           // Id  
let miFuncion = function(data,...) { .. };    // variable
```

Es suficiente con escribir:  
bajarFoto (url,miFuncion);

Paso innecesario

bajarFoto (url, **function**(data,...){miFuncion(data,...)}); *//Función inline anónima*

bajarFoto (url, (arg1,arg2,...) => { .. }); *//Función arrow*

# Ejemplo 1

---

¿Qué hace este fragmento de código?

```
function mostrar(data, type){  
    alert(data);  
    alert(type);  
}
```

```
function solicitudLenta(callback){  
    setTimeout(function(){  
        callback("<html>...", "Content-Type: text/html");  
    }, 5000);  
}
```

```
solicitudLenta(mostrar);
```



## Ejemplo 2

### Sumar 2 cajas de texto usando callback un evento click.

Nuestra función Sumar(a, b, callback), recibe 3 parámetros, los 2 primeros son los valores de la caja de texto y el segundo es el callback, este será el encargado de retornar el resultado de la sumatoria.

```
function Sumar(a, b, callback){  
    callback(a+b);  
}
```

```
document.querySelector("#operar").addEventListener('click', function(){  
    let a = parseInt(document.querySelector("#a").value),  
        b = parseInt(document.querySelector("#b").value);
```

```
    Sumar(a, b, function(r){  
        console.log('El resultado es ' + r);  
    })  
})
```

<https://codepen.io/webUnicen/pen/xjBbVR>



DEMO

# Ejercicio de ejemplo

---

# Ejercicio 1

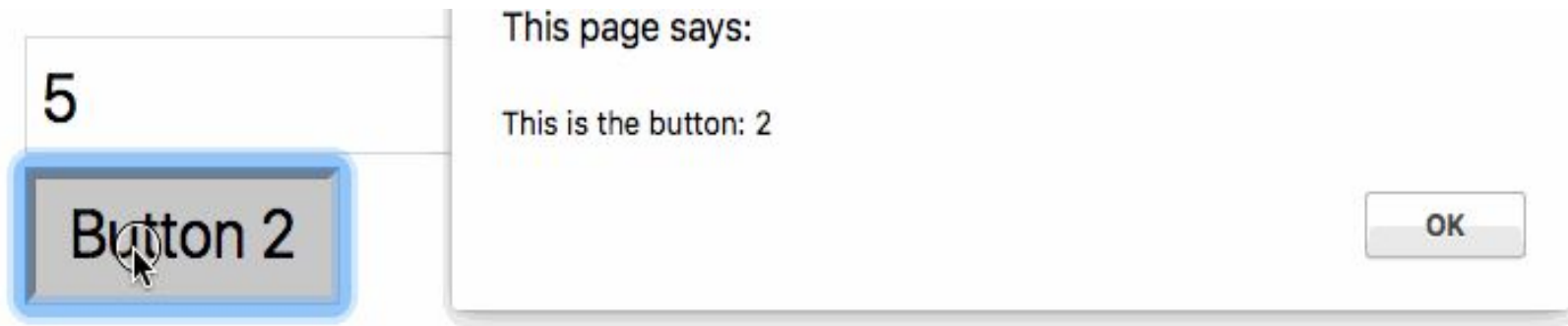
---

1. Crear una página web que solo tenga un input y un botón de Agregar Botones.
2. Al hacer clic en el botón se deben lanzar N temporizadores (N leído desde el input) cada uno con un tiempo aleatorio entre 0 y 5 segundos.
3. Al cumplirse el timer se debe crear un botón que al hacerle click muestre una alerta y el número de botón que fue (0, 1, 2, ...).
4. Todos los botones deben agregarse al final de un div que los contiene.
5. Probar el orden de los botones, y qué mensaje muestra cada uno.
6. Al hacer clic en el botón de Agregar Botones deben agregarse otros M botones (M puede ser igual o diferente de N). Debe haber ahora dos botones 0, y de cada número.
7. Probar que funcionan todos los botones (viejos y nuevos).

# Ejercicio

Una imagen vale más que mil palabras.

**Resultado Final Esperado:**



# Divide y conquista

---

Vamos a dividir el problema en partes:

- Tener muchos botones
- Que hagan un alert (todos muestran lo mismo)
- Saber cual boton es cada uno (cada uno muestra su número en el alert)

Es decir, son dos problemas diferentes:

- Donde asigno los eventos
- Como se que boton apreté

# Paso 1: Muchos botones

---



Manos a la obra

# HTML

---

Esto no tiene misterios

```
<input type="number" name="" value="" id="quantity">
<button type="button" name="create" id="create">Create
Buttons</button>
<div id="buttons">
</div>

<script src="js/createButtons.js" charset="utf-8"></script>
```



# Agregar el handler al boton

---

- ¿Cómo hacemos?
  - TBC

```
document.getElementById('create').addEventListener('click', createButton);
```

# Create Buttons

---

- La funcion createButton debería:
  - [TBC]



# Solución

```
function createButton(){  
  let div = document.getElementById('buttons'); //Busco el DIV  
  let button = document.createElement("button"); // Creo el botón  
  button.innerHTML = "Button"; // Pongo nombre a Mostrar  
  div.appendChild(button); // Lo agrego al Div  
}
```



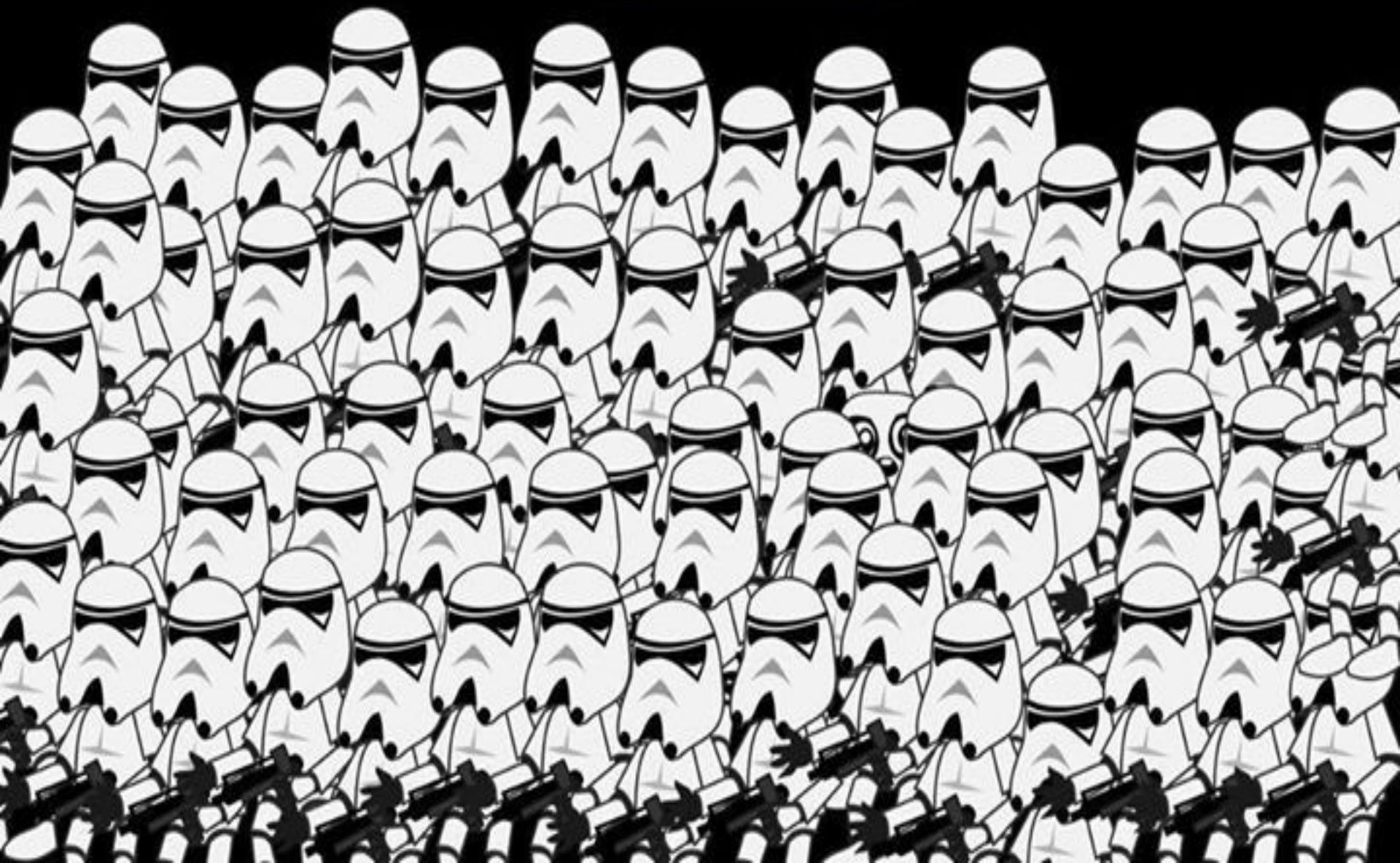
# Crear N Botones

---

- Tenemos que crear, al hacer click en el botón N botones.
- ¿Cómo lo hacemos?
  - TBC

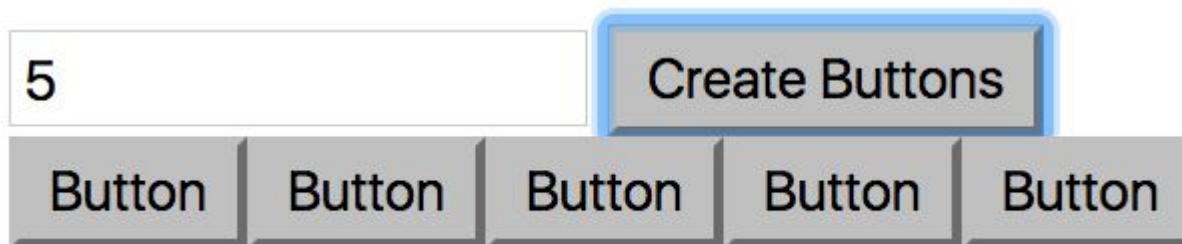


# FIND THE PANDA



# Solución

```
function createButtons() {  
  let quantity = document.getElementById('quantity').value;  
  for (let i = 0; i < quantity; i++) {  
    createButton();  
  }  
}
```



<https://codepen.io/webUnicen/pen/XqGbMZ>



# Nombre de los Botones

---

- Los nombres de los botones tienen que tener un Número.
- ¿Cómo hacemos?
  - TBC
  -

# Solución

```
function createButton(name){  
  let div = document.getElementById('buttons');  
  let button = document.createElement("button");  
  button.name = name; //valor de i pasado por parametro  
  button.innerHTML = "Button " + name;  
  div.appendChild(button);  
}
```

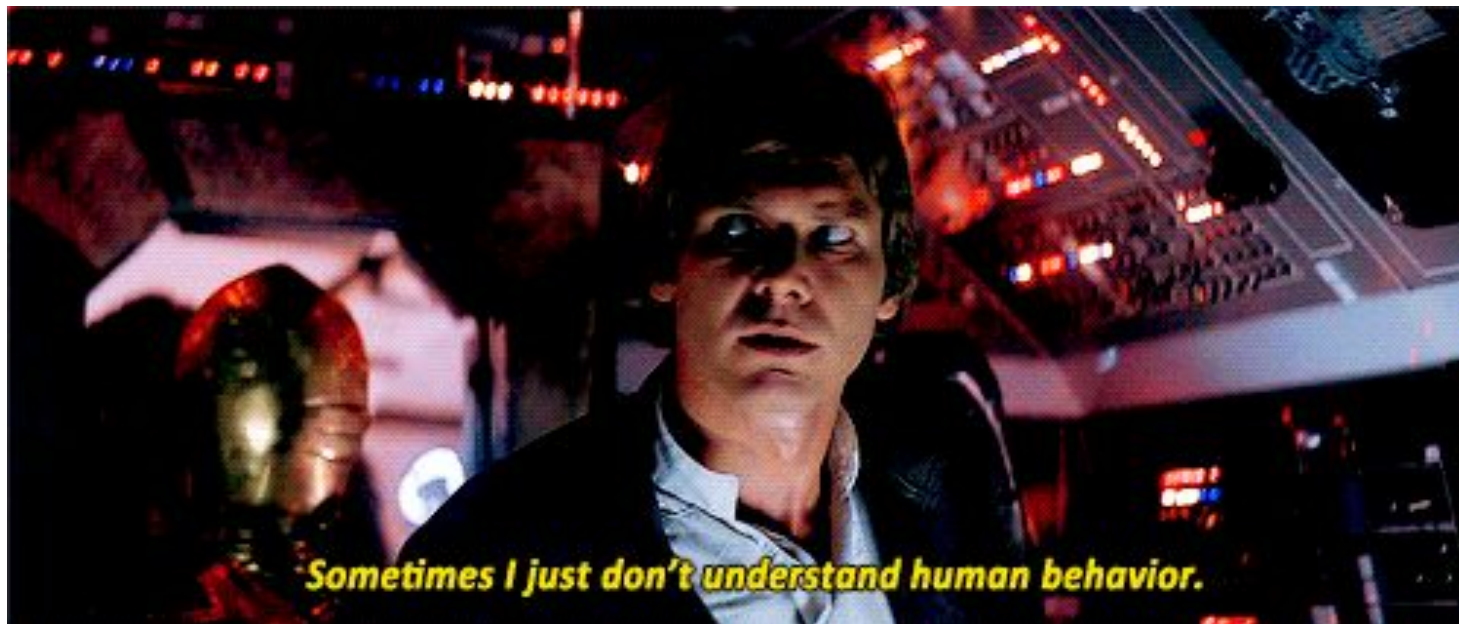
|          |                |          |          |          |
|----------|----------------|----------|----------|----------|
| 5        | Create Buttons |          |          |          |
| Button 0 | Button 1       | Button 2 | Button 3 | Button 4 |







- Los botones se crean con un delay entre 0 y 5 segundos.
- ¿Cómo hacemos?



# Eventos de tiempo (Repaso)



Se puede programar un evento, para ejecutar una función dentro de M milisegundos.

*//dispara (ejecuta bang) en 5 segundos*

**let** timer = setTimeout(bang, 5000);

# Solución

```
function createButtons() {  
  let quantity = document.getElementById('quantity').value;  
  for (let i = 0; i < quantity; i++) {  
    let randTime = Math.random() * 5000;  
    setTimeout(function(){createButton(i)}, randTime);  
  }  
}
```

DEMO

<https://codepen.io/webUnicen/pen/QroKxL>

# Eventos

---

- Los botones tienen que mostrar un alert con su nombre

## Cómo hacemos?



## Paso 2: Asignar eventos

---

# Opción 1

---

- Todos los botones que creamos tienen una clase.
- Los busco por clase y les agrego un listener.



# Solución

```
function createButton(name){
```

```
.....
```

```
button.classList.add("js-async");
```

```
div.appendChild(button);
```

```
}
```

```
function assignEvents(){
```

```
let buttons = document.querySelectorAll('.js-async');
```

```
for (let button of buttons) {
```

```
    button.addEventListener('click',() => { alert('Hiciste click') })
```

```
}
```

```
}
```

Dónde llamo esta función?



# Solución

```
function createButtons() {  
  let quantity = document.getElementById('quantity').value;  
  for (let i = 0; i < quantity; i++) {  
    let randTime = Math.random() * 5000;  
    setTimeout(function(){createButton(i)}, randTime);  
  }  
  assignEvents();  
}
```

<https://codepen.io/webUnicen/pen/OZqREd>





# ¿Por qué no anda?

---

Ideas:

- [TBC]



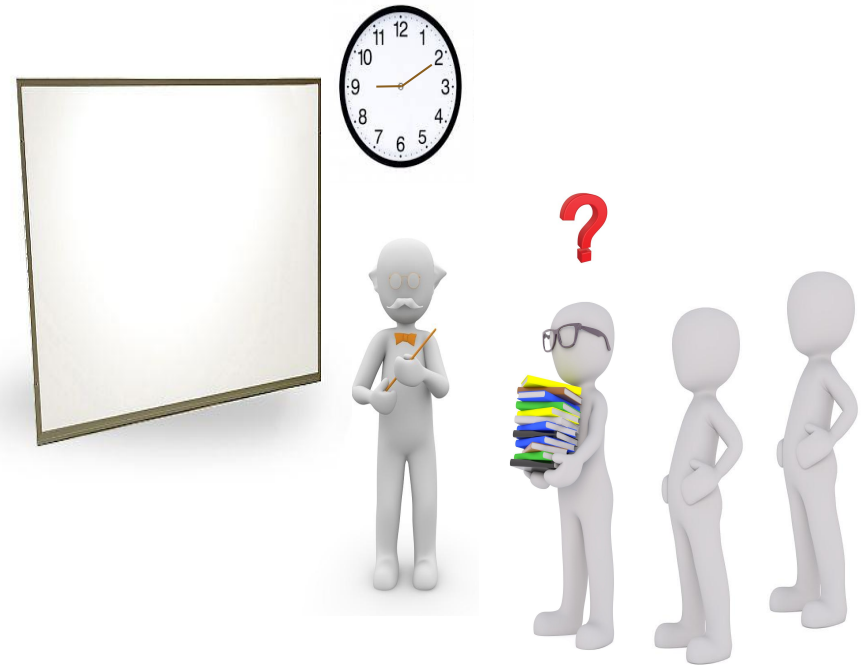


# El caso “Botón Thomson”



# El caso “Botón Thomson” (2)

¿Qué pasa si Thomson llega tarde a clases?



# El caso “Botón Thomson” (3)

¿Qué pasa si Thomson llega tarde a clases?

¿SOLUCIÓN?

Cuando entra Thomson, el profesor vuelve a decir lo mismo.



¿Pero qué pasa si hay otro Thomson en la habitación?



# El caso “Botón Thomson” - Solución

---

## SOLUCIÓN

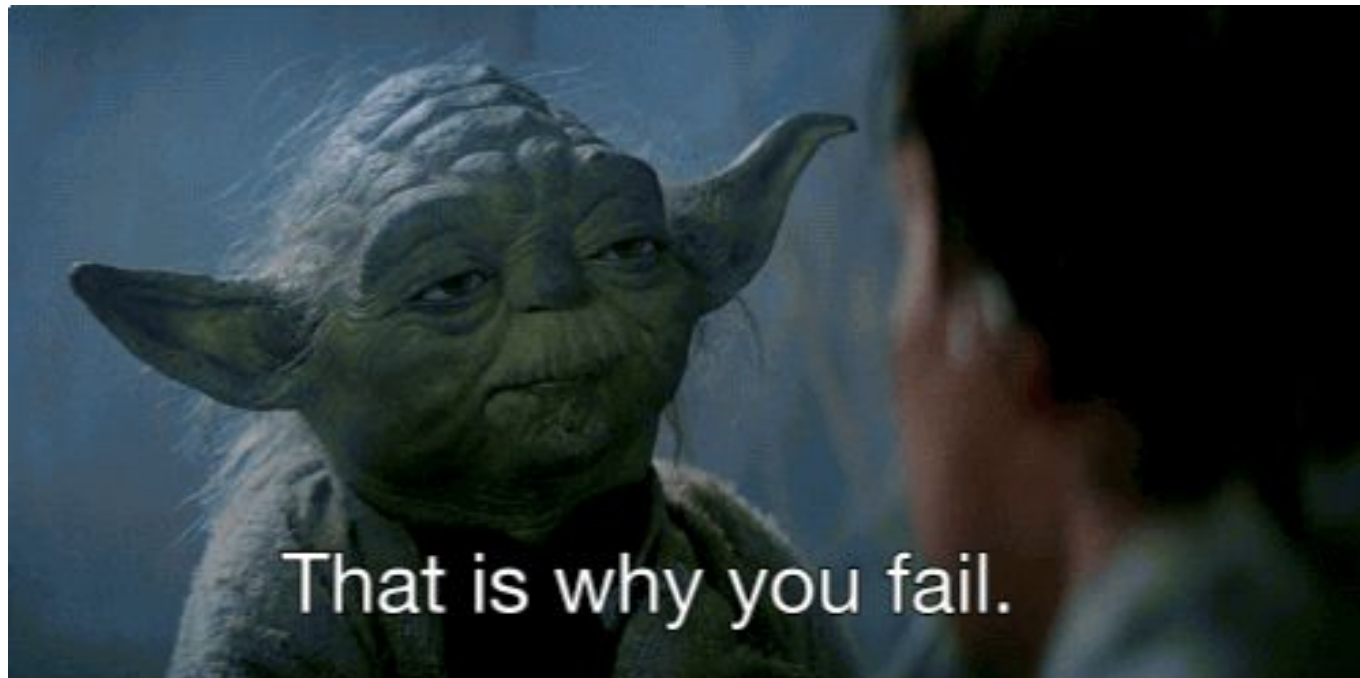
El profesor le dice en secreto solo a “los Thomson’s” **CUANDO ENTRAN** a la habitación.



## Parche #1

---

Un timeout de 10 segundos, seguro que se crearon todos los botones.



# Solución

```
document.getElementById('create').addEventListener('click',createButtons);
function createButtons() {
  let quantity = document.getElementById('quantity').value;
  for (let i = 0; i < quantity; i++) {
    let randTime = Math.random() * 5000;
    setTimeout(function(){createButton(i)}, randTime);
  }
  setTimeout(function(){assignEvents()}, 10000);
}
```



# Funciona!!! (o más o menos)

- Pudimos hacer que los botones tengan un evento.
- El alert funciona bien
- Hay que esperar que pase el tiempo...
- Que pasa cuando generamos 2 veces botones?



# Cómo lo arreglamos?

---

- Tratar de handlear el evento lo antes posible.
- Cuándo?



# Solución

```
function createButton(name){  
  let div = document.getElementById('buttons');  
  let button = document.createElement("button");  
  button.name = name;  
  button.innerHTML = "Button " + name;  
  button.addEventListener('click',() => { alert('Hiciste click')});  
  div.appendChild(button);  
}
```



# Debug

- Sigamos esto con breakpoints, para ver el orden en que se ejecuta cada sentencia



# Resumen: Diferentes opciones

---

Como crear el botón:

- Crear boton con create element
- Crear boton con HTML (regenerando todo desde un string)

Add Event Listener

- Al final: no anda porque aún no se crearon los botones
- Temporizado: no anda hasta que no pasó el tiempo
- Para todos luego de agregar cada botón (es posible duplicar event listeners)
- Para cada elemento, la dificultad está en saber como lo selecciono solo a ese:
  - Un ID para cada boton generado
  - Si lo cree con “createElement” ya es un objeto DOM

## Paso 3: Quién es quién?

---

# Quién es quién?

---

Cómo hago para saber cuál botón fue el que apreté?

# Ámbitos - Closures

REPASO

Forma de crear variables “ocultas”

<https://codepen.io/webUnicen/p/RVxROB>

```
function crearFuncionContadora() {  
    //nuevo ámbito  
    let x = 0;  
    return function() { x++; return x; }  
};  
  
//no la puedo acceder desde afuera  
let inc = crearFuncionContadora();  
inc(); //x es local a “ámbito”
```

En JS, declarar una variable es “crear una nueva cada vez que se pasa por esa sentencia”.



# Ámbitos - VAR vs LET

REPASO

Ejemplo práctico de diferencia:

```
console.log("Con var");  
for(var i = 0; i < 5; i++) {  
    setTimeout(function () {  
        console.log(i);  
    }, 0)  
}
```

```
console.log("Con let");  
for(let i = 0; i < 5; i++) {  
    setTimeout(function () {  
        console.log(i);  
    }, 0)  
}
```

El setTimeout usa la variable, pero después

Con VAR es siempre la misma variable, así que usa el último valor (5).  
Imprime 5 veces 5

Con LET cada ciclo usa una variable diferente  
Imprime del 0 al 4



DEMO

# Solución 1: Guardo quien es quien en el DOM

```
function createButton(name){  
  let div = document.getElementById('buttons');  
  let button = document.createElement("button");  
  button.name = name;  
  button.databuttonname = name;  
  button.innerHTML = "Button " + name;  
  button.addEventListener('click',() => clickEnBoton);  
  div.appendChild(button);  
}
```

Guardo algo más en el DOM

```
function clickEnBoton(){  
  alert('Hiciste click en ' + this.databuttonname);  
}
```

Uso "this"

DEMO

# this

REPASO

En el contexto de Eventos *this* representa el elemento involucrado en el evento

```
let el = document.getElementById('miDiv');  
el.addEventListener('click', function(e){  
    this.classList.toggle("clase");  
    //toggle de clase del div miDiv click  
});
```

DEMO

<https://codepen.io/webUnicen/pen/odNvKK>

## Solución 2: Uso una clojure

```
function createButton(name){  
  let div = document.getElementById('buttons');  
  let button = document.createElement("button");  
  button.name = name;  
  button.innerHTML = "Button " + name;  
  button.addEventListener('click',() => {  
    alert('Hiciste click en ' + name);  
  });  
  div.appendChild(button);  
}
```

**Funcion inline  
(anónima o arrow)**

**Puedo acceder a la  
variable directo**

**DEMO**

# Diferentes opciones

---

Donde guardo el ID

- En el closure (si o si anonima)
  - Crear boton con subfunción (anda con “var”, sino “let” si o si)
- En el DOM (permite modularizar externo)
  - No usar el ID o cualquier dato que sea propio de HTML
  - Usar algun atributo especial como “data-id”

# Recap de lo que aprendimos

---

- .....



# Bibliografía

---

- Solución:
- <https://jsparagatos.com/>
- <http://lemoncode.net/lemoncode-blog/2018/1/29/javascript-asincrono>

**AHORA LES TOCA PRACTICAR :D**

