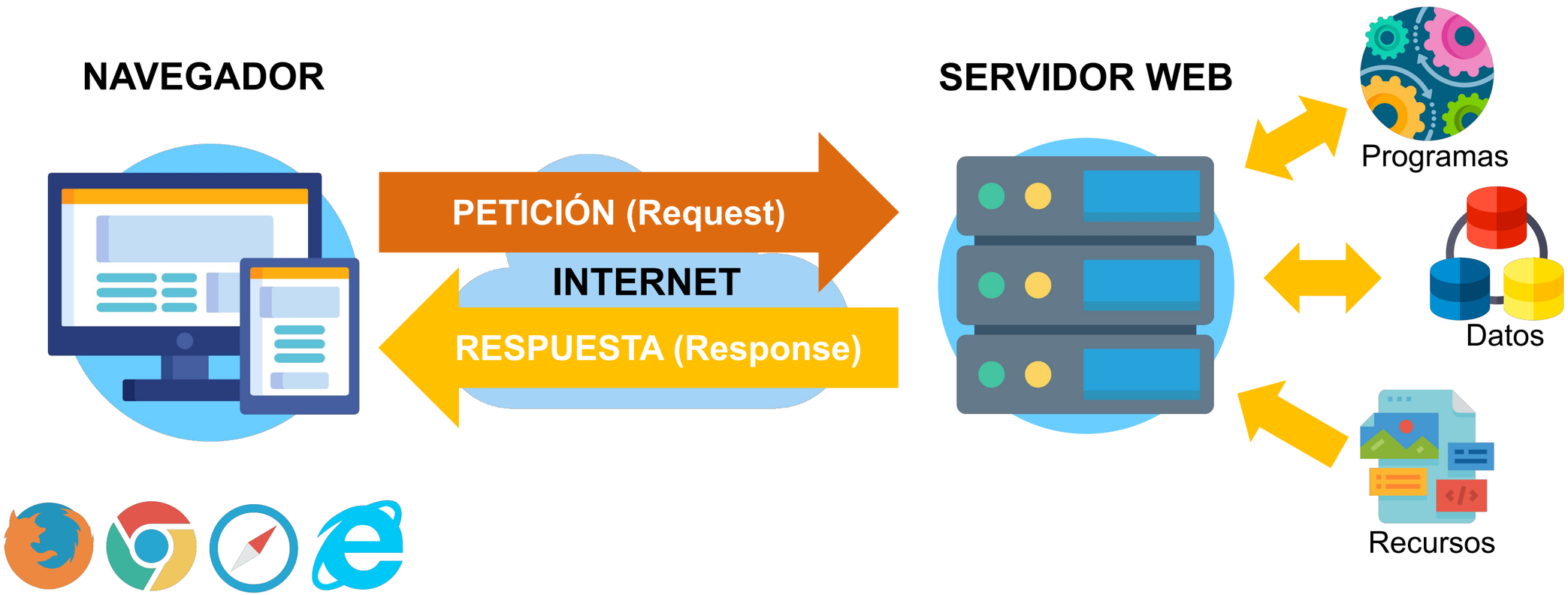
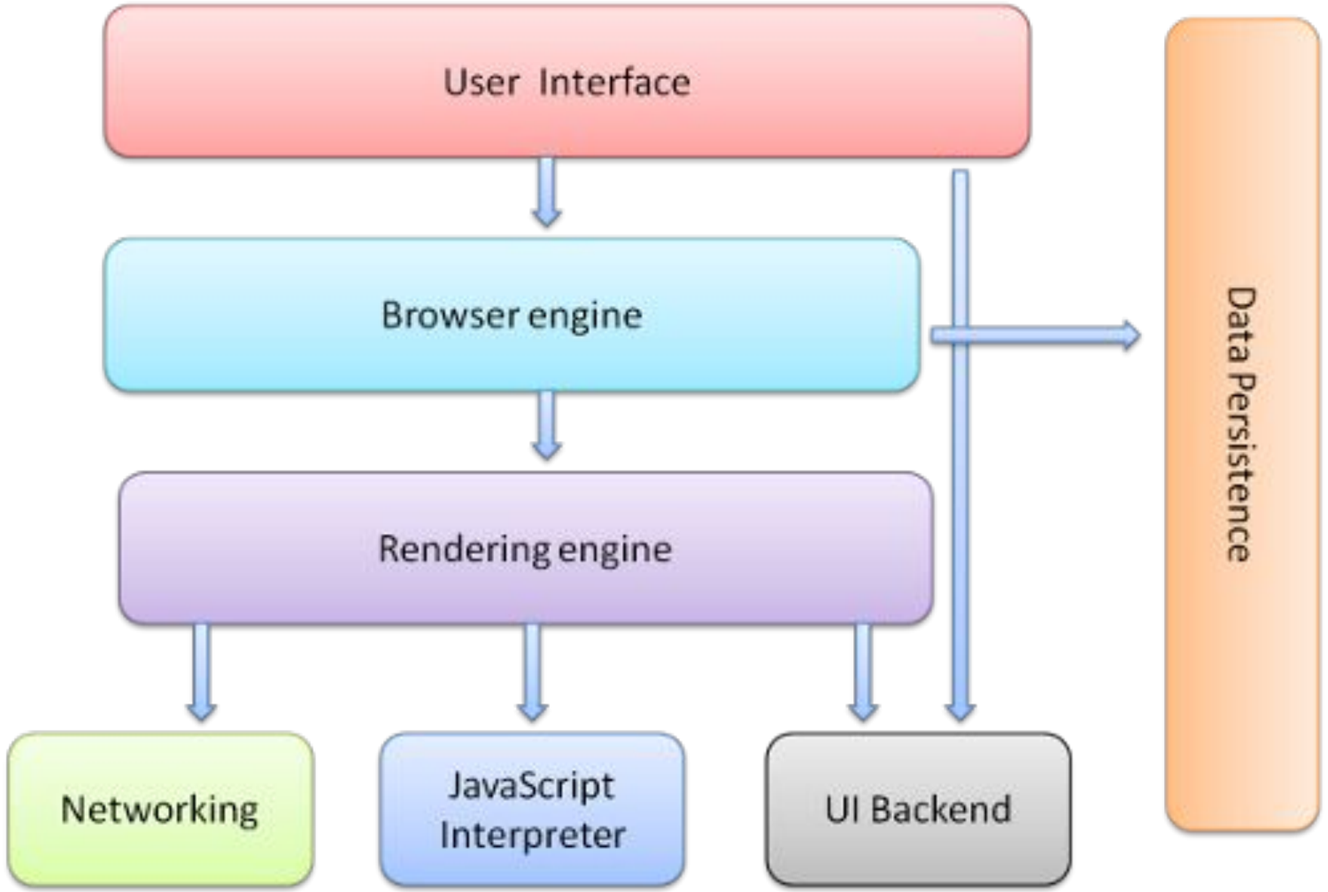

# JavaScript en el navegador

# Cómo funciona la web



# ¿Cómo funciona un navegador?




https://www.html5rocks.com/es/tutorials/internals/howbrowserswork/

# Motores e intérpretes de cada navegador

| | Rendering Engine | Js interpreter |
|---|---|---|
| **Google Chrome** | Blink | V8 |
| **Mozilla Firefox** | Gecko | SpiderMonkey |
| **Safari** | WebKit | Nitro |
| **Microsoft Edge** | EdgeHTML | ChakraCore |
| **Internet Explorer** | Trident | Chakra |

**CanIUse???**

# Hello World

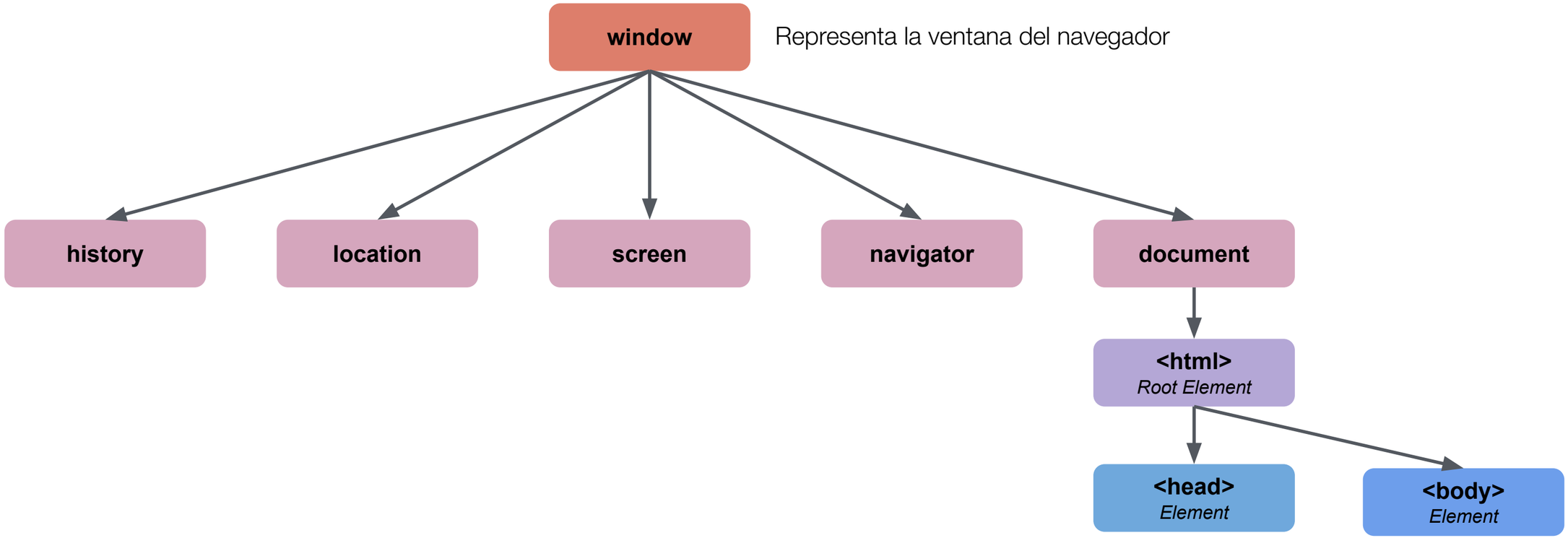
# Hello World

```html
<!DOCTYPE html>
<html>
<head>
    <title>Hello World!</title>
</head>
<body>
    <script>
        console.log('Hello World')
    </script>
    <!-- importar archivo JavaScript externo -->
    <script src="hello-world.js"></script>
    <!-- importar módulo JavaScript (para poder usar la instrucción import) -->
    <script type="module" src="hello-world-module.js"></script>
</body>
</html>
```

# Browser Object model (BOM)

# Browser Object Model (BOM)

El **BOM** ofrece un conjunto de objetos para poder controlar el navegador web.




[Bom definition]

# Navigator

appCodeName
appName
appVersion
appMinorVersion
userAgent
product
productSub

browserLanguage
language
systemLanguage
userLanguage
cpuClass
oscpu
platform
userProfile
securityPolicy

mimeTypes
plugins
cookieEnabled
javaEnabled()
onLine
preference()

https://developer.mozilla.org/es/docs/Web/API/Window/navigator

# Window

alert(),
confirm()
prompt()
print()
open()
close()
createPopup()
focus()
blur()
stop()

moveBy()
moveTo()
resizeBy()
resizeTo()
scrollBy()
scrollTo()
**setInterval()**
**setTimeout()**
clearInterval()
clearTimeout()

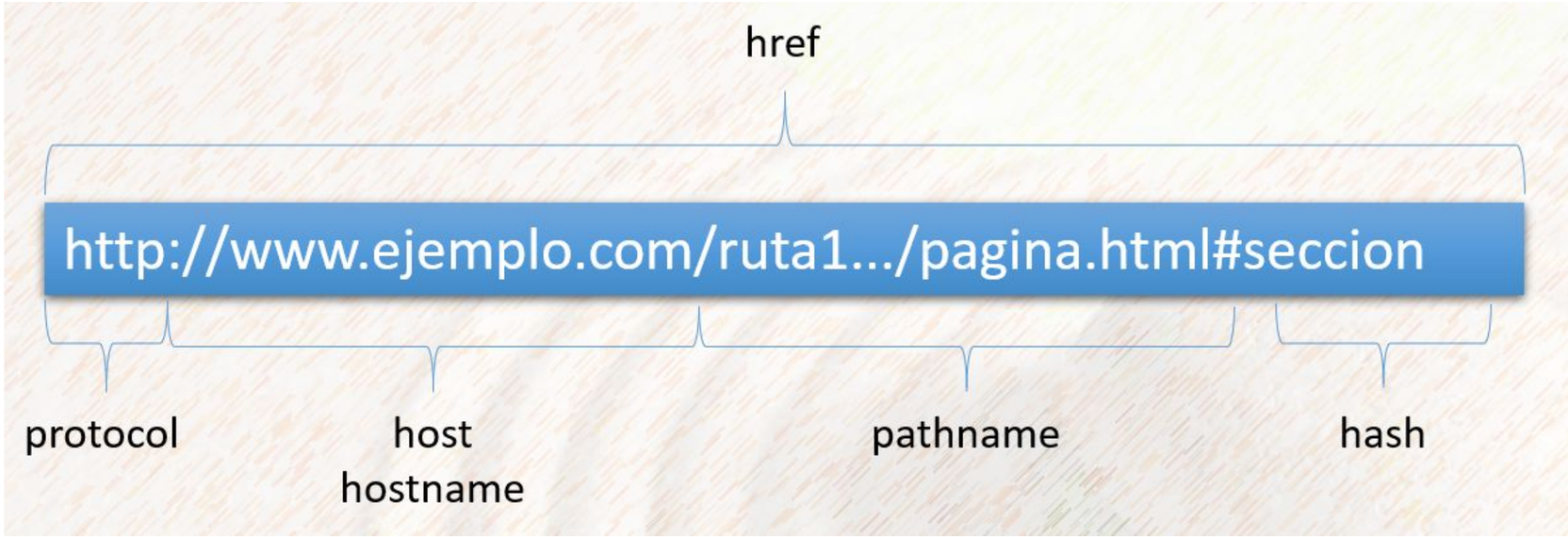
screenX
screenY
innerWidth
innerHeight
outerWidth
outerHeight

(APIS HTML5)
localStorage
sessionStorage

https://developer.mozilla.org/es/docs/Web/API/Window

# location

href
protocol
host name
pathname
hash
port
search
assign()
replace()
**reload()**




https://developer.mozilla.org/es/docs/Web/API/Window/location

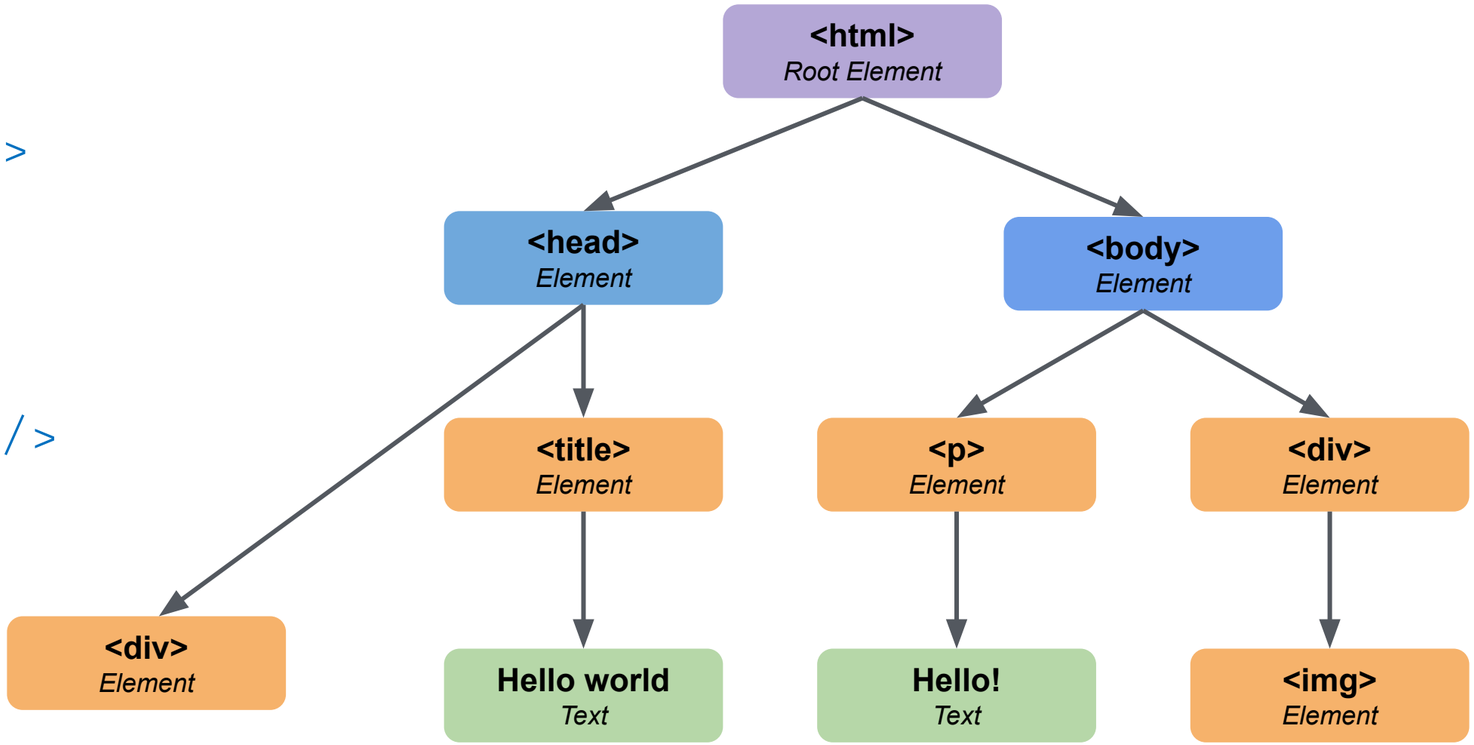
# Document Object Model (DOM)

# Document Object Model (DOM)

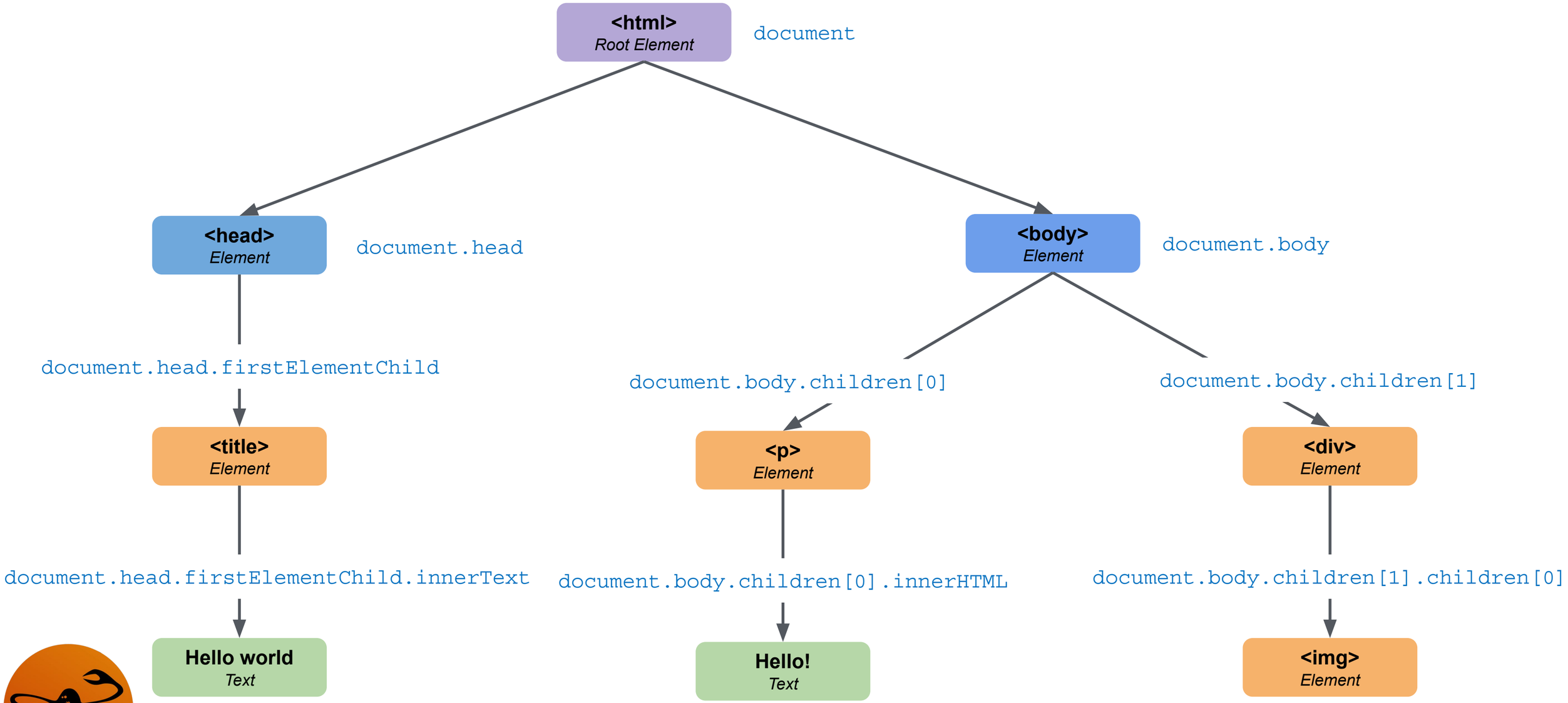
**El DOM es un árbol** que crea el navegador y lo pone en el contexto de JavaScript a través de la variable **document**.

A través **document**, podemos acceder a sus elementos, que son los elementos que hay en nuestro HTML.

```
<html>
   <head>
     <title>Hello World</title>
   </head>
   <body>
     <p>Hello!</p>
     <div>
        <img src="example.png"/>
     </div>
   </body>
</html>
```



# Navegando por el árbol



# Element

Los nodos que cuelgan de document.body o document.head, son de instancias **Element**.

La clase **Element** nos proporciona atributos y métodos para manipular estos nodos:

- **attributes:** array de atributos del nodo
- **innerHTML:** HTML contenido dentro del nodo
- **textContent:** texto incluído dentro del nodo
- **classList:** array de clases CSS que tiene el nodo
- **appendChild(<Element>):** permite añadir un elemento como hijo
- **removeChild(<Element>):** permite eliminar un elemento hijo
- **replaceWith(<Element>):** reemplaza el elemento actual por el que se pasa como parámetro

https://developer.mozilla.org/es/docs/Web/API/Element

# Seleccionando nodos

Desde cualquier nodo, podemos realizar búsquedas de elementos hijos para ir más rápido:

- `nodo.getElementById(<id>) -> [<Element>]`
- `nodo.getElementsByTagName(<tag>) -> [<Element>]`
- `nodo.getElementsByName(<name>) -> [<Element>]`
- `nodo.getElementsByClassName(<clase CSS>) -> [<Element>]`
- `nodo.querySelector(<selector CSS>) -> 1er <Element>`
- `nodo.querySelectorAll(<selector CSS>) -> [<Element>]`

# Crear y eliminar elementos

Desde cualquier nodo, podemos realizar búsquedas de elementos hijos para ir más rápido:

- `document.createElement('tag name') -> <Element>`
- `nodo.appendChild(<Element>)`
- `nodo.innerHTML = '<html code>'`
- `nodo.outerHTML = '<html code>'`
- `nodo.remove()`

# Manipulando atributos

Desde cualquier nodo, podemos realizar modificaciones de sus atributos:

- `nodo.attributes`
- `nodo.setAttribute(“<name>”, “<value>”)`
- `nodo.getAttribute(“<name>”) - > <value>`
- `nodo.hasAttribute(“<name>”) -> Boolean`
- `nodo.removeAttribute(“<name>”)`

# Manipulando estilos y clases CSS

Desde cualquier nodo, podemos realizar modificaciones de sus atributos y clases CSS:

- `nodo.style.<cssAttributeInCamelCase>`
- `nodo.className // valor del atributo class de HTML`
- `nodo.classList - > [<CSS name>]`
  - `nodo.classList.add(<CSS name>) // añade`
  - `nodo.classList.remove(<CSS name>) // elimina`
  - `nodo.classList.toggle(<CSS name>) // alterna`
  - `nodo.classList.contains(<class name>) - > Boolean`

# Eventos

# Eventos

JavaScript es un lenguaje orientado a eventos.

La filosofía general es poner callbacks o handlers a los eventos, de manera que todo funciona de manera reactiva.

Estos callbacks recibirán siempre como parámetro el objeto que identifica el evento.

https://developer.mozilla.org/es/docs/Web/Events

# Manejando eventos in-line

```
<button onclick='getElementById("demo").innerHTML=Date()'>The time is?</button>
```

```
<button onclick='doSomething()'>The time is?</button>

<script>
function doSomething() {…}
</script>
```

# Manejando eventos desde JavaScript

```
<button id='magicButton'>The time is?</button>

<script>
function doSomething() {…}

document.getElementById('magicButton').addEventListener('click', doSomething);

document.getElementById('magicButton').removeEventListener('click', doSomething);

</script>
```

# Promesas

# Involucrados en una promesa

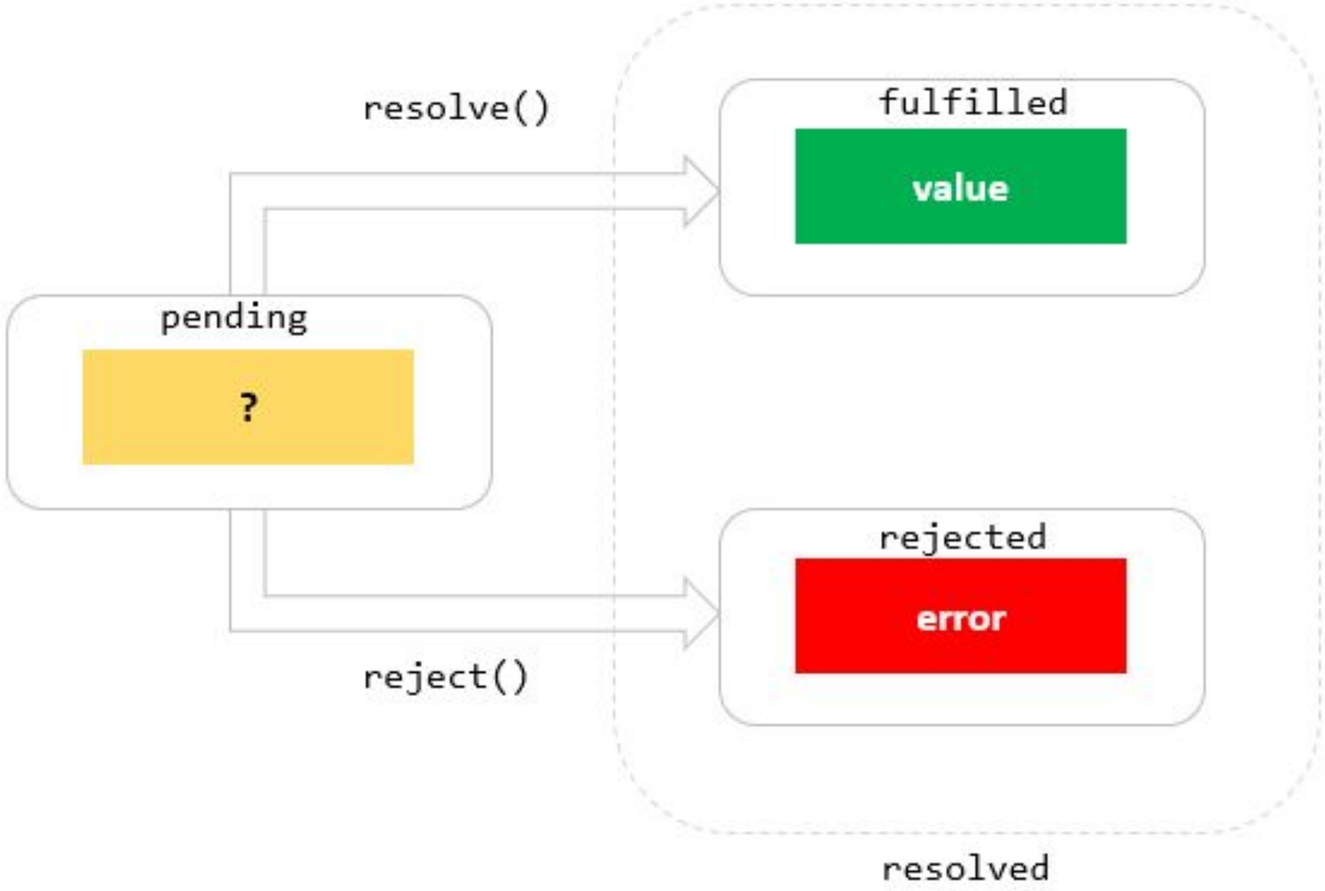
Hay 2 involucrados en una promesa. Podemos identificarlos como productor y consumidor.

- Productor: es quien gestiona un proceso asíncrono mediante una promesa y quien decide si la operación concluye bien o no.
- Consumidor: es quien reacciona ante el cambio de estado de la promesa.

# Estados

Los estados de una promesa son los siguientes

- *pending*
- *fulfilled*
- *rejected*




# Validación de formularios

# Validación de formularios

HTML5 valida los formularios por defecto. Desde JavaScript podemos hacer uso de su validation API para personalizar las validaciones y mensajes de error.

Es necesario que el formulario tenga el atributo ***novalidate***, para desactivar la validación automática del navegador y poder encargarnos nosotros de ello.

https://developer.mozilla.org/es/docs/Learn/Forms/Form_validation

# Peticiones HTTP con fetch

# Peticiones HTTP con fetch

Desde el navegador podemos realizar peticiones HTTP a servidores utilizando el API de fetch, basada en promesas.

```
fetch('https://jsonplaceholder.typicode.com/todos')
  .then(response => response.json())
  .then(data => console.log(data));
```

Y si queremos parsear el JSON...nos devuelve otra promesa.

https://developer.mozilla.org/es/docs/Web/API/Fetch_API

# Almacenamiento local

# Almacenamiento local

El estándar de HTML5 ofrece APIs JavaScript para el almacemiento local datos:

- **Cookies:** almacenamiento clave-valor que se envía y recibe del server
- **Storage:** almacenamiento clave-valor (localStorage & sessionStorage)
- **Web SQL:** base de datos relacional
- **Indexed DB:** base de datos documental

# localStorage & sessionStorage

- **localStorage:** los valores almacenados están disponibles mientras el usuario no borre los datos de navegación.
- **sessionStorage:** los valores almacenados están disponibles hasta que el usuario cierra la pestaña/ventana.

```
Storage.getItem(<key>) // devuelve <value> en <key>
Storage.setItem(<key>, <value>) // guarda <value> en <key>
Storage.removeItem(<key>) // elimina el <value> de <key>
Storage.clear() // elimina todos los valores
```

https://developer.mozilla.org/es/docs/Web/API/Storage

# localStorage & sessionStorage

- **Sólo podemos guardar valores primitivos** (no podemos guardar objetos o arrays).
- **Alternativa:** serializar/de-serializar

```
let obj = {a: 1, b: true, c: “hey”};
localStorage.setItem(‘serializedObj’, JSON.stringify(obj));
const serializedObj = localStorage.getItem(‘serializedObj’);
obj = JSON.parse(serializedObj);
```

# Twitter Clon

# GRACIAS

www.keepcoding.io