JavaScript

¿Qué es JavaScript?

- Es un lenguaje interpretado, es decir, no requiere compilación.
- Se ejecuta en el navegador del usuario. O sea se encarga de interpretar las sentencias JavaScript contenidas en una página HTML.
- Es un lenguaje orientado a eventos.
- Es un lenguaje basado en prototipos. Utiliza el concepto de prototipos para implementar o simular aspectos de la Orientación a Objetos.

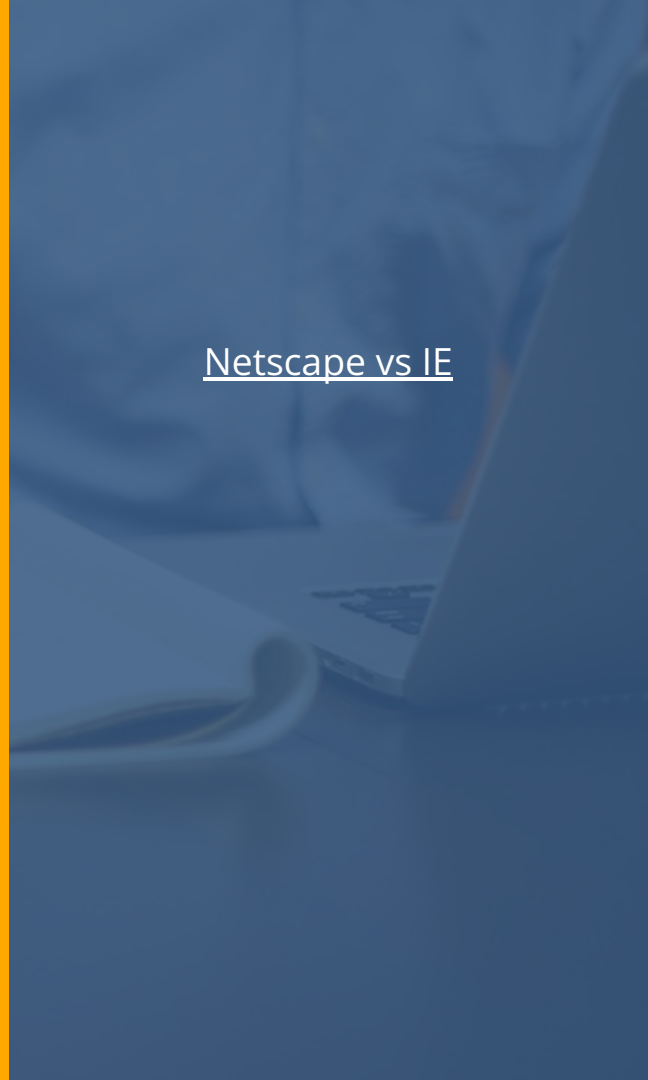
Historia de JavaScript

- Inventado por Brendan Eich (en 10 días) en 1995 en la empresa Netscape Communications.
- Apareció por primera vez en el producto de Netscape llamado Netscape Navigator.
- Inicialmente los autores lo llamaron Mocha (May. 95) y más tarde LiveScript (Sep. 95).
- Luego Livescript fue rebautizado como JavaScript en un anuncio conjunto entre Sun Microsystems y Netscape, el 4 de diciembre de 1995.

Historia

Historia de JavaScript

- En el 96 Microsoft implementa JScript, con características muy similares al JavaScript de Netscape en la ya conocida “guerra de los navegadores”.
- En junio de 1997 fue adoptado como un estándar ECMA. (como el W3C europeo) con el nombre de ECMAScript.
- ECMAScript 6 fue lanzado en 2015.
- En 2018 ya estaba soportado en todos los navegadores importantes.
- Desde el 2016, hay actualizaciones anuales.



Netscape vs IE

Uso de JavaScript

- Principal uso en el navegador.
- Aplicaciones móviles híbridas: Cordova, Ionic.
- Videojuegos: Phaser, PixiJS.
- Aplicaciones de escritorio: Electron.
- Servidor: nodeJS, deno.

Cordova

Ionic

Phaser

PixiJS

Electron

NodeJS

Deno

Incluir JavaScript

- Mediante la etiqueta <script>. Escribiendo código dentro de las etiquetas o importando un archivo externo con el atributo “src”.
- El atributo “type” es opcional. Por defecto es js.
- Se puede incluir en el <head> o en el <body>. Tener en cuenta el orden de ejecución.

```
<script type="text/javascript">
    function popup() {
        alert("Hello World")
    }
</script>

<script src=”principal.js”></script>
```

¿Dónde ponerlo?

defer y async

Sintaxis

- Similar a Java.
- Cuerpo de funciones con { }.
- Las sentencias terminan con ; (opcional).
- Cadenas de texto con “...”, ‘...’ o `...`.
- Declaraciones con **var**, **let** y **const**.
- Literales: 10, 99.99.
- Comentarios con /* */ o //
- Operadores: + - / * ** % ++ --
- Asignación: = += -= *= /= **= %=
- Relacionales: == === != !== < <= > >= ?
- Lógicos: && || !
- typeof instanceof

Sentencias

Sintaxis

let

const

Operadores

Alcance

- Existen 3 tipos de alcance: global, de función y de bloque*.
- Las variables declaradas dentro de un bloque ({...}) solo pueden ser accedidas dentro del bloque.
- Las variables declaradas dentro de una función son locales a la función.
- Las variables declaradas fuera de una función son globales.
- Asignar variables no declaradas las auto declara globalmente.

Alcance

*Solo las variables declaradas con **let** o **const** permiten alcance de bloque

Tipos de Datos

- En js tenemos tipos dinámicos.
- La misma variable puede albergar diferentes tipos.
- String, Number, Boolean, Object.
- Array, Date.
- undefined.

Tipos de Datos

typeof

Conversión de Tipos

String

Funciones de String

Number

Funciones de Number

Estructuras de Control

- if, else.
- switch.
- for, for in, for of.
- while, do while.

if else

switch

for

for in

for of

while

Try-catch-finally

- Similar a java.
- Errores predefinidos son objetos con la forma:
  {
    name: <String>,
    message: <String>
  }
- throw permite arrojar errores.

Errores

MDN - Errores

Funciones

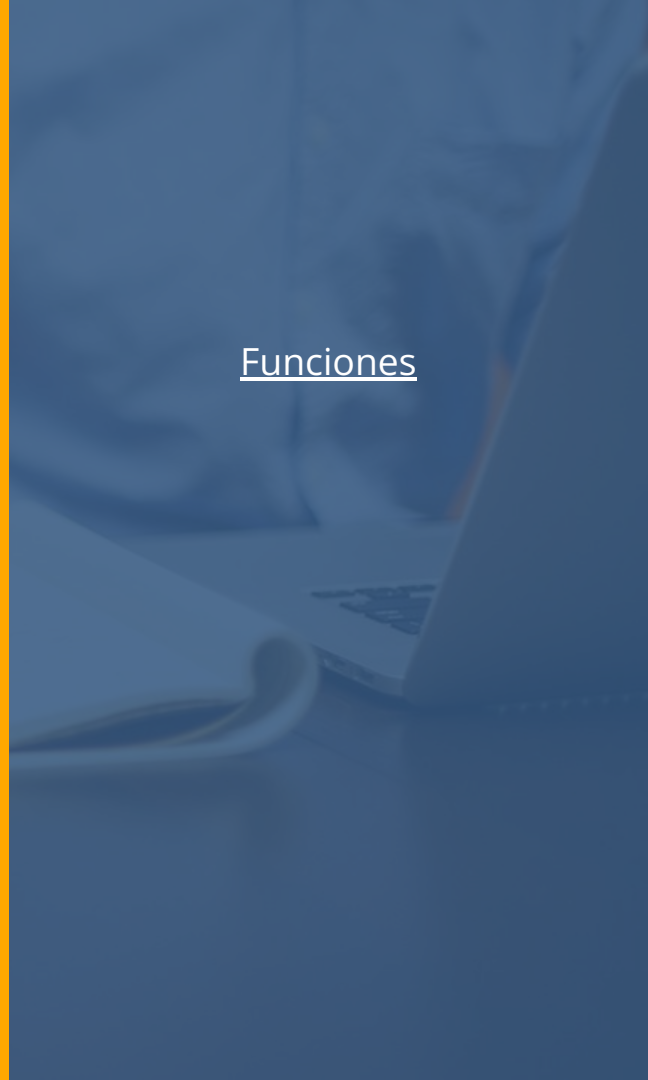
- Palabra clave **function**.

```
function multi(a, b) {
  return a * b;
}
```

- Uso de return opcional.
- Pueden ser almacenadas en una variable.

```
let multi = function(a, b) {
  return a * b;
}
let res = multi(3,2);//res => 6
```

- Tipos simples pasados por valor y objetos pasados por referencia.



Funciones

Funciones

- No se especifican los tipos de los parámetros ni del retorno.
- No hay control por cantidad de valores pasados.

```
function mult(a, b=1) {
  return a * b;
}
…
mult();//válido
mult(3);//válido
mult(3,2);//válido
mult(3,2,10);//válido
```

- Parámetros ausentes son considerados undefined. Salvo uso de valor por defecto.

Funciones

- El operador typeof aplicado a una función devuelve “function”.
- Si aplicamos .toString() devuelve la función.
- El objeto especial “arguments” es similar a un Array accesible dentro de funciones que contiene los valores de los argumentos pasados a esa función.
- Con “arguments.length” podemos saber la cantidad de parámetros recibidos en la función.

func.toString()

arguments

Arreglos

- Conjunto heterogéneo de elementos. Los elementos pueden ser de cualquier tipo.
- Índice basado en cero (0, 1, 2, ...).
- No tienen un tamaño fijo.

```
const autos = [“Fiat”, “Volvo”, “BMW”];
const nombres = [];
nombres[0] = “Pepe”;
nombres[39] = “Juan”;
```

- Los arreglos son objetos.
    - Usar: `Array.isArray(arr)`.
- `arr.push(e)` inserta al final.
- `arr.length` indica la longitud del arreglo.

Arreglos

Referencia Completa

Objetos

- Colección de valores nombrados.

```
const persona = {
  nombre: “Juan”,
  edad: 50,
};
persona.edad += 10;
console.log(“Nombre ” + persona.nombre);
console.log(“Edad ” + persona[“edad”]);//válido
console.log(“Persona: ”, persona);
```

- Pueden contener funciones también.

Objetos

Referencia Completa

Objeto Date

- Representa un momento en el tiempo.
- Contiene un número que representa los milisegundos desde el 1/1/70.
- Crear fechas con su constructor:

```
let d = new Date(2022, 11, 25);
console.log(“Navidad: ” + d);
```

- ¡Cuidado! El mes va de 0-11.
- Posee funciones getXYZ y setXYZ para obtener o establecer sus partes (segundos, minutos, día, etc.).

Fechas

Referencia Completa

Expresiones Regulares

- Permiten la búsqueda de patrones.
- Sintaxis:
  `/patrón/modificadores`
- Útil para búsqueda de patrones y para validar.
- Modificadores:
  - i: búsqueda no sensible a mayúsculas.
  - g: global, busca todas las apariciones en lugar de detenerse en la primera.
  - m: multilínea.

RegExp

Referencia Completa

Símbolo	Significado	Ejemplo	Cadenas válidas
^	Comienzo de línea	/^mo/	“moderna”, “mono”
$	Final de una línea	/ión$/	“Avión”, “Acción”
\w	Letra, Nº o _	/^\w\w\w$/	“_8_”, “sol”, “p2p”
\d	Dígito	/\d\d\d/	“123”, “456”, “555”
\D	Char NO dígito	/^\D\D$/	“do”, “re”, “mi”
\s	Espacio en blanco	/a\sa/	“a a“
( )	Agrupar	/^(sol)$/	“sol”
|	Opciones	/^(L|F)ocal$/	“Local”, “Focal”
[ ]	Conjunto	/le[aeo]/	“lea”, “lee”, “leo”

Metacaracteres

Símbolo	Significado	Ejemplo	Cadenas válidas
?	Opcional	/profe(sor)?/	“profe”, “profesor”
+	Una o más veces	/(a)+h/	“ah”, “aah”, “aaah”
*	Cero o más veces	/(o)*/	“”, “o”, “oo”, “ooo”
{n}	Exactamente *n* veces	/^w{3}/	“www”, “www….”
{n,m}	Entre *n* y *m* veces	/(ja){1,3}/	“ja”, “jaja”, “jajaja”
{n,}	Al menos *n* veces	/Gracia(s){2,}$/	“Graciass”, “Graciasssss”

Metacaracteres

Expresiones Regulares

- La función test permite ver si cumple el patrón.

```
let pat_mail = /^\w+@\w+(\.\w{2,4})+$/;
let email = "info@algo.com.ar";
pat_mail.test(email);//=>true
```

- Las funciones de string search y replace usan expresiones regulares como parámetros.

Test

Search

Replace

Cuadros de Diálogo

- Forma nativa de comunicación.
- No permite cambiar el estilo.
- Es bloqueante.
- alert(“Cartel”);
- let confirmacion = confirm(“Está seguro?”);
- let nombre = prompt(“Diga su nombre”);

Diálogos

abstract	arguments	await*	boolean
break	byte	case	catch
char	class*	const	continue
debugger	default	delete	do
double	else	enum*	eval
export*	extends*	false	final
finally	float	for	function
goto	if	implements	import*
in	instanceof	int	interface
let*	long	native	new
null	package	private	protected
public	return	short	static
super*	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	yield

Palabras Reservadas

¡Gracias!

¿Preguntas?

Plantilla de SlidesCarnival
Fotos de Startup Stock Photos