# Introdución

JavaScript é unha linguaxe que, segundo o estándar, pertence ao paradigma de Orientación a Obxectos. De todos xeitos, hai moitas referencias e moita xente que defende que se trata de unha linguaxe multiparadigma, que depende de como se utilice.

JavaScript xurde no ano 1995 en base a un encargo que lle fai *Netscape* a *Bendran Eich*. Hai unha anécdota que os detractores da linguaxe utilizan habitualmente que di que as súas supostas carencias (eu diría peculiaridades) son porque lle deron 10 días para facelo.

Inicialmente chamouse *Mocha*, despois pasou a chamarse *LiveScript* e, por último e definitivamente, denominouse JavaScript. Convén dicir que JavaScript e Java non teñen nada que ver. De feito o nome de Java(Script) foi más ben por unha cuestión de marketing xa que, en aquel momento, Java gozaba dunha moi boa reputación e tratouse de aproveitar para darlle pulo a esta nova linguaxe.

É unha linguaxe orientada para executarse nun navegador, de aí que teña características moi especiais que o definen, que o fan moi flexible e, nalgúns casos, algo raro de utiliza para xente que ten experiencia noutras linguaxes de programación máis estritas coma pode ser Java. Por tanto falamos dunha linguaxe de contorno cliente. Ben é certo que, nos últimos anos, o seu uso aumentou exponencialmente, sobre todo polas apostas de compañías como Google e polo seu uso en asincronismo. Isto fixo que xurdiran *frameworks* que lle deran un xiro a este concepto e abarquen os contornos de cliente e servidor (`node.js`).

Unha das cousas boas que ten JavaScript é que, co paso dos anos, chegou a ter un estándar controlado por ECMA e por W3C. Nos comezos isto non era así, de feito incluso había versións propias coma a de Microsoft (Jscript), pero pouco a pouco o mercado deuse conta que isto só aportaba problemas. Así que a día de hoxe todos os navegadores **intentan** implementar as especificacións que dita o estándar.

A día de hoxe, unha das versións máis utilizadas sería ECMAScript6, que xurde no ano 2015 e aporta diferentes novidades, por exemplo:

- `var -> const, let`
- `for ... of`

# Comentarios

Os comentarios son porcións de código que serán pasadas por alto polo intérprete, é dicir, non serán executados e que poden ser utilizados polos seguintes motivos:

- Para que o código sexa máis claro, matizando aquelas partes que dean lugar a dúbidas.
- Para aportar información sobre funcións, variables,...
- Para desactivar TEMPORALMENTE bloques de código que non queremos que sexa executado.
- Para deixar información de contacto.

Poden ser:

- De unha liña:

```
// Isto sería un comentario dunha liña
```

- De varias liñas:

```
/* E isto
sería
un comentario
de varias liñas */
```

Se queremos utilizar os comentarios dun xeito similar a JavaDoc en Java poderemos utilizar JSDoc

# Variables

Unha variable non é máis que un identificador asociado a un espazo en memoria no que pode variar o seu contido. De aí o seu nome. En versións antigas de JavaScript, para crear unha variable utilizábamos a palabra reservada `var` e despois un identificador que deberá cumprir cos seguintes requisitos:

- O nome da variable debe de ser suficientemente explicativo.
- Non utilizar palabras reservadas da linguaxe.
- Non utilizar espazos en branco.
- Ollo, JavaScript é case sensitive.
- Utilizar caracteres alfanuméricos máis o `_`. Non se poden utilizar `+`, `%`,...
- Non poden comezar por números.

Un exemplo:

```
var idade;
idade = 45;
```

Pero dende ECMA6 se recomenda encarecidamente utilizar `let` en vez de `var`. O motivo é que con var non se fan comprobacións como, por exemplo:

- Redeclaracións (volver a declarar máis abaixo a mesma variable).

```
var idade = 5;
var idade = 8;
```

Sen embargo, con `let` isto non ocorre. O código anterior, se usamos `let`, dará un erro.

En JavaScript non hai que indicar un tipo de dato explicitamente, é dicir, é unha linguaxe debilmente tipada ou dinámica, xa que determinarase en tempo de execución. Aínda que temos os seguintes tipos de datos soportados:

- **Booleanos ou valores lóxicos**.
- **Números**.
- **Cadeas.**
- **Arrays**.

```
let coches = ['Ford', 'Nissan', 'Skoda'];
console.log(coches[0]); // Amosará por consola Ford
```

- **Obxectos**.

```
let coches = { marca: 'Ford', modelo: 'Focus' };
console.log(coches.marca); // Amosará por consola Ford
```

O ámbito dunha variable é a zona na que se define a variable. Pode ser: global ou local. No caso de que exista conflito terá preferencia o ámbito local fronte o global.

- Global:

```
var mensaxe = "Mensaxe de proba";

function amosaMensaxe() {
	alert(mensaxe);
}
```

- Local:

```
var mensaxe = "Mensaxe de proba";

function amosaMensaxe() {
	var mensaxe = "Outra mensaxe de proba";
	alert(mensaxe);
}
```

Tamén existe un xeito diferente de declarar variables, sería coa palabra reservada `let`. Nun principio sería intercambiable con `var` pero `let` define un ámbito máis local. Por exemplo:

```
var a = 5;

if (a === 5) {
	let a = 4; // Alcance dentro do if
	alert(a);
}

alert(a);
```

Relacionado con isto, se executamos o seguinte código tamén atoparemos diferenzas:

```
alert(a); // Non da erro pero amosa undefined
alert(b); // Danos un erro

var a = 5;
let b = 3;
```

Outro tipo de "variables" serían as `const`, que serían variables que non van mudar de valor durante a execución do código. De feito se se tratan de mudar dará un erro.

```
const NUMERO = 7;
NUMERO = 9; // ERRO !!
```

Tamén debe asignarse sempre un valor na súa definición.

```
const NUMERO;
```

```
NUMERO = 79; // ERRO !!
```

# Conversións

Poden realizarse conversións entre tipos. Algúns exemplos:

- `integer + float = float`
- `numero + cadea = cadea`

```
console.log( 1 + 2 + '3'); // Amosará por consola 33
```

- `parseInt("32") -> integer`
- `parseFloat("32.1") -> float`
- `"" + numero -> cadea`

# Operadores

- Comparación

| Sintaxe | Nome | Operandos | Resultados |
|---|---|---|---|
| == | Igualdade | Todos | Boolean |
| != | Distinto | Todos | Boolean |
| === | Igualdade estrita | Todos | Boolean |
| !== | Desigualdade estrita | Todos | Boolean |
| > | Maior que | Todos | Boolean |
| >= | Maior ou igual que | Todos | Boolean |
| < | Menor que | Todos | Boolean |
| <= | Menor ou igual que | Todos | Boolean |

- Aritméticos

| Sintaxe | Nome | Operandos | Resultados |
|---|---|---|---|
| + | Máis | `integer, float, string` | `integer, float, string` |
| - | Menos | `integer, float` | `integer, float` |
| * | Multiplicación | `integer, float` | `integer, float` |
| / | División | `integer, float` | `integer, float` |
| % | Módulo | `integer, float` | `integer, float` |
| ++ | Incremento | `integer, float` | `integer, float` |
| -- | Decremento | `integer, float` | `integer, float` |
| +valor | Positivo | `integer, float, string` | `integer, float` |
| -valor | Negativo | `integer, float, string` | `integer, float` |

- Asignación

| Sintaxe | Nome | Exemplo | Significado |
|---|---|---|---|
| = | Asignación | x = y | |
| += | Suma un valor | x += y | x = x + y |
| -= | Resta un valor | x -= y | x = x - y |
| *= | Multiplica un valor | x *= y | x = x * y |
| **= | Exponenciar un valor | x **= y | x = x ** y |
| /= | Dividir un valor | x /= y | x = x / y |
| %= | Módulo de un valor | x %= y | x = x % y |
| <<= | Desprazar bits á esquerda | x <<= y | x = x << y |
| >>= | Desprazar bits á dereita | x >>= y | x = x >> y |
| >>>= | Desprazar bits á dereita sen signo | x >>>= y | x = x >>> y |
| &= | AND bit a bit | x &= y | x = x & y |
| \|= | OR bit a bit | x \|= y | x = x \| y |
| ^= | XOR bit a bit | x ^= y | x = x ^ y |

- Lóxicos

| Sintaxe | Nome | Operandos | Resultados |
|---|---|---|---|
| && | AND | Boolean | Boolean |
| \|\| | OR | Boolean | Boolean |
| ! | NOT | Boolean | Boolean |

- Bit a bit

| Sintaxe | Nome | Uso |
|---|---|---|
| & | AND bit a bit | x & y |
| \| | OR bit a bit | x \| y |
| ^ | XOR bit a bit | x ^ y |
| ~ | NOT bit a bit | ~x |
| << | Desprazar bits á esquerda | x << y |
| >> | Desprazar bits á dereita | x >> y |
| >>> | Desprazar bits á dereita con recheo de ceros | x >>> y |

- Ternario

  É a forma reducida da expresión `if ... else ...`

  A sintaxe é:

  condición ? expresión se se cumpre a condición: expresión se non se cumpre;

```
let estado = (idade >= 18) ? "adulto" : "menor";
```

- Unarios
  - typeof: permite amosar o tipo dun dato.

```
let numero = 23;
console.log(typeof numero); // number

let cadea = '23';
console.log(typeof cadea); // string
```

## Template Strings

Xa coñecemos un xeito de concatenar cadeas visto no apartado de operadores (+) pero, nalgún caso, pódese facer máis doado utilizar estes padróns que permiten o uso de expresións ou variables incrustadas. Un exemplo sería:

```
let cadea = `A miña cadea de texto`;
```

De momento o único que cambia sería o tipo de comiñas que, en vez de ser dobre ou simple, será a inclinada que aparece na tecla de [.

Os cambios aparecen cando queremos concatenar ou incluír variables dentro dela Por exemplo:

```
let cadea1 = ”Ola, o meu nome é “ + nome + “ e vivo en “ + cidade;
let cadea2 = `Ola, o meu nome é ${nome} e vivo en ${cidade}`;
```

Como se pode apreciar resultan moito máis cómodas e máis axeitadas cando queremos incluír moitas variables dentro da cadea, en vez de estar abrindo e pechando continuamente a cadea.

Os template strings aparecen con ECMAScript6.

# Consola

A consola do navegador é un elemento moi importante á hora de desenvolver o noso código JavaScript. Por un lado teremos o editor, que nos vai amosando erros, pero a consola permite facer seguimentos/debug en tempo de execución. Poderemos acceder a ela con diferentes atallos de teclado, teclas de función (F12), opcións de menú,... Teremos un obxecto que nos permitirá enviarlle mensaxes, será `console`. Podemos establecer diferentes niveis como son:

- `info`
- `log`
- `warn`
- `error`

Despois poderemos configurar a consola do noso navegador para que amose só os niveis que nos consideremos axeitados.

Outro elemento interesante será o comando `debugger`. Funcionará coma un punto de ruptura. A partires dese momento poderemos escoller na consola se queremos executar o noso código paso a paso.

# Estruturas de control

- Simple

  Sintaxe:

```
if (condición) {
	// instrucións a executar se se cumpre a condición
}
```

  Exemplo:

```
if (idade >= 18) {
	alert("A votar");
}
```

- Dobre

  Sintaxe:

```
if (condición) {
	// instrucións a executar se se cumpre a condición
} else {
	// instrucións a executar se NON se cumpre a condición
}
```

  Exemplo:

```
if (idade >= 18) {
	alert("Adulto");
} else {
	alert("Menor");
}
```

- Switch

  Sintaxe:

```
switch (expresion) {
	case et1:
		sentenzas1
		[break;] // Se se omite pasa á seguinte
	case et2:
		sentenzas2
		[break;]
	...
	default:
		sentenzas por defecto
		[break;]
```

```
}
```

Exemplo:

```
switch (tipoFruta) {
      case "Naranjas":
            console.log("Naranjas cuestan 0,59€ el kilo.");
            break;
      case "Manzanas":
            console.log("Manzanas cuestan 0,32€ el kilo.");
            break;
      case "Plátanos":
            console.log("Plátanos cuestan 0,48€ el kilo.");
            break;
      case "Cerezas":
            console.log("Cerezas cuestan 3,00€ el kilo.");
            break;
      case "Mangos":
            console.log("Mangos cuestan 0,56€ el kilo.");
            break;
      case "Papayas":
            console.log("Mangos y papayas cuestan 2,79€ el kilo.");
            break;
      default:
            console.log("Disculpa, no tenemos el tipo de fruta " + tipoFruta + ".");
}
console.log("¿Te gustaría tomar algo?");
```

# Bucles

Un bucle, ciclo ou loop é unha execución repetida dun anaco ou bloque de código ben definido ata que se cumpre unha condición.

- For: permite repetir un bloque de instrucións un número limitado de veces.

  Sintaxe:

```
for (expresión inicial; condición; incremento) {
      // Instrucións a executar dentro do bucle
}
```

  Exemplo:

```
for (let  i=0;  i<=20;  i++) {
      console.log("i: " + i);
}

// Amosará na consola: 0, 1,... 20
```

- While: permite repetir un bloque de instrucións un número indefinido de de veces MENTRES se cumpre unha condición.

  Sintaxe:

```
while (condición) {
      // Instrucións a executar dentro do bucle
}
```

  Exemplo:

```
let i=0;
while  (i<=20) {
      console.log("i: " + i);
      i++; // Non podemos esquecer modificar a variable de control
}

// Amosará na consola: 0, 1,... 20
```

- Do … While: permite repetir un bloque de instrucións un número indefinido de de veces mentres se cumpre unha condición, tamén. A diferenza está na avaliación da condición do bucle, xa que no `while` faise antes de entrar no bucle e no `do while` esta comprobación faise despois.

  Sintaxe:

```
do {
	// Instrucións a executar dentro do bucle
} while (condición);
```

  Exemplo:

```
let i=0;
do {
	console.log("i: " + i);
	i++; // Non podemos esquecer modificar a variable de control
} while (i<=20);

// Amosará na consola: 0, 1,... 20
```

## Bibliografía:


- https://www.edu.xunta.es/fpadistancia
- https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia
- https://developer.mozilla.org/es/docs/Web/JavaScript/Guide
- https://www.w3schools.com/js/default.asp
- https://uniwebsidad.com/libros/javascript
- https://openclassrooms.com/en/
- https://www.arkaitzgarro.com/javascript/index.html
- https://openwebinars.net/
- https://desarrolloweb.com/
- https://caniuse.com
- https://enlacima.co/
- https://www.arkaitzgarro.com/javascript/
- https://javascript.info/bubbling-and-capturing
- https://stackoverflow.com/
- https://codepen.io/
- Os meus alumnxs.