Índice

Funcións

Segundo os principios da programación modular baseados no concepto de “divide e vencerás” cada subproblema é desexable que sexa independente dos demais. A este subproblema se lle denomina módulo e, en moitos casos, vese implementado por procedementos e funcións. Estes módulos ou funcións tamén favorecen a optimización do código xa que permiten:

- Eliminar liñas repetidas.
- Modificacións nun único punto.

A sintaxe das funcións en JavaScript será:

```
function nomeFuncion([parametros]) {
	// Accións a realizar
}
```

E un exemplo podería ser:

```
let resultado;
let n1 = 5;
let n2 = 10;
function suma() {
	resultado = n1 + n2;
	alert("O resultado é " + resultado);
}
```

Neste caso non se lle pasan parámetros pero tamén se poden enviar valores do seguinte xeito:

```
function nomeFuncion(p1, p2,...) {
	// Accións a realizar
}
```

E modificando o exemplo anterior:

```
function suma(n1, n2) {
	let resultado = n1 + n2;
	alert("O resultado é " + resultado);
}
```

Esa sería a definición da función. A chamada sería do seguinte xeito:

```
suma();
```

e

```
suma(5,10);
```

O número de parámetros na chamada debe coincidir co número de parámetros da definición, aínda que JavaScript non da erro se non se fai así. De todos xeitos, dende ECMAScript6 aparecen os parámetros por defecto para previr isto:

```
function suma(n1, n2 = 0) {
	let resultado = n1 + n2;
	alert("O resultado é " + resultado);
}
```

Tamén pode darse o caso de que quereramos que a comunicación coa función sexa de saída e non só de entrada. Nese caso a función pode devolver un valor. O único que teremos que indicar, ao final do código da función, será a palabra reservada `return` e o valor que queremos que devolva. Por exemplo:

```
function suma(n1, n2) {
	let resultado = n1 + n2;
	alert("O resultado é " + resultado);
	return resultado;
}
```

Despois existen as denominadas como ***funcións callback***, que son funcións que se pasan como parámetro de outras funcións). Un exemplo de uso sería:

```
function exemplo(miFun) {
	miFun();
}
exemplo(function() {
	alert("ola mundo");
});
```

Pode parecer moi enrevesado este uso pero para casos concretos coma `forEach` en Arrays, xestores de eventos,... resulta moi útil (consultar exemplo no apartado de Arrays).

Poden levar parámetros (coma o valor e o índice no caso dos Arrays):

```
function exemplo(miFun) {
	let p = "Luis";
	miFun(p);
}
exemplo(function(p) {
	alert("ola " + p);
});
```

Temos a posibilidade de utilizar as ***funcións anónimas***, que non é máis que asignar a unha variable o resultado dunha función que non ten identificador:

```
let cadrado = function (n) { return n * n };
let resultado = cadrado(2); // chamada
```

E temos as ***funcións arrow*** ou ***funcións frecha***, cuxo paso dende as funcións anónimas case é directo:

```
let cadrado = (n) => { return n * n };
let resultado = cadrado(2); // chamada
```

Estamos substituíndo a palabra chave `function` por unha sintaxe máis directa. E aínda se pode reducir máis:

```
let cadrado = (n) => n * n;
let resultado = cadrado(2); // chamada
```

As chaves as utilizaremos no caso de que queiramos que a lóxica da función sexa máis complicada e leve máis dunha instrución.

As funcións frecha teñen sentido ou presentan vantaxes fronte as funcións “convencionais” porque pode que se requira algún tipo de peche, chamada ou vinculación para garantir que se execute nun ámbito axeitado. Como se recolle nun exemplo de MDN:

```
window.age = 10;

function Person() {
  this.age = 42;
  setTimeout(function () { // A función tradicional execútase no ámbito de window
    console.log("this.age", this.age); // Amosa "10" polo ámbito
  }, 100);
}

let p = new Person();
```

```
window.age = 10;

function Person() {
  this.age = 42;
  setTimeout(() => { // Función frecha executándose no ámbito de "p"
    console.log("this.age", this.age); // Amosa "42" polo ámbito
  }, 100);
}

let p = new Person();
```

Arrays

Un Array permite almacenar varios valores do mesmo tipo nunha mesma variable. De todos xeitos en JavaScript podemos incluír valores de diferentes tipos (cadeas, números,...). Por exemplo:

```
let coches = ["Saab", "Volvo", "Honda"];
let misc = ["Saab", "Volvo", 46];
```

Aínda que tamén se poden crear dun xeito similar á teoría de obxectos.

```
let coches = new Array ("Saab", "Volvo", "Honda");
```

De todos xeitos, en diferente documentación, se recomenda non utilizar esta sintaxe fronte á do primeiro exemplo. Algúns dos motivos son por simplicidade, facilidade de lectura e velocidade de execución.

Para acceder aos elementos do Array farémolo utilizando os índices ou posición que ocupan no mesmo. Hai que ter en conta que comezarán sempre na posición 0.

```
let arr_coche = coches[0]; // Será o Saab
```

E tamén poderemos modificar o contido de cada un dos elementos do Array.

```
coches[0] = "Ford";
```

Outro xeito de facelo, tamén próximo á teoría de obxectos, sería o seguinte:

```
let coches = {marca:"Saab", modelo:"Volvo"};
...
alert(coches.marca);
alert(coches["modelo"]);
```

Tede en conta que se aproveitamos esta capacidade estaremos indicando explicitamente que o Array é un obxecto e haberá propiedades (coma `length`) que deixarán de funcionar correctamente e dará un erro. Así que se utilizamos Arrays usaremos números como índices e se utilizamos obxectos usaremos nomes coma índices.

Engadir elementos tamén resulta moi sinxelo coa propiedade push/unshift.

```
coches.push("Ford");
```

Ollo porque se tratamos de facelo a través de índices poderemos ter problemas deixando buratos no medio.

Ao traballar con Arrays podemos utilizar propiedades propias deles como, por exemplo:

- **length**: devolve a lonxitude ou o número de elementos do array.

E métodos coma:

- **valueOf()**: devolve TODO o contido do Array.
- **indexOf()**: devolve a posición do elemento que lle indicamos. No caso de que non exista devolverá -1.

- **sort()**: ordena alfabeticamente os elementos do array.
- **reverse()**: inverte a posición dos elementos dun Array.
- **slice()**: devolve unha copia dunha parte do Array dentro dun novo Array comezando por inicio ata fin (non incluído).

```
let nomes = ['Victoria', 'Eva', 'Adrián', 'Julio', 'Manuel'];
let mulleres = nomes.slice(0, 2);

// mulleres contén ['Victoria','Eva']
```

- **push()**: inserta un elemento o final do Array.
- **pop()**: elimina un elemento ao final do Array.
- **shift()**: elimina un elemento ao comezo do Array.
- **unshift()**: inserta un elemento ao comezo do Array.

En canto aos bucles, existen varios xeitos de percorrer un array:

- O clásico:

```
let coches = ["Saab", "Volvo", "Honda"];
for (i = 0; i < coches.length; i++) {
      coches[i]...
}
```

- Un propio dos arrays e máis doado (pese a que a que se desaconsella o seu uso por *deprecated*):

```
var coches = ["Saab", "Volvo", "Honda"];
for each (i in coches) {
      coches[i]...
}
```

- De feito é mellor utilizar este método:

```
let coches = ["Saab", "Volvo", "Honda"];
for (i in coches) {
      coches[i]...
}
```

- Outro propio dos arrays e máis utilizado:

```
texto = "<ul>";
coches.forEach(miFuncion);
texto += "</ul>";
document.getElementById("demo").innerHTML = texto;

function miFuncion(valor) {
      texto += "<li>" + valor + "</li>";
}
```

- ou tamén:

```
let coches = ["Saab", "Volvo", "Honda"];
coches.forEach(miFuncion);

function miFuncion(elemento, indice) {
	alert(indice + ": " + elemento);
}
```

- En JavaScript, tal e como anunciamos no apartado de funcións, pódese incluír como parámetro a implementación dunha función. De feito, este é o xeito máis usual:

```
const coches = ["Saab", "Volvo", "Honda"];

coches.forEach(function (elemento, indice) {
	alert(indice + ": " + elemento);
});
```

- A partires de ECMAScript6 aparece `for ... of`.

```
const coches = ["Saab", "Volvo", "Honda"];

for (const item of coches) {
	console.log(item); // item xa non é un índice senón que contén o valor
});
```

- E non esquezamos que un *String* non deixa de ser un *Array* tamén.

```
const coches = "Saab";

for (const item of coches) {
	console.log(item); // Amosará:
                       // S
                       // a
                       // a
                       // b
});
```

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.