Funciones JS

Las funciones son una parte fundamental de JavaScript y de cualquier otro lenguaje de programación. Son un conjunto de instrucciones que se pueden ejecutar repetidamente en diferentes partes de un programa. Las funciones tienen varias ventajas:

- Permiten dividir un programa en partes más manejables y organizadas.
- Ayudan a reutilizar código evitando tener que escribir el mismo código varias veces.
- Facilitan el depurado y la mantención de un programa, ya que permite localizar y corregir errores en una sola parte del código en lugar de tener que buscarlos en todo el programa.

La sintaxis de una función en JavaScript es la siguiente:

```
function nombreDeLaFunción(parametros) {
  // código a ejecutar
}
```

Los parámetros son variables que se pasan a la función y que se pueden utilizar dentro de ella. Una función puede tener cero o más parámetros.

Para llamar a una función, se utiliza su nombre seguido de paréntesis y, en algunos casos, de argumentos entre ellos. Los argumentos son valores que se pasan a la función y que se asignan a los parámetros de la función.

Aquí tienes un ejemplo de una función que suma dos números:

```
function suma(a, b) {
  return a + b;
}
```

```
let resultado = suma(3, 4); // resultado = 7
```

En este caso, la función `suma` tiene dos parámetros: `a` y `b`. Al llamar a la función con `suma(3, 4)`, se asignan los valores 3 y 4 a los parámetros `a` y `b`, respectivamente. La función devuelve la suma de `a` y `b`, que en este caso es 7.

Espero que esto te haya ayudado a entender cómo funcionan las funciones en JavaScript. Si tienes alguna duda o necesitas más información, no dudes en preguntar.

Declaración de funciones

Para declarar una función tenemos que usar la palabra reservada **function** + el nombre de mi función **name-function** + unos paréntesis, que es donde nos llegarán nuestros parámetros **( )** y finalmente las llaves **{ }** en las cuales realizaremos nuestras operaciones.

```
function pokemonAttack() {
  console.log( '¡Pikachu Impact Trueno!');
}
```

La palabra clave **function** va primero, luego va el nombre de función, luego una lista de parámetros entre paréntesis **( )** y finalmente el código de la función, también llamado “el cuerpo de la función” , entre llaves.

```
function pokemonAttackParams(name , attack) {
  console.log(`¡${name} ${attack}! `);
}
```

Si os fijáis en esta función por **parámetro** entran dos elementos o parámetros que son name y attack, de esto modo nuestra función sacará por consola los

parámetros introducidos.

Una vez tenemos nuestras funciones declaradas tenemos que invocarlas o ejecutarlas para que realicen la funcionalidad programada.

```
pokemonAttack();
pokemonAttackParams('Charmander', 'Ascuas');
pokemonAttackParams('Squirtel', 'Pistola de agua');
```

Valores predeterminados

Si una función es llamada pero no se le proporciona un argumento o parámetro , su valor correspondiente se convierte en **undefined**.

```
function pokemonAttackParams(name , attack) {
  console.log(`¡${name} ${attack}!`);
}

pokemonAttackParams('bulbasaur');
```

Aquí estamos invocando la función pero le estamos pasando solamente un argumento por parámetro, ¿Qué sucederá? Nos mostrará **"¡Bulbasaur undefined!"**. Como no se pasa un valor de **attack**, este se vuelve **undefined**.

Imaginar querer tener un ataque por defecto cuando no le pasemos el argumento **attack** a nuestra función. Podemos especificar un valor llamado “default” (predeterminado) en la declaración de función usando **=**.

```
function pokemonAttackParamsDefault(name, attack = 'Ataque ar
  console.log(` ¡${name} ${attack}! `);
}

pokemonAttackParamsDefault('Onix'); // ¡Onix Ataque arena!
```

Ahora, si no existe el parámetro **attack**, obtendrá el valor **'Ataque arena'** .

Aquí **'Ataque arena'** es un string, pero puede ser una expresión más compleja, la cual solo es evaluada y asignada si el argumento pasado por parámetro falta.

```
function pokemonAttackParamsDefault(name, attack = defaultAtt
  // defaultAttack() solo se ejecuta si attack no fué asignad
  // su resultado se convierte en el valor de texto
    console.log(` ¡${name} ${attack}! );
}

function defaultAttack() {
    return "Ataque arena"
}
```

Retorno en las funciones

Hasta ahora nuestras funciones simplemente manipulaban datos y los mostraban por consola. ¿Qué sucede si queremos manipular o transformar un dato y devolver un valor nuevo? Tenemos que retornar el elemento manipulado o transformado y lo hacemos a través del **return**.

```
function suma (numA, numB) {
    return numA + numB;
}

let result = suma(5, 20); // valor es 25
```

> **Estudiantes nunca debemos añadir código después de nuestro *return* puesto que nunca se ejecutará.**

Es posible utilizar return sin ningún valor. Eso hace que la función salga o termine inmediatamente. Si checkAge(age) devuelve false, entonces

showMovie no mostrará el valor por consola.

```
function showMovie(age) {
  if ( !checkAge(age) ) {
    return;
  }

  console.log( "Mostrándote la película" );
}
```

Alternativa a parámetros indeterminados

A veces tiene sentido asignar valores predeterminados, no en la declaración de función ,sino en un estadio posterior. Podemos verificar si un parámetro es pasado durante la ejecución de la función comparándolo con **undefined** .

```
function showPokemon(name) {
  if (name === undefined) { // si falta el parámetro
    name = 'Magikarp';
  }

  console.log(name);
}

showPokemon(); // Magikarp
```

O podemos usar el operador **||** .

```
function showPokemon(name) {
  // si name es indefinida o falsa, la establece a 'Magikarp'
  name = name || 'Magikarp';
}
```

Intérpretes JavaScript modernos soportan el operador **nullish coalescing** , que es representado con **??** .

```
function showCount(count) {
  // si count es undefined o null, muestra "desconocido"
  console.log(count ?? "unknown");
}

showCount(0); // 0
showCount(null); // desconocido
showCount(); // desconocido
```

Nombrado de Funciones

Las funciones son acciones. Entonces su nombre suele ser un verbo. Debe ser breve, lo más preciso posible y describir lo que hace la función, para que alguien que lea el código obtenga una indicación de lo que hace la función.

```
showMessage(..)     // muestra un mensaje
getAge(..)          // devuelve la edad (la obtiene de alguna
calcSum(..)         // calcula una suma y devuelve el resulta
createForm(..)      // crea un formulario (y usualmente lo de
checkPermission(..) // revisa permisos, y devuelve true/false
```

Arrow functions

Es una manera de generar funciones más compacta y además nos ayudará a entender mejor los problemas del **scope**. Al ver la sintaxis, ya no hace falta escribir la palabra **function**, lo sustituimos por la flecha **⇒** .

Las arrow function tienen la capacidad de capturar el objeto **this** del contexto donde la **arrow** se ejecuta y así utilizarlo dentro de su bloque de sentencias. Esto lo trataremos en otro post con algunos conceptos de Javascript.

```
const getName = () => {
  console.log('Devolviendo nombre');
  return 'Carlos';
};
```

```
const name = getName();

console.log(name);

// Ejemplo inline (omitiendo el return)
const getSurname = () => 'Martín';

const surname = getSurname();

console.log(surname);
```

En el siguiente ejemplo, la función nos devuelve un mensaje 'hola mundo' sin la necesidad de tener un return.

```
const helloWorld = () => 'hola mundo';
const hello = helloWorld();

console.log(hello);
```

En caso de querer o tener que usar un return.

```
const helloWorld = () => {
  const messageToWorld = 'hello world';
  return messageToWorld;
}
```

En el caso de querer devolver un **object** inline, la sintaxis deberá ser la siguiente.

```
const myObjt = () => ({ atribute: 'atribute', atribute: 'atri
```

Argumentos en las arrow functions

Como ya sabéis en las funciones pueden entrar parámetros que indiquen los valores con los que vamos a trabajar en el caso de las arrow functions.

```
const multiTwo = x => x * 2;

const result = multiTwo(3);
```

En el resto de casos sí que necesitaremos especificar los paréntesis.

```
const multi = (a, b) => a * b;

const multiplication = multi(2, 2);
```

Existe la opción de tener valores por defecto.

```
// b siempre será 3

const multiDefault = (a, b = 3) => a * b;

const operation = multiDefault(2);
```

Limitaciones Arrow functions

Las funciones "flecha" o "arrow functions" son una característica de JavaScript introducida en la versión 6 de ECMAScript. Se caracterizan por tener una sintaxis más corta y legible que las funciones tradicionales.

Aunque las arrow functions tienen muchas ventajas, también tienen algunas limitaciones y diferencias respecto a las funciones tradicionales. A continuación, te menciono algunas de ellas:

1. **No tienen una palabra clave** `this`: En una función tradicional, la palabra clave `this` se refiere al objeto que llama a la función. Sin embargo, en una arrow function, `this` se refiere al objeto que contiene la función. Esto

puede ser un problema en algunos casos, como cuando se necesita acceder al objeto que llama a la función.

2. **No pueden ser usadas como constructores**: Las arrow functions no pueden ser utilizadas con la palabra clave `new`. Si se intenta hacer esto, se lanzará un error.
3. **No tienen una palabra clave** `arguments`: La palabra clave `arguments` es una variable local que se crea en todas las funciones y que contiene una lista de los argumentos pasados a la función. En una arrow function, no se puede acceder a `arguments`.
4. **No pueden ser usadas como métodos**: Las arrow functions no pueden ser utilizadas como métodos de objetos. Si se intenta hacer esto, se lanzará un error.

Aquí tienes un ejemplo de una arrow function que tiene una de estas limitaciones:

```
const obj = {
  name: 'John',
  sayHi: () => {
    console.log(`Hi, my name is ${this.name}`);
  }
};

obj.sayHi(); // "Hi, my name is undefined"
```

En este caso, la arrow function `sayHi` no puede acceder al objeto `obj` utilizando la palabra clave `this`. Como resultado, el mensaje que se imprime en la consola es "Hi, my name is undefined".

Espero que esto te haya ayudado a entender algunas de las limitaciones de las arrow functions en JavaScript. Si tienes alguna duda o necesitas más información, no dudes en preguntar.