A function is a block of code designed to perform a task.

Functions are like recipes. They accept data, perform actions on that data, and return a result. The beauty of functions is that they allow us to write a block of code once, then we can reuse it over and over without rewriting the same code.

## Función con variable cerrojo: *Calculator ON/OFF*

1. We create a function named: **pressPowerButton**

* *const* **pressPowerButton 🡪** creates a variable with a given name given in *lowerCamelCase.*
* The variable is then set to equal **=** to a set of parentheses followed by an arrow token **() =>** , indicating the variable stores a function. This sintax is known as *arrow function sintax: const* **pressPowerButton =** **() =>**
* Finally between the curly braces **{}** is the function body or the JavaScript statements that define the function. This is followed by a semi-colon **;** in JavaScript, any code between curly braces is also known as a *block.*

1. Inside the function body is an **if / else** statement
2. On the last few lines, we call the function by writing its name followed by a semi-colon**;** this executes the function, running all code within the function body.

***let* calculatorIsOn = false;**

***const* pressPowerButton = () => {**

**if (calculatorIsOn)**

**{ *console*.log("Calculator turning off."); calculatorIsOn = false; }**

**else**

**{ *console*.log ("Calculator turning on."); calculatorIsOn = true; }**

**};**

**pressPowerButton();** //Output: Calculator turning on.

**pressPowerButton();** //Output: Calculator turning off.

1. We execute the code in the function body twice without having to write the same set of instructions twice. Functions can make code reusable!

The purpose of a function is to take some input, perform some task on that input, then return a result. Using **return** keyword is generally a best practice when writing functions, as it makes your code more maintainable and flexible.

Functions in JavaScript are generally declared as either a *function declaration* or a *function expression.* Function expressions end with a semi-colon since they are stored in a variable. Arrow function syntax is a shorter syntax for a function expression. You can identify arrow functions through the use of parentheses and the arrow token **() =>**.

Function declarations do not end in a semi-colon, they require the ***keyword*** ***function***, a name, and a function body.

if a function takes zero or multiple parameters, parentheses are required. A function composed of a sole single-line block is automatically returned: in this case: curly braces (brackets) and return keyword can be omitted. A function composed of a sole single-line block is automatically returned. This is referred to as *implicit return*.

*Parameters* are variables in a function definition that represent data we can input into de function. Then we call the function by its name and the argument(s) which are the value(s) of the parameter(s). Block scope means that a variable defined in the block is only accessible within the curly braces. The best practice is to declare the variables of the inside of the block scope of the function.

### Declaramos la función

***function* miProcedimiento( *parameter(s)* )**

**{**ámbito del procedimiento / qué es lo que hace**}**

El ***parámetro*** formal es como si fuera una variable pero que sólo existe dentro de la función.

**Scope/ámbito:** lo que pasa dentro de la función se queda en la función.

### LLamamos al procedimiento

**miProcedimiento**( El ***argumento*** efectivo que queremos que ejecute el ámbito de la función lo definimos cada vez que la llamamos).

Entre el nombre de la función/procedimiento y el paréntesis del argument/parámetro NO HAY ESPACIO!!!. Cuando la secuencia de código devuelve un valor, son funciones; en caso contrario son procedimientos. En ambos casos se declaran con la palabra reservada ***function***

### Ejemplos

***function* esPar( *theNumber* )** **{**

**if** **(** theNumber **%** **2** **==** **0)**

**{ return true; }**

**else**

**{ return false; } }**

***console*.log( esPar( 8 ) ); muestra true en pantalla**

**return** es necesario siempre que vayamos a pedirle a la función que me devuelva algo, sin embargo, lo que le pidamos que devuelva es proceso (PROCESS). Cuando pongo un return se rompe el flujo de ejecución (como con el break en el switch case).

***function* sumarNNumeros() {**

***var*** total **= 0;  
 *var*** i;

**for (**i **= 0;** i **< *arguments***.length**;** i**++) //.length** without (): is an attribute, not a method

**{** total **+= *arguments*[i];}**

**return** total**;**

**}**

***document*.write( sumarNNumeros(2, 3, 6, 10) );**

**arguments** es una palabra que tiene reservada Js (keyword) para definir un array dentro de una función.

/\* función con variable cerrojo \*/

***let* calculatorIsOn = false;**

***const* pressPowerButton = () => {**

**if (calculatorIsOn)**

**{**

***console*.log("Calculator turning off.");**

**calculatorIsOn = false;**

**}**

**Else**

**{**

***console*.log("Calculator turning on.");**

**calculatorIsOn = true;**

**}**

**};**

**pressPowerButton();** //Output: Calculator turning on.

**pressPowerButton();** //Output: Calculator turning off.

/\* llamada de argumentos y funciones \*/

***let* orderCount = 0;** //inicialización de variable

***const* takeOrder = (topping, crustType) => {**

**orderCount = orderCount + 1;** //número de pedidos o de llamadas de la función

***console*.log('Order: ' + crustType + ' pizza topped with ' + topping);**

**};**

**takeOrder("mushrooms", "thin crust");** // primera llamada función

**takeOrder("bacon", "classic");** // segunda

**takeOrder("chicken", "american style");** // se incrementa la variable orderCount

***const* getSubTotal = ( itemCount ) => {**

**return itemCount \* 7.5;** // Devuelve el parámetro que se le pase en la llamada \* 7.5

***console*.log( "Precio sin impuestos = " + getSubTotal( orderCount ) );**

**};**

***const* getTax = () => {** // devuelve una llamada a otra función

**return getSubTotal( orderCount ) \* 0.06;**

***console*.log( "impuestos = " + getTax() );**

**}**

***const* getTotal = () => {**  //devuelve una operación tras llamar a dos funciones

**return getSubTotal(orderCount) + getTax();**

**}**

***console*.log( getTotal() );**

<https://www.infojobs.net/tres-cantos/ings.-electronica-espacio-potencia-test-egses/of-id268276f854619a73cbdac7aa7587a?applicationOrigin=search-new%7Celement%7E32159557532>

## FUNCIONES

Becarios: les declaramos lo que queremos que haga cuando les llamemos y cómo (argument). Las funciones son verbos. Los objetos sustantivos. Un proceso es un flujo de ejecución de datos. El proceso son verbos o acciones mientras que los datos son sustantivos.

|  |  |  |  |
| --- | --- | --- | --- |
| ACTION | | EFFECT | |
| Event | Data: **Element Trigger** | Data: **Element Target** | **Change** |
| Process: on-event | 2 states: Boolean   * ON * OFF | Boolean Light   * OFF * ON | Process: {scope} |

Cuando el elemento trigger está en ON, el target está en OFF y viceversa. Todos los eventos del elemento trigger se pueden definir en HTML con un atributo que comienza con on…=”handler”

# Ejercicio 1

**Se pide implementar el codigo necesario para mostrar:**

**- Una imagen**

**- Un boton con dos estados: apagado y encendido**

**Y el siguiente comportamiento:**

**- Cuando el usuario hace click en el boton...**

**--> El boton alterna su aspecto (estilo y contenido)**

**--> La imagen se cambia por otra**

***Con una variable global y una sola <img>***

***var* mySwitch = true;**

***function* toogle() {** // SCOPE FUNCTION: block

**if ( mySwitch == true )**

**{**

**mySwitch = false;  
*document*.getElementById(“myButton”).style.backgroundColor = “cambio de color”;**

***document*.getElementById("myButton").innerHTML = "cambio de contenido";**

***document*.getElementById("image").src = "cambio de ruta de la imagen";**

**}**

**else**

**{**

**mySwitch = true;  
*document*.getElementById(“myButton”).style.backgroundColor = “cambio de color”;**

***document*.getElementById("myButton").innerHTML = "cambio de contenido";**

***document*.getElementById("image").src = "cambio de ruta de la imagen";**

**}**

**}**

**<img> 🡪 CSS style: display;**

**<button** **Onclick=”toogle()**”**>** EVENT LISTENER (Handler)

***Con una variable global y dos <img>***

Variable global = variable cerrojo booleana.

***var* mySwitch = true;**

***function* toogle() {** // SCOPE FUNCTION: block

***var* button = *document*.getElementById("myButton");**

***var* img1 = *document*.getElementById("myImg1");**

***var* img2 = *document*.getElementById("myImg2");**

**if ( mySwitch == true )**

**{**

**mySwitch = false;  
button.style.backgroundColor = “cambio de color”;**

**button.inner = "cambio de contenido";**

**img1.style.display = "none";**

**img2. style.display = "inline";**

**}**

**else**

**{**

**mySwitch = true;  
button.style.backgroundColor = “cambio de color”;**

**button.inner = "cambio de contenido";**

**img1.style.display = "inline";**

**img2.style.display = "none";**

**}**

**}**

**<img** **src** **>**

**<button** **Onclick=”toogle()**”**> //** EVENT LISTENER (Handler)

***Sin variables globales y una sola <img>***

***function* toogle(*element*) {** // SCOPE FUNCTION: block

**if ( element.style.backgroundColor = “rgb(x, y, z)”** ) //color definido en el HTML

**{**

**element.style.backgroundColor = “cambio de color”;  
*document*.getElementById("myButton").innerHTML = "cambio de contenido";**

***document*.getElementById("image").src = "cambio de ruta de la imagen";**

**}**

**else**

**{**

**element.style.backgroundColor = “rgb(x, y, z)”;  
*document*.getElementById("myButton").innerHTML = "cambio de contenido";**

***document*.getElementById("image").src = "cambio de ruta de la imagen";**

**}**

**<button** **Onclick=”toogle(*this*)**” **style="background-color: rgb(x,y,z);">**

La palabra reservada ***keyword*** ***this*** es el parámetro de la ***function* toogle(*element*) {**y hace referencia al *handler* o escuchador de eventos, que es el elemento button. El *trigger* es el onclick del button y los target son el elemento img y el propio button.

**$( document ).ready( … );**

**$ ( SELECTOR ).JFUNCTION( PARAMETERS );**

* jQuery understand all kind of CSS Selectors. Selector can be the trigger or the target.

# Mismo ejercicio con jQuery

**$( document ).ready(function() {**

**$( "#myButton" ).click( function() {**

**if( !$( *this* ).hasClass( "enabledButton" ))** //Si estoy en OFF

**{**

//1. El propio botón cambia su color y contenido (2 opciones)

**$( *this* ).css("background-color", "new value");**

**$( *this* ).css("property", "new value");**

**$( *this* ).addClass( "enabledButton" );** // selector de la nueva regla CSS

**$( *this* ).html( "ON" );**

//2. La imagen cambia su src

**$( "#myImage" ).attr( "src", "img/sun.jpg");**

**}**

**else**

**{**

**$( *this* ).css("property", "inherit");** //css value

**$( *this* ).removeClass( "enabledButton" );** // clase con la regla CSS

**$( *this* ).html( "OFF" );**

**$( "#image" ).attr( "src", "img/moon.jpg");**

**}**

**});**

**});**

La referencia a un objeto ***this*** the JavaScript depende del ámbito en el que este. En este caso conseguimos que **#myButton** sea el *trigger* y él mismo **( *this* )** sea también el *target*

La función jQuery se conoce como (*callback* ) y la maneja el evento **onclick**  en el elemento **#myButton** que será el escuchador de eventos (*handler*).

JFunction **ready()** is used with the selector document. **ready()** is a function whose parameter is another function (*the callback*).

**$( document ).ready( function() { } );**

The function is no longer defined as the handler. In jQuery is called “*callback”*. JFunction **ready()** is *the trigger*. **click()** is another Jfunction (a function from the jQuery library).

<https://www.arkaitzgarro.com/javascript/capitulo-18.html>