



Nuestro compromiso es con el *futuro*.

Introducción a la programación

Repaso

Funciones

Una función:

1. Puede recibir algo (**parámetros**)
2. Ejecuta algo (**instrucciones**)
3. Devuelve algo (un **valor**)

```
1 // declaración de una función
2 function saludar(nombre, ciudad) {
3     return "¡Hola!, mi nombre es " + nombre + " y vivo en " + ciudad;
4 }
```

Funciones

Para obtener el valor de retorno de una función, la **invocamos** de la siguiente manera:

nombre(**parámetros**)

```
6 // invocación o llamada de una función
7 saludar("Marcos", "Buenos Aires");
8 // ¡Hola!, mi nombre es Marcos y vivo en Buenos Aires
```

Funciones

También podemos crear una **variable** dentro del cuerpo (scope) de una función y retornarla.

```
10 // retornar el valor de una variable
11 function saludar(nombre, ciudad) {
12     let saludo = "¡Hola!, mi nombre es " + nombre + " y vivo en " + ciudad;
13
14     return saludo;
15 }
```

Funciones

Además, es posible **encapsular** el valor de retorno de una variable.

```
19  // encapsulamiento
20  const saludo = function saludar(nombre, ciudad) {
21  |    return "¡Hola!, mi nombre es " + nombre + " y vivo en " + ciudad;
22  }
```

Funciones

Por supuesto, dentro de una función podemos utilizar **estructuras de control**

```
27 // bloque condicional dentro de una función
28 const saludoMarcos = function saludarMarcos(nombre) {
29     if (nombre === 'Marcos') {
30         return "¡Hola, " + nombre + "!";
31     } else {
32         return "Lo siento, " + nombre + ", sólo me han programado para saludar a Marcos"
33     }
34 }
```


Funciones

Además, las funciones pueden ser innominadas, es decir, **no tener nombre**.

```
37 // función innominada
38 const saludar = function (nombre) {
39     return "¡Hola, " + nombre + "!";
40 }
```

Por último, los **parámetros** tampoco son obligatorios: podemos crearlos dentro del cuerpo de la función

```
43 // función sin parámetros
44 const saludoPredefinido = function () {
45     let nombre = "Marcos";
46     return "¡Hola, " + nombre + "!";
47 }
```

Clase 3

Módulos

¿Qué son los módulos en JavaScript / Nodejs?

Son una manera que tenemos de **compartir código** entre varios archivos, de esta manera podemos literalmente “modularizar” nuestra aplicación, dividiendo los contenidos en fragmentos o componentes más simples de leer y comprender, y poder también poder reutilizarlos en varios lugares sin tener que repetir muchas veces el mismo código.

Módulos

En Node.js vamos a utilizar un tipo de declaración de módulos que se llama *CommonJS*, que si bien no es la forma más moderna, es la que encontraremos casi siempre, debido a la compatibilidad que necesita tener node.js con los servidores que existen en la actualidad.

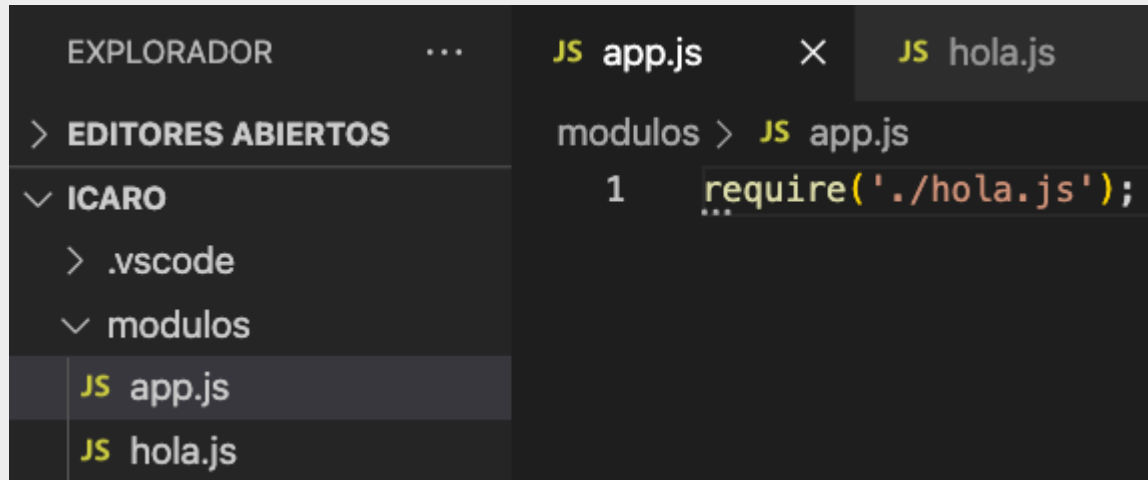
Más adelante aprenderemos también cómo utilizar la forma más moderna, llamada ES2015 Modules o ESM, para acortar.

```
const express = require('express')
```

```
import express from 'express'
```

Módulos

require recibe como argumento **'un string'** que es la ubicación donde se encuentra el código (módulo) que queremos agregar.
Como *hola.js* está en el mismo directorio que *app.js*, se utiliza *./* al principio



The screenshot shows the VS Code interface. On the left, the Explorer sidebar is open, showing a file tree with 'ICARO' as the root. Under 'ICARO', there are two folders: '.vscode' and 'modulos'. The 'modulos' folder is expanded, showing two files: 'app.js' and 'hola.js'. On the right, the Editor pane shows the 'app.js' file open. The code in 'app.js' is as follows:

```
modulos > JS app.js
1  require('./hola.js');
```

Módulos

En síntesis, los módulos son un **bloque de código reusable**, cuya existencia **no altera accidentalmente** el comportamiento de otros bloques de código

<pre>JS hola.js × ... modulos > JS hola.js 1 console.log("¡Hola!");</pre>	<pre>JS app.js × modulos > JS app.js 1 require('./hola.js');</pre>
---	---

Vamos a practicar!

Aplicación de calculadora

En la clase de hoy vamos a codear una app de JavaScript con Nodejs que nos permitirá realizar cálculos entre dos parámetros que le brindemos a una determinada función. En clases posteriores haremos posible que esta aplicación pueda correrse desde una terminal.

Aplicación de calculadora

Nuestra app contará con cuatro funciones básicas:

- Sumar
- Restar
- Multiplicar
- Dividir

Pasos a seguir: 1

- 1- Crear un archivo sumar.js que contenga una función llamada sumar, la cual debe ser exportada al final del archivo. Esta función deberá recibir 2 parámetros y retornar la suma de los mismos.

Pasos a seguir: 2

2- Crear un archivo restar.js que contenga una función llamada restar, la cual debe ser exportada al final del archivo. Esta función deberá recibir 2 parámetros y retornar la resta de los mismos.

Pasos a seguir: 3

3- Crear un archivo `multiplicar.js` que contenga una función llamada `multiplicar`, la cual debe ser exportada al final del archivo. Esta función deberá recibir 2 parámetros y retornar la multiplicación de los mismos.

Contemplar el escenario donde si alguno de los dos parámetros es cero, la función retornará cero.

Pasos a seguir: 4

- 4- Crear un archivo dividir.js que contenga una función llamada dividir, la cual debe ser exportada al final del archivo. Esta función deberá recibir 2 parámetros y retornar la división de los mismos.
- Contemplar el escenario donde si alguno de los dos parámetros es cero, la función retornará "No se puede dividir por cero".

Pasos a seguir: 4

Para verificar que hasta aquí viene todo bien, sería recomendable probar cada una de las funciones y testear su correcto funcionamiento por separado antes de continuar con la integración.

Pasos a seguir: 5

5- Crear un archivo calculadora.js en el cual deberemos requerir los cuatros
archivos
hechos con anterioridad.

Pasos a seguir: 6

6- Ejecutar la función que permite sumar y la función que permite restar, pasando como argumentos dos números cualesquiera. Mostrar en consola los resultados.

Pasos a seguir: 7

7- Ejecutar la función que permite multiplicar, pasando como argumentos dos números cualesquiera. Mostrar en consola el resultado.

Pasos a seguir: 8

8- Ejecutar la función que permite multiplicar, pasando ahora como uno de los dos argumentos, el número cero. Mostrar en consola el resultado.

Pasos a seguir: 9

9- Ejecutar la función que permite dividir, pasando como argumentos dos números cualesquiera. Mostrar en consola el resultado.

Pasos a seguir: 10

10- Ejecutar la función que permite dividir, pasando ahora como uno de los dos argumentos, el número cero. Mostrar en consola el resultado.



JS

Ahora les toca a ustedes!

Éxito!

Si todo funcionó correctamente, deberíamos tener una aplicación que nos permita realizar algunas operaciones aritméticas básicas correctamente!

Felicitaciones!

Algunas consideraciones

- ¿Qué hubiese sucedido si, en vez de generar un archivo por cada operación matemática, hubiésemos programado todo en un mismo archivo?
- ¿Por qué el mejor camino es generar distintos archivos y luego requerirlos en uno solo?
- ¿Será esta metodología de trabajo una constante de aquí en adelante?

Muchas gracias!



ICARO Asociación Civil
CUIT 30716564815
info@icaro.org.ar
www.icaro.org.ar