

## Cargar y ejecutar un archivo \*.js

Desde la aplicación de consola de MongoDB (shell), se pueden ejecutar comandos JavaScript en línea que pueden estar en un archivo \*.js y se pueden cargar, simplemente, mediante un comando. Para implementar un script de mediana o gran complejidad, lo más adecuado es utilizar un editor de texto y grabarlo en un archivo \*.js

### Cargar y ejecutar un archivo \*.js

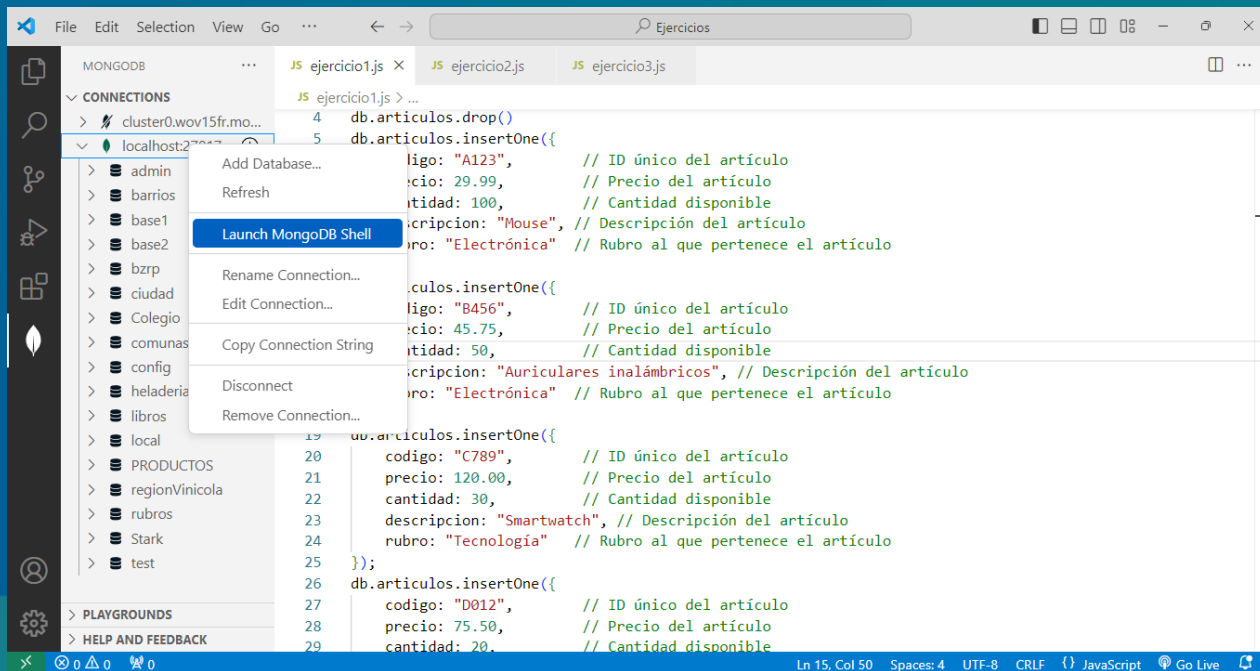
Se codifica el siguiente bloque dentro del archivo 'ejercicio1.js' y se guarda en la carpeta actual.

```
db.articulos.drop()
db.articulos.insertOne({
  codigo: "A123",           // ID único del artículo
  precio: 29.99,            // Precio del artículo
  cantidad: 100,            // Cantidad disponible
  descripcion: "Mouse",     // Descripción del artículo
  rubro: "Electrónica",     // Rubro al que pertenece el artículo
});
db.articulos.insertOne({
  codigo: "B456",           // ID único del artículo
  precio: 45.75,            // Precio del artículo
  cantidad: 50,             // Cantidad disponible
  descripcion: "Auriculares inalámbricos", // Descripción del artículo
  rubro: "Electrónica",     // Rubro al que pertenece el artículo
});
db.articulos.insertOne({
  codigo: "C789",           // ID único del artículo
  precio: 120.00,           // Precio del artículo
  cantidad: 30,             // Cantidad disponible
  descripcion: "Smartwatch", // Descripción del artículo
  rubro: "Tecnología",      // Rubro al que pertenece el artículo
});
```



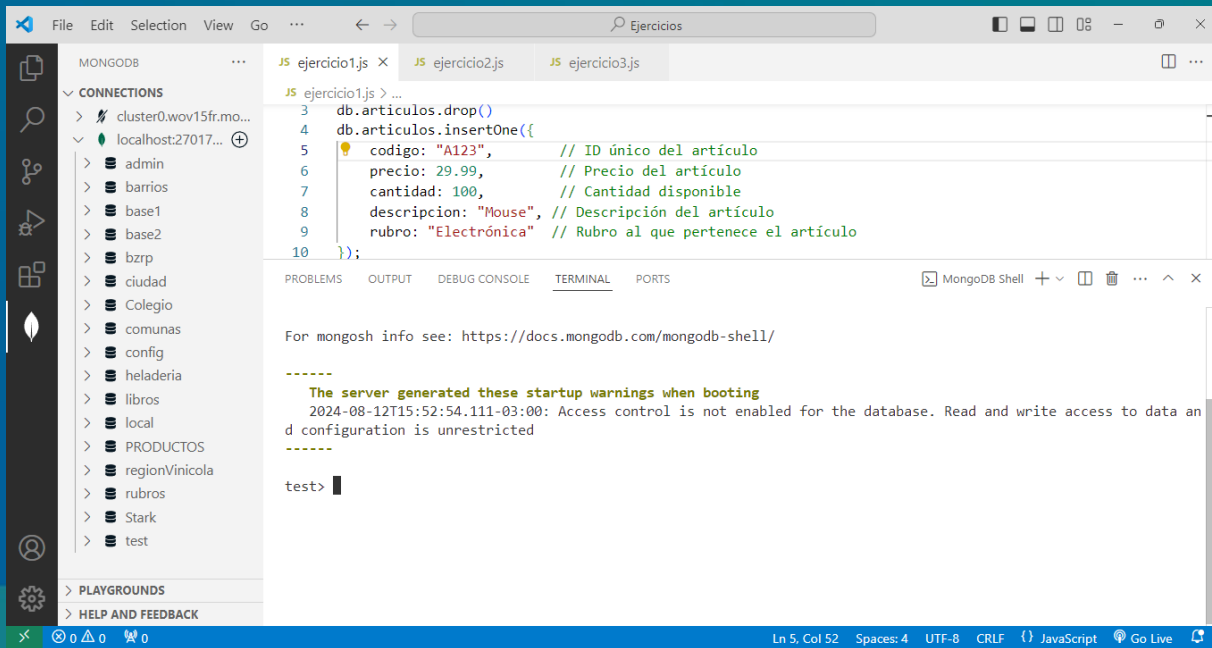
### Cargar y ejecutar un archivo \*.js

Abrir la consola de VSC y ejecutar Mongo Shell dentro de ella.



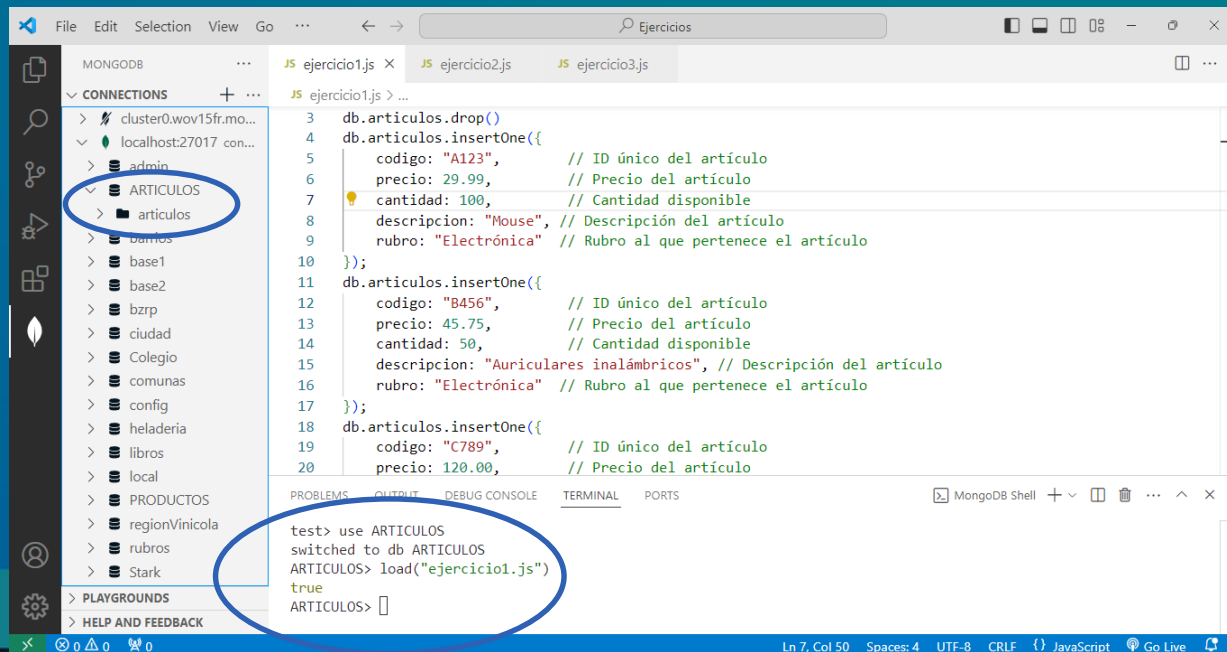
### Cargar y ejecutar un archivo \*.js

Abrir la consola de VSC y ejecutar Mongo Shell dentro de ella.



### Cargar y ejecutar un archivo \*.js

Previamente creadas la Base de Datos y la colección, se hace use de la base de datos en la terminal y se realiza el load("ejercicio1.js"), que realiza los insert.



The screenshot shows the MongoDB Shell interface. On the left, the 'CONNECTIONS' panel lists various databases and collections. The 'ARTICULOS' collection is highlighted with a blue circle. The main editor shows a JavaScript file named 'ejercicio1.js' with the following code:

```

3 db.articulos.drop()
4 db.articulos.insertOne({
5   codigo: "A123",      // ID único del artículo
6   precio: 29.99,       // Precio del artículo
7   cantidad: 100,       // Cantidad disponible
8   descripcion: "Mouse", // Descripción del artículo
9   rubro: "Electrónica" // Rubro al que pertenece el artículo
10 });
11 db.articulos.insertOne({
12   codigo: "B456",      // ID único del artículo
13   precio: 45.75,       // Precio del artículo
14   cantidad: 50,        // Cantidad disponible
15   descripcion: "Auriculares inalámbricos", // Descripción del artículo
16   rubro: "Electrónica" // Rubro al que pertenece el artículo
17 });
18 db.articulos.insertOne({
19   codigo: "C789",      // ID único del artículo
20   precio: 120.00,      // Precio del artículo

```

At the bottom, the 'TERMINAL' panel shows the following commands and output:

```

test> use ARTICULOS
switched to db ARTICULOS
ARTICULOS> load("ejercicio1.js")
true
ARTICULOS>

```

The 'load' command output 'true' is highlighted with a blue circle.

## Cargar y ejecutar un archivo \*.js

Verificar el insert de los documentos en la colección desde MongoSh:

```
ARTICULOS> db.articulos.findOne()  
{  
  _id: ObjectId('66bbbf12957cda4bd68bf202'),  
  codigo: 'A123',  
  precio: 29.99,  
  cantidad: 100,  
  descripcion: 'Mouse',  
  rubro: 'Electrónica'  
}  
ARTICULOS> █
```

## Comandos de MongoDB en un archivo \*.js

En un script no se pueden utilizar directamente los comandos `show dbs`, `use`, `show collections`, etc. pero se pueden sustituir llamando a métodos:

<code>use base1</code>	<code>db = db.getSiblingDB('base1')</code>
<code>show dbs, show databases</code>	<code>db.adminCommand('listDatabases')</code>
<code>show collections</code>	<code>db.getCollectionNames()</code>
<code>show users</code>	<code>db.getUsers()</code>
<code>show roles</code>	<code>db.getRoles({showBuiltinRoles: true})</code>
<code>show log</code>	<code>db.adminCommand({ 'getLog' : '' })</code>
<code>show logs</code>	<code>db.adminCommand({ 'getLog' : '*' })</code>
<code>it</code>	<pre>cursor = db.collection.find(); while ( cursor.hasNext() ) {   printjson( cursor.next() ); }</pre>

## Comandos de MongoDB en un archivo \*.js

Vamos a crear el archivo verColeccion.js para:

- 1-Ver las Base de Datos (show dbs / adminCommand('listDatabases'))
- 2-Posicionarse en una Base de datos (use / getSiblingDB)
- 3-Ver las colecciones para esa Base de Datos (show collections / getCollectionNames())
- 4-Cargar un find a un cursor para recorrer y mostrar.
- 5-Recorrer el cursor para ver la coleccion, de 3 formas diferentes (Mediante el método **forEach**, Mediante los métodos **hasNext** y **Next** y Mediante el método **toArray**)



## Comandos de MongoDB en un archivo \*.js

Vamos a crear el archivo verColeccion.js para:

- 1-Ver las Base de Datos (show dbs / adminCommand('listDatabases'))
- 2-Posicionarse en una Base de datos (use / getSiblingDB)
- 3-Ver las colecciones para esa Base de Datos (show collections / getCollectionNames())
- 4-Cargar un find a un cursor para recorrer y mostrar.
- 5-Recorrer el cursor para ver la coleccion, de 3 formas diferentes (Mediante el método **forEach**, Mediante los métodos **hasNext** y **Next** y Mediante el método **toArray**)

## Comandos de MongoDB en un archivo \*.js

Vamos a crear el archivo verColeccion.js para:

El código siguiente es igual para los 3 casos a abordar

```
//1-Ver las Base de Datos (show dbs - adminCommand('listDatabases'))  
printjson(db.adminCommand('listDatabases'))
```

```
//2-Posicionarse en una Base de datos (use - getSiblingDB)  
db = db.getSiblingDB('ARTICULOS')
```

```
//3-Ver las colecciones para esa Base de Datos (show collections -  
getCollectionNames())  
print(db.getCollectionNames())
```

```
//4-Cargar un find a un cursor para recorrer y mostrar  
cursor = db.articulos.find()
```

## Comandos de MongoDB en un archivo \*.js – Método forEach()

Vamos a crear el archivo verColeccion.js para:

```
printjson(db.adminCommand('listDatabases'))  
db = db.getSiblingDB('ARTICULOS')  
print(db.getCollectionNames())  
cursor = db.articulos.find()
```



/\*Cada vez que se llama al método find de una colección, retorna un objeto de la clase Cursor.

Mediante el método **forEach** del Cursor es posible recorrer todo el contenido del cursor en forma secuencial hasta agotarlo

function(d) { printjson(d) }: Esta es una función de callback que se ejecuta para cada documento en el cursor. Aquí, d representa cada documento individualmente a medida que se itera sobre el cursor.\*

```
cursor.forEach(function(d) { printjson(d) });
```

## Comandos de MongoDB en un archivo \*.js – Método hasNext() / next()

Vamos a crear el archivo verColeccion.js para:

```
printjson(db.adminCommand('listDatabases'))  
db = db.getSiblingDB('ARTICULOS')  
print(db.getCollectionNames())  
cursor = db.articulos.find()
```



/\*Mediante los métodos hasNext y Next: Se usan en forma combinada para acceder al contenido de un cursor en forma secuencial hasta agotarlo.\*/\*

```
while (cursor.hasNext()) {  
    print(cursor.next())  
}
```

## Comandos de MongoDB en un archivo \*.js – Método toArray()

Vamos a crear el archivo verColeccion.js para:

```
printjson(db.adminCommand('listDatabases'))
db = db.getSiblingDB('ARTICULOS')
print(db.getCollectionNames())
cursor = db.articulos.find()
```



/\*Acceder en forma aleatoria al contenido de un cursor, a través del método **toArray()** de los cursores.\*/

```
var documentArray = cursor.toArray()
var count = documentArray.length

var document = documentArray[1]
print("Muestro el elemento en la posicion 1 /n")
printjson(document)

print("Recorro el array")
for (i = 0; i<count; i++)
{
    document = documentArray[i]
    printjson(document)
}
```