**JavaScript 😊**

**Preparamos nuestro entorno de ejecución de JavaScript:**

**Visual Studio Code ->** Editor código e instalo un plugin *Code Runner* (necesito tener instalado Node.js).

**Node.js 🡪** Es un entorno que **permite ejecutar código JavaScript fuera del navegador.** Ya sabemos que JavaScript es el lenguaje por defecto del navegador, pero con node podemos ejecutarlos en la terminal y crear herramientas de líneas de comando. Otro uso muy habitual es crear **API REST** sin olvidar que podemos crear app en tiempo real.

Comencemos……

Ejecuto mi primer programa prueba:

1. Creo mi archivo con extensión .js:

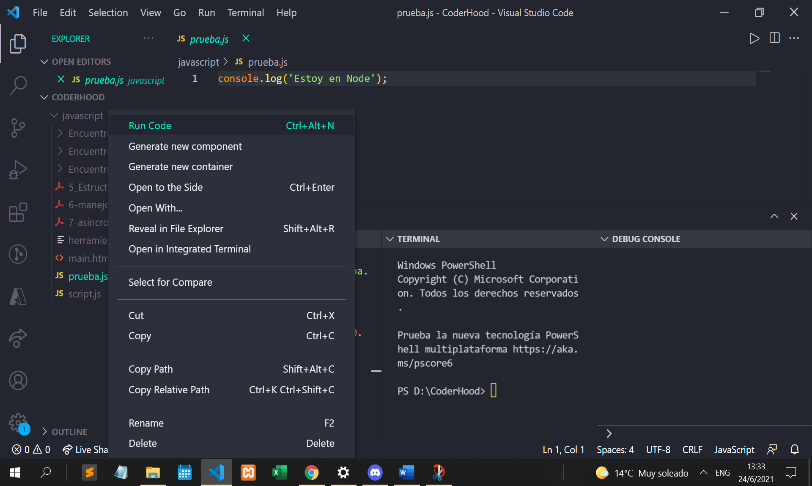
a.1) File > New File

a.2) Ctrl + S -> Guardar como: nombreArchivo.js

b) console.log(‘Estoy en Node);

c) Correr el código puedo elegir 2 maneras con el plugin instalado o con la consola a lo “*macho pecho peludo..” 😊 .*

*Opc1.*

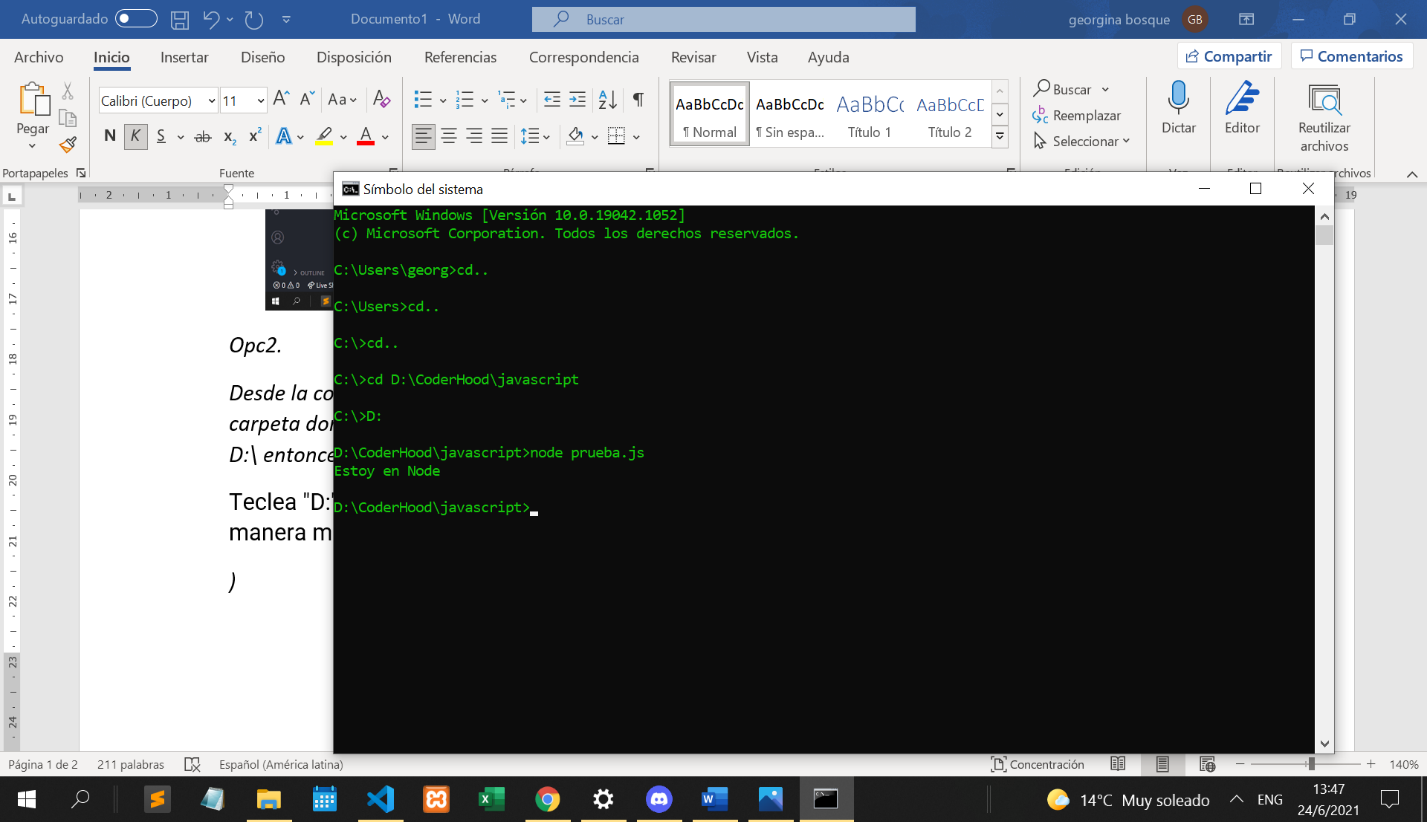


*Opc2.*

*Desde la consola de Windows -> para ejecutar los comandos de node tengo que estar dentro de la carpeta donde cree mi archivo.js. (tenía mi consola corriendo en el disco C:\ pero yo tenía mi carpeta en D:\ entonces cambien la unidad:*

Teclea "D:" y presiona "Enter". Esto cambiará el comando de leer en "C: >" a "D: >". Esta es la manera más básica de cambiar la unidad de línea de comando.

*).*



## [plataformasParaPracticarJavascript](D:\\CoderHood\\javascript\\plataformasParaPracticarJS.txt).

**Estructuras de Datos:**

* Object -> La clase Object representa uno de los tipos de datos de Javascript. Es es usado para guardar una colección de datos definidos y entidades más complejas.
* Arrays
* OBJETOS

**Declaramos y definimos objetos:**

**Recordar**: Que es un **lenguajes de tipado dinámico** son aquellos (como [JavaScript](https://developer.mozilla.org/es/docs/Glossary/JavaScript)) donde el intérprete asigna a las [variables](https://developer.mozilla.org/es/docs/Glossary/Variable) un tipo durante el tiempo de ejecución basado en su [valor](https://developer.mozilla.org/es/docs/Glossary/Value) en ese momento.

let georgina={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

console.log(georgina)

Mostramos por consola:

[Running] node "d:\CoderHood\javascript\estructurasDeDatos\obj.js"

{edad: 31, nombre: 'Georgina', tieneGato: true}

[Done] exited with code=0 in 0.289 seconds

**Como obtenemos los datos que están en un objeto?**

Hay 2 formas, accedemos a los campos con un punto “ . ” (*tal y como lo venia haciendo con los* ***struct*** *de c++*) o entre corchetes y el nombre del campo que queramos acceder:

let georgina ={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

//La más recomendada

console.log(georgina.edad)

//opcional puede servir para casos particulares

 console.log(georgina['nombre'])

**Y como agregamos o modificamos datos en un objeto?**

let georgina ={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

//tambien puedo solo declarar un objeto no lo defino osea seria la firma

let usuario={}

georgina.edad = 30; //modificamos datos

usuario.nombre = 'nahuel' // agregamos datos

console.log(georgina);

console.log(usuario)

**Como removemos datos de un objeto:**

let georgina ={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

delete georgina.edad //delete = eliminar en ingles

console.log(georgina);

**¿Sabes cómo recorrer un objeto en JavaScript?** 🤔

1. Bucle for…in

let georgina ={

    edad:31,

**nombre: 'Georgina',**

    tieneGato: true,

}

for(let key in georgina){

    console.log(georgina[key])

}

1. Object.keys()
2. let georgina ={
3. edad:31,
4. nombre: 'Georgina',
5. tieneGato: true,
6. }
7. //retorna un array de strings que representan todas las propiedades
8. // del objeto.
9. let keys= Object.keys(georgina)
11. keys.forEach(key =>{
12. console.log(georgina[key])
13. })

3) Object.values()

let georgina ={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

// devuelve un array con los valores correspondientes a las propiedades enumerables de un objeto

let values = Object.values(georgina)

values.forEach(value=>{

    console.log(value)

})

4) Object.entries()

let georgina ={

    edad:31,

    nombre: 'Georgina',

    tieneGato: true,

}

// devuelve una matriz de pares propios de una propiedad enumerable.

let entries = Object.entries(georgina);

for(let [key, value] of entries){

    console.log(key, value)

}

//Compilando…

[Running] node "d:\CoderHood\javascript\estructurasDeDatos\obj.js"

edad 31

nombre Georgina

tieneGato true

**¿Por qué necesitas saber filtrar y extraer propiedades de un objeto JavaScript?**

Porque cuando tenemos una colección de datos (la típica respuesta una petición a cualquier API), a veces necesitamos extraer todos sus valores en base a una propiedad determinada.

Por ejemplo, “necesito todos los apellidos únicos de la colección usuarios” o “dame todos los títulos de la colección post”.

Aunque hay librerías que incorporan esta función (Underscore\_pluck), nosotros vamos a crearla desde cero para luego refactorizarla.

**La colección**

Valor duplicado

let usuarios = [

    {nombre:'Juan', profesion: 'Dev'}

    {nombre:'Laura', profesion: 'Ux'}

    {nombre:'Teresa', profesion: 'Mk'}

    {nombre:'Andres', profesion: 'Dev'}

    {nombre:'Jorge', profesion: 'Account'}

    {nombre:'Ivan', profesion: 'Product'}

    {nombre:'Martina', profesion: 'Planner'}

]

Esta lista usuario/as (una matriz que contiene objetos) representa el contenido sobre el que trabajaremos.

2) Creamos la función y Buscamos las coincidencias:

//creamos la funcion que tiene dos parametros: uno que indica la propiedad  a buscar

// (prop) y otro que indica donde buscar (coleccion)

function extraer(prop, coleccion){

    let coincide = coleccion.map(function (item){

        return item[prop]

    })

}

Utilizando el método map de las matrices, creamos una nueva matriz (coincide) **con todos los objetos de la colección que contienen la propiedad a buscar.**

**3) Descartar repeticiones**

function extraer(prop, coleccion){

    let coincide = coleccion.map(function (item){

        return item[prop]

    })

    coincide = new Set(coincide)

}

Para eliminar posibles duplicados, creamos un *Set (que solo puede contener valores únicos).*

**4) Retornar valor final**

function extraer(prop, coleccion){

    let coincide = coleccion.map(function (item){

        return item[prop]

    })

    coincide = new Set(coincide)

    coincide = Array.from(coincide)

    return coincide

}

Usando el método Array.from, creamos una matriz con el contenido del *set del paso anterior, y la retornamos.*

*5) poniéndolo a prueba*

*Invocamos y mostramos la función*

console. log(extraer('profesion', usuarios))

No hay duplicado.

[Running] node "d:\CoderHood\javascript\estructurasDeDatos\obj.js"

[ 'Dev', 'Ux', 'Mk', 'Account', 'Product', 'Planner' ]

**Refactorizando…**

const extraerA = (prop, coleccion)=>

     Array.from(new Set(coleccion.map(item=>item[prop])))

Gracias a las funciones flecha y su return implícito, podemos refactorizar la función extraer a una única sentencia.

**¿Por qué necesitas aprender a desestructurar objetos JavaScript?** 👇

El uso más común para la nueva desestructuración JavaScript surge cuando queremos extraer propiedades de un objeto.

La forma en lo que lo hacíamos hasta ahora (pre ES6) requiere crear una variable por cada propiedad a extraer.

Hasta ahora, esto lo que hacíamos:

let usuarios ={

    nombre:'Juan',

    provincia: "valencia"

}

let nombre = usuario.nombre

let nombre = usuario.provincia

Esta forma (pre ES6) requiere crear una variable por cada propiedad a extraer.

Algo **poco extensible** y puede dar lugar a error.

A**prenderemos una forma mejor,** más rápida y mas simple de llegar al mismo resultado.

let usuarios ={

    nombre:'Juan',

    provincia: "valencia"

}

let {nombre} = usuario //Juan

Asignando un **identificador** (nombre) **que corresponda a la propiedad de un objeto** (usuario)

Propiedad existente.

Objeto donde esa propiedad existe.

Creamos una nueva variable extrayendo el valor de la propiedad.

**Extraer varias propiedades**

let usuarios ={

    nombre:'Juan',

    provincia: "valencia"

}

let {nombre, provincia} = usuario //Juan, Valencia

Por supuesto, **podemos seguir enumerando más propiedades del objeto** para crear nuevas variables

Con sus valores extraídos.

**Valor por defecto**

let usuarios ={

    nombre:'Juan',

    provincia: "valencia"

}

let {lenguaje = 'JS'} = usuario //"JS"

Si intentamos desestructurar una propiedad que no existe (y cuyo valor seria *undefined*), podemos declarar un valor por defecto a través del operador de asignación (=).

**Utilizar alias**

let usuarios ={

    nombre:'Juan',

    provincia: "valencia"

}

let {lenguaje: lang = 'JS'} = usuarios //"JS"

console.log(lang) // "JS"

Para crear variables con nombres diferentes a las propiedades a extraer (para evitar colisiones con otras variables), puedes crear un alias especificándolo con dos puntos (:).

**Acceso Profundo**

let usuarios ={

    nombre:'Juan', provincia: "valencia",

    stack: {

        lenguaje: 'JS', editor: 'VsCode',

    }

}

let {stack: {editor}} = usuarios

console.log(editor) // Vscode

Si queremos **extraer una propiedad que contiene otro objeto,** debemos referenciar la propiedad que lo contiene (*stack*) así como la propiedad que queremos extraer (*editor*)**.**

**Obtener el resto**

let usuarios ={

    nombre:'Juan', provincia: "valencia",

    stack : {

        lenguaje: 'JS', editor: 'VsCode',

    }

}

let {nombre, ...resto} = usuarios

console.log(resto) // Vscode

//{ provincia: 'valencia', stack: { lenguaje: 'JS', editor: 'VsCode' } }

Podemos usar la sintaxis rest(…) para **recolectar el resto de propiedades** del objeto a desestructurar.