

COD - EANDO

Aquí empieza nuestro desafío





Nuestro Recorrido

01

Intro a la Programación

Moviendo nuestra neuronas.
Aprendiendo JS

02

El FrontEnd

Las Aplicaciones Web, estética y
dinamismo. HTML + Css y JS

03

Controlando el código

Una herramienta super-poderosa
llamada Git

04

El BackEnd

Procesando la información.
Ahora JS en el BackEnd con Node

05

Bases de Datos

Un poco de Sql con Mysql
Nuestro repositorio de datos.

06

Mejorando el FrontEnd

La Sazón con ReactJs
JS en front es más ordenado.



01

Intro a la Programación

Clase 4





Agenda

Repaso

Fabricas o funciones

Funciones como expresión, anónimas.

Funciones de flecha

Métodos de array.

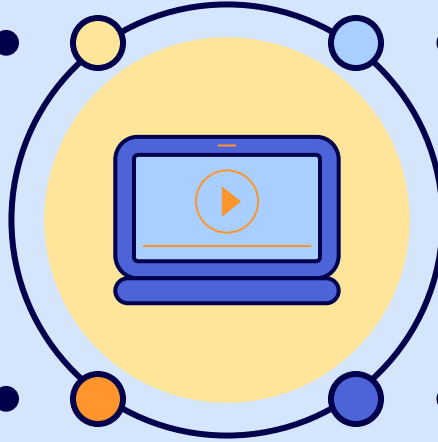
Repasando

Condicionales

Ciclos

Ambito de
Variables

Arrays y ciclos



Puntos importantes


La mayoría de las palabras reservadas son en inglés

Puedo usar un ciclo en incremento o decremento




Los ciclos deben tener una condición de fin

Palabras reservadas





Usualmente queremos
tener código para usarlo
en varios lugares





Las Funciones

Las Funciones

```
function nombre( ) {  
  //lo que se va a ejecutar  
  return xxx  
}  
  
function saludar() {  
  return 'Hola mundo'  
}  
  
saludar()  
  
let mensaje = saludar()  
console.log(mensaje)
```

Estamos declarando una función usando la palabra reservada **function** e identificando con **nombre**.

Lleva paréntesis y llaves para encerrar el código.

Tiene una palabra reservada **return**, que se usa para devolver el resultado.

Es importantísimo que una función siempre devuelva algo.

Para ejecutar o llamar a la función, usamos su nombre y luego paréntesis.
La guardo y consologeo.

Las Funciones y sus parámetros

```
function saludarPersona (nombre) {  
    return 'Hola: ' + nombre  
}
```

```
function saludarPersonaV (nombre)  
{  
    if (!nombre) {  
        return 'Hola Desconocido'  
    }  
    return 'Hola: ' + nombre  
}
```

Declaramos la función y le definimos que va a recibir un nombre.

Este parámetro o argumento sólo vive dentro de la función.

También podemos validarlo, por si no llega nada y devolver algo distinto.

Un pequeño ejercicio

1. Crear el archivo funciones.html
2. Crear el archivo funciones.js
3. Crea un array de objetos literales de alumnos que contengan: nombre, apellido, edad

Crea funciones para:

4. Devolver la cantidad de alumnos.
5. Devolver cuántos alumnos son mayor de edad.
6. Devolver los alumnos que son mayor de edad.
7. Devolver los alumnos agregandoles un atributo nuevo llamado nacionalidad y valor "Venezolano"



Las Funciones y sus parámetros

```
function saludarPersona0(nombre  
= 'desconocido') {  
    return 'Hola: ' + nombre  
}
```

```
function sumarDos(a, b = 0) {  
    return a + b  
}
```

Los parámetros pueden ser opcionales y darles un valor por si no llegan.

Le asignamos el valor string **desconocido** Así no tenemos que validar que llegue. Sólo para casos que sean opcionales.

Siempre el valor opcional debería de ir al final.

Pueden haber varios valores opcionales.

Las Funciones anónimas

```
let algo = function( ) {  
  //lo que se va a ejecutar  
  return xxx  
}
```

```
let sumar = function(a, b) {  
  return a + b  
}
```

```
console.log(sumar(1, 2))
```

También llamadas funciones como expresión.
Se guardan en variables.

Tambien reciben parámetros y deben devolver algo.

Se ejecuta de la misma manera

Las Funciones de flecha

```
let algo = () => {  
  //lo que se va a ejecutar  
  return xxx  
}
```

```
let sumar = (a, b) => {  
  return a + b  
}
```

```
let sumarFlecha = (a, b) => a + b
```

Son funciones anónimas.

No llevan la palabra function.

Luego de los paréntesis llave un ==>

Muy similar a la función anónima.

Tiene otra ventajas, pero poco a poco las entenderemos.

Aquí la sazón, se pueden omitir las llaves y la palabra return, solo en casos que no se realicen más instrucciones de código.

Las Funciones autoejecutables

```
(function () {  
    //lo que se va a ejecutar  
    return xxx  
}) ()  
  
let result = (function (a,b) {  
    return a + b  
}) (1,2)
```

Son funciones anónimas.


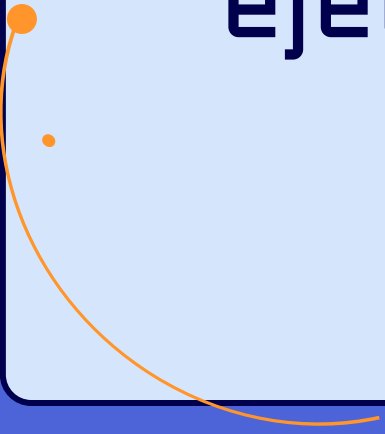


Llevan paréntesis englobando a la función

Luego de esos paréntesis, llevan otros dos para ejecutarla.


Este tipo de funciones se crean y ejecutan en el momento.

Se usan para englobar código, sin que moleste otras variables o funciones.



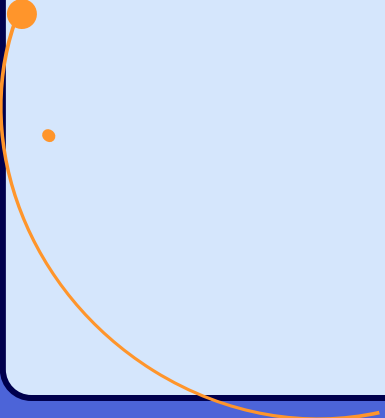
El resultado lo puedo guardar en una variable.



Podemos crear
funciones una vez y
ejecutar donde sea y
cuando sea.



Cuando tenemos arrays
vamos a querer filtrar ú
obtener datos
modificados.





Para arrays usamos
.forEach .filter .map
.reduce



```
[🐮, 🥔, 🐔, 🌽].map(cook) ⇒ [🍔, 🍟, 🍗, 🍷]  
[🍔, 🍟, 🍗, 🍷].filter(isVegetarian) ⇒ [🍟, 🍷]  
[🍔, 🍟, 🍗, 🍷].reduce(eat) ⇒ 💩
```

Métodos de array

```
array.metodo()
```

```
array.metodo(function() {})
```

```
const res = array.metodo(() => {})
```

```
function miCallback() {}
```

```
const res =  
array.metodo(miCallback)
```

Son funciones que tienen los array.
Se hace más fácil el filtrado, manipulación, etc.
Cada una de las funciones se accede mediante el operador punto y luego el nombre del método.

La mayoría recibe como primer parámetro una función anónima, de flecha o callback.

Muchos métodos devuelven un resultado y eso lo podemos guardar en una variable.

Un callback es usar el nombre de una función dentro de alguien que se encargue de ejecutarla

Métodos de array: filter

```
const words = ['spray', 'limit',  
               'elite', 'exuberant', 'destruction',  
               'present'];  
  
const result =  
words.filter(function(word) {  
    return word.length > 6  
});
```

Tomando un array de palabras.

Aplicamos el filter, donde aplicaremos una función anónima, cada elemento se convierte en word y esta función anónima debe retornar la condición de filtrado, en esta caso sólo las palabras mayores a 6 letras.

Esto lo guardo en otra variable.

Métodos de array: reduce

```
const letras = ['e', 's', 't', 'o',  
'y', ' ', 'p', 'r', 'o', 'g', 'a',  
'm', 'a', 'n', 'd', 'o', ' ', 'e',  
'n', ' ', 'j', 's'];
```

```
function unirLetras (acum, letra) {  
    return acum + letra;  
}
```

```
const palabra =  
letras.reduce(unirLetras, '');
```

Tomando un array de letras.

Vamos a obtener la palabra, en terminos generales concatenaremos las letras

Aplicamos el reduce, este recibe un callback ó función anónima que tiene al menos dos parámetros, el primero el acumulador y el segundo el valor actual y como segundo parámetro del reduce vamos a iniciar el acumulador como un string vacio.

Métodos de array: map

```
const numeros = [3, 5, 20, 43, 8, 10,  
15, 60, 8];
```

```
const porDos = numeros.map((num) =>  
num * 2);
```

Tomando un array de números.
Vamos a transformar el array multiplicándolo por dos a cada elemento.

Aplicamos el método map, esta vez aplicando una función de flecha, donde num va a ser cada elemento y va a devolver num multiplicado por dos.

Terminamos de guardarlo en una variable.

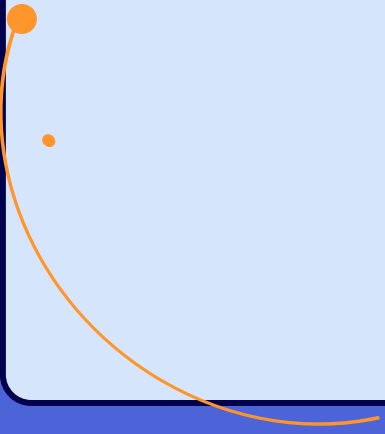
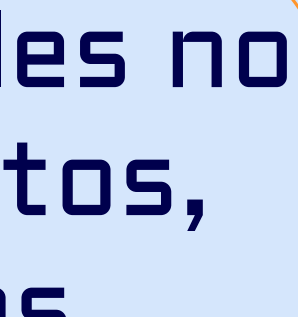

Métodos de array: forEach

```
alumnos.forEach(function(alumno) {  
    //aquí no devuelvo nada, pero  
    puedo:  
    //enviar un email, guardar en la  
    base de datos, etc  
})
```


Tomando un array de números.

Si necesitamos hacer algo con cada uno de esos elementos, sin filtrarlos ni cambiarlos.

Aplicamos el método forEach, que funciona igual que el for o el for of. Este método recibe un callback o función anónima y no devuelve nada



En los objetos literales no
solo existen atributos,
tambien podemos
agregarles
funcionalidades



Funciones en objetos literales

```
const persona = {  
  nombre: 'pepe',  
  saludar: () => 'Hola Buen Día'  
}
```

```
persona.saludar()
```

Un Objeto literal puede contener:
Atributos y Funciones.

Estas funciones se declaran tal como un atributo y dentro tienen una función anónima o una función de flecha.

Definimos la función saludar que devuelve un string.

Para llamar al método saludar, accedemos usando el operador punto.

Funciones en objetos literales

```
const persona = {  
  nombre: 'pepe',  
  saludarNombre: function () {  
    return 'Hola me llamo ' +  
    this.nombre  
  }  
}  
  
persona.saludarNombre()
```

Pero si en algún momento queremos usar los atributos dentro de un método del objeto literal.

Lo más recomendable es usar una función anónima, así podemos acceder a los atributos dentro del objeto usando una palabra reservada llamada `this`.

Esta hace referencia al objeto literal.

Para llamar al método `saludarNombre`, accedemos usando el operador punto.

Fin por hoy!



Sigan practicando

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, infographics & images by **Freepik**

