# Javascript

# Programa del curso

- Clase 1: Introducción y sintaxis
- **Clase 2: Funciones, arrays y objetos**
- Clase 3: Javascript integrado a HTML
- Clase 4: DOM, selectores y elementos
- Clase 5: Eventos
- Clase 6: Formularios
- Clase 7: Ajax
- Clase 8: Ejercicio integrador

# 1. Funciones

# Definición de funciones

Una función es un bloque de código designado para hacer una tarea.

```
function elProducto(n1, n2) {
    return n1 * n2; // Devuelve el producto de n1 con n2
}
```

Una función es ejecutada cuando alguien la invoca.

```
elProducto(2, 5); // Esto me devolverá 10
```

# Funciones como expresión

Podemos asignar una función anónima a una variable

```
var nombreFuncion = function() {
        console.log("Hola");
};
```

```
nombreFuncion();
```

Cómo hacemos para realizar funciones matemáticas en javascript?

Google

funciones matematicas javascript

# Scope

El scope de una variable es el contexto donde dicha variable es visible.

```
function myFunction() {
    var a = 4;
    return a * a;
}
```

En este ejemplo **a** es una variable **local**

```
var a = 4;
function myFunction() {
    return a * a;
}
```

En este ejemplo a es una variable **global**

# Funciones anidadas

Podemos tener una función dentro de otra función.
Esta función anidada será visible dentro de la función padre solamente.

```
function circunferencia (radio)
{
   function diametro()   // función anidada
   {
     return 2*radio;   // radio es una variable de la función padre
    }

   return Math.PI * diametro(); // invocamos la función
}
```

# Veamos este ejemplo

```
function a(){
  console.log( 'a viene primero');
}
function b(){
  console.log( 'b viene segundo' );
}
a();
b();
```

Esto resultaría:

```
a viene primero
b viene segundo
```

# ¿ Qué pasa en el siguiente ejemplo?

```
function a(){
  setTimeout( function(){ //explayaremos este método
    console.log( 'a viene primero');
  }, 1000 );
}
function b(){
  console.log( 'b viene segundo' );
}
a();
b();
```

Quedaría:

```
b viene segundo
a viene primero
```

# *Callbacks*

# Callbacks

Se denomina callback a una función que es pasada como parámetro a otra función que será ejecutada al finalizar la función anterior.

Javascript es un lenguaje asincrónico. Esto significa que un llamado puede ejecutarse y no sé cuándo termina. Para poder manejar esta situación se utiliza el patrón de diseño **callback**.

Los callbacks se pueden utilizar de varias formas, en el próximo ejemplo utilizamos un callback utilizando una función anónima.

```
function cocinarBizcochuelo(callback){
        setTimeout( function(){
                console.log( 'cocinarBizcochuelo viene primero');
                callback();
        }, 3000 );
}
function decorarTorta(){
        console.log( 'decorarTorta viene segundo' );
}

cocinarBizcochuelo( decorarTorta );
```

```
'cocinarBizcochuelo viene primero
'decorarTorta viene segundo
```

# A practicar!

Práctica 2 - **Funciones**

# 2.
# Arrays

# Métodos de arrays - forEach

Ejemplo:

```
[1, 5, 7].forEach(function(value, index) {
  console.log("En el indice: " + index + " está el valor: " + value);
});
```

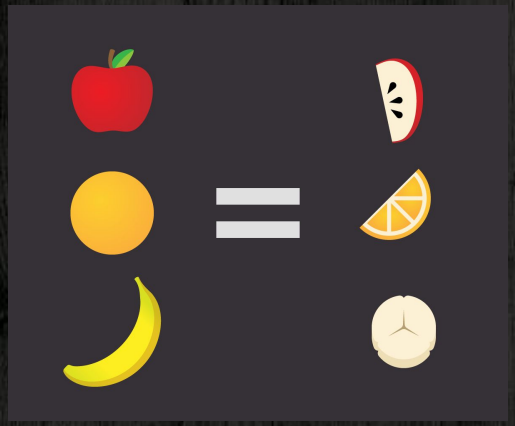
Resultado:

```
En el índice: 0 está el valor: 1
En el índice: 1 está el valor: 5
En el índice: 2 está el valor: 7
```

# Métodos de arrays - map

Usamos el **map** cuando dado un array queremos modificar **cada uno** de los ítems utilizando una función determinada.

```
[1, 5, 7].map(function(numero) {
        return numero * 2;
});
```



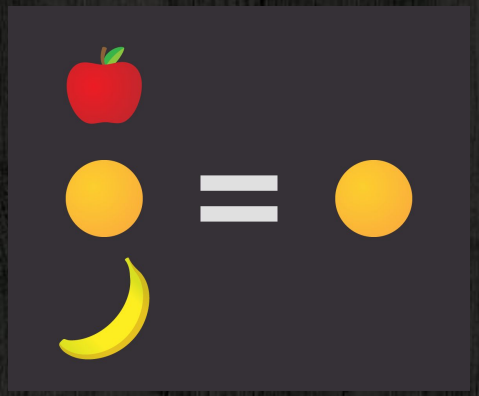
El resultado es un nuevo array.

```
[2, 10, 14]
```

# Métodos de arrays - filter

Usamos el **filter** cuando queremos filtrar algunos elementos de un array.

```
[13,18,20].filter(function(numero) {
        return (numero >= 18);
});
```



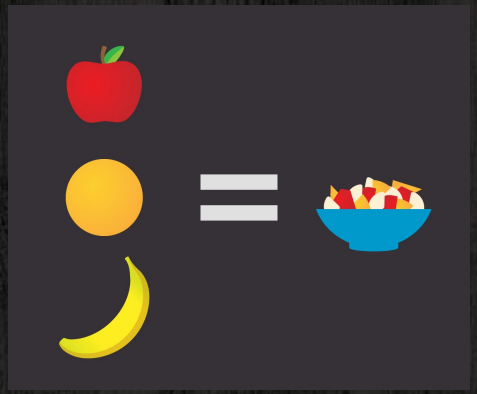
El resultado es un nuevo array.

```
[18, 20]
```

# Métodos de arrays - reduce

Usamos el **reduce** cuando dado un array queremos reducirlo a un valor único utilizando una función determinada.

```
[1, 5, 7].reduce(function(total, numero) {
        return total + numero;
});
```



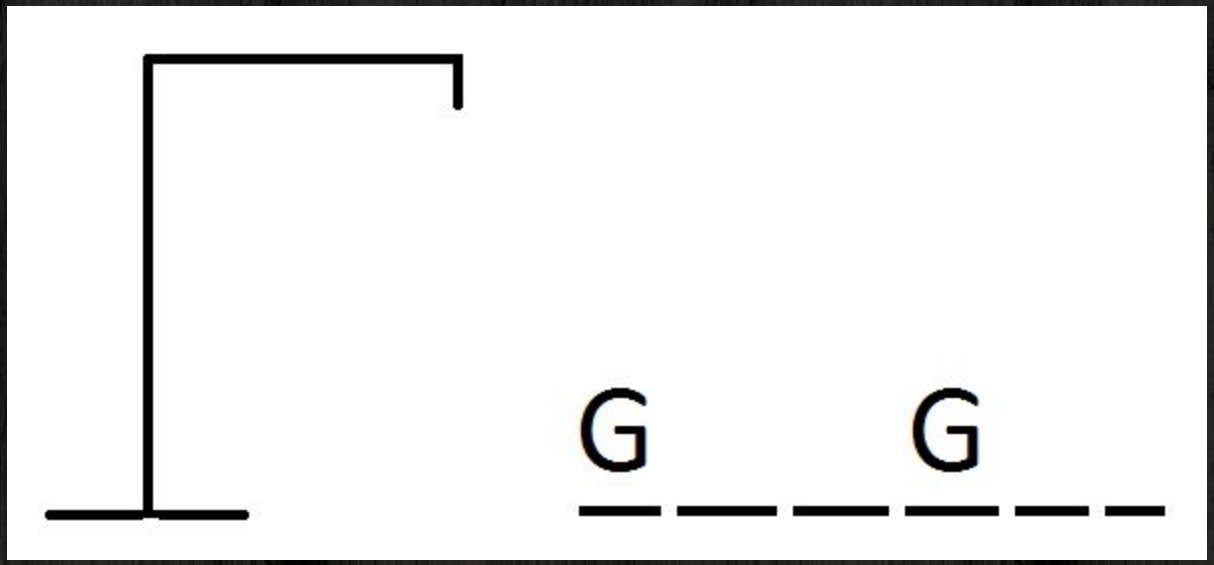
El resultado es un valor único.

```
13
```


NAH, TOO EASY...
memegenerator.net

Dado un array de lo que se les ocurra:
¿Cómo harían si tienen que obtener la cantidad de ítems que contiene?

# Una Pista...



# A practicar!

Práctica 2 - **Arrays**

# 3.
# Objeto literal

# Definición de objetos

Un **objeto** es cualquier cosa de la vida representado en código.

```
var auto = {
        marca: "Chevrolet",
        modelo: "Corsa",
        kilometraje: 65000,
        color: "Blanco",
        titulares: ["Luciana", "Juan", "Josesito"]
};
```

Los objetos tienen **propiedades** que se utilizan para describirlo.

Estas pueden ser de distinto tipo (number, string, array, object, etc)

# Propiedades de objetos

Podemos acceder a una **propiedad** de un objeto de varias formas:

- auto.marca; // "Chevrolet”
- auto["marca"]; // "Chevrolet”

Podemos acceder utilizando variables:

```
var prop = "marca";
auto[prop]; // "Chevrolet"
```

# Propiedades de objetos

Establecer el valor de una propiedad:

```
auto.color = "Rojo";
auto["color"] = "Rojo";
```

A un objeto se le pueden agregar propiedades:

```
auto.velocidadMax = 200;
auto[“velocidadMax”] = 200;
```

# Métodos de objetos

Los objetos también tienen **métodos** para interactuar con ellos.

Un método es una propiedad del objeto que tiene asignado una **función**.

```
var auto = {
        marca: "Chevrolet",
        arrancar: function(nombre){ console.log("brum brum"); },
        modelo: "Corsa"
};
auto.arrancar();
// Se puede utilizar auto["arrancar"]();
```

```
“brum brum”
```

# Métodos de objetos y parámetros

Los métodos también pueden recibir parámetros ya que son una función.

```
var auto = {
        marca: "Chevrolet",
        conductor: function(nombre){
                console.log("Está manejando "+ nombre);
        },
        modelo: "Corsa"
};
auto.conductor("Pepito");
```

“Está manejando Pepito”

# A practicar!

Práctica 2 - **Objeto literal**

*Gracias!*

# Preguntas?