

# DESARROLLO DE SOFTWARE



## POR DÓNDE ANDAMOS

Arrancamos con la  
revisión de conceptos  
de Full Stack y  
aplicaciones en capas

1

Hoy comenzamos a  
profundizar en el  
lenguaje de  
programación Javascript

3

A partir de allí nos  
dedicamos de lleno a  
la construcción del  
Backend

5

Revisamos  
herramientas para  
maquetar lo que ve el  
usuario en un browser.

2

Mañana integramos  
Javascript con lo visto  
en HTML además de  
agregar conceptos de  
POO

4



# AGENDA PARA HOY

REALIZAR UNA PRIMERA APROXIMACIÓN  
AL LENGUAJE DE PROGRAMACIÓN  
JAVASCRIPT





# ¿CÓMO ABORDAMOS UN NUEVO LENGUAJE DE PROGRAMACIÓN?

## Cómo manejamos datos...

Variables, asignación, tipos de datos, conversión de tipos y operadores.

Colecciones y trabajo sobre estructuras de datos.

## Cómo organizamos y reutilizamos el código...

Estructuras de control;  
Modularización  
funciones y **objetos**;  
Estrategias de resolución de problemas y procesamiento de datos.

## Como ejecutamos y probamos

Con esto ya comenzamos:  
Consola del browser o archivo de Script.  
Continuamos con ambas alternativas.

1.

# LENGUAJE DE PROGRAMACIÓN JAVASCRIPT

Seguimos avanzando con el lenguaje.



## ALGUNAS CARACTERÍSTICAS ➔ HABÍAMOS DICHO:

- Difícil encasillarlo porque se puede hacer de todo! 😊
- ✓ Cross-platform
  - ✓ Orientados a Objetos o también, Enfocado en Prog. Funcional
  - ✓ Ejecutado por un entorno virtual (que puede radicar en el navegador o en el sistema operativo local)
  - ✓ Utilizado inicialmente para dar vida a lo que puede consumirse con un navegador de internet
  - ✓ Con capacidades que van desde realizar cálculos hasta producir animaciones, juegos o conectarse con servidores remotos para consumir datos
  - ✓ Hoy en día también utilizado para programar el back-end de las aplicaciones en entornos independientes del navegador como NodeJS

## AHORA EN CONCRETO:

Características generales! 😊 ... Manejalo como quieras...

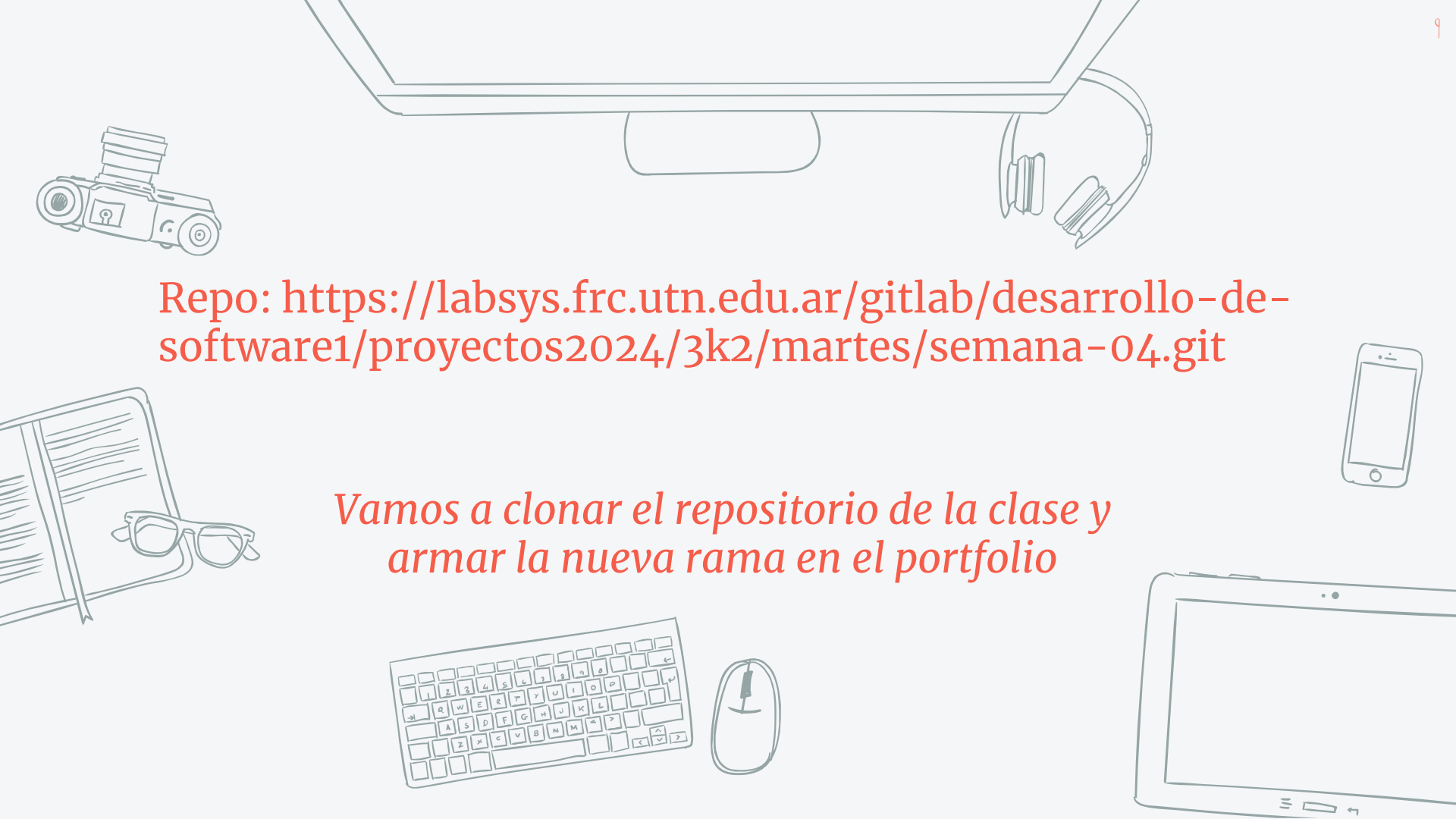
- ✓ Es completamente case sensitive, lo poco concreto jeje
- ✓ En Javascript en lugar de **Sentencias** deberíamos hablar de **Declaraciones**, pero Ok
- ✓ Las sentencias o declaraciones terminan con “;” ... o no... ?
- ✓ Los bloques se delimitan con llaves { ... } aunque puede cuando el bloque está constituido por una sola sentencia no son necesarias
- ✓ Tipado dinámico de datos



# COMENCEMOS

Clonar repo de la clase, pushear en el portfolio y comenzamos a trabajar!





Repo: <https://labsys.frc.utn.edu.ar/gitlab/desarrollo-de-software1/proyectos2024/3k2/martes/semana-04.git>

*Vamos a clonar el repositorio de la clase y  
armar la nueva rama en el portfolio*

## DECLARACIÓN DE VARIABLES...

### let

Declara una variable local en el contexto del bloque donde está expresado, opcionalmente la inicializa con un valor.

### const

Declara una variable local en el contexto del bloque donde está expresado asignado el valor inicial.

### ¿QUÉ PASA CON **var**?

En versiones anteriores de JS se utilizaba para declarar variables y publicarlás a un contexto superior.

Siempre asignamos valor utilizando el operador **=**

Tipo	Descripción
number	puede contener números, enteros o de punto flotante y también valores simbólicos como +Infinity, -Infinity y NaN
string	representa a las cadenas de caracteres
boolean	representa un valor lógico y puede contener los valores <b>true</b> y <b>false</b>
bigint	permite almacenar enteros que exceden los límites impuestos para el tipo number. Para especificar un literal bigint se agrega n al número que representa.
undefined	es el valor que contiene una variable que no ha sido inicializada y el tipo de la misma.
null	No es un tipo en esencia, representa un <b>valor</b> nulo o vacío, tiene solamente un valor: <b>null</b>
symbol	es un valor primitivo único e inmutable y se puede utilizar como clave de una propiedad de objeto.
Object	...

## TIPOS DE DATOS EN JAVASCRIPT

Se puede ver como cada capa interactúa con las capas adyacentes, se comunica, envía y recibe datos/información, para llevar a cabo la función que le corresponde.

## Operador

## Descripción

<code>+, -, *, /, %</code>	operadores aritméticos naturales, notar que no figura la potenciación ni la división entera
<code>+=, -=, *=, /=</code>	operadores resumidos, funcionan igual que python
<code>++, --</code>	operador unario de incremento y decremento
<code>==, !=</code>	igualdad y desigualdad estricta
<code>&lt;, &gt;, &lt;=, &gt;=</code>	operadores relacionales naturales
<code>&amp;&amp;,   , !</code>	operadores lógicos: and, or y not
<code>falsy</code>	¡No es un operador! ...conjunto de valores que resultan falsos al ser evaluados como booleanos: <b>false</b> , <b>0</b> , cadena vacía, <b>null</b> , <b>undefined</b> , <b>NaN</b>

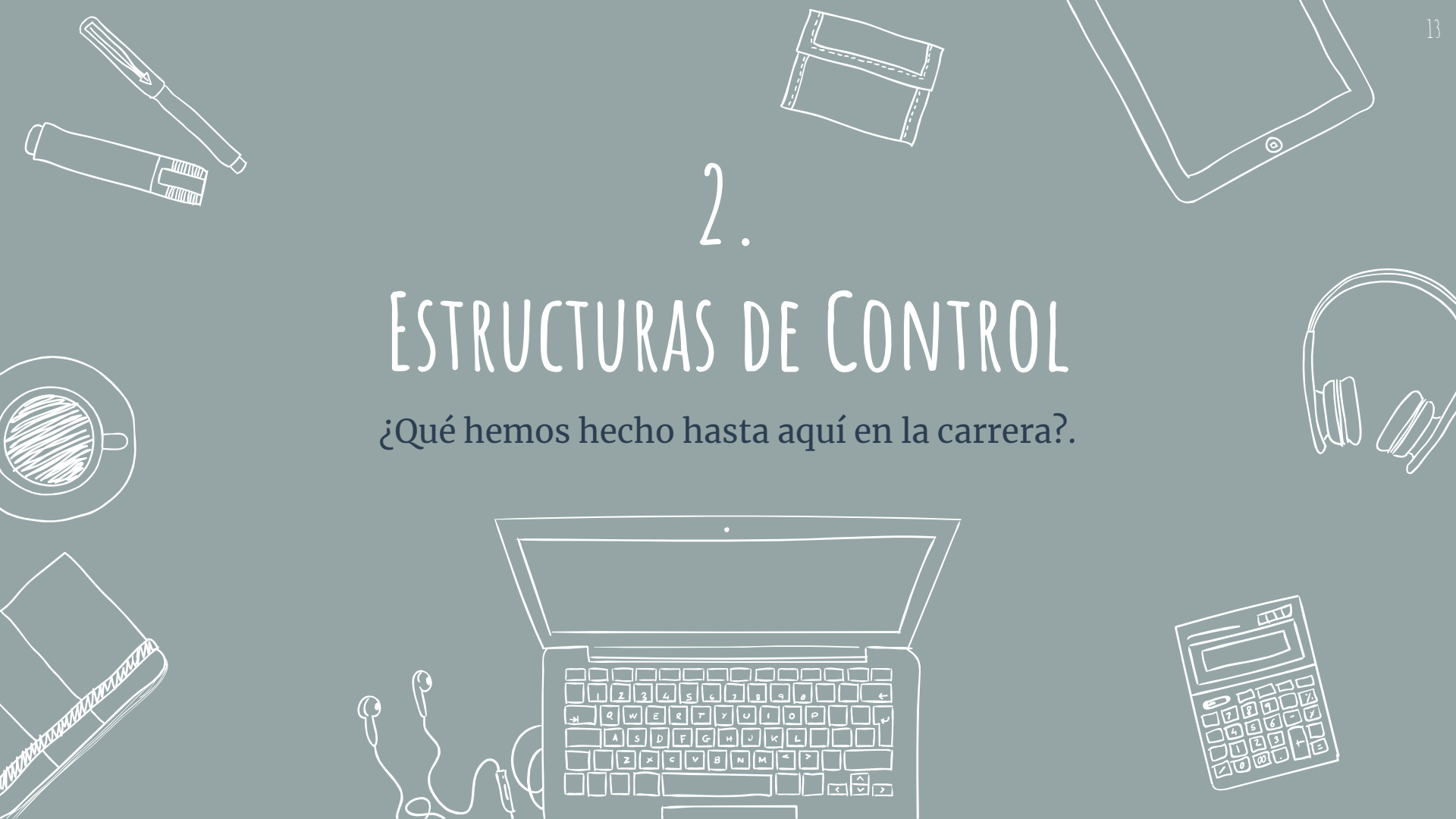
# OPERADORES

Los operadores responden en base a los tipos de los operandos e incluso infieren tipos para lograr obtener resultados.

2.

# ESTRUCTURAS DE CONTROL

¿Qué hemos hecho hasta aquí en la carrera?.



Estructura	Descripción
<code>if () {}</code>	condicionamiento de bloque o condicional simple
<code>if () {} else {}</code>	condición natural con alternativas por verdadero y falso
<code>switch () {   case   default }</code>	Estructura nueva, denominada condicional múltiple, reemplaza la cadena de <code>if / elif / elif / else</code> que utilizábamos en Python. Puede ser remplazada en javascript por <code>if + else if</code> pero su construcción es más clara y soporta mayor cantidad de alternativas sin generar código ilegible.
<code>while () {}</code>	Ciclo mientras, ciclo 0 a N
<code>do {} while ();</code>	Ciclo 1 a N natural, también itera mientras la condición es verdadera
<code>for( ; ; ) {}</code>	Ciclo for, ciclo exacto o alguna de sus múltiples formas
<code>() ? :</code>	Operador ternario, muy utilizado en para realizar asignaciones alternativas en línea o envío alternativo de parámetros en línea

NO HAY MUCHO PARA AGREGAR A LO QUE SABEMOS...

En general las estructuras de control son las mismas que conocemos de python, se suman el switch y el do while y el operador ternario cambia de forma.

## ALGUNAS NOVEDADES

### ( condición )

La condición debe expresarse entre paréntesis y estos determinan desde donde y hasta donde se evaluará la condición.

### break y continue

Funcionan de forma similar a Python y en caso de break se vuelve completamente necesario en la implementación de switch.

3.

# MODULARIZACIÓN → FUNCIONES

Vamos a construir subrutinas pero en Javascript lo podemos hacer de una y mil maneras veamos...





*El concepto de función como unidad de reutilización con parámetros de entrada y retorno de resultados no cambia.*

*A lo que nos vamos a tener que acostumbrar es a que hay muchas formas de definir estas funciones en Javascript.*

## ALTERNATIVA BÁSICA

```
5 ✓ function saludar() {  
6   |   console.log("¡Hola!");  
7   |   }  
8
```

- Utilizamos la palabra reservada **function**.
- Agregamos un nombre o identificador de la función.
- Agregamos la lista de parámetros ( ... ) o el par de paréntesis si no recibe parámetros.
- Encerramos el cuerpo de la función entre llaves { ... }.
- Notar que no es obligatorio el **return**, si no hace falta simplemente devuelve **undefined**

```
18 function multiplicar(x, y) {  
19   |   return x * y;  
20   |   }  
21
```

## INCLUSO PODEMOS DEFINIR UNA FUNCIÓN SIN NOMBRE

- Podemos crear funciones con una cantidad variable de parámetros.

```
28 function calcularPromedio(... numeros) {  
29     let suma = 0;  
30     for (let num of numeros) {  
31         suma += num;  
32     }  
33     return suma / numeros.length;  
34 }  
25
```

- Y también crear funciones con parámetros opcionales con valor por defecto:

```
10 function saludarA(nombre = "amigo") {  
11     console.log("¡Hola", nombre + "!");  
12 }  
13
```

También podemos crear una función que se va a ejecutar apenas termina su bloque de código.

- Tiene su fundamentación en la posibilidad de administrar variables locales
- Y es la manera en la que generalmente se define la función inicial en los programas Javascript.

```
58 (function main(){
59     // Llamada a la función sin parámetros.
60     saludar(); // Imprimirá "¡Hola!" en la consola.
61
62     // Llamada a la función sin especificar el parámetro.
63     saludarA(); // Imprimirá "¡Hola amigo!" en la consola.
64
65     // Llamada a la función especificando un nombre.
66     saludarA("Juan"); // Imprimirá "¡Hola Juan!" en la consola.
67
68     // Llamada a la función con parámetros.
69     sumar(5, 3); // Imprimirá "La suma de 5 y 3 es: 8" en la consola.
70
71     // Llamada a la función con parámetros y captura del resultado en una variable.
72     let resultadoMultiplicacion = multiplicar(4, 6);
73     console.log("El resultado de la multiplicación es:", resultadoMultiplicacion);
74
75     // Llamada a la función con un número variable de parámetros.
76     let promedio = calcularPromedio(4, 6, 8, 10);
77     console.log("El promedio es:", promedio); // Imprimirá "El promedio es: 7" en la
78 })();
```

# FUNCIONES ANÓNIMAS

```
6 const saludar = function (nombre = "amigo") {  
7     console.log("¡Hola", nombre + "!");  
8 };
```

- Utilizamos la palabra reservada **function**.
- En este caso no llevan nombre por lo que las vamos a asignar a una variable o constante para poder referenciarlas.
- Agregamos la lista de parámetros ( ... ) o el par de paréntesis si no recibe parámetros.
- Encerramos el cuerpo de la función entre llaves { ... }.
- Al disponer de estas funciones así podemos asignar la función como entidad a otra variable o realizar la ejecución. Para esto vamos a agregar los parámetros o los paréntesis vacíos.

```
32 let interactuar = (Math.random() > 0.5)  
33     ? despedir  
34     : saludar; // Asignará una función  
35 console.log(interactuar);  
36  
37 interactuar(); // Invocará a la función que se
```

# FUNCIONES FLECHA

- En este caso vamos a intercambiar la palabra **function** por el operador **=>**.
- La función sigue siendo anónima por lo que la vamos a asignar a una variable o constante
- Agregamos la lista de parámetros ( ... ) o el par de paréntesis si no recibe parámetros, en el caso de un solo parámetro se pueden omitir los paréntesis.
- Encerramos el cuerpo de la función entre llaves { ... }.

```
22 const dividir = (x, y) => {  
23     if (y == 0) return 0;  
24     return x / y;  
25 };
```

```
27 const cuadrado = a => a * a;
```

- En el caso en que la única línea del cuerpo de la función fuera el return se pueden omitir las llaves y la palabra reservada return

```
17 // const sumar = (a, b) => {return a + b;};  
18 const sumar = (a, b) => a + b;      You, 4 hou  
19
```

4.

# ESTRUCTURAS DE DATOS → ARRAYS

Ahora comenzamos a trabajar con la estructura de datos lineal general de Javascript, los arrys...

# ESTRUCTURAS DE DATOS EN JAVASCRIPT!

## Colecciones indexadas

Son similares a las listas utilizadas en Python y por lo tanto los primeros que comenzaremos a trabajar.

Además existe un tipo especial de arrays denominados Arreglos tipados pero exeden al contexto de la asignatura.

**Como siempre en las estructuras de datos nos interesan fundamentalmente las operaciones: agregar elemento, eliminar elemento, iterar todos los elementos e identificar un elemento.**

## Colecciones con clave Maps y Sets

Se agregaron en ECMAScript 6.

Los set son conjuntos de valores independientes que permiten la implementación de operaciones de conjunto y no mantienen orden ni admiten repetidos.

Los maps son colecciones de pares ordenados [clave:valor] donde las claves son un conjunto y los valores están asociados a estas.



# ARRAYS → CREACIÓN

➤ Para crear un array en Javascript tenemos varias alternativas

➤ Crear un array estático.

```
3 // Crear un array estático
4 let estatico = ['a', 'b', 'c', 'd'];
5 console.log(estatico);
6
```

➤ Crear un array vacío y agregar elementos.

```
7 // Crear un array vacío y agregar elementos al
8 let vacio = [];
9 console.log("Antes", vacio);
10 vacio.push('a');
11 vacio.push('b');
12 vacio.push('c');
13 vacio.push('d');
14 console.log("Luego", vacio);
15
```

➤ Crear un array con elementos y agregar valor a cada posición.

```
16 // Crear un array de longitud definida
17 let definido = [];
18 definido.length = 4;
19 console.log("Antes c module definido do);
20 let definido: any[]
21 for (let i = 0; i < definido.length; i++) {
22   console.log(typeof definido[i]);
23   definido[i] = estatico[i];
24 }
25 console.log(definido);
26
```

➤ O utilizar un esquema funcional

```
27 // Crear un array mediante un esquema funcional
28 let funcional = Array.from({ length: 5 }, () ⇒ Math.random());
29 console.log(funcional);
```

Método	Descripción
length	propiedad que permite el acceso a la cantidad de elementos del array
join	genera una cadena definiendo el separador de elementos
slice	permite obtener un subarray, como el operador de corte de python
keys	permite iterar todos los índices con valor del array
includes(x)	busca el valor de x en el array y retorna true si lo encuentra y false en caso contrario
indexOf(x)	busca x en el array y retorna el subíndice si lo encuentra y -1 en caso contrario
sort	ordena el array
push	agrega un elemento al final
pop	quita el último elemento del array y retorna su valor

## PRINCIPALES MÉTODOS DE LOS ARRAYS EN JAVASCRIPT...

Una breve lista de los métodos principales de manejo de arrays en javascript.

## Método

## Descripción

<code>for( ; ; )</code>	iterar por subíndice, si bien no se utiliza casi es la primera y más elemental forma de hacerlo
<code>forEach</code>	método que itera cada uno de los elementos <b>definidos</b> del array
<code>filter</code>	permite obtener un nuevo array con todos los elementos para los que la función enviada retorne true
<code>find</code>	permite obtener el primer elemento para el que la función enviada retorne true
<code>reduce</code>	obtiene un resultado de aplicar recursivamente la operación enviada como función a todos los elementos del array
<code>every / some</code>	determina si todos / algún elemento del array cumple con la función enviada
<code>map</code>	aplica la función enviada como parámetro a todos los elementos del array

## MÉTODOS FUNCIONALES DE ITERADO Y PROCESO DE ARRAYS...

Una breve lista de los métodos principales de manejo de arrays en javascript.

# Y LA PLABRA Desafío COMO SE ATACA?



## LIBRERÍA → INSTALAR E IMPORTAR

- Para instalar la librería vamos a utilizar un componente de node **npm**
- npm es quién permite administrar las dependencias de los scripts en node
- Para instalar seedrandom a nivel local global vamos a hacer:

```
philip@hpZbook-Philip MINGW64 /d/Facu/Grado/DDS/Clases/semana-04/scriptsClase
$ npm install -g seedrandom
```

- Luego para poder usarlo en nuestro script vamos a hacer disponible el path de instalación.

```
philip@hpZbook-Philip MINGW64 /d/Facu/Grado/DDS/Clases/semana-04/scriptsClase
$ export NODE_PATH=$(npm root -g)
```

- Si conocemos el directorio también lo podemos hacer con la cadena

```
philip@hpZbook-Philip MINGW64 /d/Facu/Grado/DDS/Clases/semana-04/scriptsClase
$ npm root -g
C:\Users\Public\Roaming\nodejs\node_modules
```

## NUESTRO CÓDIGO → USAR LA LIBRERÍA

- Para importar la librería por ahora vamos a usar require.

```
3  const seedrandom = require("seedrandom");
```

- Luego creamos la variable que nos permitirá generar los números aleatorios.

```
5  var random = seedrandom(1763519);
```

- Finalmente llenamos el array con los valores aleatorios, aquí una opción

```
7  // Generar 1000 números aleatorios en un array.  
8  let numerosAleatorios = Array.from({ length: 100 }, () => random.int32());  
9
```

GRACIAS!  
**Preguntas?**

