



UNIVERSIDAD
DE SANTIAGO
DE CHILE



Asociación de
Informáticos
UTE-USACH A.G.



/PRIMEROS PASOS CON JAVASCRIPT

Curso de FrontEnd

Sábado 16 de Noviembre 2024



LABOR LÆTITIA NOSTRA



/AGENDA



/01

/bienvenida

/02

/revisión tarea n° 4

/03

Condicionales JS

/04

Métodos JS

/05

/Callback Historia

/06

/Promesas

/07

/Then & Catch

/08

/Async - Await

/09

/tarea n°5



Horario

09hrs - Entrada
09:30hrs - Módulo 1
10:00hrs - Módulo 2
10:15hrs - Módulo 3
10:30hrs - Módulo 4
11hrs - Break
11:15hrs - Módulo 5
11:30hrs - Módulo 6
12:00hrs - Módulo 7
12:30hrs - Módulo 8
13:00hr - Cierre

Condicionales

JS

En programación, si queremos ejecutar un código cuando se cumple una condición o condiciones particulares, entonces, en tales casos, hacemos uso de algo llamado declaraciones **if**.

Por ejemplo, imagina cómo podríamos hacer un programa que nos diga si un número es mayor o menor a diez. Si es mayor a 10 debería imprimir una cosa, pero si es menor debería imprimir otra.

```
1 if (<condición>) {  
2   // código que se ejecuta si se cumple la condición  
3 }
```

If / else / else if / operador ternario

JS

If: Ejecuta una sentencia si una condición especificada es evaluada como verdadera. Si la condición es evaluada como falsa, otra sentencia puede ser ejecutada.

Else: La cláusula else (no obligatoria) sirve para indicar instrucciones a realizar en caso de no cumplirse la condición

```
if (condición) sentencia1 [else sentencia2]
```

If / else / else if / operador ternario

JS

Else If: Habrá momentos en los que desees probar múltiples condiciones. Ahí es donde entra el bloque else if.

```
if (la condición 1 es verdadera) {  
    // el código se ejecuta  
} else if (la condición 2 es verdadera) {  
    // el código se ejecuta  
} else {  
    // el código se ejecuta  
}
```

If / else / else if / operador ternario

JS

Operador ternario: El operador condicional (ternario) es el único operador en JavaScript que tiene tres operandos. Este operador se usa con frecuencia como atajo para la instrucción **if**.

```
condición ? expr1 : expr2
```

Métodos en JS

Métodos: Un método es una función la cual es propiedad de un Objeto. Existen dos tipos de métodos: Métodos de Instancia los cuales son tareas integradas realizadas por la instancia de un objeto, y los Métodos Estáticos que son tareas que pueden ser llamadas directamente en el constructor de un objeto.

Numéricos

```
//métodos para números  
  
// parseFloat() *  
  
// parseInt() *  
  
// toFixed()  
  
// toPrecision()
```

Strings

```
//métodos para strings  
  
// .charAt()  
//.concat(variable2);  
//.indexOf();*  
//.lastIndexOf();*  
//.replace("texto a encontrar", "texto que sustituye el original")  
//.slice("donde empiezo", "donde acabo");
```

Arrays

```
//métodos para arrays  
  
//.length *  
  
//.push()  
  
//.unshift()
```

Ejemplo: https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_tofixed
https://www.w3schools.com/jsref/tryit.asp?filename=tryjsref_number_parsefloat1

Ciclo for

For: Crea un bucle que consiste en tres expresiones opcionales, encerradas en paréntesis y separadas por puntos y comas, seguidas de una sentencia ejecutada en un bucle.

```
for (var i = 0; i < 9; i++) {  
    n += i;  
    mifuncion(n);  
}
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

For Of

For Of: La sentencia `for of` ejecuta un bloque de código para cada elemento de un objeto iterable, como lo son: `String`, `Array`, objetos similares a array (por ejemplo, `arguments` or `NodeList`), `TypedArray`, `Map`, `Set` e iterables definidos por el usuario.

```
for (variable of iterable) {  
    statement  
}
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

For Each

For Each: El método `forEach()` ejecuta la función indicada una vez por cada elemento del array.



```
1 const array = [1,2,3,4,5,6,7]
2 const resultArray = array.filter(element => element === 3 || element === 7)
3 console.log(resultArray) // [3, 7]
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

Map

Map: El método `map()` crea un nuevo array con los resultados de la llamada a la función indicada aplicados a cada uno de sus elementos.



```
1 const array = [1,2,3,4,5,6,7]
2 const resultArray = array.map(element => element + 10)
3 console.log(resultArray) // [11,12,13,14,15,16,17]
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)



Diferencias

¿Qué diferencias hay entre Map
y ForEach?

Filter

Filter: El método `filter()` crea un nuevo array con todos los elementos que cumplan la condición implementada por la función dada.



```
1 const array = [1,2,3,4,5,6,7]
2 const resultArray = array.filter(element => element === 3 || element === 7)
3 console.log(resultArray) // [3, 7]
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

Find

Find: El método `find()` devuelve el valor del primer elemento del array que cumple la función de prueba proporcionada.



```
1 const array = [1,2,3,4,5,6,7]
2 const resultArray = array.find(element => element > 3 )
3 console.log(resultArray) // [4]
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)



Diferencias

**¿Qué diferencias hay entre
Filter y Map?**

Some

Some: El método `some()` comprueba si al menos un elemento del array cumple con la condición implementada por la función proporcionada.

```
1 const array = [1,2,3,4,5,6,7]
2
3 // Los elementos son mayores que 4
4 const isGreaterThanFour = array.some(element => element > 4)
5 console.log(isGreaterThanFour) // true
6
7 // Los elementos son menores que 0
8 const isLessThanTen = array.some(element => element < 0)
9 console.log(isLessThanTen) // false
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

Every

Every: Determina si todos los elementos en el array satisfacen una condición.

```
1  const array = [1,2,3,4,5,6,7]
2
3  // Los elementos son mayores que 4
4  const isGreaterThanFour = array.every(element => element > 4)
5  console.log(isGreaterThanFour) // false
6
7  // Los elementos son menores que 10
8  const isLessThanTen = array.every(element => element < 10)
9  console.log(isLessThanTen) // true
```

1. ¿A qué se aplica este método? (Tipo de dato)
2. ¿Qué recibe este método? (Parámetro)
3. ¿Qué se obtiene? (Retorno o resultado)

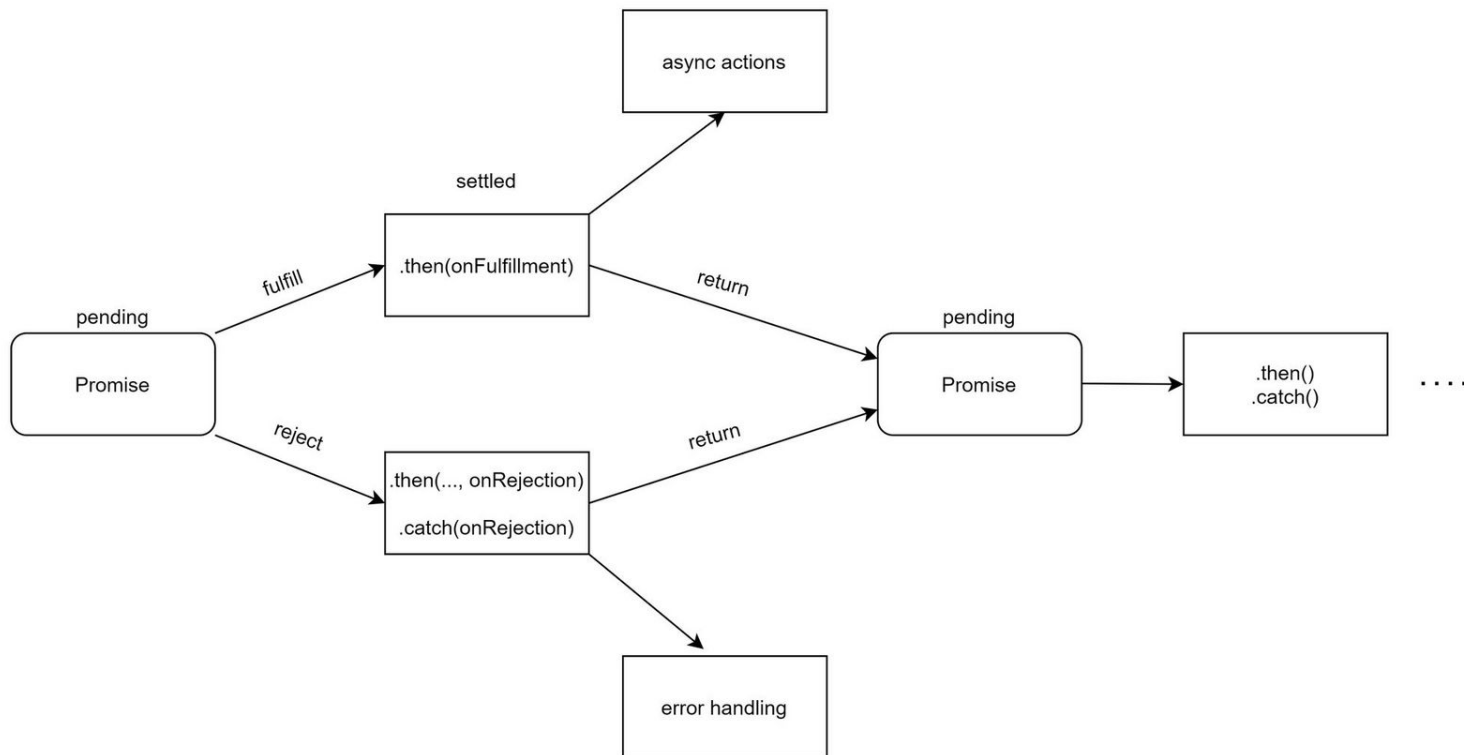
Promesas

Una **Promise** (promesa en castellano) es un objeto que representa la terminación o el fracaso de una operación asíncrona.

Esencialmente, una promesa **es un objeto devuelto al cuál se adjuntan funciones callback**, en lugar de pasar callbacks a una función.

```
function exitoCallback(resultado) {  
  console.log("Archivo de audio disponible en la URL " +  
    resultado);  
}  
  
function falloCallback(error) {  
  console.log("Error generando archivo de audio " + error);  
}  
  
crearArchivoAudioAsync(audioConfig, exitoCallback, falloCallback);
```

Promesas



Constructor new promise

El constructor **Promise** se utiliza principalmente para ajustar funciones que aún no admiten promesas.

```
const myFirstPromise = new Promise((resolve, reject) => {  
  // hacer algo asíncronico que eventualmente llame a:  
  //  
  // resolver (algúnValor) // cumplido  
  // or  
  // rechazar ("motivo de falla") // rechazado  
});
```

Then / Catch

Se utiliza para el manejo de promesas. El método `catch()` retorna una `Promise` y solo se ejecuta en los casos en los que la promesa se marca como `Reject`. Se comporta igual que al llamar `Promise.prototype.then(undefined, onRejected)` (de hecho, al llamar `obj.catch(onRejected)` internamente llama a `obj.then(undefined, onRejected)`).

```
p.catch(onRejected);

p.catch(function(reason) {
  // rejection
});
```

Async / Await

Se utiliza para trabajar con promesas de forma asíncrona.

```
function scaryClown() {  
  return new Promise(resolve => {  
    setTimeout(() => {  
      resolve('😱');  
    }, 2000);  
  });  
}  
  
async function msg() {  
  const msg = await scaryClown();  
  console.log('Message:', msg);  
}  
  
msg(); // Message: 😱 ← after 2 seconds
```



Operadores lógicos

Operador	Nombre	Ejemplo	Descripción
+	Suma	$5 + 6$	Suma dos números
-	Substracción	$7 - 9$	Resta dos números
*	Multiplicación	$6 * 3$	Multiplica dos números
/	División	$4 / 8$	Divide dos números
%	Módulo: el resto después de la división	$7 \% 2$	Devuelve el resto de dividir ambos números, en este ejemplo el resultado es 1
++	Incremento.	$a++$	Suma 1 al contenido de una variable.
--	Decremento.	$a--$	Resta 1 al contenido de una variable.
-	Invierte el signo de un operando.	$-a$	Invierte el signo de un operando.

Operadores lógicos



```
console.log( 0 == 0 );           // true
console.log( "" == 0 );          // true
console.log( false == 0 );       // true
console.log( undefined != 0 );   // true
console.log( null == 0 );        // false

console.log( '1' === 1 );        // false
console.log( 2 === 2 );          // true
console.log( 'abc' !== 123 );    // true
```



Tarea N°6

1. Mostrar en consola la secuencia de Fibonacci:
 - a. Entre los números 0 y 1000.
 - b. Números pares entre 0 y 1000.
 - c. Números impares entre 0 y 1000.

Hint: Puedes usar recursividad o algún ciclo o método iterador visto en clase

2. Del siguiente arreglo de strings retornar otro arreglo con todo a mayúsculas.
3. Del siguiente arreglo de objetos, retornar otro arreglo con los pokemon tipo fuego.

Nota: Arreglos en la siguiente diapositiva.

Enviar la tarea usando el formulario: softwarelibrechile.cl/G24-S2-06-Tarea

Importante Todas las tareas se deben subir en Github, no se acepta el envío de .zip mediante e-mail



//2.- Del siguiente arreglo de strings retornar otro arreglo con todo a mayúsculas.

```
let pokemon =  
  
[  
  'Pikachu',  
  'Charmander',  
  'Bulbasaur',  
  'Squirtle'  
]
```

//3.- Del siguiente arreglo de objetos, retornar otro arreglo con los pokemon tipo fuego.

```
let pokemon = [  
  {  
    nombre: 'Pikachu',  
    tipo: 'Electrico'  
  },  
  {  
    nombre: 'Charmander',  
    tipo: 'Fuego',  
  },  
  {  
    nombre: 'Bulbasaur',  
    tipo: 'Planta'  
  },  
  {  
    nombre: 'Squirtle',  
    tipo: 'Agua'  
  },  
  {  
    nombre: 'Charmeleon',  
    tipo: 'Fuego'  
  },  
  {  
    nombre: 'Weedle',  
    tipo: 'bicho'  
  },  
  {  
    nombre: 'Charizard',  
    tipo: 'Fuego'  
  }  
]
```

Rúbrica Tarea N°6



Rúbrica Tarea N°6 Evaluación para Tarea: JavaScript Básico			
Suma el porcentaje alcanzado en cada criterio para obtener una nota del 0 al 10. Esta rúbrica permite evaluar de manera clara y justa los distintos aspectos de la tarea, enfocándose en la correcta implementación de las soluciones, el uso de GitHub, y la aplicación de conceptos básicos de JavaScript.			
Criterio	Descripción	Ponderación	Notas
Entrega en GitHub	La tarea fue correctamente subida en un repositorio en GitHub según las instrucciones, sin uso de archivos .zip.	15%	0: No se entrega en GitHub. 5-10: Subida incompleta o con errores. 11-14: Correcta pero con detalles menores. 15: Entrega en GitHub sin problemas.
Fibonacci entre 0 y 1000	Secuencia de Fibonacci completa y correcta mostrada en consola.	20%	0: No implementada. 5-10: Parcial o con errores en los cálculos. 15-19: Correcta, pero con detalles menores. 20: Secuencia completa y bien implementada.
Fibonacci pares entre 0 y 1000	Cálculo correcto de los números pares de la secuencia y mostrados en consola.	15%	0: No implementada. 5-10: Parcial o con errores en la selección de pares. 11-14: Correcta con detalles menores. 15: Completa y bien implementada.
Fibonacci impares entre 0 y 1000	Cálculo correcto de los números impares de la secuencia y mostrados en consola.	15%	0: No implementada. 5-10: Parcial o con errores en la selección de impares. 11-14: Correcta con detalles menores. 15: Completa y bien implementada.
Arreglo de strings a mayúsculas	Uso correcto de métodos para transformar el arreglo de strings a un nuevo arreglo en mayúsculas.	15%	0: No implementado. 5-10: Parcial o con errores en el manejo del arreglo. 11-14: Transformación correcta con detalles menores. 15: Implementado perfectamente.
Arreglo de Pokémon tipo fuego	Uso correcto de métodos para filtrar objetos del arreglo de Pokémon y obtener solo los de tipo fuego.	20%	0: No implementado. 5-10: Parcial o con errores en el filtrado. 15-19: Correcto con detalles menores. 20: Filtrado completo y bien implementado.

/Porcentajes a cumplir en el curso

A continuación les dejamos información acorde al % de asistencia, entregas de tarea y proyecto final, todo esto de **carácter obligatorio** para obtener el certificado de aprobación.

/CLASES	/TAREAS	/PROYECTO FINAL
ASISTENCIA 80%	ENTREGAS 80%	ENTREGA 100%
11 clases	8 tareas	1 proyecto
Debe existir asistencia mínima de 9 clases	Deben entregarse mínimo 7 tareas o más	El proyecto final se hará de forma grupal, 2 personas, ambas obteniendo la misma nota.

/APROBACIÓN Y ENTREGA CERTIFICADO



/TAREAS

El promedio de las notas de la tareas equivale a un

/50%



/PROYECTO FINAL

La nota en el proyecto final equivale a un

/50%



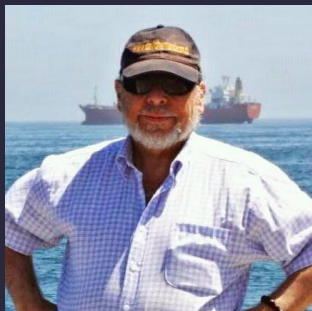
/NOTA FINAL

La nota debe ser igual o mayor a 6

/100%



Curso FrontEnd: Relatores



David Hernández

Director Asociación de
Informáticos UTE-USACH
A.G. | Paleo Informático
david.hernandezm@usach.cl
+56998246832

<https://www.linkedin.com/in/david-hern%C3%A1ndez-23205b3a>



Gonzalo Flemming

Tech Lead en Falabella
Financiero

<https://www.linkedin.com/in/gfleming-garrido/>



Sebastian Becerra

Gerente de Operaciones en
PropulsoW, Ingeniero
Informático, voluntario
Mozilla Chile ayudante en
taller Joomla dictado en la
Usach el 2014- 2015

<https://www.linkedin.com/in/sebaebc/>



Scarlet Melgarejo

Encargada de Canales
Digitales B2B - B2C en
Productos Torre , Ayudante
en Desafío Latam, voluntario
Mozilla Chile ayudante en
taller Joomla dictado en la
Usach el 2011 - 2015 2015.

<https://www.linkedin.com/in/scarlett-melgarejo-venegas-38805626/>



Curso FrontEnd: Relatores



Carolina Pirela

Ingeniera en Sistemas -
Front End Developer React
| Javascript | Sass |
Bootstrap
Dev UI Mobile @ Seiza

<https://www.linkedin.com/in/caropi30/>



Cristian Pavéz

Head of AI and Data Innovation
en Soluciones - an AVOS Tech
Company
Ingeniero en Informática

<https://www.linkedin.com/in/cristian-pavez>



Asociación de
Informáticos
UTE-USACH A.G.



UNIVERSIDAD
DE SANTIAGO
DE CHILE



<gracias!>