

ACÀMICA

Agenda

Daily

Programo: Promise ALL, Promise Race y Promise Chaining

Buenas prácticas: Largo de líneas

Break

Programamos: Vengan pokemones

Programan: Promise ALL, Promise Race y Promise Chaining

Cierre



Todo sobre promesas

Hoy veremos métodos que nos permiten ampliar el trabajo con promesas.

Podremos capturar todas las promesas, la primera que finalice de un grupo o trabajar con promesas en cadena

Daily



Daily



Sincronizando...

Bitácora



¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Challenge



¿Cómo te ha ido?
¿Obstáculos?
¿Cómo seguimos?

Promise ALL



Promise ALL

Promise.all se cumple cuando todas las promesas se han cumplido.

Si al menos una sola promesa es rechazada, promise.all será rechazada.



promise.all success

```

> let promesa1 = new Promise((resolve, reject) => {
  resolve("uno");
});
let promesa2 = new Promise((resolve, reject) => {
  resolve("dos");
});
let promesa3 = new Promise((resolve, reject) => {
  resolve("tres");
});

Promise.all([promesa1, promesa2, promesa3]).then(function(respuesta) {
  console.log(respuesta);
}).catch(function(error){
  console.log('Error: ' + error);
});

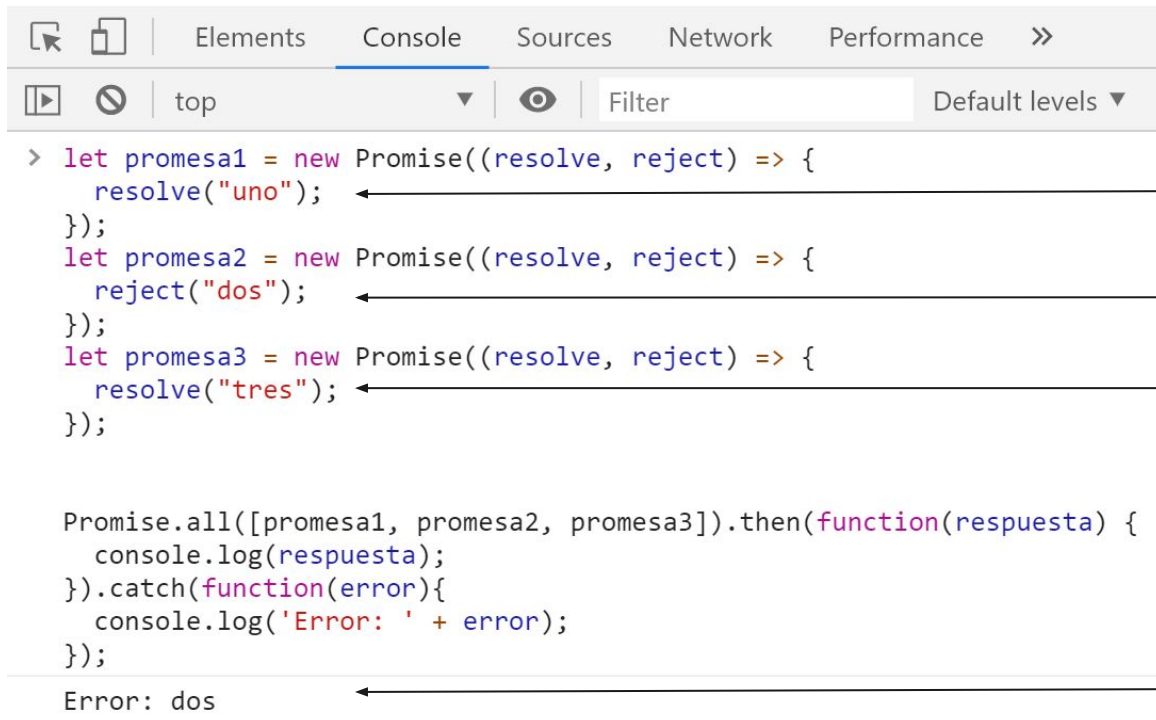
▶ (3) ["uno", "dos", "tres"]

```

Las 3 promesas
finalizan con `resolve()`

Mediante un array
recibimos las
respuestas de las 3
promesas

promise.all reject



The screenshot shows a web browser's developer console with the 'Console' tab selected. The console displays the following JavaScript code and its output:

```
> let promesa1 = new Promise((resolve, reject) => {  
  resolve("uno");  
});  
let promesa2 = new Promise((resolve, reject) => {  
  reject("dos");  
});  
let promesa3 = new Promise((resolve, reject) => {  
  resolve("tres");  
});  
  
Promise.all([promesa1, promesa2, promesa3]).then(function(respuesta) {  
  console.log(respuesta);  
}).catch(function(error){  
  console.log('Error: ' + error);  
});
```

The output in the console is:

```
Error: dos
```

Arrows in the original image point from the text annotations to the `reject("dos")` line in the code and the `Error: dos` output.

La segunda promesa finaliza con un `reject()`

El `reject()` es capturado por la función `catch()`

Promise.race

Promise.race obtiene la primera promesa que sea resuelta o rechazada.



promise.race

```
< Elements Console Sources Network Performance >>
top Filter Default levels ▼
> let promesa1 = new Promise((resolve, reject) => {
  setTimeout(function(){ resolve("uno") }, 500);
});
let promesa2 = new Promise((resolve, reject) => {
  setTimeout(function(){ resolve("dos") }, 300);
});
let promesa3 = new Promise((resolve, reject) => {
  setTimeout(function(){ resolve("uno") }, 1000);
});

Promise.race([promesa1, promesa2, promesa3]).then(function(respuesta) {
  console.log(respuesta);
}).catch(function(error){
  console.log('Error: ' + error);
});
◀ ▶ Promise {<pending>}
dos
```

3 setTimeout
diferentes

Mediante un array
recibimos las
respuestas de las 3
promesas

respuesta = a la
primera promesa
resuelta

Promesas en cadena

Existe la necesidad de ejecutar dos o más operaciones asíncronas con un orden específico, donde cada operación se ejecuta cuando la anterior tiene éxito. Logramos esto creando una cadena de objetos **promises**.



Promesas en cadena

```

> let promesa1 = new Promise((resolve, reject) => {
    resolve("uno");
  });
let promesa2 = new Promise((resolve, reject) => {
  resolve("dos");
});
let promesa3 = new Promise((resolve, reject) => {
  resolve("tres");
});

promesa1.then(
  function(respuesta){
    console.log(respuesta);
    return promesa2;
  }).then(function(segunda){
    console.log(segunda);
    return promesa3;
  }).then(function(tercera){
    console.log(tercera);
  }).catch(function(error){
    /*Captura cualquier error */
    console.log('Error: ' + error);
  });

```

uno

dos

tres

Retorna la promesa al siguiente **.then**

.catch captura cualquier error en la ejecución de alguna de las promesas

Programo

mentores/as



Promesas

Veamos estas 3 nuevas funcionalidades de promesas en acción.



Buenas prácticas



Limita el largo de las líneas

Una línea de código con una extensión prolongada dificulta su lectura.

Si debes realizar un condicional con muchas condiciones, escribe 1 condición por línea.



```
> if(es_mayor_de_edad === true && tiene_licencia_de_conducir === true && licencia_vigente === true && tiene_seguro === true){
  console.log("Puede conducir");
}

if(es_mayor_de_edad === true
  && tiene_licencia_de_conducir === true
  && licencia_vigente === true
  && tiene_seguro === true){
  console.log("Puede conducir");
}
```

A close-up photograph of a white ceramic cup filled with a latte. The surface of the milk is decorated with intricate latte art, featuring a central heart shape surrounded by concentric, wavy lines. The cup is placed on a matching white saucer. In the background, a white napkin and a silver fork are visible, though they are out of focus. The overall lighting is soft and even, highlighting the textures of the coffee and the smooth surface of the cup.

¡BREAK!



Programamos

todos/as



Javascript

Programamos

Ejecuta 3 llamadas diferentes al siguiente endpoint:

<https://pokeapi.co/api/v2/pokemon/> + (número entero)

Cuando todas las llamadas estén completas, muestra los resultados por pantalla modificando el DOM.

machop #66



paras #46



tentacool #72



Programan

estudiantes



Promesas

Genera un array con 10 posiciones numéricas.

Toma de manera aleatoria 3 elementos.

Lanza 3 fetch al mismo Endpoint de Pokemon y muestra solamente la primera respuesta.



Promesas

Cambia uno de los elementos de tu array a un valor tipo string.

Repite el proceso: toma de manera aleatoria 3 elementos.

Lanza 3 fetch al mismo Endpoint de Pokemon y muestra los siguientes resultados:

- Si los 3 request finalizaron correctamente, muestra los resultados por pantalla.
- Si algún request falló (te tocó el valor string en la elección aleatoria), muestra un error por pantalla.



Promesas

Genera una interfaz donde haya 3 inputs y un botón de buscar.

Valida que los 3 inputs estén completos y sean numéricos al momento de presionar buscar.

Busca el pokemon del ID del primer input recién cuando finalice busca el segundo. Haz lo mismo con el tercero.

Lanza cada una de las llamadas cuando estés seguro/a que la anterior finalizó.



Para la próxima

- 1) Termina el ejercicio del encuentro de hoy.
- 2) Lee la bitácora 31 y carga las dudas que tengas al Trello.
- 3) Resuelve el challenge.

En el encuentro que viene uno/a de ustedes será seleccionado para mostrar el ejercicio de hoy y otro/a mostrará cómo resolvió el challenge de la bitácora. De esta manera, ¡aprendemos todos/as de (y con) todos/as, así que vengan preparados/as!

ACÀMICA