ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS
INTRODUCCIÓN AL DESARROLLO WEB
2025 II
LABORATORIO 16 – JAVASCRIPT: MANEJO DE EXCEPCIONES Y ASINCRONÍA

I

OBJETIVOS

- Comprender la importancia de JavaScript en el desarrollo web
- Comprender los fundamentos del manejo de excepciones y la asincronía en JavaScript
- Solucionar ejercicios aplicando los conocimientos obtenidos

TIEMPO ESTIMADO: 2 horas

II

CONSIDERACIONES DE EVALUACIÓN

- Se deberán utilizar los conocimientos impartidos en las clases teóricas
- Deberá utilizar nombre de variables significativos
- Deberá realizar pruebas adicionales
- El alumno deberá indicar en su código con quien colaboró, así sea la IA
- El alumno será requerido de realizar modificaciones en su código y responder a preguntas sobre el mismo
- Los ejercicios deberán realizarse en el laboratorio, a su ritmo y subirlos al aula virtual como AVANCE antes de finalizar la clase
- Todos los ejercicios completos, deberán ser subidos al aula virtual como TAREA, para lo cual se les dará un tiempo adecuado y deben cumplir el deadline estipulado. Esto se deberá cumplir, aunque en el laboratorio ya se hayan terminado todos los ejercicios
- El formato a usar para los avances y tareas será .zip, que contenga todos los archivos requeridos
- Utilizar Git con GitHub para el manejo de versiones de su trabajo, ocasionalmente se le solicitará que compartan sus repositorios

III

POLITICA DE COLABORACION

La política del curso es simple, a menos que se exprese lo contrario en el laboratorio, siéntase libre de colaborar con sus compañeros en todos los laboratorios, pero debe notificar expresamente con quien ha colaborado. La colaboración con alumnos, que no están matriculados en el curso está prohibida. Los laboratorios y asignaciones han sido desarrollados para ayudarlo a comprender el material. Conozca su código y esté preparado para revisiones individuales de código. Durante las revisiones es probable que se le pida realizar modificaciones y justificar sus decisiones de programación. Cada uno de sus ejercicios debe iniciar de la siguiente forma:

```
// Laboratorio  Nro x - Ejerciciox
// Autor: mi nombre
// Colaboró : el nombre
// Tiempo :
```

INDICACIONES GENERALES

a. En cada sesión de laboratorio, los ejercicios propuestos deberán ser guardados en la misma carpeta/directorio
b. La carpeta deberá tener el nombre del Laboratorio y el nombre del alumno, así por ejemplo:
Laboratorio 11 – Juan Perez
c. Utilice nombres significativos
d. Su código deberá estar correctamente indentado y preferentemente documentado
e. Deberá ser debidamente probado

MARCO TEORICO

1. ¿Qué es una excepción?

Es un error que ocurre durante la ejecución del programa y que interrumpe el flujo normal del código si no es manejado.

Tipo de error	¿Cuándo ocurre?	Ejemplo
SyntaxError	Cuando el código tiene un error de sintaxis	if (true {
ReferenceError	Se usa una variable no declarada	console.log(nombre);
TypeError	Cuando se usa un tipo incorrecto	"hola".push("mundo");
RangeError	Un número está fuera del rango permitido	new Array(-5)
URIError	Error en funciones de URI (como decodeURI)	decodeURI("%");

2. ¿Por qué manejar excepciones?

- Evitar que la aplicación se detenga inesperadamente
- Guiar al usuario cuando ocurre un error
- Registrar problemas en el sistema
- Intentar recuperar una operación fallida

3. Sintaxis

```
try {
    // Código que puede generar un error
} catch (error) {
    // Código para manejar el error
} finally {
    // Código que se ejecuta siempre (opcional)
}
```

4. Ejemplos

```
try {
    // Código que produce un error
    let resultado = x + 1;   // x no está definida
} catch (e) {
    console.log("Ocurrió un error");
}
```

```
try {
    let texto = "{nombre: 'Juan'}";  // JSON inválido
    let obj = JSON.parse(texto);     // Lanzará un SyntaxError
    console.log("Objeto:", obj);
} catch (e) {
    console.log("Error al convertir JSON");
}
```

```
try {
    let numero = "hola";
    let resultado = numero * 2;   // Error: resultado será NaN
    if (isNaN(resultado)) {
        throw new Error("La operación no es válida");
    }
    console.log("Resultado:", resultado);
} catch (e) {
    console.log("Ocurrió un error:", e.message);
}
```

```
function dividir(a, b) {
    if (b === 0) {
        throw new Error("No se puede dividir entre cero.");
    }
    return a / b;
}

try {
    dividir(10, 0);
} catch (e) {
    console.log(e.message);
}
```

```
function dividir(a, b) {
    if (b === 0) {
        throw new Error("No se puede dividir entre cero.");
    }
    return a / b;
}

try {
    let resultado = dividir(20, 0);   // denominador cero
    console.log("Resultado:", resultado);
} catch (e) {
    console.log("Ocurrió un error:", e.message);
}
```

```
function procesarJSON(texto) {
    try {
        // 1. Intentar convertir el JSON (puede lanzar SyntaxError)
        let obj = JSON.parse(texto);

        // 2. Provocar un TypeError si obj no es lo esperado
        // (por ejemplo, acceder a una propiedad de undefined)
        console.log("Nombre:", obj.nombre.toUpperCase());
        // Si obj.nombre es undefined → TypeError

        console.log("Procesamiento exitoso:", obj);

    } catch (e) {

        if (e instanceof SyntaxError) {
            console.log("SyntaxError: JSON inválido");
            console.log("Mensaje:", e.message);

        } else if (e instanceof TypeError) {
            console.log("TypeError: El dato no tiene la estructura esperada");
            console.log("Mensaje:", e.message);

        } else {
            console.log("Error general");
            console.log("Mensaje:", e.message);
        }
    }
}
// PRUEBAS
// 1. JSON inválido → SyntaxError
procesarJSON("{nombre: Juan}");

// 2. JSON válido pero sin propiedad 'nombre' → TypeError
procesarJSON('{"edad": 30}');
```

5. Asincronía

Es uno de los conceptos más importantes del lenguaje, especialmente porque JavaScript se ejecuta en un solo hilo (single-thread), pero aun así puede manejar tareas largas sin bloquear la ejecución.
Se ejecuta instrucciones una detrás de otra, en un único hilo. Esto significa que, si una tarea tarda mucho (por ejemplo, leer un archivo o esperar una respuesta HTTP), el programa se bloquearía.

JavaScript usa mecanismos asincrónicos, que permiten:

- Delegar tareas al sistema o al navegador
- Continuar ejecutando otras instrucciones
- Recibir el resultado cuando esté listo

Así, una operación no detiene la ejecución del programa. El código continúa y, cuando la tarea esté lista, se ejecuta la instrucción correspondiente.

6. ¿Por qué es necesaria la asincronía?

Porque muchas operaciones son lentas, como:

- Peticiones HTTP (fetch)
- Lectura/escritura de archivos
- Temporizadores (setTimeout)
- Consultas a bases de datos
- Animaciones
- Eventos de usuario (clicks, teclas, etc.)

Si fueran síncronas, bloquearían todo el programa. JavaScript utiliza 3 mecanismos para manejar la asincronía.

7. Callbacks

Función que se pasa como argumento a otra función y que se ejecuta cuando la operación asincrónica termina.

```
function tareaAsincrona(callback) {
    console.log("Iniciando tarea...");

    setTimeout(() => {
        console.log("Tarea completada.");
        callback("Datos cargados correctamente");
    }, 2000);
}

tareaAsincrona(resultado => {
    console.log("Callback recibió:", resultado);
});
```

8. Promesas

Son un mecanismo moderno para manejar la asincronía, creado para reemplazar el uso excesivo de callbacks.
Una Promesa es un objeto que representa una operación asíncrona y cuyo resultado aún no se conoce en el momento en que se crea.

```
function tareaAsincrona() {
    console.log("Iniciando tarea...");

    return new Promise((resolve, reject) => {
        setTimeout(() => {
            console.log("Tarea completada.");
            resolve("Datos cargados correctamente");
        }, 2000);
    });
}

tareaAsincrona()
    .then(resultado => {
        console.log("Promesa recibió:", resultado);
    })
    .catch(error => {
        console.log("Error:", error);
    });
```

9. Async / Await

Permiten escribir código asincrónico con una sintaxis similar al código síncrono, más limpia y legible que las promesas.
Código moderno y limpio.

```
async function tareaAsincrona() {
    console.log("Iniciando tarea...");

    const resultado = await new Promise(resolve => {
        setTimeout(() => {
            console.log("Tarea completada.");
            resolve("Datos cargados correctamente");
        }, 2000);
    });

    console.log("Async/Await recibió:", resultado);
}

tareaAsincrona();
```

EJERCICIOS PROPUESTOS

1. Crear un directorio que tenga su nombre y un subdirectorio laboratorio17. Recomendación: usar minúsculas, sin espacios, sin tildes ni “ñ” y con guiones medios o bajos
2. Utilizar los atajos de teclado o combinación de teclas para agilizar su trabajo

   Crear el html y js que manipule el DOM para cada ejercicio:
3. Provocar un error de cualquier tipo e imprimir su mensaje
4. Generar un SyntaxError usando JSON.parse con texto inválido. En el catch mostrar e.name y e.message
5. Que console.log genere un error de cualquier tipo, en el catch imprimir “falló” y en finally imprimir “siempre se ejecuta”
6. Definir una función validarEdad(edad), si edad < 0 o no es número, lanzar un error ("Edad inválida"). Capturar y mostrar el error
7. Dentro de try provocar un TypeError. Ejemplo: let x = null; x.nombre;
   En catch verificar con instanceof si es TypeError. Imprimir un mensaje distinto según el tipo
8. En un catch dentro de una función imprimir un mensaje de error y luego volver a lanzar el mismo error con throw hacia otra función que la invocó. El error también será capturado en esa otra función y finalmente será capturado en un catch externo a cualquier de las funciones, el cual lo recibe e imprime. Comprueba que el error sigue propagándose
   Salida:
   Nivel 2 atrapó el error: x is not defined
   Nivel 1 recibió el error: x is not defined
   ERROR FINAL capturado en el nivel superior: x is not defined
9. Crear una función llamada cargarMensaje que reciba un callback, espere 1 segundo utilizando setTimeout y llame al callback con el mensaje: "Mensaje cargado" que deberá ser impreso en pantalla
10. Crea una función llamada cargarUsuario que reciba un callback. La función debe esperar entre 800 ms y 1500 ms (tiempo aleatorio usando setTimeout). Luego debe llamar al callback con un objeto que represente un usuario (id y nombre). El callback debe imprimir el usuario formateado, por ejemplo: Usuario cargado: Juancito (ID: 1)
11. Crea una función dividirAsync que reciba a, b y un callback. Espere 1.5 segundos. Si b === 0, llame a callback(new Error("No se puede dividir entre cero"), null); Si no, llame a: callback(null, a / b);
12. Crea una función llamada procesarLista. Debe recibir un arreglo de números, un callback, por cada número del arreglo esperar entre 500 y 1500 ms usando setTimeout. Imprimir "Procesando <numero>..." y cuando todos los números estén procesados llamar al callback final con el mensaje "Proceso completado"
13. Ejercicio 9 con promesas
14. Ejercicio 10 con promesas
15. Ejercicio 11 con promesas
16. Ejercicio 12 con promesas
17. Ejercicio 9 con async/await
18. Ejercicio 10 con async/await
19. Ejercicio 11 con async/await
20. Ejercicio 12 con async/await
21. Crear un repositorio remoto en GitHub y subir tu repositorio local. Compartir URL y pdf con captura de pantalla del código de los archivos js y de la ejecución

I. TAREA PARA LA CASA: Complete todos los ejercicios.

Crear un documento docx/pdf con la solución de los ejercicios.

Subir el documento a la tarea **Tarea 17** del Aula Virtual respetando las fechas indicadas.