UNIVERSIDAD NACIONAL DE SAN AGUSTÍN
FACULTAD DE INGENIERÍA DE PRODUCCIÓN Y SERVICIOS
ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

LAB11 - JAVASCRIPT

RESOLUCIÓN DEL LABORATORIO

ASIGNATURA:

PROGRAMACIÓN WEB 1

DOCENTE:

CORRALES DELGADO, CARLO JOSE LUIS

PRESENTADO POR:

MESTAS ZEGARRA, CHRISTIAN RAUL

AREQUIPA - PERÚ

2023

ACTIVIDADES:

1. Estudiar Javascript en w3schools y realizar un resumen de lo aprendido.
2. Agregar ejemplos de la función fetch.

RESUMEN:

Sintaxis básica

- El tipado de todas las variables es dinámico.
- La creación de variables se hace con las sintaxis: var nombre, let nombre o const nombre.
- Las asignaciones y operadores funcionan como en otros lenguajes.
- Los comentarios son señalados con // o /* */

Funciones

- Las funciones son declaradas con la sintaxis: function nombre(arg1, arg2, …) { }
- Las funciones pueden retornar valores con la palabra clave return.
- Las funciones se llaman con el operador (), donde se envían los argumentos.

Tipos y estructuras de datos

- Existen los tipos de datos primitivos comunes, como string, number, bigint, boolean, undefined, null, symbol, y existen los objetos (para POO).
- Existen estructuras de datos ya implementadas como arreglos, que permiten almacenar grupos de variables del mismo tipo.

Sintaxis de sentencias

- Las sentencias condicionales usan las sintaxis: if (condición) { } else { } else if (condición) { }
- La sentencia condicional múltiple usa la sintaxis: switch (expresión) { caso x: break; caso y: break; … }
- Las sentencias repetitivas usan las sintaxis: while (condición) { } o for (inicializacion; condicion; paso) { }
- Las sentencias repetitivas de objetos iterables usan la sintaxis: for (variable in objeto) { }

JavaScript y el DOM

- JavaScript puede manipular los elementos tanto del HTML como del CSS de una página.
- Se puede seleccionar estos elementos mediante métodos como getElementById, getElementsByClassName, getElementsByTagName y se puede modificar su html con la propiedad innerHTML o su estilo con la propiedad style.property.
- También se puede crear nuevos elementos y ponerlos en el HTML de una página con los métodos createElement y appendChild.

JavaScript asíncrono

- JavaScript permite enviar métodos como argumentos de otro método, así permitiendo llamar a estos una vez el método principal lo decida, a esto se le llama un *callback*.
- Un ejemplo de una función de *callback* es la funcion setTimeout(función, tiempo), que retrasa la ejecución de la función por el tiempo especificado.

- JavaScript también permite crear funciones promesas que devuelven un resultado de acuerdo a si la función terminó su ejecución correctamente. Estas usan la sintaxis: let promesa = new Promise(función(estado1, estado2, …)).
- Una vez recibida la respuesta esta se puede resolver con la sintaxis: promesa.then(función(estado1), función(estado2), …).
- Este proceso de promesas se puede resumir con las palabras clave async y await, que son el equivalente a retornar la promesa creada y resolverla.

AJAX

- Con la misma idea de la asincronía, AJAX es una técnica que permite enviar una solicitud a un servidor web, y esperar a su respuesta para realizar una acción.
- Todo el proceso ocurre con ayuda del objeto XMLHttpRequest, que posee los métodos open y send para enviar la solicitud, y el evento onload para procesar la respuesta del pedido.
- Esta técnica está fuera de uso ya que fue reemplazada por la API más simple, fetch

FETCH

- Fetch es una API que reduce el proceso de enviar la solicitud al servidor, parte de AJAX, todo ocurre dentro del método fetch, al que se envía un objeto Request como argumento y la respuesta se maneja asincrónicamente con .then y .catch.
- Ejemplo 1:

```
fetch('https://api.ejemplo.com/datos')
    .then(response => {
        if (!response.ok) {
            throw new Error(`Error de red: ${response.status}`);
        }

        return response.json();
    })
    .then(data => {
        console.log('Datos recibidos:', data);
    })
    .catch(error => {
        console.error('Error durante la solicitud:', error);
    });
```

  En este ejemplo, se usa el método fetch para recibir datos de una API, donde se verifica que la respuesta se haya recibido correctamente y luego se procesa la respuesta.
- Ejemplo 2:

```
const url = 'https://pagina.com/ejemplo.pl';

const requestOptions = {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
```

```
    },
    body: JSON.stringify({ key: 'value' }),
};

fetch(url, requestOptions)
    .then(response => {
        if (!response.ok) {
            throw new Error(`Error de red: ${response.status}`);
        }
        return response.json();
    })
    .then(data => {
        console.log('Respuesta del script Perl:', data);
    })
    .catch(error => {
        console.error('Error durante la solicitud:', error);
    });
```

En este ejemplo, se usa el método fetch para recibir la respuesta generada por un script de perl, donde se crean opciones personalizadas para el objeto Request, se verifica que la respuesta se haya recibido correctamente y luego se procesa la respuesta.