Ignacio Redondo Alfaro
C06420

1. ¿Cuáles son los data types que soporta javascript ?

String
Number
Bigint
Boolean
Undefined
Null
Symbol
Object: objects, arrays, dates, maps, sets, intarrays, floatarrays, promises.

2. ¿Cómo se puede crear un objeto en javascript? De un ejemplo

Hay varias formas de crear objetos en js:

```
/* Usando notacion literal*/
const car = {
  type: "Fiat",
  model: "500",
  color: "white"
};

/* Usando un constructor*/

const cars = new Object();
cars.type = "Fiat";
cars.model = "500";
cars.color = "white";

/* Usando una funcion*/
function Car(type, model, color) {
  this.type = type;
  this.model = model;
  this.color = color;
}

const car1 = new Car("Fiat", "500", "white");
```

3. ¿Cuáles son los alcances (scope) de las variables en javascript?

Tenemos las variables de alcance global, que son declaradas fuera de cualquier función o bloque y tiene inferencia en todo el código.

```
var globalVar = "Global";

function mostrar() {
  console.log(globalVar);  // se puede accesar
}
mostrar();
console.log(globalVar);  // se puede accesar
```

Variable locales que solo tiene inherencia en el rango de la función

```
function miFuncion() {
  var localVar = "Local";
  console.log(localVar);
}

miFuncion();
// console.log(localVar); // Error
```

Y las variables con las palabras reservadas let y const que solo pueden ser accedidas dentro del bloque de llaves {}

```
function miFuncion() {

  if (true) {
    let bloqueVar = "local";
    const bloqueConst = "También local";
    console.log(bloqueVar);
    console.log(bloqueConst);
  }
  console.log(bloqueVar);  // Error
  console.log(bloqueConst); // Error

}
```

4. ¿Cuál es la diferencia entre undefined y null?

Undefined es el valor predeterminado para variables sin inicializar mientras null es un valor dado por los programadores. Ambas significan que la variable está vacía pero una ya fue manipulada mientras la otra no.

5. ¿Qué es el DOM?

El DOM es la forma en que el navegador entiende y organiza el HTML para que se pueda interactuar de manera dinámica usando JavaScript. De esta manera podemos acceder y modificar elementos de la página sin necesidad de recargar.

En el siguiente ejemplo modificamos el texto de parrafo:

```html
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Ejemplo de DOM</title>
</head>
<body>

  <p id="miParrafo">Este es un párrafo.</p>
  <button onclick="cambiarTexto()">Cambiar Texto</button>

  <script>
    function cambiarTexto() {
      // Accede al elemento del DOM con el id "miParrafo"
      var parrafo = document.getElementById("miParrafo");
      // Modifica el contenido del párrafo
      parrafo.textContent = "El texto cambia!";
    }
  </script>
</body>
</html>
```

6. Usando Javascript se puede acceder a diferentes elementos del DOM. ¿Qué hacen, que retorna y para qué funcionan las funciones getElement y querySelector? Cree un ejemplo

getElementById: Se utiliza para buscar un elemento del DOM único por su id.

querySelector: Se utiliza para buscar el primer elemento que coincida con un selector CSS.

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo de getElementById y querySelector</title>
  </head>
  <body>
    <h1 id="tituloPrincipal">Bienvenido a mi sitio</h1>
    <p class="descripcion">Este es un sitio....</p>
    <button id="btnCambiar">Cambiar contenido</button>

    <script>
      // Obtenemos el elemento h1 por su id usando getElementById
      const titulo = document.getElementById("tituloPrincipal");
      // Obtenemos el primer párrafo que tenga la clase "descripcion" con querySelector
      const parrafo = document.querySelector(".descripcion");

      // Agregamos un evento de clic al botón para cambiar el contenido de los elementos
      document.getElementById("btnCambiar").addEventListener("click", function() {
        titulo.textContent = "Título actualizado";
        parrafo.textContent = "El contenido del párrafo ha sido modificado ...";
      });
    </script>
  </body>
</html>
```

7. Investigue cómo se pueden crear nuevos elementos en el DOM usando Javascript. De un ejemplo

Para esto se utiliza la función document.createElement().
Un ejemplo sería:

```html
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ejemplo de Agregar </title>
  </head>
  <body>
    <h1>Lista de Tareas</h1>
    <ul id="listaTareas">
      <li>Tarea 1</li>
      <li>Tarea 2</li>
    </ul>
    <button id="btnAgregar">Agregar Tarea</button>

    <script>
      // Se selecciona el botón y se añade un manejador de eventos para el clic
      document.getElementById("btnAgregar").addEventListener("click", function() {
        // Crear un nuevo elemento <li>
        const nuevaTarea = document.createElement("li");
        // Establecer el texto del nuevo elemento
        nuevaTarea.textContent = "Nueva tarea";
        // Agregar el nuevo elemento a la lista existente
        document.getElementById("listaTareas").appendChild(nuevaTarea);
      });
    </script>
  </body>
</html>
```

8. ¿Cuál es el propósito del operador this?

El propósito del operador this es identificar de quién se habla. Por ejemplo en objetos es común para indicar que la variable pertenece a un atributo de objeto determinado y no es una variable global o local.

9. ¿Qué es un promise en Javascript?De un ejemplo

La promesa se crea para manejar una operación asíncrona. Mientras la operación se ejecuta, la promesa está en estado "pending". Cuando la operación termina, se ejecuta la función correspondiente según si fue exitosa (resolve) o fallida (reject). Esto es justamente lo que se conoce como establecer un plan de acción para una operación cuyo resultado aún no se conoce.

```javascript
const miPromesa = new Promise((resolve, reject) => {
  // Simulamos una operación asíncrona, como una petición a un servidor.
  setTimeout(() => {
    const exito = true;
    if (exito) {
      resolve("Datos recibidos");
    } else {
      reject("Error en la operación");
    }
  }, 2000);
});

miPromesa
  .then(resultado => {
    console.log("La promesa se cumplió con:", resultado);
  })
  .catch(error => {
    console.error("La promesa fue rechazada:", error);
  });
```

10. ¿Qué es Fetch en Javascript? De un ejemplo

Fetch es una función integrada en JavaScript que permite pedir datos a un servidor y recibir una respuesta.

```javascript
fetch('https://linkdesupreferencia')
  .then(response => response.json())
  .then(data => {
    console.log("Datos recibidos:", data);
  })
  .catch(error => {
    console.error("Error en la petición:", error);
  });
```

11. ¿Qué es Asynch/Await en Javascript ? De un ejemplo

Async/await es una forma de manejar la asincronía en JavaScript de forma más clara, esperando que las operaciones asíncronas se completen antes de continuar con la siguiente línea de código.

```
async function obtenerDatos() {
  try {
    // Se espera hasta que fetch retorne una respuesta
    const respuesta = await fetch('https://ellinkmasbonitodelmundo.com');
    // Se espera hasta que la respuesta se convierta a JSON
    const datos = await respuesta.json();
    console.log('Datos recibidos:', datos);
  } catch (error) {
    console.error('Error al obtener datos:', error);
  }
}

obtenerDatos();
```

12. ¿Qué es un Callback? De un ejemplo

Un callback es una función que se pasa como argumento a otra función y se ejecuta después de que se complete una determinada tarea o proceso. Este mecanismo permite que una vez que se termine una acción, se ejecute otra función, facilitando el manejo de tareas asíncronas.

```
function cocinar(plato, callback) {
  console.log(`Empezamos a cocinar ${plato}...`);

  // Simulamos que cocinar toma 2 segundos
  setTimeout(() => {
    console.log(`${plato} está listo!`);
    callback(plato); // Una vez terminado, llamamos al callback
  }, 2000);
}

function servir(plato) {
  console.log(`Ahora se sirve el ${plato} en la mesa. ¡Disfruta tu comida!`);
}

// Llamamos a la función cocinar y le pasamos la función servir como callback
cocinar("pasta", servir);
```

13. ¿Qué es Clousure?

Un closure permite encapsular y ocultar datos dentro de una función, de modo que sólo puedan ser accedidos o modificados a través de métodos controlados. Esto ayuda a proteger la información y garantizar que solo se pueda modificar de la manera que uno decida.

14. ¿Cómo se puede crear un cookie usando Javascript?

Una cookie es un pequeño archivo de texto que un sitio web almacena en el navegador del usuario. Y se crean en js usando document.cookie = “”;

15. ¿Cuál es la diferencia entre var, let y const?

var: Tiene alcance de función/global, permite redeclaración.
let: Tiene alcance de bloque y permite reasignación.
const: Tiene alcance de bloque y no permite reasignación,