Programación Web
Ejercicios de examen

- Se recomienda que se intenten resolver los ejercicios por cuenta propia. Igualmente, recapacitar sobre posibles errores o deficiencias encontradas para su resolución durante el examen (si se realizó).
- No se muestran ejercicios cuya resolución supone el conocimiento o aplicación directa de la teoría.
- Se separan los enunciados de la posible solución para facilitar que se resuelvan los ejercicios sin dejarse influir.

Ejercicio 1

Desarrolle una función **`mapaDeHuecos`** que reciba un array disperso y devuelva un objeto donde las claves sean los índices de inicio de los huecos consecutivos y los valores, la longitud de cada secuencia de huecos.

Por ejemplo, para `[1, , , 4, , , , 5, 6, ,]`, debe retornar `{1: 2, 4: 3, 9: 1}`

Ejercicio 2

Cree una función llamada **`crearObjeto`** que tome como argumentos un nombre y una edad, y devuelva un objeto con:

1. Una propiedad **`nombre`** con el valor pasado como argumento.
2. Una propiedad **`edad`** con el valor pasado como argumento.
3. Un método **`actualizarEdad`** que reciba un número y actualice la propiedad edad.

Demuestre su uso actualizando la edad de un objeto creado.

Ejercicio 3

Sobrescribe el método **`toString`** del prototipo de **`Array`** para que devuelva una cadena con los elementos del array separados por **`|`** (carácter barra horizontal) en lugar de la coma por defecto. Demuestra el resultado con el array **`[10, 20, 30]`**

Ejercicio 4

Una cadena de producción en una fábrica de tecnología necesita controlar la calidad de los productos ensamblados. Sus datos están organizados en un array de objetos, donde cada objeto representa un producto que cuenta con los siguientes atributos:

- `id`: Identificador único del producto.
- `components`: Un array con los componentes del producto. Cada componente es representado por un objeto con:
  - `type`: tipo del componente ("chip", "sensor", "case", etc.).
  - `status`: estado del componente ("OK", "DAMAGED" o "MISSING").

El sistema debe verificar si el producto está defectuoso (esto es, si al menos uno de sus componentes tiene estado "DAMAGED" o "MISSING") o es válido (estado "OK"). Escriba una función en JavaScript que reciba el array de productos y devuelva un array con los identificadores (`id`) de los productos defectuosos. No está

permitido el uso del bucle for clásico (`for (..;..;..)`), ni de métodos `map`(), `reduce`() o `filter`(), en caso de conocerlos).

Considerando la siguiente entrada

```
const products = [
    {   id: "P001",
        components: [
            { type: "chip", status: "OK" },
            { type: "sensor", status: "DAMAGED" }
        ]
    },
    {   id: "P002",
        components: [
            { type: "chip", status: "OK" },
            { type: "sensor", status: "OK" },
            { type: "case", status: "OK" }
        ]
    },
    {   id: "P003",
        components: [
            { type: "chip", status: "MISSING" },
            { type: "sensor", status: "OK" }
        ]
    } ];
```

Se esperaría la siguiente salida como resultado: `["P001", "P003"]`

Ejercicio 5

Defina un objeto como función constructora, `Counter`, que de alguna forma encapsule un contador privado y proporcione un método único que, al ser invocado, devuelva un objeto con dos métodos: `increment()` y `getValue()`. El método `increment()` debe incrementar el contador privado, y `getValue()` debe retornar el valor actual del contador. La peculiaridad es que cada invocación al método que expone `increment()` y `getValue()` debe devolver nuevos invocadores independientes que mantengan su propio estado del contador.

Código de demostración de uso:

```
const myCounter = new Counter();
const counterMethods = myCounter.getCounterMethods();
counterMethods.increment();
counterMethods.increment();
const counterMethods2 = myCounter.getCounterMethods();
counterMethods2.increment();

console.log("1: " + counterMethods.getValue());  // 1: 2
console.log("2: " + counterMethods2.getValue()); // 2: 1
```

Ejercicio 6

Implemente herencia entre dos funciones constructoras, `Animal` (padre) y `Perro` (hijo), sin usar la palabra clave `class`.

- `Animal` recibe `nombre` y lo asigna a propiedad pública del mismo nombre. Tiene un método `comer()` público a todas las instancias de `Animal`. Este método genera un mensaje en consola `"¡Qué rico!"`.
- `Perro` recibe `nombre` y `raza`.
- Establezca la cadena de prototipos correctamente para que `Perro` herede de `Animal` y corrija la propiedad constructor.
- Inicializa un objeto `Perro` de ejemplo.

Ejercicio 7

Escriba una función asíncrona `obtenerUsuarios(url)` que use `fetch`. La función debe:

- Realizar la petición.
- Lanzar un error manualmente con el mensaje "`Error HTTP`" si el servidor responde con un código 404 o 500.
- Devolver los datos parseados como JSON si todo va bien.
- Manejar errores de red o de parsing

RESPUESTAS

Ver esta sección solo después de intentar realizar los ejercicios de forma autónoma.

Ejercicio 1

```
function mapaDeHuecos(arr) {
   // Se necesita un objeto que contenga los pares <índice: num_huecos>
   const mapa = {};
   // Inicio de una secuencia de huecos
   let inicio = null;
   // Registro de la longitud de la secuencia
   let longitud = 0;

   // Se recorre el array en toda su longitud
   for (let i = 0; i < arr.length; i++) {
      // Se comprueba que el hueco está vacío, evitando comparaciones con
      'undefined', 'null' u otros valores que podrían ser valores válidos de la
      posición del array
        if (!(i in arr)) {
      // Si inicio es 'null' indica que es el comienzo de una nueva secuencia, por
      lo que se asigna el valor del índice i
            if (inicio === null) inicio = i;
      // Y la longitude se va incrementando de uno en uno mientras que haya huecos
      consecutivos
            longitud++;
      // Se encuentra un valor en el array que no es un hueco
        } else {
      // Si se estaba contabilizando un hueco consecutivo…
            if (inicio !== null) {
            // Se registra en el objeto con la clave 'inicio' y valor 'longitud'
                mapa[inicio] = longitud;
            // Y se reestablecen los valores de 'inicio' y 'longitud' para buscar
            el siguiente hueco que pudiera haber
                inicio = null;
                longitud = 0;
            }
        }
    }

// Se termina el bucle, por lo que hay que comprobar que no se estuvieran
contabilizando huecos. En ese caso, se registra en el objeto y se devuelve
    if (inicio !== null) {
        mapa[inicio] = longitud;
    }

    return mapa;
}

// Se comprueba el funcionamiento
const arr = [1, , , 4, , , , 5, 6, ,];
console.log(mapaDeHuecos(arr)); // {1: 2, 4: 3, 9: 1}
```

Ejercicio 2

Para realizar este ejercicio, hay que tener en cuenta que se está pidiendo que el método devuelva un objeto literal, por lo que no se debe caer en implementar una función constructora, sino una función convencional que devuelve un objeto `{ .. }`

```
// La función recibe dos parámetros que representan las propiedades del objeto.
function crearObjeto(nombre, edad) {
      // Se devuelve un objeto literal, que contiene las propiedades de los
      argumentos (nombre, edad) y un método que recibe la nueva edad, n.
    return {
        nombre,
        edad,
        actualizarEdad(n) {
      // Para actualizar la propiedad 'edad', se debe acceder al contexto del
      objeto, por lo que es necesario el uso de 'this', que se refiere al objeto
      actual (el devuelto por crearObjeto)
            this.edad = n;
        },
    };
}

// Para probar la creación del objeto, es importante considerar que NO es una
función constructura (no se utiliza 'new') y, por tanto, se invocará como una
función convencional
const persona = crearObjeto("Raúl", 10);
// Para posteriormente invocar a actualizarEdad
persona.actualizarEdad(15);
console.log(persona);
```

Ejercicio 3

```
// Se debe entender que los métodos públicos de un objeto (‘Array’ en este caso) están en el ‘prototype’. Dicho de otra forma: todos los objetos basados en el prototipo comparten métodos. Por tanto, la función ‘toString’ se debe redefinir en el prototipo de todos los objetos creados a partir de ‘Array’.
Array.prototype.toString = function () {
      // La función ‘join’ combina todos los elementos de un array en una cadena utilizando el carácter entre paréntesis. En esta ocasión, ‘this’ hacer referencia al array que ha invocado el método
    return this.join('|');
};

// Se prueba con el array propuesto
const num = [10, 20, 30];
console.log(num.toString()); // "10|20|30"
```

Ejercicio 4

```
// Se define una función que recibe un array de productos y devuelve un array
// con identificadores de los productos que tengan algún componente en estado
// "DAMAGED" (dañado) o "MISSING" (faltante)
function findDefectiveProducts(products) {
    // Array que almacena los identificadores de productos defectuosos
    const defectiveProducts = [];

    // Iteramos sobre cada producto del array -->bucle for..of
    for (const product of products) {
        // Variable tipo bandera para determinar si el producto es defectuoso
        let isDefective = false;

        // Iteramos sobre cada componente del producto
        for (const component of product.components) {
            // Si el componente tiene estado "DAMAGED" o "MISSING"
            if (component.status === "DAMAGED" ||
                component.status === "MISSING") {
                // Marcamos el producto como defectuoso
                isDefective = true;
                // Y salimos del bucle de componentes (no seguimos comprobando)
                break;
            }
        }

        // Si el producto fue marcado como defectuoso, agregamos su id
        if (isDefective) {
            defectiveProducts.push(product.id);
        }
    }

    // Devolvemos el array de productos defectuosos
    return defectiveProducts;
}
```

Ejercicio 5

```
// Función constructora del objeto Counter
function Counter() {
  // Método que expone los métodos solicitados y el contador privado
  this.getCounterMethods = function () {
    // Variable privada para estado del contador
    // (no accesible desde fuera de esta función)
    let count = 0;

    // Se retorna un objeto literal con los métodos increment y getValue
    return {
      // Método para incrementar el valor del contador
      increment: function () {
        count++;
      },

      // Método que devuelve el valor del contador
      getValue: function () {
        return count;
      },
    };
  };
}
```

Ejercicio 6

Para responder este ejercicio hay que considerar que en clase no se han estudiado ciertos métodos como `Object.create()` o `call()`, por lo que la solución debe ofrecerse sin estas invocaciones. Se valorarán las competencias referidas al conocimiento de las funciones constructoras, uso del `this` y métodos en `prototype`, cadena de prototipos e inicialización de objetos.

```
function Animal(nombre) {
    this.nombre = nombre || "John Dog"; // Valor por defecto para el prototipo
}
Animal.prototype.comer = function() {
      console.log("¡Qué rico!");
};

function Perro(nombre, raza) {
    // Sin .call(), no podemos "pedir prestado" el constructor del padre para 'this'.
    // Debemos asignar la propiedad manualmente, lo que hace "sombra" (shadowing)
    // a la propiedad 'nombre' que existirá en el prototipo.
    this.nombre = nombre;
    this.raza = raza;
}

// Usamos una instancia real de Animal como prototipo.
// Esto conecta la cadena de prototipos: Perro -> instancia Animal -> Animal.prototype
Perro.prototype = new Animal();

// Corregimos el constructor
Perro.prototype.constructor = Perro;

// Prueba
const p = new Perro("Chucho", "Catalburun");
p.comer();
```

Ejercicio 7

Para responder este ejercicio hay que considerar que se pide una función asíncrona, esto es, no se debe utilizar la notación propia de JS para *promises*, sino el *syntax sugar* `async`/`await`. Se valorará (a) el uso correcto de `async`/`await`; (b) la validación de errores mediante `response.ok`; (c) el parseado correcto mediante el método síncrono `.json()`; (d) el control de excepciones (errores manuales y de red) mediante `try`/`catch`.

```
async function obtenerDatos(url) {
    try {
        const response = await fetch(url);

        // Validación manual del estado HTTP – importante el uso de .ok
        if (!response.ok) {
            throw new Error(`Error HTTP: ${response.status}`);
        }

        const data = await response.json();
        return data;
    } catch (error) {
        console.error("Fallo en la petición:", error);
    }
}
```