Examen UD5, UD6 y UD7 - Grupo B

Nombre: **Apellido:**

El examen consta de 3 partes:

- **Preguntas de Desarrollo:** Se hacen en papel y sin conexión internet. **(30 minutos)**
- **Desarrollo de ejercicios en JavaScript:** Se hará con conexión a internet. **(2:00 horas)**

Los resultados de aprendizaje alcanzados con este examen quedan detallados de la siguiente forma:

UT 5 → 100% UD5 + 100% RA6 UT6 → 50% RA7 UT7 → 50% RA7

Y a su vez con la realización de este examen se puntúa sobre el **60%** de la nota final de cada uno de los resultados de la primera evaluación.

Preguntas de Desarrollo / 7 preg. / 0.428 ptos-pregunta / 3 ptos

Desarrollo 1: De los métodos o propiedades que tenemos para recorrer y modificar un árbol DOM indique cuales son las acordes a las siguientes descripciones :

a) Obtener una referencia al nodo padre:
b) Obtener una referencia al último nodo hijo:
c) Obtener una referencia al nodo anterior en el mismo nivel jerárquico:
d) Método para comprobar si un nodo tiene hijos:

Desarrollo 2: Complemente el siguiente código en el cual se implementa una función capaz de crear un nodo párrafo, con texto, con un atributo “class” y por último, se añade este nodo párrafo como hijo del nodo que se pasa por parámetro. Para su implementación debe de usar los métodos y propiedades estudiados para tal cometido:

```
function crearParrafo( nodo ) {
            //Crear elemento párrafo:
            let parrafo =  ;
            //Crear contenido para el párrafo con el texto "Párrafo":
            let nodoTexto = ;
            //Añadir el contenido al nuevo nodo párrafo:
            //Añadir al párrafo el atributo "class" con el valor "clasecss":
            //Añadir párrafo al nodo pasado por parámetro:
      }

      crearParrafo(document.body);  //Ejemplo de uso
```

Nombre: Apellidos:

Desarrollo 3: Añada al nodo **h1** dos eventos: **"onmouseover"** y "**onmouseout**". Aplícales a ambos eventos **la misma función callback** con el nombre que quieras. En dicha función, debes de implementar la lógica para detectar cual de los dos eventos ha disparado la función. Muestra por consola un mensaje con el nombre del evento que haya "triggeado" la función.

```
<body>
      <h1 id="eventos">Obtener información de un evento</h1>
      <script>
      // Código aquí:
      </script>
</body>
```

Desarrollo 4: ¿De qué otra forma podemos controlar los estados del objeto XMLHttpRequest cuando hacemos una implementación alternativa al *"httpRequest.onreadystatechange"* ? Explica la alternativa

Desarrollo 5: Implemente en el siguiente código el código necesario para ejecutar la función **"doTask"** y sea capaz de mostrar por consola, si todo va bien, el campo **"value"**, y si hay un error, el campo **"message"**.

```
function doTask (iterations) {
      return new Promise( (resolve, reject) => {
            const numbers = [];
            for (let i = 0; i < iterations; i++) {
                const number = 1 + Math.floor(Math.random() * 6);
                numbers.push(number);
                if (number === 6) {
                    reject({
                        error: true,
                        message: "Se ha sacado un 6"
                    });
                }
            }
            resolve({
                error: false,
                value: numbers ,
                valores : [ wfwef, effwef, wewef, wefwef]
            });
        })};
```

Desarrollo 6: Explique para qué se utilizan los métodos **Promise.any() y Promise.race()**. Deje bien claro **la diferencia principal** que existe entre estos 2 métodos estáticos del objeto Promise.

Desarrollo 7: Dado el siguiente código, ¿Cómo podemos **"desacoplar"** (**eliminar dependencias**) el código para hacer genérica la función llamadaServidor y así poder usarla desde otro archivo, y tener de alguna

Nombre: Apellidos:

forma, su resultado en el método desde el que se llama? Asegúrese de mostrar por consola también un posible error.

Código a desacoplar: libreriaDatos.js

```
function llamadaServidor(url )
{
        fetch( url )
        .then(  response =>  {
            if (response.status == 200) {
                return response.json();
            }
            else {
                throw new Error(response.status);
            }
        })
        .then(   json =>
             obtenerDatosAlumnos(json);
        )
}

function obtenerDatosAlumnos( json )
{
    console.log(  json.nombre );
}

llamadaServidor("alumnos.json");
```

Y aquí el desacoplado a completar:

libreriaDatos.js

```
<script srt="libreriaNetwork.js"></script>


function obtenerDatosAlumnos( ...  )
{

     console.log(  .....   .nombre );




}


llamadaServidor( ... );
```

libreriaNetwork.js

```
function llamadaServidor(    )
{
        fetch(url)
        .then(  response =>  {
            if (response.status == 200) {

                return response.json();

            }
            else {

                throw new Error(response.status);

            }
        })
        .then(   json =>  {

        } )
}
```

Nombre: Apellidos:

Desarrollo 1:

Desarrollo 2:

```
function crearParrafo( nodo ) {
            //Crear elemento párrafo:
            let parrafo =

            //Crear contenido para el párrafo con el texto "Párrafo":
            let nodoTexto =

            //Añadir el contenido al nuevo nodo párrafo:

            //Añadir al párrafo el atributo "class" con el valor "clasecss":

            //Añadir párrafo al nodo pasado por parámetro:

      }

      crearParrafo(document.body);  //Ejemplo de uso
```

Desarrollo 3:

```
<body>
      <h1 id="eventos">Obtener información de un evento</h1>
      <script>  // Código aquí:




      </script>
</body>
```

Desarrollo 4:

Nombre: Apellidos:

Desarrollo 5:
Desarrollo 6:
Desarrollo 7: (Contestar en el enunciado)

Nombre: Apellidos:

Desarrollo de ejercicios en JavaScript:

Ejercicio 1: **Ejercicio 1: 2.5 puntos**

TAREA 1.1. Debe de construir haciendo uso de **"callbacks"** la ejecución de los siguientes 3 métodos que deben de ejecutarse "anidádamente", es decir, uno detrás del otro, y cada uno de ellos enviará al siguiente método información con los métodos callbacks propios de las promesas, ya que deseamos que el día de mañana estos 3 métodos puedan ser reutilizables e independientes del código donde se puedan usar.

Función 1: `obtenerDatosTriangulos( url )`

Dicha función recibirá, como mínimo, un parámetro, la url donde se encuentra el archivo JSON "datosTriangulos.json". En dicha función debe de implementar la lógica para realizar una "simulada" llamada a servidor pero que sea capaz de cargar el archivo JSON almacenado en local. El código debe de garantizar la asincronicidad del proceso.

La función no devuelve mediante "return" , ni guarda en ninguna variable global el contenido del archivo.

La función debe "devolver" mediante "métodos de las promesas" solo el **array "triangulos"** (no el archivo entero).

Función 2 :

```
function calcularHipotenusa( triangulos ){
            setTimeout ( function() {
             let hipotenusa = Math.hypot(  <<<longCateto1>>>   , <<<longCateto2>>> );
            } , 2000);
        }
```

Dicha función recibe como mínimo un parámetro el cual es el array de los triángulos. La función calcula el valor de la hipotenusa partiendo de los valores de cada "cateto". Muy importante, el valor de la hipotenusa será guardada en el mismo objeto triangulo que se recibe con una propiedad denominada "hipotenusa". Esto se realizará para todos los triángulos.

Nombre: Apellidos:

La función no devuelve mediante "return" , ni guarda en ninguna variable global el contenido del nuevo triángulo.

La función debe "devolver" mediante "los métodos propios de las promesas" el objeto "triángulos" que se le pasó inicialmente pero con las modificaciones indicadas.

Función 3 : La siguiente función recibe el array con todos los triángulos como parámetro y no mostrará nada por consola. La función debe de devolver mediante la técnica de desacoplamiento indicada en el ejercicio para que devuelva dicha cadena de texto formateada para cada uno de los triángulos:

```
function formatearTexto( triangulos ){
            setTimeout ( function() {
                let cadena = "Cateto 1 = " + <<<longCateto1>>> + "\n";
                cadena += "Cateto 2 = " + <<<longCateto2>>> + "\n";
                cadena += "Hipotenusa = " + <<<hipotenusa>>> + "\n" ;
                cadena += "Nombre = " + <<<nombre>>> + "\n";
            } , 2000);
        }
```

Función 4: Esta función solo debe de ser capaz de mostrar por consola el contenido devuelto por la Función 3: formatearTexto (). De hecho, no hace falta que sea una función, simplemente que imprima la respuesta de la función formatearTexto.

El resultado final del ejercicio es el siguiente:

```
Cateto 1 = 2.5
Cateto 2 = 2.5
Hipotenusa = 3.5355339059327378
Nombre = equilátero
```

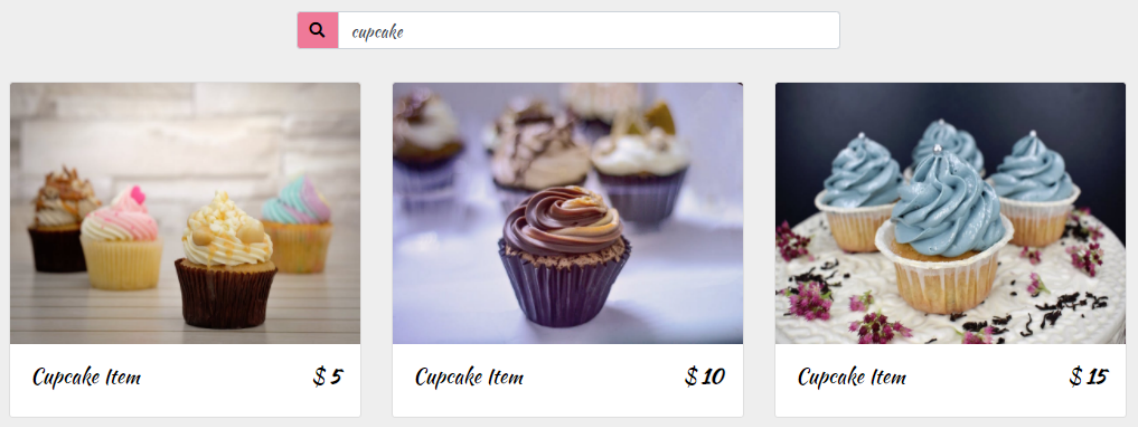
```
Cateto 1 = 3.5
Cateto 2 = 2.5
Hipotenusa = 4.301162633521313
Nombre = isósceles
```

```
Cateto 1 = 5.5
Cateto 2 = 2.5
Hipotenusa = 6.041522986797285
Nombre = escaleno
```

Ejercicio 2: **Ejercicio 2: 1.5 puntos**

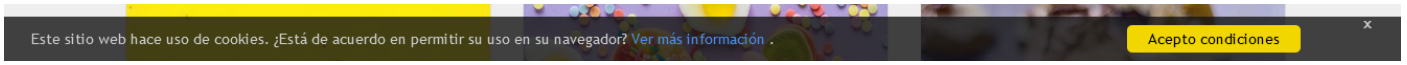
A partir del código facilitado que puede encontrar en la plataforma Moodle debe de implementar la funcionalidad a la lista de pastelitos para que el campo "buscar" pueda mostrar/ocultar (repito, mostrar/ocultar) aquellos pasteles que no "contengan" (repito, "contengan") el texto introducido en el campo buscar. Si el usuario pulsa el botón y no hay nada escrito (cadena vacía) en el campo de texto, debe de volver a mostrar todos los elementos de la lista de pastelitos. Ejem: si se busca "cupcake":

Nombre: Apellidos:



Ejercicio 3: **Ejercicio 3: 1.25 puntos**

A partir del código facilitado que puede encontrar en la plataforma Moodle debe de implementar la funcionalidad a la lista de pastelitos para que la siguiente ventana de aviso de uso de "almacenamiento" (da igual que sean cookies u otro tipo de almacenamiento de datos) tenga la siguiente funcionalidad:



➔ Si el usuario cierra pulsando sobre la "X", se "ocultará" el mensaje, pero la siguiente vez que se refresca la página, vuelve a aparecer el mensaje de aviso.

➔ Si el usuario hace click sobre "Aceptar condiciones", se debe de ocultar el mensaje, y las sucesivas veces que el usuario refresque la página, ya no se mostrará nunca más esta ventana/mensaje de aviso.

El código asociado a dicha ventana lo encontrará al final del archivo "index.html" y es el siguiente:

```
<div id="consentimientoCookies">
    <div id="closeconsentimientoCookies">X</div>
    Este sitio web hace uso de cookies. ¿Está de acuerdo en permitir su uso en su
navegador?  <a href="#" target="_blank">  Ver más información </a>. <a
id="consentimientoCookiesOK" class="consentimientoCookiesOK">Acepto condiciones</a>
</div>
```

Ejercicio 4: **Ejercicio 4: 1.75 puntos**

En el siguiente ejercicio debe de hacer uso de la función **"Fetch"** para cargar un archivo **"datos.json"** el cual contiene información sobre 3 nuevos pasteles. Debe de implementar la lógica para que nada más cargar la página, además de mostrar los pasteles que ya existen en el "html", se añadan al final los 3 pastelitos nuevos.

➔ El formato de la presentación de los datos tiene que ser el mismo que el del resto de pasteles. Las imágenes ya están contenidas dentro de la carpeta **/img**.

Nombre: Apellidos:

Pastel De Unicornio $5

Pastel De Lazos $6

Pastel De Arcoiris $7