FACULTAD DE CIENCIAS EXACTAS
UNIVERSIDAD NACIONAL DEL CENTRO DE LA PROVINCIA DE BUENOS AIRES

TUDAI
Parcial - WEB 1 (B)

16 JUNIO 2022

Alumno	DNI	Firma	#hojas
Comisión Teórica: Javier / Matias		Comisión Práctica (1-15):	

Nota:

- *Es necesario al menos un avance parcial en JS para aprobar*
- *El código debe estar prolijo e indentado.*

1- Verdadero y Falso

Marque verdadero o falso y justifique todas sus respuestas brevemente. Puede emplear un ejemplo para la justificación. Sin justificación la respuesta no tiene validez

a. Tiene sentido usar la sentencia: let r = document.querySelectorAll("#titulo")

   FALSO, no tiene sentido porque se seleccionan múltiples elemento que se guardan en un arreglo y un id #titulo es unico.

b. JSON puede tener variables de distinto tipo.

   VERDADERO, un JSON puede tener arreglos, strings, booleanos, numericos, incluso incluir objetos

c. Las media-queries solo funcionan para desktop.

   FALSO, funcionan para cualquier dimensión que se definan que se definan, puede ser con min-width usando mobile first para cualquier medida

d. Una página web escrita con buenas prácticas en cuanto a semántica es independiente del estilo que se aplique

   VERDADERO, la semántica se construye con las etiquetas HTML5, no tiene que ver con el estilo. No cambian el comportamiento visual de un sitio.

e. Partial Render y REST ambos utilizan AJAX entonces no pueden emplearse en el mismo sitio.

   FALSO, pueden usarse para ambos casos, si bien se maneja de distinto modo la información, es posible combinarlas.

2- Seleccione la respuesta correcta.

No es necesario justificar. Marcar sobre la misma hoja.
Respuesta mal contestada resta -0.25, sin contestar 0.00 y bien contestada +1.00.

A.¿Cual de estas cosas **NO mejoran** la semántica en una página web?

1. El uso de <figure>
2. **El uso de div's**
3. El uso de <nav> y <footer>

B. Para el diseño responsive, cual de las afirmaciones es **NO es correcta**

1. Siempre se utiliza el BOX MODEL
2. Se pueden usar unidades absolutas y relativas
3. **Es indistinto incluir meta name="viewport" dentro de HTML**

C. Determine cuál afirmación [redacted] **es correcta** para AJAX

1. Usa Promesas y es sincrónico
2. **Es una técnica asincrónica que utiliza una combinación de tecnologías**
3. Es un lenguaje con el que se puede hacer Partial Render y REST

D. Cual de las siguientes opciones considera **mala práctica en JS**

1. Usar "use strict"
2. **Definir variables globales**
3. Usar funciones anónimas

E. De acuerdo al siguiente código css responsive, decida la opción de cómo serán las vistas

```
body {
   background-color: blue;
   font-style: italic;
}

h1 {
   font-size: 15px;
}

@media only screen and (min-width: 600px) {
   h1 {
      font-style: normal;
      font-size: 25px;
      color: white;
   }
}
```

1. **Mobile:** Fondo azul - Texto h1, 25px de color negro Fuente estilo normal
   Desktop: Fondo azul - h1, 15 px color blanco
2. Mobile: Fondo Azul - Texto h1, 15 px color negro Fuente estilo itálica
   Desktop: Fondo azul - Texto h1, 25 px color blanco, Fuente estilo normal
3. **Mobile:** Fondo Azul - Texto h1, 25 px color blanco Fuente estilo normal
   Desktop: Fondo azul - Texto h1, 15 px color negro, Fuente estilo Italica
4. **Mobile:** Fondo Azul - Texto h1, 15 px color blanco Fuente estilo itálica
   Desktop: Fondo azul - Texto h1, 25 px color negro, Fuente estilo normal

3. HTML + CSS

A. Analice el siguiente **código HTML + CSS** y responda la pregunta:

```
<p>Texto 1</p>
<p class="resaltado">Texto 2</p>
<p>
   Texto 3
   <span class="resaltado">Texto 4</span> y
   <span class="suave">Texto 5</span>
</p>
```

```
p {
   color: black;
}
p.resaltado {
   color: green;
}
.resaltado {
   color: pink;
}
p .suave {
   color: blue;
}
```

- ¿De qué color queda cada uno de los textos? (Texto 1, Texto 2, Texto 3, Texto 4 y Texto 5). Justifiqué su respuesta mencionando que propiedades y mecanismos CSS se aplican para calcular el color resultante.

RESOLUCION

Texto 1 : Negro: no hay colisiones
Texto 2 : Verde: hay colisiones, CASCADA resuelve por ESPECIFICIDAD. Regla 2 es más específica que regla 3
Texto 3 : Negro: no hay colisiones
Texto 4: Rosa hay colisión por herencia. CASCADA resuelve por ESPECIFIDAD
Texto 5: Azul hay colisión por herencia. CASCADA resuelve por ESPECIFIDAD

B. Escriba solo el **código CSS** para crear algunos estilos generales a un sitio web. **Utilice tags semánticos**.

i. La tipografía del sitio completo debe ser “OpenSans” tamaño 15px, salvo los h2 que se encuentren dentro de un pie de página que debe tener tipografía “OpensSansThin” y con tamaño 11px.
ii. Todos los encabezados (h1, h2…) deben tener grosor de fuente normal, salvo los que estén dentro de un artículo con la clase **.noticia** que deben estar en **negrita**.

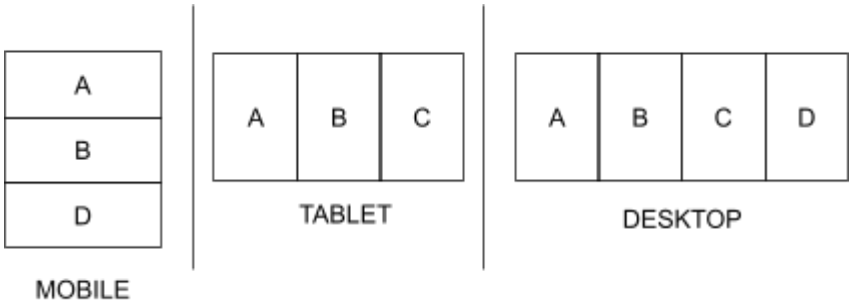
RESOLUCION

```
 /*1*/
body{
   font-family: 'OpenSans';
   font-size: 15px;
}

footer h2{
   font-family: 'OpenSansThin';
   font-size: 11px;
}

/*2*/
h1, h2, h3, h4, h5, h6 {
   font-weight: normal;
}

article.noticia h1, article.noticia h2,
article.noticia h3, article.noticia h4,
article.noticia h5, article.noticia h6 {
   font-weight: bold;
}
```

4. Diseño responsive

A. Escriba el código CSS necesario para crear el siguiente layout responsive a partir del HTML propuesto.
   - Se debe realizar utilizando **Mobile First.**
   - No se deben realizar cambios en el HTML

   Nota: no se puede utilizar ningún framework CSS




```
<div class="container">
   <div class="a">A</div>
   <div class="b">B</div>
   <div class="c">C</div>
   <div class="d">D</div>
</div>
```

RESOLUCION

Notas:

- NO IMPORTAN LAS DIMENSIONES, SOLO LAYOUT.
- NO IMPORTAN LOS BORDES
- LOS BREAKPOINTS PUEDEN USAR CUALQUIERA MIENTRAS SEAN LÓGICOS DE TAMAÑOS DE DISPOSITIVOS, NO PRETENDEMOS QUE SE LOS ACUERDEN DE MEMORIA

Alternativa FLEX simple 1

```
.container {
   display: flex;
   flex-direction: column;
}

.c {
   display: none;
}


@media only screen and (min-width: 576px) {
   .container {
       flex-direction: row;
   }

   .c {
      display: block; /* volver a mostrar */
   }

   .d {
       display: none;
   }
}

@media only screen and (min-width: 992px) {
   .d {
       display: block;
   }
}
```

Alternativa FLEX simple 2

```
.c {
   display: none;
}

@media only screen and (min-width: 576px) {
   .container {
       display: flex;
   }
    .c {
      display: block; /* volver a mostrar */
   }
    .d {
       display: none;
   }
}

 @media only screen and (min-width: 992px) {
   .d {
       display: block;
   }
}
```

En esta en mobile pueden dejar el comportamiento default al container

Alternativa GRID simple 1	Alternativa GRID CON AREAS

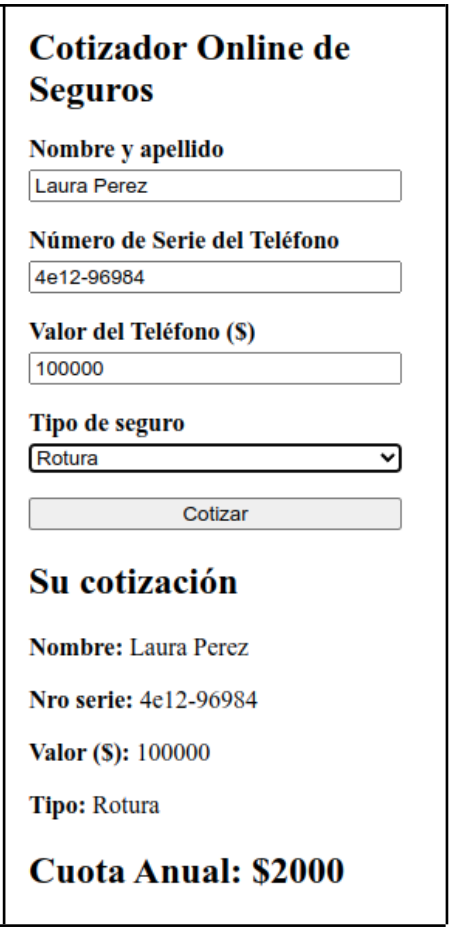
```
.container {
   display: grid;
}

.c { display: none; }

@media only screen and (min-width: 576px) {
   .container {
       grid-template-columns: repeat(3, auto);
       /* puede ser cualquier valor para lograr
encolumnar */
   }

   .c { display: block; }

   .d {
       display: none;
   }
}

@media only screen and (min-width: 992px) {
   .container {
       grid-template-columns: repeat(4, auto);
   }

   .d {
       display: block;
   }
}
```

```
.container {
   display: grid;
   grid-template-areas:"A"
                       "B"
                       "D";
}

.a { grid-area: A }
.b { grid-area: B }
.c { grid-area: C }
.d { grid-area: D }

.c { display: none; }

@media only screen and (min-width: 576px) {
   .container {
       grid-template-areas: "A B C D";
   }
   .c { display: block; }

   .d { display: none; }
}

@media only screen and (min-width: 992px) {
   .d {
       display: block;
   }
}
```

5- Javascript

La empresa de seguros Alliansur SA desea desarrollar un cotizador online de pólizas de seguros para teléfonos móviles. En este sistema web se podrán cargar los datos del teléfono y del propietario para que el calculador online genere de manera automática una cotización por un seguro.

```html
<h1>Cotizador Online de Seguros</h1>
<form id="form_cotizador">
   <label for="cliente">Nombre y apellido</label>
   <input id="cliente" name="cliente" type="text">

   <label for="nroserie">Número de Serie del Teléfono</label>
   <input id="nroserie" name="nroserie" type="text">

   <label for="valor">Valor del Teléfono ($)</label>
   <input id="valor" name="valor" type="number">

   <label for="tipo">Tipo de seguro</label>
   <select id="tipo" name="tipo">
       <option value="robo">Robo</option>
       <option value="rotura">Rotura</option>
   </select>

   <button id="btn_cotizar" type="submit">Cotizar</button>
</form>

<h2>Su cotización</h2>
<section class="contenedor_cotizacion">
 <!-- contenedor para escribir los datos de la cotización -->
</section>
```



El valor de la cuota anual se calcula de la siguiente manera:

- 0.1 % si es del tipo “Robo”
- 0.2 % si es del tipo “Rotura”

Escribir el código para cumplir los siguientes requerimientos:

- Se deben mostrar los datos de la cotización como indica la imagen una vez que se ingresaron los datos y se oprime Cotizar.
- Todos los campos son obligatorios y se deben validar (deben no estar vacíos al cotizar). Puede utilizar validaciones HTML5 si lo desea.
- El color de fondo del formulario <form> debe cambiar según el tipo de seguro:
    - **Rojo** para tipo “Robo”
    - **Gris** para tipo “Rotura”
- Se pueden repetir las cotizaciones cuantas veces quiera el usuario.

NOTA:

- si utiliza validaciones HTML5 agregue las propiedades al html en esta misma hoja
- escriba la(s) clase(s) CSS necesaria(s) si las necesita

AYUDA: El valor de un select se lee de la misma manera que un input

RESOLUCION

Se puede hacer de varias formas, dejamos dos modelos clásicos: usando validaciones HTML5 y validando en JS. Si usan HTML5 tiene que escuchar SI O SI el evento del submit y no del click.

ALTERNATIVA 1: C/VALIDACIONES HTML

JS:

```
"use strict"

/**
* Versión con FormData. No es necesario validar en JS.
*/

let container = document.querySelector(".contenedor_cotizacion");
let form = document.querySelector("#form_cotizador");

form.addEventListener('submit', cotizar);
function cotizar(e) {
   e.preventDefault(); // si o si el prevent default

   let formData = new FormData(form);

   let cliente = formData.get("cliente");
   let nroserie = formData.get("nroserie");
   let valor = formData.get("valor");
   let tipo = formData.get("tipo");

   /* usando FormData las validaciones quedan por HTML5,
   solo era necesario que agreguen los atributos "required"
   en el html */

   /* de este IF las variantes que se les ocurran estan OK
   incluso si repiten el if (tipo === ) para la parte de la clase
   no es un error que se considera grave en web 1.  */

   // en este caso no es necesario parsear "valor" a int, la conversion es implicita, si lo hacen está bien
igual
   let cuota = 0;
   if (tipo === "robo") {
       cuota = valor * 0.1/100;
   } else {
       cuota = valor * 0.2/100;
   }

   /* para imprimir en el DOM aceptamos cualquier forma,
   mientras pongan y concatenen los datos correctos */

   // el contenedor lo pueden agarrar acá directo también
   container.innerHTML = `
       <p>Nombre: ${cliente}</p>
       <p>Nro Serie: ${nroserie}</p>
       <p>Valor ($): ${valor}</p>
       <p>Tipo: ${tipo}</p>
       <h2>Cuota Anual: $${cuota}</h2>
   `;

   /* pueden usar el valor del select para definir la clase,
   o pueden armar una como crean necesario, incluso "fondo-azul" o "fondo-verde"
   Lo importante que dejen SOLO la clase que necesiten */
   form.classList.remove('robo');
   form.classList.remove('rotura');
   form.classList.add(tipo);
}
```

CSS

```
/* no es obligacion usar un combinado */
#form_cotizador.robo {
   background-color: lightblue;
}

#form_cotizador.rotura {
   background-color: lightgreen;
}
```

ALTERNATIVA 2: C/VALIDACIONES EN JS

```
"use strict"
```

```
/**
* Version sin FormData. Es necesario validar en JS.
*/

let container = document.querySelector(".contenedor_cotizacion");
let form = document.querySelector("#form_cotizador");
let btn = document.querySelector("#btn_cotizar");

btn.addEventListener('click', cotizar);
function cotizar(e) {
   e.preventDefault(); // el prevent default si o si xq el boton tiene type="submit"

   let cliente = document.querySelector("#cliente").value; // OJO SI TOMAN EL .value GLOBAL
   let nroserie = document.querySelector("#nroserie").value;
   let valor = document.querySelector("#valor").value;
   let tipo = document.querySelector("#tipo").value;

   /* las validaciones HTML5 no funcionan con esta forma, deben validar a mano */
   if (!cliente || !nroserie || !valor) {
       alert("Debe completar datos requeridos");
       return; // se permite hasta esta desprolijidad ;)
   }

   /* de este IF las variantes que se les ocurran estan OK
   incluso si repiten el if (tipo === ) para la parte de la clase
   no es un error que se considera grave en web 1.  */

    // en este caso no es necesario parsear "valor" a int, la conversion es implicita, si lo hacen está bien
igual
   let cuota = 0;
   if (tipo === "robo") {
       cuota = valor * 0.1/100;
   } else {
       cuota = valor * 0.2/100;
   }

   /* para imprimir en el DOM aceptamos cualquier forma,
   mientras pongan y concatenen los datos correctos */

   // el contenedor lo pueden agarrar acá directo también
   container.innerHTML = `
       <p>Nombre: ${cliente}</p>
       <p>Nro Serie: ${nroserie}</p>
       <p>Valor ($): ${valor}</p>
       <p>Tipo: ${tipo}</p>
       <h2>Cuota Anual: $${cuota}</h2>
   `;

   /* pueden usar el valor del select para definir la clase,
   o pueden armar una como crean necesario, incluso "fondo-azul" o "fondo-verde"
   Lo importante que dejen SOLO la clase que necesiten */
   form.classList.remove('robo');
   form.classList.remove('rotura');
   form.classList.add(tipo);
}
```

6. Javascript + JSON

Ahora el cotizador online debe ir agregando los datos de nombre y número de serie del teléfono a un arreglo local “consultas” cada vez que se realiza una cotización.

A continuación se describe el formato que tiene el arreglo consultas.

let consultas = [
{
"cliente": "Juan Garcia",
"nroserie": "01101289"
},
{
"cliente": "Martina Rodriguez",
"nroserie": "04103212"
}
];
	1. Escriba una función que agregue al arreglo el siguiente dato:
	"cliente": "Martin Garcia",
	" nroserie": "51101289"
	
	2. Escriba una función que muestre por consola: los números de serie de los teléfonos por los que hicieron consultas y el total de consultas realizadas.

NOTA: No debe modificar el ejercicio anterior, solamente escriba las dos funciones indicadas una abajo de otra.

RESOLUCION

1.

```
function agregarData(){
let renglon = {
       "nombre": "Martin Garcia",
       "nroserie": "51101289"
}
consultas.push(renglon);
}
```

2.

```
function mostrarNroSerie(){
for(let item of consultas){
       console.log(item.nroserie);
}
console.log(consultas.length);
}
```

Otra alternativa

```
function mostrarNroSerie(){
for(let i=0; i<consultas.length; i++){
       console.log(consultas[i].nroserie);
}
console.log(consultas.length);
}
```