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

TUDAI
Parcial - WEB 1 (A)

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. La sentencia: `let r = document.querySelectorAll('.social');` selecciona un elemento con id "social".

   FALSO, selecciona todos los elementos con clase social y los guarda en un arreglo.

b. La estructura de JSON es equivalente a la estructura de un arreglo.

   FALSO, no es la misma estructura, un arreglo es [a , b , c, ...] JSON puede tener elementos tipo objeto, objetos anidados, arreglos, arreglos de objetos.

c. Para una media-query no es lo mismo usar min-width que max-width

   VERDADERO, no es lo mismo el umbral min-width se usa para mobile first y aplica estilos cuando se supera dicho tamaño, en cambio max-width se usa como umbral para un tamaño menor.

d. HTML5 y semántica web son sinónimos

   FALSO, no es lo mismo, semántica aplica para dar sentido con etiquetas como <footer>, <nav>, HTML5 aplica al estandar, alguna etiquetas para HTML5 que permiten contenido multimedia son audio, canvas, video.

e. Una promesa siempre representa el resultado de una operación sincrónica.

   FALSO, la operación de la promesa es asincronica

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. Si en el archivo html pongo header después de footer en el navegador se va a ver:

1. El header arriba y el footer siempre abajo del todo.
2. **Dependerá de lo que diga el CSS.**
3. El footer antes del header.

B. Seleccione la opción que considere correcta respecto a **FLEX y GRID en un diseño responsive**

1. Solo se puede usar uno, FLEX o GRID
2. Se puede usar FLEX solo para diseños simples y GRID para diseños avanzados.
3. **Se pueden usar ambos en combinación, o simplemente uno de los dos.**

C. Cual de las siguientes **NO es una ventaja de AJAX**

1. **Las promesas se completan en un plazo de tiempo concreto**
2. Mejora la experiencia de usuario y la velocidad
3. Disminuye el volumen de información para transmitir

D. Cual de las siguientes afirmaciones **NO aplica para JS**

1. **Es un lenguaje en el que debe definirse el tipo de cada variable**
2. Es un lenguaje orientado a eventos y no pueden predecirse exactamente el orden de ejecución
3. Existen diferencias al declarar variables con var y con let

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

```
body {
    background-color: blue;
    font-family: Arial;
}

h1 {
    font-size: 18px;
}

@media only screen and (min-width: 600px) {
    h1 {
        font-family:'Times';
        font-size: 30px;
        color: red;
    }
}
```

1. **Mobile:** Fondo azul - Texto h1, 30 px de color rojo fuente Times
   Desktop: Fondo azul - Texto h1, 18 px color negro
2. **Mobile:** Fondo Azul - Texto h1, 18 px color negro Fuente Times
   Desktop: Fondo azul - Texto h1, 30 px color negro, Fuente Arial
3. **Mobile:** Fondo Azul - Texto h1, 18 px color negro Fuente Arial
   Desktop: Fondo azul - Texto h1, 30 px color rojo, Fuente Times
4. **Mobile:** Fondo Azul - Texto h1, 30 px color negro Fuente Arial
   Desktop: Fondo azul - Texto h1, 18 px color rojo, Fuente Times

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: red;
}
.resaltado {
   color: green;
}
p.suave {
   color: gray;
}
```

- ¿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 colisión. CASCADA resuelve por ESPECIFICIDAD (por orden tmb gana pero no llega a aplicarlo)
Texto 3 : Negro: no hay colisiones
Texto 4: Rojo: hay colisión. CASCADA resuelve por ESPECIFICIDAD (es más específica la regla 2)
Texto 5: Negro: no hay colisiones

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 14px, salvo los párrafos que se encuentren dentro de un pie de página que debe tener tipografía “OpensSansThin” y con tamaño 12px.
ii. Todos los encabezados (h1, h2…) deben tener grosor de fuente normal, salvo los que estén dentro de un div con la clase **.evento** que deben estar en **negrita**.

RESOLUCION

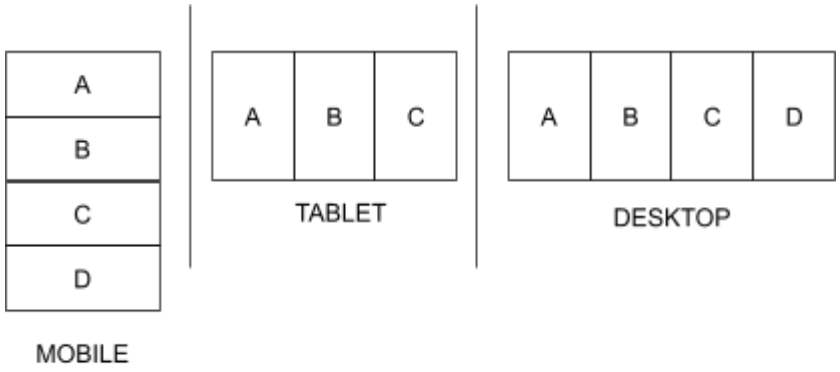
```
/*1*/
body { // o html
   font-family: 'OpenSans';
font-size: 14px;
}

footer p{
   font-family: 'OpenSansThin';
```

```
   font-size: 12px;
}

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

div.evento h1, div.evento h2,
div.evento h3, div.evento h4,
div.evento h5, div.evento 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	Alternativa FLEX simple 2
`.container { display: flex; flex-direction: column; } @media only screen and (min-width: 576px) { .container { flex-direction: row; } .d { display: none; } } @media only screen and (min-width: 992px) { .d { display: block; /* volver a mostrar */ } }`	`@media only screen and (min-width: 576px) { .container { display: flex; } .d { display: none; } } @media only screen and (min-width: 992px) { .d { display: block; /* volver a mostrar */ } }` En esta en mobile pueden dejar el comportamiento default al container

Alternativa GRID simple 1	Alternativa GRID CON AREAS

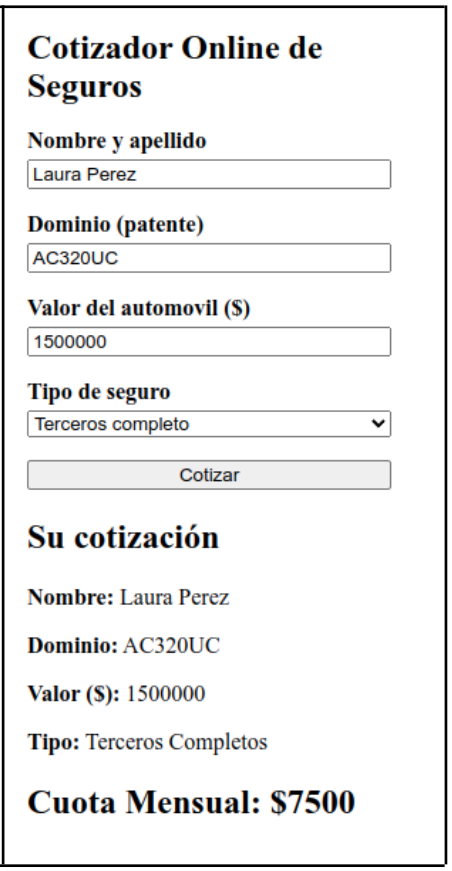
```
.container {
   display: grid;
}

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

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

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

   .d {
       display: none;
   }
}

@media only screen and (min-width: 992px) {
   .container {
       grid-template-areas: "A B C D";
   }

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

5- Javascript

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

```
<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="dominio">Dominio (patente)</label>
   <input id="dominio" name="dominio" type="text">

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

   <label for="tipo">Tipo de seguro</label>
   <select id="tipo" name="tipo">
       <option value="tercero">Terceros completo</option>
       <option value="todo_riesgo">Todo riesgo</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 mensual se calcula de la siguiente manera:

- 0.5 % si es del tipo “Terceros completos”
- 1 % si es del tipo “Todo Riesgo”

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:
  - **Azul** para “Terceros completos”
  - **Verde** para tipo “Todo Riesgo”
- 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 dominio = formData.get("dominio");
   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;
   let textoTipo = "";
   if (tipo === "tercero") {
       textoTipo = "Terceros completos";
       cuota = valor * 0.5/100;
   } else {
       textoTipo = "Todo riesgo";
       cuota = valor * 1/100;
   }

   /* 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>Dominio: ${dominio}</p>
       <p>Valor ($): ${valor}</p>
       <p>Tipo: ${textoTipo}</p>
       <h2>Cuota Mensual: $${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('tercero');
   form.classList.remove('todo_riesgo');
   form.classList.add(tipo);
}
```

CSS

```css
/* no es obligacion usar un combinado */

#form_cotizador.tercero {
   background-color: lightblue;
}

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

ALTERNATIVA 2: C/VALIDACIONES EN JS

```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 dominio = document.querySelector("#dominio").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 || !dominio || !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;
   let textoTipo = "";
   if (tipo === "tercero") {
       textoTipo = "Terceros completos";
       cuota = valor * 0.5/100;
   } else {
       textoTipo = "Todo riesgo";
       cuota = valor * 1/100;
   }

   /* 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>Dominio: ${dominio}</p>
       <p>Valor ($): ${valor}</p>
       <p>Tipo: ${textoTipo}</p>
       <h2>Cuota Mensual: $${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('tercero');
   form.classList.remove('todo_riesgo');
   form.classList.add(tipo);
}
```

6. Javascript + JSON

Ahora el cotizador online debe ir agregando los datos de nombre y dominio 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 = [
{
"nombre": "Juan Garcia",
"dominio": "ACF12UCT"
},
{
"nombre": "Martina Rodriguez",
"dominio": "KUR43CTD"
}
];
	1. Escriba una función que agregue al arreglo el siguiente dato:
	"nombre": "Martin Garcia",
	" dominio": "UTF54FSD"
	2. Escriba una función que muestre por consola: los dominios que hicieron consultas y el total de consultas realizadas.

RESOLUCION

1.

```
function agregarData(){
	let renglon = {
		"nombre": "Martin Garcia",
		"dominio": "UTF54FSD"
	}
	consultas.push(renglon);
}
```

2.

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

Otra alternativa

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