

Sprint 1 – Ejercicios

Índice

Índice	1
Objetivo	3
Comentarios generales	3
Ejercicio 1	4
Ejercicio 2	6
Ejercicio 3	7
Ejercicio 4	8
Ejercicio 5	9
Ejercicio 6	9
Ejercicio 7	10
Ejercicio 8	11
Ejercicio 9	12
Ejercicio 10	13
Ejercicio 11	15
Ejercicio 12	16
Ejercicio 13	16
Ejercicio 14	17
Ejercicio 15	18
Ejercicio 16	19
Ejercicio 17	20
Ejercicio 18	21
Ejercicio 19	22
Ejercicio 20	23

Ejercicio 21	24
Ejercicio 22	25
Ejercicio 23	26
Ejercicio 24 (Extra)	27
Ejercicio 25 (Extra)	29
Ejercicio 26 (Extra)	30
Ejercicio 27 (Extra)	31
Ejercicio 28 (Extra)	32
Ejercicio 29 (Extra)	33

Objetivo

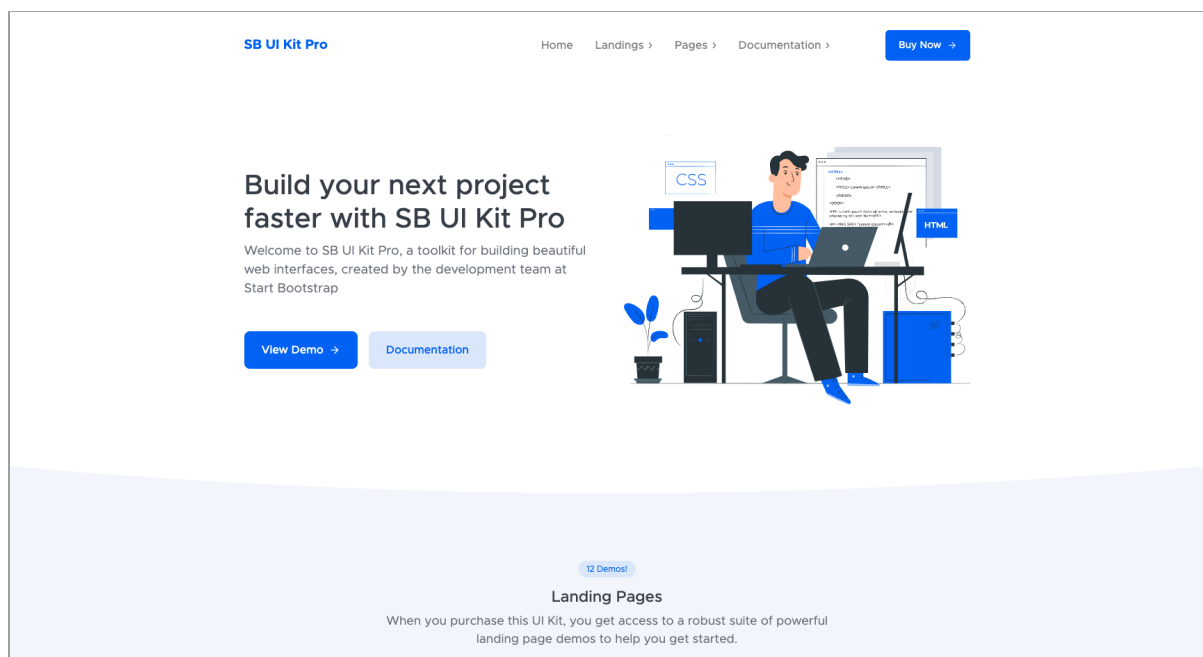
El objetivo de este Sprint es repasar los siguientes temas:

- Maquetación – [HTML](#) y [CSS](#).
- [Bootstrap](#).
- Programación en [JavaScript](#).
- Manipulación del [DOM](#) (preferentemente sin usar [jQuery](#)).

Comentarios generales

- **Leer en detalle** la pauta de cada ejercicio **antes de empezar a escribir código**.
- Notar que algunos ejercicios requieren que se haya dictado una clase previa (**teórico**) antes de poder resolverlos. Estos ejercicios estarán debidamente señalizados.
- En caso de **dudas**, pueden recurrir a sus compañeros, docentes (por Teams/Discord) y/o sitios en Internet (ej: Stack Overflow). Recuerden la importancia de apoyarse entre ustedes ya que una gran forma de aprender y reforzar conocimientos es explicarle a otro.

Ejercicio 1



Pauta:

- El ejercicio consiste en **maquetar** la siguiente página:
<https://sb-ui-kit-pro.startbootstrap.com/index.html>.
- Deberá ser hecho en forma **individual**, aunque podrán consultar sus compañeros, docentes o en Internet. Recuerden que ayudar a sus compañeros es una gran forma de aprender.
- Sólo es necesario replicar la **Home**. Pueden copiar las imágenes del sitio o usar otras de su preferencia.
- Verificar que el sitio se vea bien en **mobile**.
- ⚠ Intentar no copiar el código fuente (del link anterior).
- Hay varias partes de la página que tal vez aún no las saben hacer. ¡Para eso deberán **investigar** sobre nuevos temas (teóricos)! Por ejemplo, es probable que tengan que leer sobre:
 - [CSS Before](#) y [CSS After](#).
 - [CSS Transitions](#).
 - Imágenes [SVG](#). Ver [video explicativo](#). Junto con esto, algo que les puede ser útil es la herramienta [Figma](#) para crear imágenes vectoriales

y que se pueden exportar a SVG. Si están muy trancados para lograr la “panza” (curva), pueden mirar este [ejemplo de ayuda](#).

- [Flexbox](#). Recomendamos completar todos los niveles de [este juego](#).

Notas importantes sobre los ejercicios de maquetación

Menos es Más 🙌

En este tipo de ejercicios de maquetación, hemos visto con recurrencia que los alumnos utilizan ciertas **etiquetas en exceso** (principalmente *divs*), así como también hemos visto un **uso excesivo de clases CSS**. Es muy común que los alumnos se compliquen mucho más de lo necesario.

Por lo tanto, recomendamos que traten de ser más **austeros**. Es importante que se tomen unos minutos para pensar bien si realmente vale la pena agregar el pedazo de código que están pensando agregar. Sean **pacientes**. No se dejen llevar por la ansiedad de escribir código.

Terminar todo el ejercicio no es lo más importante 100

Terminar el 100% del problema no es lo esencial, particularmente en estos ejercicios de maquetación. Tampoco existe una carrera para ver quién es el alumno que termina primero.

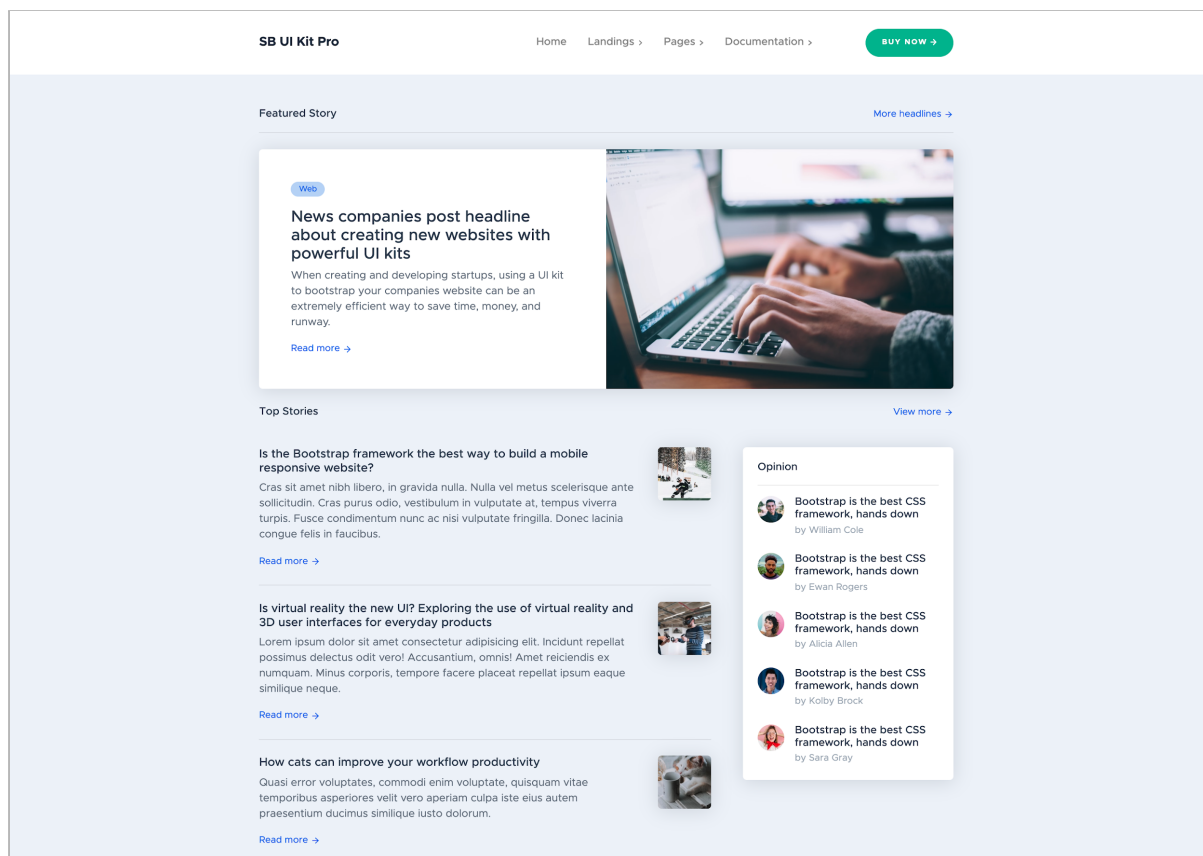
Lo que sí importa es que: **lo que hayan hecho, lo hayan hecho bien** (de forma prolija, ordenada y entendiendo cada una de las líneas de código que escribieron).

Por lo tanto, si pudieron maquetar toda la página pero, por ejemplo, les faltó hacer el `footer`, ¡no se desanimen! Valoren lo que sí hicieron y lo que aprendieron.

Ejercicio 2

- Investigar sobre **metodologías ágiles**.
 - ¿Qué ventajas / desventajas tienen sobre la gestión tradicional de proyectos?
 - ¿Qué tipos de metodologías ágiles existen?
- Investigar sobre **Scrum**.
 - ¿Qué ventajas y desventajas tiene?

Ejercicio 3



Pauta:

- Este es un ejercicio similar al Ejercicio 1. Se deberá maquetar la siguiente [página](#).
- Deberán trabajar en **equipos** de 2 o 3 alumnos (armados por el docente). Por lo tanto, deberán **dividirse las tareas** y elegir los canales de **comunicación** que consideren pertinentes. La idea es trabajar de la forma más sincronizada posible, como un "EQUIPO", con todas las letras y en mayúscula.
- La idea es que al finalizar el ejercicio, cada alumno deberá tener una copia idéntica de la solución.
- ⚠ Intentar no copiar el código fuente (del link anterior).

Ejercicio 4

Clase previa: “Línea de Comandos”.

Pauta:

Usando una terminal (ej: Hyper), preferentemente con una shell como Bash, Zsh o PowerShell:

1. **Crear** una carpeta llamada `EjercicioTerminal`.
2. **Crear** un archivo llamado `alumnos.txt`.
3. **Agregar** al archivo un listado con los nombres de los compañeros de clase.
4. **Guardar** el archivo.
5. **Crear** otro archivo llamado `tecnologias.txt`.
6. **Agregar** al archivo anterior un listado con algunas de las tecnologías que usaremos en el curso (ej: HTML, CSS, JavaScript, Node.js, etc).
7. **Listar** en la terminal todo el contenido de la carpeta `EjercicioTerminal`.
8. **Mover** dicha carpeta a otra ubicación de la máquina (por ejemplo, moverla a la raíz del disco o al Escritorio).
9. Hacer una **copia** de la carpeta llamada `EjercicioTerminal_v2`. Al realizar esto, deberían tener estas dos carpetas en la misma ubicación:
 - `EjercicioTerminal`
 - `EjercicioTerminal_v2`
10. **Borrar** todos los archivos dentro de la carpeta `EjercicioTerminal_v2`.

Ejercicio 5

Clase previa: “Línea de Comandos”.

Investigar sobre los siguientes comandos:

- `sudo` → ¿qué es? ¿para qué sirve? ¿cuándo usarlo?
- `rm -r`
- `df`
- `cat`
- `head`
- `tail`
- `curl`
- `grep`

👉 Un sitio que les puede ser de ayuda es [este](#) y [este](#). También les será de ayuda lo que se conoce como “[Command-line completion](#)” o simplemente “Tab completion”.

Ejercicio 6

Clase previa: “Git”.

Pauta:

Investigar sobre **Git**.

- ¿Para qué sirve?
- ¿Qué ventajas / desventajas tiene? ¿Vale la pena?
- ¿Qué hacían los desarrolladores antes de tener Git?
- ¿Qué diferencia hay entre Git y GitHub?

👉 Un alumno (elegido al azar) deberá contarle a toda la clase el resultado de su investigación. Si lo desean, pueden preparar una **pequeña presentación** con diapositivas (ej: Google Slides).

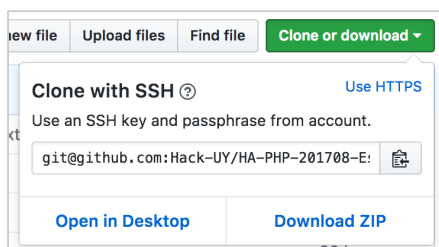
🔑 Si aún no tienen, háganse una cuenta en [GitHub](#).

Ejercicio 7

Clase previa: “Git”.

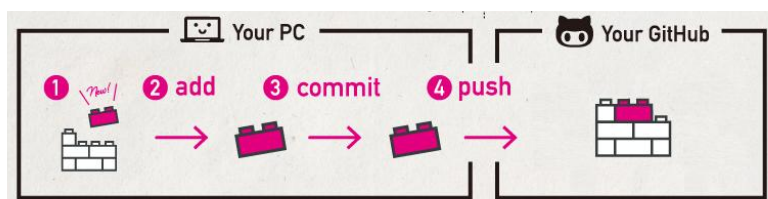
Pauta:

1. Crearse una cuenta en **GitHub**: <https://github.com>.
2. Descargar **GitHub Desktop** (Windows / Mac).
3. Crear un repositorio privado en **GitHub**.
4. Clonar dicho repositorio. Se puede hacer haciendo click en “Open in Desktop”.



Esto hace una copia del código del proyecto en su computadora (repositorio local). GitHub Desktop les preguntará dónde quieren crear la carpeta del proyecto dentro de su equipo.

5. Crear algún archivo (ej: un `index.html`) dentro de la carpeta del proyecto.
6. Hacer `commit` del archivo.
7. Hacer `push` de los cambios.



8. Ver el repositorio en GitHub (en la web).
9. Realizar modificaciones al archivo.
10. Hacer `commit` y `push` nuevamente.
11. Ver los cambios en GitHub.

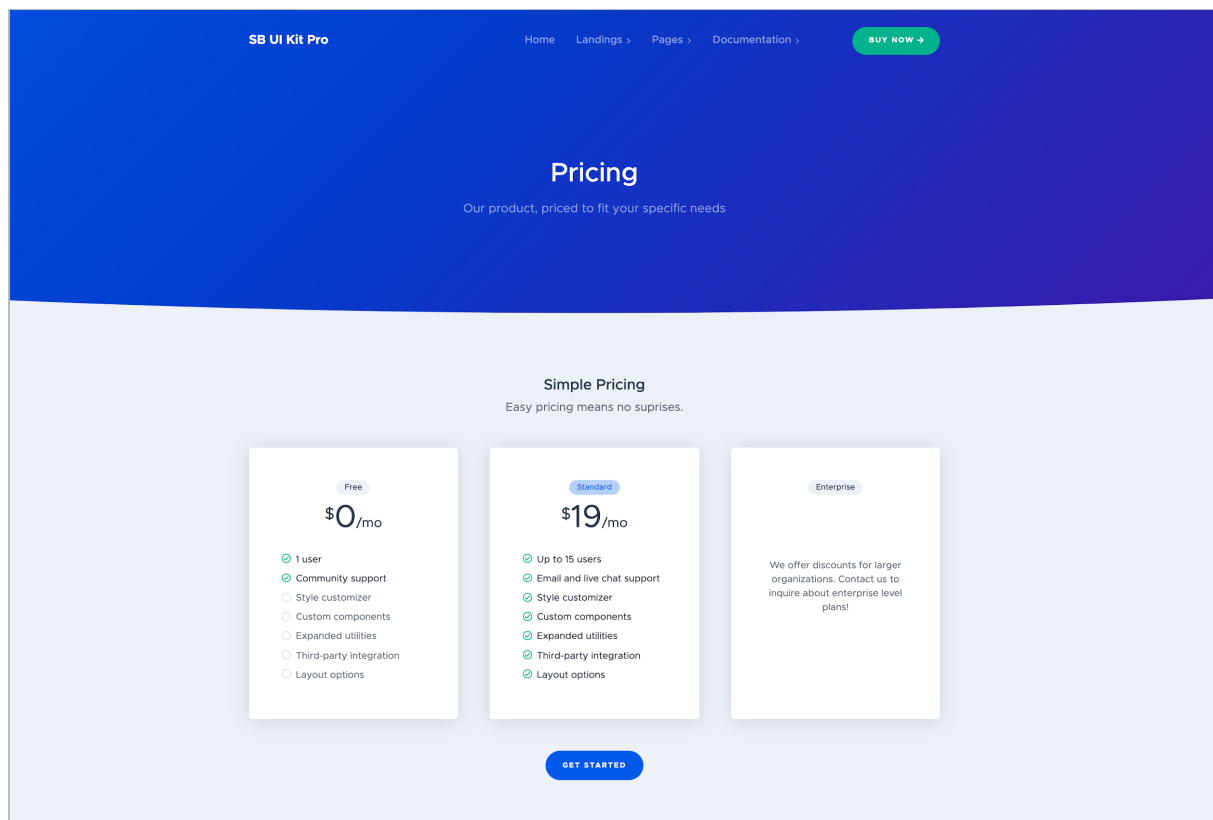
Ejercicio 8

Pauta:

- Darle acceso a su **repositorio** (el que crearon en el ejercicio anterior) a un compañero de la clase. Darle permisos de escritura (no solamente de lectura).
- A su vez, otro compañero les dará acceso a ustedes.
- La idea es que se hagan cambios cruzados en sus proyectos.
- ¿Qué pasa si dos personas hacen cambios sobre el mismo archivo? ¿Cómo se soluciona? ¡Pruébenlo!

Ejercicio 9

Clase previa: “Git”.



Pauta:

- Este es un ejercicio similar al Ejercicio 1 y Ejercicio 3. Se deberá maquetar la siguiente [página](#).
- Se deberá trabajar en **equipos** de 2 o 3 alumnos (armados por el docente). Por lo tanto deberán **dividirse las tareas**, elegir canales de **comunicación** y trabajar con **GitHub** para que el proceso sea mucho más fluido.
- Deberán crear un **repositorio privado** en GitHub, al cual sólo tendrán acceso los alumnos del equipo y, eventualmente, uno o más docentes.
- La idea es que al finalizar el ejercicio, cada alumno deberá tener una copia idéntica de la solución.
- ⚠ Intentar no copiar el código fuente (del link anterior).

Ejercicio 10

Para este curso es fundamental tener una mejor noción de cómo funciona Internet. Por lo tanto les recomendamos darle una leída a los siguientes links:

- [How the Internet Works in 5 Minutes](#) [Video]
- [A Packet's Tale. How Does the Internet Work?](#) [Video]
- [MDN – How does the Internet work?](#)
- [MDN – What is the difference between webpage, website, web server, and search engine?](#)
- En un sitio web es posible determinar los datos de un visitante. Prueben [este link](#).

Notas importantes sobre los ejercicios de JavaScript

1. Es fundamental **leer bien la letra** de cada ejercicio. Se dice que entender bien lo que se pide, es la **mitad de la solución**. Así que no se apuren en empezar a escribir código. Tómense su tiempo.
2. Algunos ejercicios son como una especie de **acertijo** y su resolución puede requerir **varias horas** de trabajo (si bien la cantidad de código final puede ser relativamente pequeña).
3. A veces, puede ser útil agarrar una hoja de **papel** y un **lápiz**, y empezar a “dibujar” soluciones. Hacer un diagrama puede ayudar mucho a entender el problema e idear una solución.
4. Deberán **probar** sus códigos de forma **exhaustiva**. Deberán hacer un buen **testing** de las funciones que escriban. Deberán verificar que realmente hagan lo solicitado. Desde la academia les daremos algunos **casos de prueba**, pero pueden no estar completos.
5. Los **ejercicios no son una competencia** para ver quién termina primero. En el pasado hemos tenido alumnos que han llegado a soluciones de forma muy rápida pero de manera muy rebuscada (excesivamente compleja), difícil de entender y a veces incluso muy poco *performante*. La **calidad del código** es igual de importante que llegar a la solución correcta. Entre otras cosas, esto implica que el código debe estar ordenado y prolijo; debe ser fácil de entender. Se deben usar nombres de variables y funciones que sean mnemotécnicos (descriptivos). En general: *“menos es más”*.
6. Resolver un ejercicio con ayuda directa de un familiar, amigo, colega, etc, suele carecer de sentido. Nunca pierdan de vista que **lo más importante** del Bootcamp es el **aprendizaje** de los alumnos y su preparación para el mundo laboral. No se trata de resolver los ejercicios “a cualquier precio”.

Ejercicio 11

Crear una función en JavaScript llamada `removeVocales` que reciba como parámetro un *string* y retorne un nuevo *string* que sea igual que el recibido pero sin las vocales.

Ejemplos:

Input	Output
<code>removeVocales("Hola")</code>	<code>"Hl "</code>
<code>removeVocales("Hola Mundo")</code>	<code>"Hl Mnd"</code>
<code>removeVocales("Hola URUGUAY")</code>	<code>"Hl RGY"</code>

Ejercicio 12

Crear una función en JavaScript llamada `sumarMultiplos` que reciba como parámetro un número natural (entero positivo) y retorne la suma de todos los números múltiplos de 3 o 5, que sean menores al número recibido por parámetro.

Si un número fuese múltiplo de 3 y 5, se considerará una sola vez.

Ejemplos:

Input	Output
<code>sumarMultiplos(1)</code>	0
<code>sumarMultiplos(2)</code>	0
<code>sumarMultiplos(3)</code>	0
<code>sumarMultiplos(6)</code>	8
<code>sumarMultiplos(10)</code>	23

Ejercicio 13

Crear una función en JavaScript llamada `encontrarImpar` que reciba como parámetro un *array* de números enteros (positivos y/o negativos) y retorna el número del *array* que aparezca un número impar de veces.

Siempre se recibirá un *array* con un sólo número con esas características.

Ejemplos:

Input	Output
<code>encontrarImpar([8])</code>	8
<code>encontrarImpar([2,2,2,2,8,2,2])</code>	8
<code>encontrarImpar([20,1,-1,2,-2,3,3,5,5,1,2,4,20,4,-1,-2,5])</code>	5

Ejercicio 14

Crear una función llamada `caminoOptimo` que recibe como parámetro un *array* de *strings*, que pueden ser: "NORTE", "SUR", "ESTE" y "OESTE". Este *array* indica el camino que debe seguir un viajero para llegar a determinado destino. El problema es que el camino propuesto por el *array* no es óptimo.

Por ejemplo, si el *array* de entrada es ["NORTE", "SUR"] el camino no es óptimo, ya que antes que avanzar y retroceder al mismo lugar, es preferible quedarse quieto. Por lo tanto, la función en este caso debería retornar un *array* vacío [].

Considerar que "NORTE" y "SUR" son opuestos, al igual que "ESTE" y "OESTE", y se cancelan si están uno inmediatamente después del otro.

Ejemplos:

Input	Output
<code>caminoOptimo(["NORTE", "SUR"])</code>	<code>[]</code>
<code>caminoOptimo(["NORTE", "SUR", "SUR"])</code>	<code>["SUR"]</code>
<code>caminoOptimo(["NORTE", "SUR", "SUR", "NORTE"])</code>	<code>[]</code>
<code>caminoOptimo(["NORTE", "SUR", "SUR", "SUR"])</code>	<code>["SUR", "SUR"]</code>
<code>caminoOptimo(["NORTE", "SUR", "SUR", "ESTE", "OESTE", "NORTE", "OESTE"])</code>	<code>["OESTE"]</code>
<code>caminoOptimo(["NORTE", "OESTE", "SUR", "ESTE"])</code>	<code>["NORTE", "OESTE", "SUR", "ESTE"]</code>

Ejercicio 15

Crear una función en JavaScript llamada `parentesisCorrectos` que reciba como parámetro un *string* compuesto por paréntesis curvos "(" y/o ")", y retorne `true` en caso de que los paréntesis estén ordenados de la forma correcta (que se abran y cierren de forma consistente). En caso de no estarlo, la función deberá retornar `false`.

Ejemplos:

Input	Output
<code>parentesisCorrectos("()")</code>	<code>true</code>
<code>parentesisCorrectos(")(")</code>	<code>false</code>
<code>parentesisCorrectos("()())")</code>	<code>false</code>
<code>parentesisCorrectos("(")</code>	<code>false</code>
<code>parentesisCorrectos("(()((()())())")</code>	<code>true</code>

Ejercicio 16

Crear una función en JavaScript llamada `encriptar13` que reciba como parámetro un *string* y retorne como resultado un nuevo *string* pero “encriptado”.

El nuevo *string* deberá tener las letras “corridas” 13 lugares. Por ejemplo, a la letra “A” le corresponderá la “N”, así como a la “a” le corresponderá la “n”. A la letra “T” le corresponderá la “G” y a la letra “t” la “g”. Es decir, las minúsculas se transforman en otra letra minúscula y las mayúsculas también se transforman en mayúscula.

Sólo se deberán transformar (“correr”) letras del alfabeto inglés. Cualquier otro carácter deberá quedar incambiado.

Ejemplos:

Input	Output
<code>encriptar13("hola")</code>	<code>"ubyn"</code>
<code>encriptar13("CHAU")</code>	<code>"PUNH"</code>
<code>encriptar13("Título")</code>	<code>"Gíghyb"</code>
<code>encriptar13("HACK academy 2021")</code>	<code>"UNPX npnqrz1 2021"</code>

Ejercicio 17

Vamos a crear un pequeño juego, versión de “Piedra, Papel o Tijera”, para jugar contra la computadora.

Para eso deberán crear una función llamada `jugar` que recibe como parámetro un *string* que puede ser “Piedra”, “Papel” o “Tijera”.

La función deberá definir qué movimiento hizo la computadora y en base a eso determinar un ganador.

Ejemplos:

Input	Output
<code>jugar("Piedra")</code>	"La computadora eligió Papel. Perdiste."
<code>jugar("Piedra")</code>	"La computadora eligió Tijeras. Ganaste."
<code>jugar("Papel")</code>	"La computadora eligió Papel. Empataron."
<code>jugar("Papel")</code>	"La computadora eligió Tijeras. Perdiste."

👉 Al finalizar, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 18

Crear una función llamada `duracionParaHumanos` que reciba como parámetro un número entero (representando una cantidad de segundos) y retorne un *string* con un texto que indique la cantidad de tiempo que transcurrió, pero con un formato fácil de leer por humanos.

Para este ejercicio, un año está compuesto por 365 días.

Ejemplos:

Input	Output
<code>duracionParaHumanos(0)</code>	"ahora"
<code>duracionParaHumanos(62)</code>	"1 minuto y 2 segundos"
<code>duracionParaHumanos(3662)</code>	"1 hora, 1 minuto y 2 segundos"
<code>duracionParaHumanos(43424234)</code>	"1 año, 137 días, 14 horas, 17 minutos y 14 segundos"
<code>duracionParaHumanos(4342440)</code>	"50 días, 6 horas y 14 minutos"

Notar que no son correctos formatos como:

- "14 horas y 2 años" (orden incorrecto, debería ser "2 años y 14 horas").
- "40 horas, 20 horas y 3 minutos" (la componente de horas debería aparecer una sola vez, por ejemplo: "2 días, 12 horas y 3 minutos").
- "14 horas, 2 minutos" (debería ser "14 horas y 2 minutos").
- "68 minutos" (debería ser "1 hora y 8 minutos").

👉 Al finalizar, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 19

En este ejercicio vamos a crear una batería (el instrumento musical).

Pauta:

- Crear una página web que tenga 7 botones, los cuales estarán asociados a 7 sonidos distintos de batería.
- Cada botón debe estar asociado a una tecla del teclado y a un sonido. Además, cada botón tendrá su propia imagen (correspondiente a la componente de la batería que produce dicho sonido).

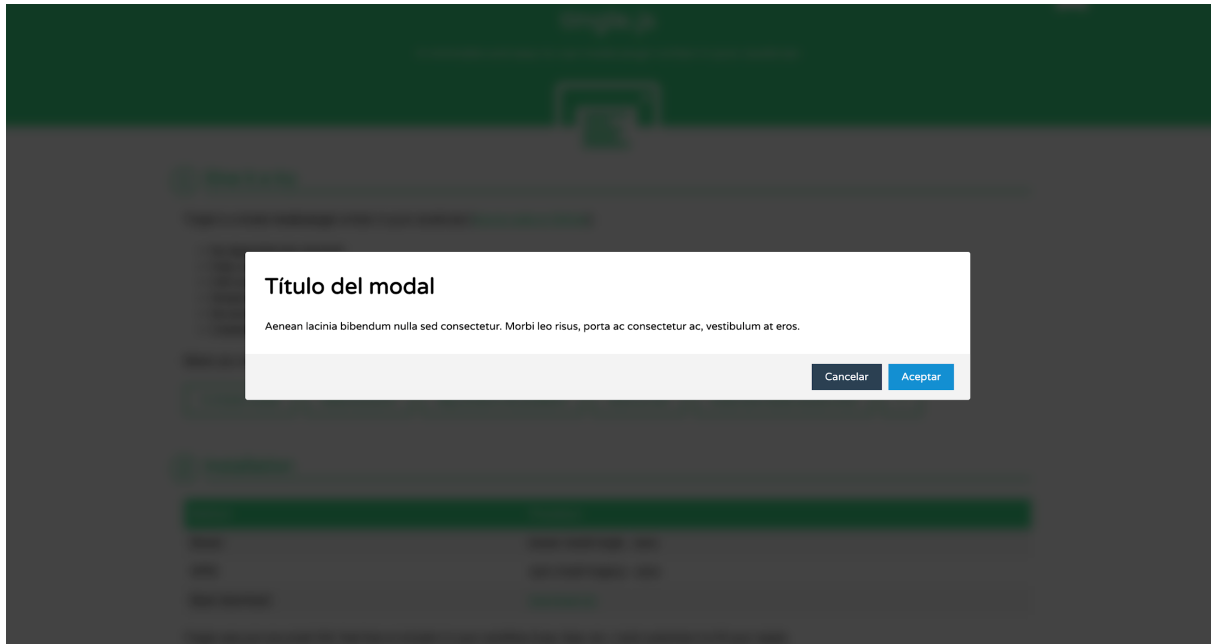
Botón	Tecla	Sonido
1	A	Tom 1
2	S	Tom 2
3	D	Tom 3
4	F	Tom 4
5	J	Snare
6	K	Crash
7	L	Kick

- Al hacer click sobre un botón o al presionar la letra correspondiente, se deberá emitir un sonido (ej: un platillo). Ver [documentación en MDN](#).
- Los 7 sonidos (mp3) e imágenes que precisan los podrán descargar de [este link](#).

👉 Al finalizar, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 20

Crear un **modal** para una página web, sin usar algo pronto como los modales que trae Bootstrap. Se puede usar jQuery.



Pauta:

- Al hacer click sobre algún elemento de la página (ej: un botón), hacer aparecer un modal (similar al de la imagen anterior).
- El modal debe aparecer centrado en la ventana, tanto vertical como horizontalmente.
- El fondo de la página se debe oscurecer y tener un aspecto difuminado (*blur*).
- Para quitar el modal el usuario podrá hacer click en alguno de los dos botones o en la zona oscura de la página.
- Verificar que el modal se vea bien en pantallas pequeñas.

Ejercicio 21

Crear una página web que contenga una [tabla HTML](#), similar al siguiente ejemplo:

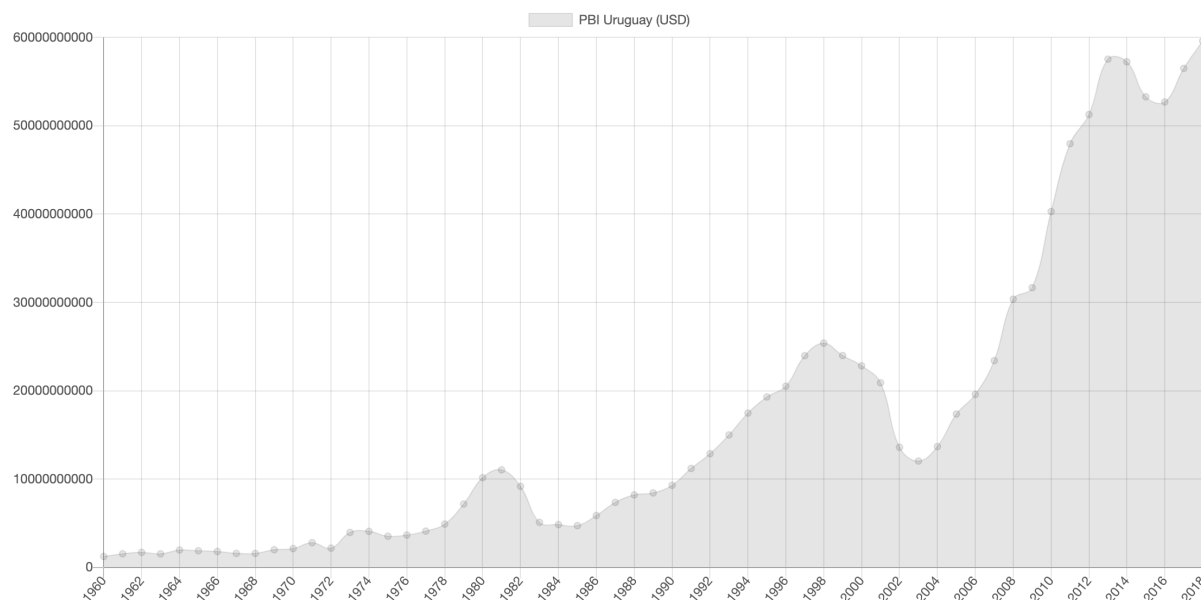
ID	Nombre	Apellido	Edad
1	María	López	34
2	Victoria	Rodríguez	12
3	José	González	75
4	Francisco	Rodríguez	50
5	Juan	Fernández	11
6	Federico	Domínguez	18
7	Martín	López	22
8	Andrea	Hernández	32
9	Paula	Gómez	27
10	Manuel	Jiménez	46

Pauta:

- La tabla debe ser creada utilizando la etiqueta html `<table>`.
- Arriba de la tabla debe aparecer un campo de texto `<input>`.
- A medida que el usuario escribe el campo de texto, la tabla se debe “acomodar” y sólo mostrar las filas que contienen el texto buscado.
- En caso de buscar un texto que no tenga ocurrencias en la tabla, se debe mostrar la tabla vacía y mostrar un mensaje de error.
- Los valores de la tabla se deben popular de [este JSON](#), haciendo una llamada AJAX.
- Inicialmente, la tabla debe mostrar todos los datos.
- No deberán usar soluciones prontas, pero pueden inspirarse visualmente en plugins como [DataTables](#).

Ejercicio 22

Crear una gráfica utilizando un plugin llamado **Chart.js**. Vamos a mostrar la evolución del PBI de Uruguay desde 1960 a 2018.



Es necesario linkear el siguiente script:

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.3.2/chart.min.js"></script>
```

Los datos se deben obtener de [este JSON](#), haciendo una llamada AJAX.

👉 Al finalizar, pedirle a un **compañero** (alguien con el que no hayan trabajado antes) que les **corrija/revise** el código y viceversa. Además de que el código funcione, verificar que esté prolijo y entendible.

Ejercicio 23

Construir su **sitio web personal**. Sería como una especie de CV online.

Pueden buscar inspiración en sitios como:

- <https://mattfarley.ca>.
- <https://pierre.io>.
- <https://timmyomahony.com>.
- <https://caferati.me>.
- <https://jonny.me>.



Un alumno (elegido al azar) deberá mostrarle su solución a toda la clase.

Deberán responder preguntas como:

- ¿Qué tecnologías utilizaron? ¿Algún plugin?
- ¿Qué dificultades tuvieron?
- ¿Qué mejoras harían?



El sitio se deberá subir al hosting [Netlify](#), pero previamente deberán subir su código a GitHub. La idea es que vinculen su cuenta de Netlify con su cuenta de GitHub, y que las mismas queden “sincronizadas”. La idea es que cada vez que hagan un cambio en su repositorio de GitHub, el sitio web (alojado en Netlify) se actualice automáticamente. Por más información, leer la [documentación oficial](#).

Ejercicio 24 (Extra)

Crear una función en JavaScript llamada `validarSudoku` que reciba como parámetro una matriz 9x9 conteniendo dígitos entre 1 y 9. La función debe retornar `true` o `false`, dependiendo de si la matriz corresponde a un tablero resuelto de Sudoku válido o no.

Si nunca jugaron al Sudoku, ingresar [aquí](#) para leer las reglas del juego.

En programación, una matriz se suele representar como un array de arrays. Por lo tanto, para crear una matriz 9x9 se necesita un array de 9 elementos, donde cada uno de esos elementos es a su vez un array de 9 elementos.

Ejemplo de una matriz 9x9:

```
var matriz = [  
  [5, 3, 4, 6, 7, 8, 9, 1, 2],  
  [6, 7, 2, 1, 9, 5, 3, 4, 8],  
  [1, 9, 8, 3, 4, 2, 5, 6, 7],  
  [8, 5, 9, 7, 6, 1, 4, 2, 3],  
  [4, 2, 6, 8, 5, 3, 7, 9, 1],  
  [7, 1, 3, 9, 2, 4, 8, 5, 6],  
  [9, 6, 1, 5, 3, 7, 2, 8, 4],  
  [2, 8, 7, 4, 1, 9, 6, 3, 5],  
  [3, 4, 5, 2, 8, 6, 1, 7, 9]  
];
```

Para acceder a un elemento de la matriz podemos escribir:

`matriz[4][3]` que en este ejemplo corresponde a **8**.

Si a la función `validarSudoku` le pasamos la matriz anterior, debería retornar `true`, ya que la misma corresponde con una solución de Sudoku válida.

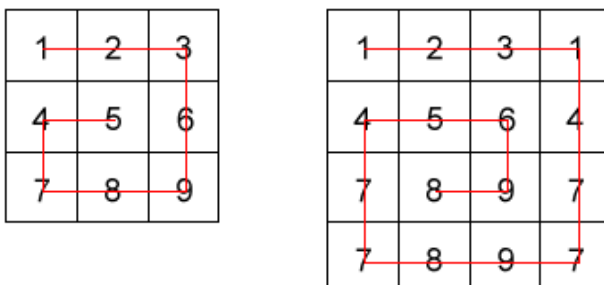
```
validarSudoku([  
    [5, 3, 4, 6, 7, 8, 9, 1, 2],  
    [6, 7, 2, 1, 9, 5, 3, 4, 8],  
    [1, 9, 8, 3, 4, 2, 5, 6, 7],  
    [8, 5, 9, 7, 6, 1, 4, 2, 3],  
    [4, 2, 6, 8, 5, 3, 7, 9, 1],  
    [7, 1, 3, 9, 2, 4, 8, 5, 6],  
    [9, 6, 1, 5, 3, 7, 2, 8, 4],  
    [2, 8, 7, 4, 1, 9, 6, 3, 5],  
    [3, 4, 5, 2, 8, 6, 1, 7, 9]  
]); // Debería retornar true.
```

Ejercicio 25 (Extra)

Crear una función en JavaScript llamada `linealizarEspiral` que reciba como parámetro una matriz NxN conteniendo dígitos entre 1 y 9.

La función debe retornar un nuevo array de tamaño $1 \times N^2$ conteniendo los mismos elementos del array original. Por eso decimos que esta función “linealiza” a la matriz.

La linealización se debe hacer de tal forma que los elementos de la matriz se recorran en formato espiral. Ver ejemplos:



Supongamos que la matriz de entrada es de tamaño 3x3:

```
var arrayLineal = linealizarEspiral([  
  [1, 2, 3],  
  [4, 5, 6],  
  [7, 8, 9]  
]);
```

En este caso, la solución sería:

```
[1, 2, 3, 6, 9, 8, 7, 4, 5]
```

Nota: Una matriz vacía (de tamaño 0x0) se representa así: `[]`

Ejercicio 26 (Extra)

Crear una función en JavaScript llamada `sumarDigitos` que reciba como parámetro un número entero y retorne como resultado otro número entero que se calcula sumando los dígitos del número recibido.

Si el resultado de la suma es un número de más de un dígito, se deberá repetir la suma la cantidad de veces que sea necesaria hasta obtener un número de un sólo dígito.

Ejemplos:

Input	Output
<code>sumarDigitos(0)</code>	0
<code>sumarDigitos(4)</code>	4
<code>sumarDigitos(62)</code>	8
<code>sumarDigitos(942)</code>	6
<code>sumarDigitos(132189)</code>	6
<code>sumarDigitos(493193)</code>	2

Detalle de los ejemplos:

- $0 \rightarrow 0$
- $4 \rightarrow 4$
- $62 \rightarrow 6 + 2 = 8$
- $942 \rightarrow 9 + 4 + 2 = 15 \rightarrow 1 + 5 = 6$
- $132189 \rightarrow 1 + 3 + 2 + 1 + 8 + 9 = 24 \rightarrow 2 + 4 = 6$
- $493193 \rightarrow 4 + 9 + 3 + 1 + 9 + 3 = 29 \rightarrow 2 + 9 = 11 \rightarrow 1 + 1 = 2$

Ejercicio 27 (Extra)

En matemáticas, la sucesión o serie de **Fibonacci** es la siguiente sucesión infinita de números naturales:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597,...

La serie comienza con 0 y 1, y luego cada uno de los siguientes números se calcula sumando los dos anteriores.

Por ejemplo, 21 es resultado de sumar 8 y 13. El número 34 es resultado de sumar 13 y 21.

En este ejercicio se deberá crear una función en JavaScript llamada `serieFibonacci` que reciba como parámetro un número natural, mayor que 0, y retorne como resultado un *string* conteniendo una serie de Fibonacci con dicha cantidad de elementos.

Ejemplos:

Input	Output
<code>serieFibonacci(1)</code>	"0"
<code>serieFibonacci(2)</code>	"0, 1"
<code>serieFibonacci(3)</code>	"0, 1, 1"
<code>serieFibonacci(4)</code>	"0, 1, 1, 2"
<code>serieFibonacci(5)</code>	"0, 1, 1, 2, 3"
<code>serieFibonacci(6)</code>	"0, 1, 1, 2, 3, 5"
<code>serieFibonacci(7)</code>	"0, 1, 1, 2, 3, 5, 8"
<code>serieFibonacci(8)</code>	"0, 1, 1, 2, 3, 5, 8, 13"

Ejercicio 28 (Extra)

Crear una función en JavaScript llamada `numeroFibonacci` que reciba como parámetro un número natural, mayor que 0, y retorne como resultado el número que ocupa la posición indicada por el parámetro en la serie de **Fibonacci**.

Ejemplos:

Input	Output
<code>numeroFibonacci(1)</code>	0
<code>numeroFibonacci(2)</code>	1
<code>numeroFibonacci(3)</code>	1
<code>numeroFibonacci(4)</code>	2
<code>numeroFibonacci(5)</code>	3
<code>numeroFibonacci(6)</code>	5
<code>numeroFibonacci(7)</code>	8
<code>numeroFibonacci(30)</code>	514229

👉 ¿Qué pasa si se llama a la función con un número muy grande? Ejemplo:

`numeroFibonacci(500)`.

Ejercicio 29 (Extra)

Crear una función en JavaScript llamada `sumarStrings` que reciba como parámetros dos *strings* (que representan dos números naturales) y retorna un nuevo *string* que representa la suma de los números.

La dificultad de este ejercicio está en poder sumar números realmente grandes y poder mostrar la suma completa (con todos los dígitos). Ver detalles [aquí](#).

Ejemplos:

Input	Output
<code>sumarStrings("1", "6")</code>	<code>"7"</code>
<code>sumarStrings("0", "4")</code>	<code>"4"</code>
<code>sumarStrings("0034", "000056")</code>	<code>"90"</code>
<code>sumarStrings("73283718237137219313432", "8934342432")</code>	<code>"73283718237146153655864"</code>