*Prueba de nivel de programación.*

HTML, CSS y Bootstrap

Preguntas test

**1.6. ¿Cuál tag HTML se utiliza generalmente para marcar un texto de una página web como el más grande/importante**?

**1.7. ¿Qué propiedad determina el espacio entre un contenedor y los elementos que contiene?**

**1.8. ¿Qué tag HTML se utiliza para crear una lista punteada (bullet list)?**

**1.9. ¿Cuál es la propiedad para seleccionar el color de fondo de una imagen con fondo transparente?**

**1. 10. ¿En CSS, que propiedad indica el espacio entre líneas de texto?**

**1.11. ¿En bootstrap, qué clase proporciona un contenedor de ancho fijo?**

1. container
2. container-fluid
3. container-fixed

**1.12. ¿En cuántas columnas está basado el grid de bootstrap?**

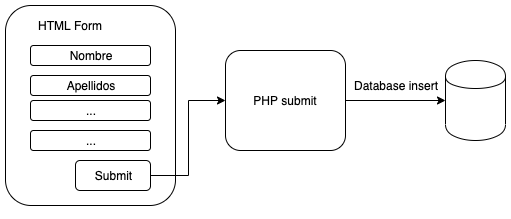
Maquetación de componentes

Se requiere el código responsive que responda el siguiente componente de diseño.



Desarrollo de código

**2. Creación y submit de un formulario.**

****

Necesitamos un formulario para registrar usuarios en una página web. El usuario introduce: Nombre, Apellidos, E-mail, Teléfono, Código postal y debe aceptar las políticas de privacidad (checkbox). Todos los campos son obligatorios.

**2.1. Desarrollar el código html de la página con el formulario POST mediante el cual el usuario se va a registrar**

**2.2. Desarrollar el código PHP del submit del formulario en el que:**

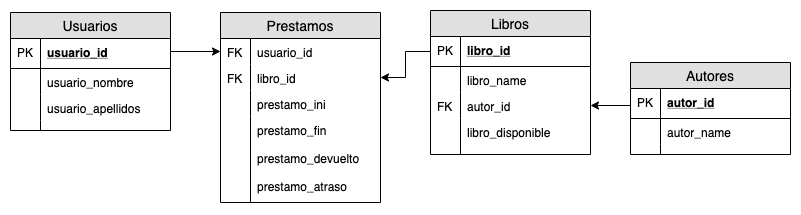
* Se lean todos los campos del formulario.
* Se compruebe que la política de privacidad está aceptada.
* No haya campos vacíos y e-mail, teléfono y código postal (español) sean válidos en cuanto a formato.
* Se deduzca la provincia del usuario (mediante su código postal).
* Se compruebe que un usuario con ese mismo e-mail no se ha registrado en las últimas 24 horas.
* Si todos los datos y las comprobaciones mencionadas son correctos, se ha de insertar toda la información del usuario en base de datos.

Para este apartado contamos con funciones de PHP ya dadas (funciones que no hay que desarrollar, sólo usar como auxiliares):

* *getProvince()*: Dado el código de provincia (dos primeras cifras del código postal), devuelve un string con el nombre de la provincia.
* *check24h():* Dada una dirección de e-mail, comprueba que no se haya insertado en base de datos en las últimas 24 horas (devuelve true si no se ha insertado, false si existe).
* *insertLead()*: Inserta todos los datos del usuario en base de datos (orden de datos irrelevante).

Base de datos (SQL)

Sistema de catálogo y préstamos de una biblioteca.



**PK**: Primary key **FK**: Foreign key **Negrita**: Usado para ids: PK, integer, unique, not null, autoincrement.

Tenemos una base de datos con:

* **Usuarios:** Tabla que contiene todos los usuarios de la biblioteca identificados por su **id**. Además tiene campos varchar para nombre y apellidos del usuario.
* **Autores:** Similar a Usuarios.
* **Libros:** Tabla con todos los libros de la biblioteca. Contiene la **id** del libro, el nombre (varchar), la disponibilidad del libro (bool: 1 si no está en préstamo o 0 si lo está) y la id del autor del libro como FK.
* **Préstamos:** Tabla sin PK. Log de todos los préstamos efectuados. Contiene como FK la id del usuario que ha cogido el libro en préstamo y la id del libro en préstamo. Además contiene el período de tiempo en que se presta el libro (prestamo\_ini y prestamo\_fin son datetime que se generan en el momento del préstamo), el estado actual del préstamo (prestamo\_devuelto: 0 si no, 1 si ha sido devuelto) y el tiempo de atraso de la devolución (si un libro no se ha devuelto en el momento del vencimiento del préstamo, por cada día prestamo\_atraso (integer), inicializado a 0, suma 1).

3. Desarrollar las siguientes consultas con la base de datos dada:

**3.1.** Una consulta que devuelva el nombre del libro y del autor de los 10 libros más prestados de la biblioteca en el año 2019 (un libro puede ser prestado n veces).

**3.2.** Una consulta que devuelva un ránking del número de libros publicados en el catálogo por cada uno de los autores (y también el nombre del autor).

**3.3.** Una consulta SQL que devuelva el nombre y los apellidos de los usuarios que actualmente tengan en préstamo (vencido o no) algún libro de un autor cuya id es AUTH\_ID.

**3.4.** Una consulta que devuelva el nombre, apellidos, nombre del libro, nombre del autor y días de atraso de los usuarios que tienen libros en préstamo y no los han devuelto (un usuario puede tener n libros en préstamo a la vez) ordenados por id de usuario.

**3.5.** Una consulta que devuelva el nombre del libro y del autor de todos los libros que están en préstamo actualmente (vencidos o no).

4. Este ejercicio solo tiene la intención de medir tú lógica para resolver un problema. Puedes realizarlo con un diagrama UML o con un script en PHP o JS.  
El script debe resolver lo indicado en la sección **Output**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
   
Blue y Orange son robots amigables. Una mente maestra malvada de la computadora los ha encerrado en pasillos separados para probarlos y luego posiblemente darles pastel.

Cada pasillo contiene 100 botones etiquetados con los números enteros positivos {1, 2, ..., 100}. El botón k siempre está a k metros del inicio del pasillo, y ambos robots comienzan en el botón 1. Durante un segundo, un robot puede caminar un metro en cualquier dirección, o puede presionar el botón en su posición una vez, o puede permanecer en su posición y no presionar el botón. Para completar la prueba, los robots deben presionar una determinada secuencia de botones en un orden determinado. Ambos robots conocen la secuencia completa de antemano. ¿Qué tan rápido pueden completarlo?

Por ejemplo, consideremos la siguiente secuencia de botones:

O 2, B 1, B 2, O 4

Aquí, O 2 significa el botón 2 en el pasillo de Orange, B 1 significa el botón 1 en el pasillo de Blue, y así sucesivamente. Los robots pueden presionar esta secuencia de botones en 6 segundos usando la estrategia que se muestra a continuación:  
  
 Time | Orange | Blue

----------+------------------------------------------+-----------------

1 | Move to button 2 | Stay at button 1

2 | Push button 2 | Stay at button 1

3 | Move to button 3 | Push button 1

4 | Move to button 4 | Move to button 2

5 | Stay at button 4 | Push button 2

6 | Push button 4 | Stay at button 2

Tenga en cuenta que Blue tiene que esperar hasta que Orange haya terminado de empujar por completo O 2 antes de poder empezar a empujar B 1.  
  
Input

La primera línea de la entrada da el número de casos de prueba, T . Siguen los casos de prueba T.

Cada caso de prueba consta de una sola línea que comienza con un número entero positivo N , que representa la cantidad de botones que deben presionarse. A esto le siguen N términos de la forma "Ri Pi" , donde Ri es un color de robot (siempre 'O' o 'B'),y Pi es una posición de botón.

Output

Para cada caso de prueba, genere una línea que contenga  
 **"Caso #x: y",**  
donde x es el número de caso (Iniciando desde 1) y es la cantidad mínima de segundos necesarios para que los robots presionen los botones dados, en orden.

Limites

1 ≤ P i ≤ 100 para todo i.

Conjunto de datos pequeño

1 ≤ T ≤ 20.

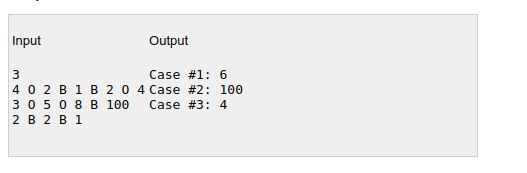
1 ≤ norte ≤ 10.

Conjunto de datos Grandes

1 ≤ V ≤ 100.

1 ≤ N ≤ 100.

Ejemplo



**Se debe enviar como respuestas a este test:  
- Un archivo zip con las respuestas de las preguntas teóricas,  
- Documento con código HTML, código PHP, código CSS y JS .  
- Archivo con la lógica desarrollada para resolver la pregunta 4.**