

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como, "Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar".

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un

tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como, "Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como, "Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica

Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como, "Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar".

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica

Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como,

"Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar

Construye tu Declaración de Perdón Ahora construyes una Declaración de Perdón mediante la combinación de las frases que has creado anteriormente. Ejemplo 1: Quiero perdonar a Janet. Ahora elijo liberar mis sentimientos de amargura y resentimiento. Reconozco que el perdón me beneficia ya que me voy a sentir más feliz, más sano y más pacífico. Me comprometo a perdonar a Janet y acepto la paz y la libertad que trae el perdón. Ejemplo 2: Quiero perdonar a mi padre por no amarme lo suficiente. Ahora elijo liberar mis sentimientos de enojo, desilusión y resentimiento. Reconozco que el perdón me beneficia ya que me sentiré libre, amoroso y más vivo. Me comprometo a perdonar a mi padre y yo acepto las maneras en que esto me permite ser más feliz y más amoroso hacia los demás. Práctica Elige cuánto tiempo vas a trabajar con los cuatro pasos (7 días, 21 días, etc.) y en qué momento(s) del día lo vas a usar. Hazlo al menos tres veces en cada sesión, escribe los pasos, si es posible, o dílos en voz alta o en silencio en tu mente en cada sesión. Conforme sigues los pasos puedes notar que tus sentimientos cambiarán (es decir, en el Paso 2 la ira cambia a frustración y así sucesivamente). Si esto sucede, simplemente cambia tu redacción para que coincida con tus sentimientos reales en el momento lo mejor que puedas. Tú puedes encontrar que después de pasar por todos los pasos un par de veces que tus sentimientos sobre el deseo de perdonar serán mucho más fuertes. Esta es una buena señal, especialmente si es porque estás empezando a ver todos los beneficios que vendrán a ti (y aquellos cercanos a ti) a medida que aprendas a perdonar. Después de un tiempo es posible que no sienta que es necesario hacer los cuatro pasos y puedes entonces sólo tener que utilizar la Declaración de Perdón del Paso Cuatro hasta que te sientas completo. Como parte de liberarte de viejos sentimientos, es posible que tengas que hacer otro tipo de liberación emocional, tales como hablar las cosas con un amigo o incluso ver a un terapeuta. A medida que trabajas con esto, puedes encontrar sentimientos inesperados y recuerdos olvidados hace mucho tiempo surgiendo. Sólo déjalos pasar o encuentra apoyo si lo necesitas. Si crees en un poder espiritual superior es simplemente natural que desees que eso sea parte de tu proceso de perdón. Sólo tienes que añadir una frase al final, como, "Pido la ayuda de Dios en el perdón y en ser libre", o "yo invito y acepto la gracia de Dios para ayudar a que me ayude a perdonar

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en

forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Los creadores de esta web entienden por dominio público “el conjunto de bienes y derechos de titularidad pública no poseídos en forma privativa”. Por ejemplo, el rubro cultural lo componen obras que pueden ser reproducidas por cualquiera, de manera gratuita o lucrativa sin tener obligación de pagar por ello. Esto puede ser por caducidad de los derechos de autor o por decisión de este.

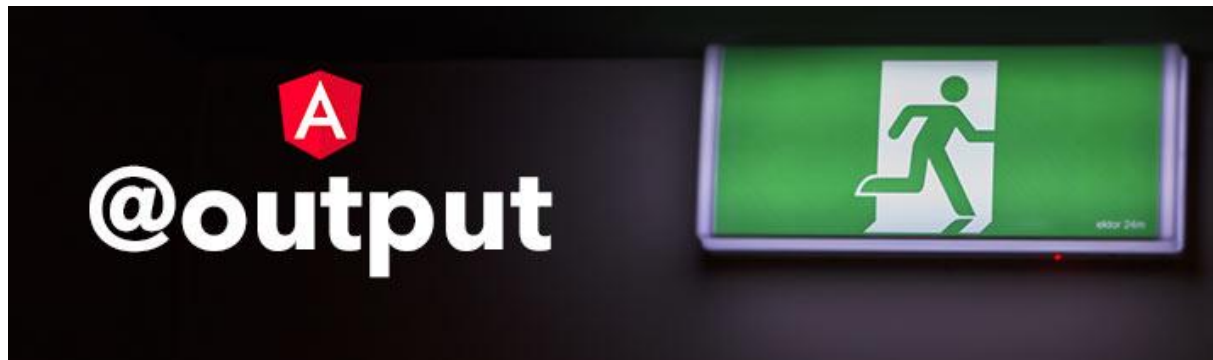
Este sitio nace con la idea de agrupar cuanta información sea posible sobre cualquier tema, y en la medida que el disco duro aguante, tener disponible en formato digital cuantas obras ya en dominio público se puedan albergar.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

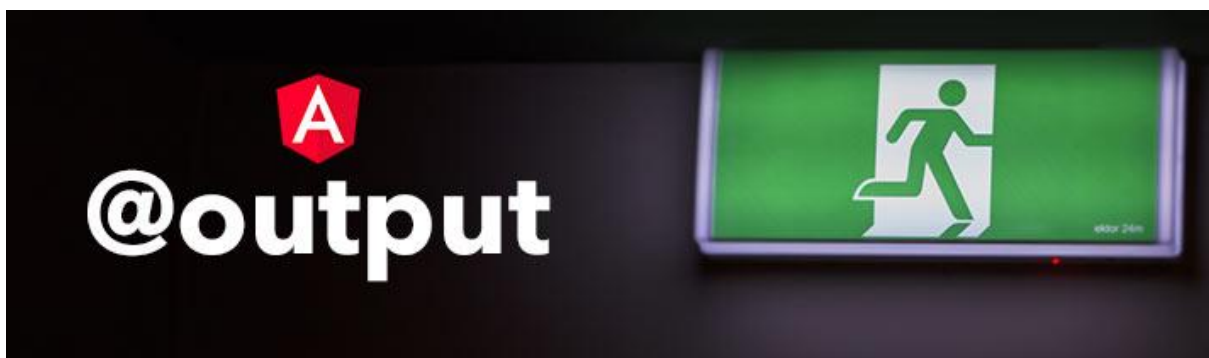
Explicamos la comunicación de cambios de los hijos hacia los

padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output ()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

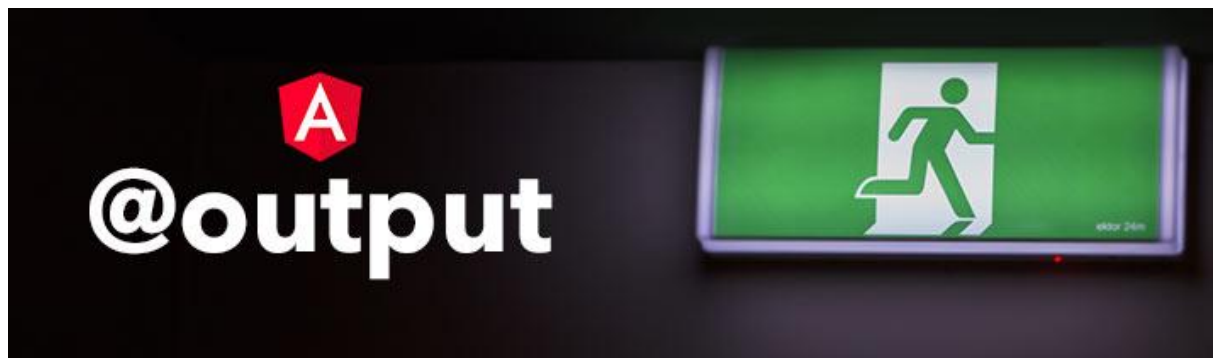
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar

un dato que el padre deba conocer, relacionado lógicamente con ese suceso.

- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar

un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output ()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

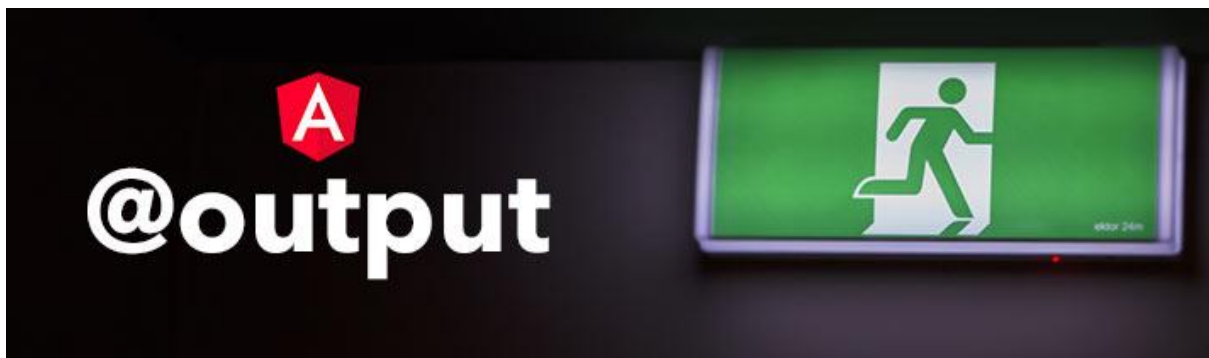
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su

comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.


```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador `@Output()` es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

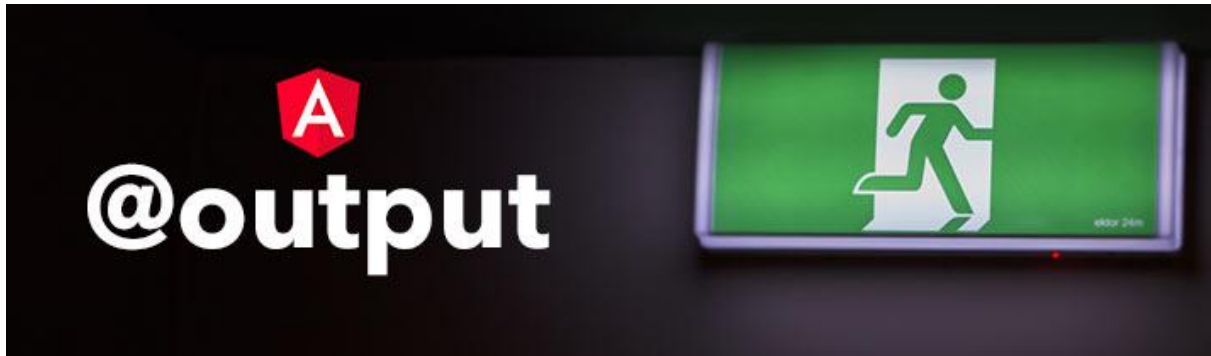
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output ()
```

```
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

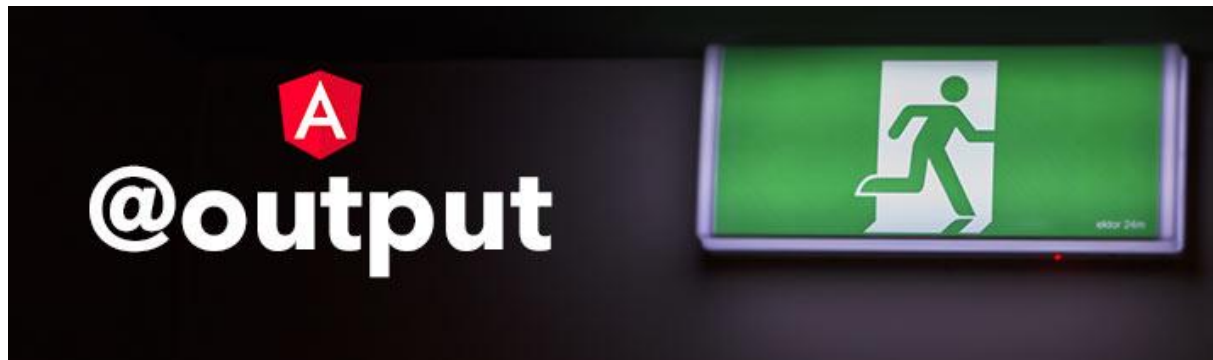
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

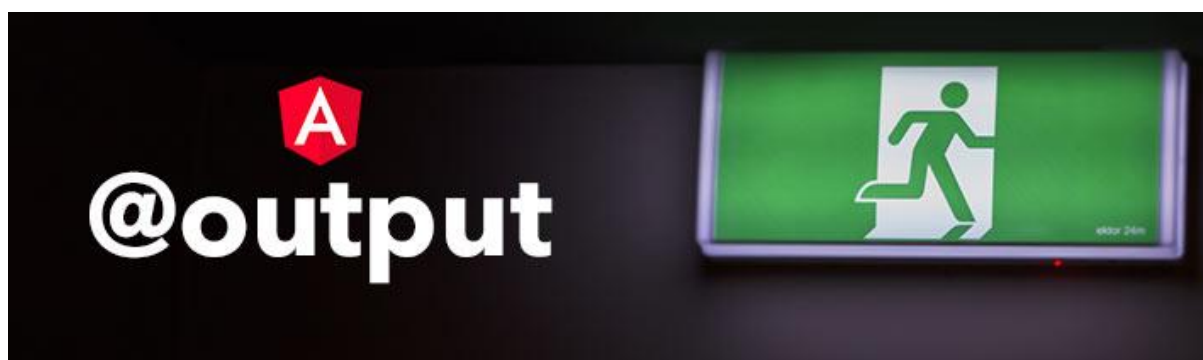
Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos

personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- El **componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- El **componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar

un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output ()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

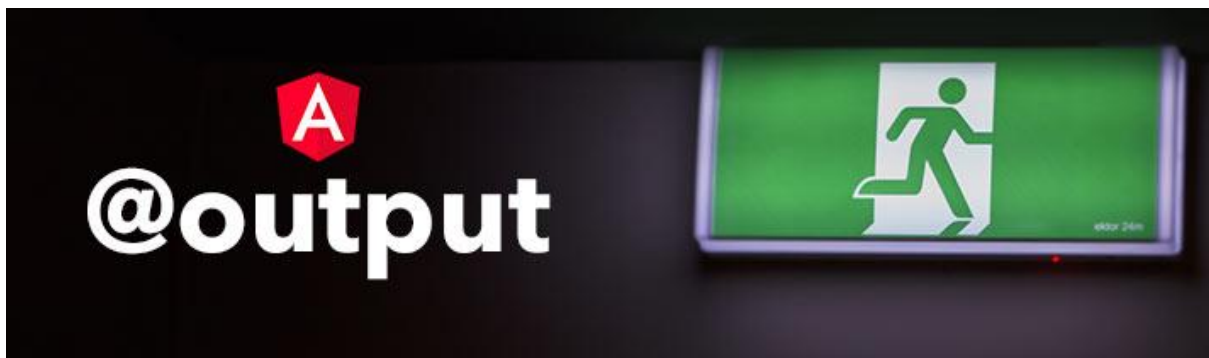
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su

comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador `@Output()` es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

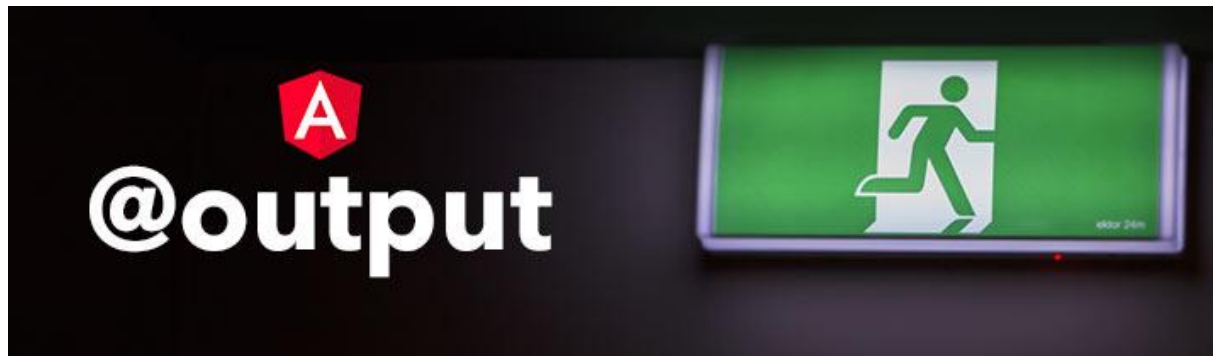
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:


```
@Output ()
```

```
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

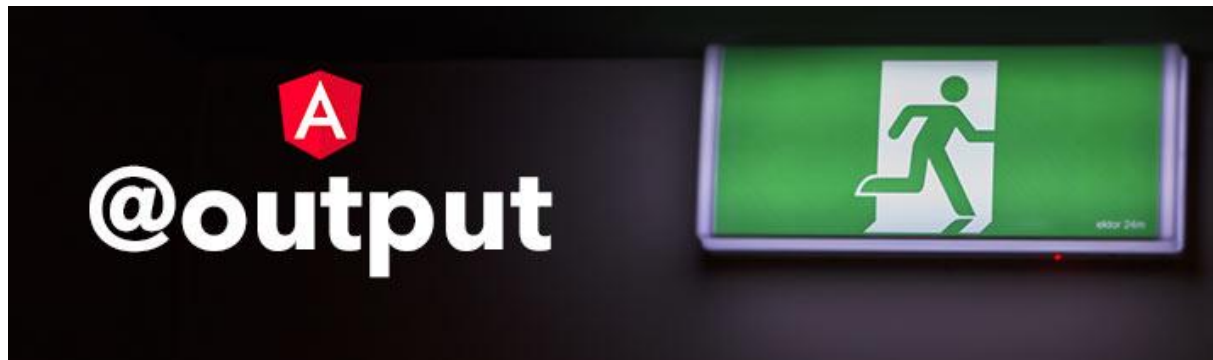
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

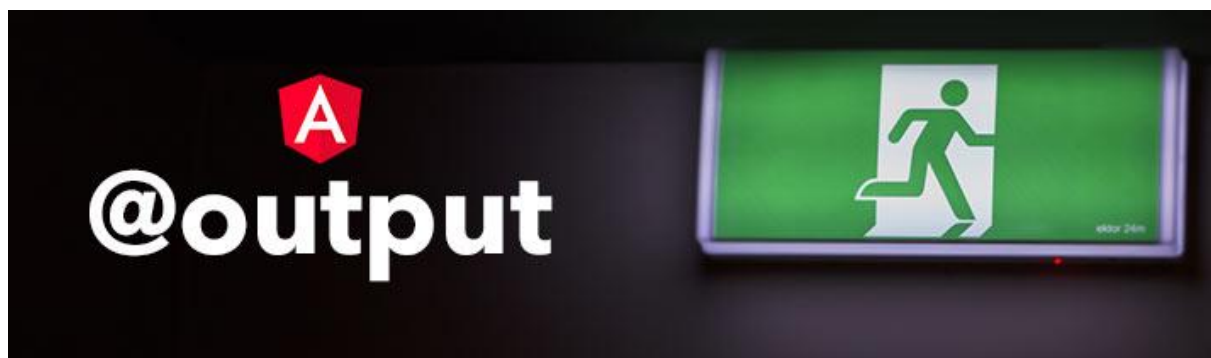
Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos

personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar

un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from '@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método emit() del objeto EventEmitter.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto EventEmitter.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades @Output.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

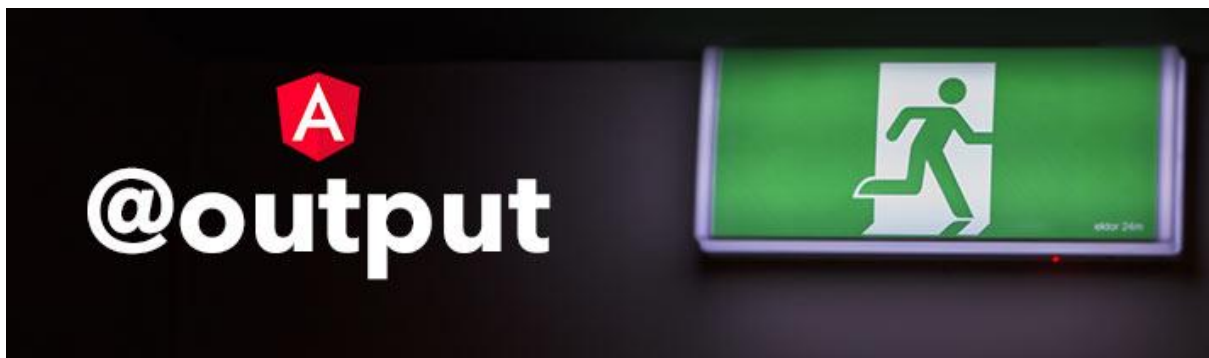
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su

comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador `@Output()` es el siguiente:

```
@Output()  
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

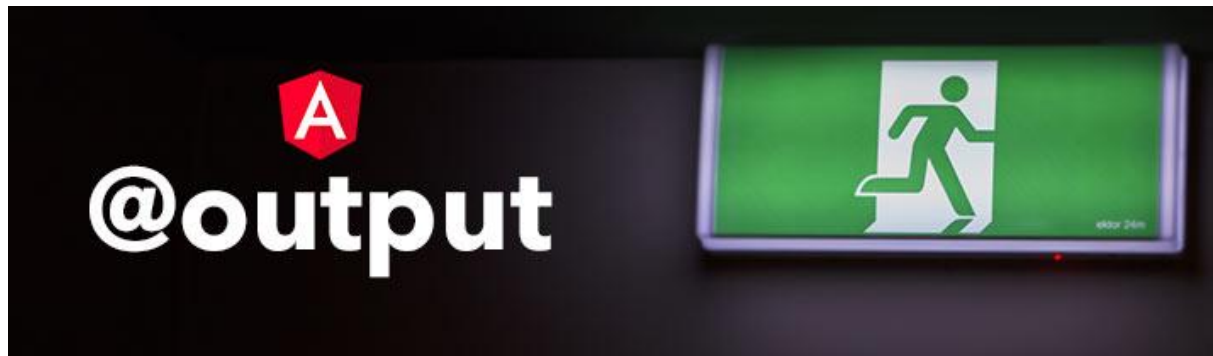
Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

Explicamos la comunicación de cambios de los hijos hacia los padres por medio de eventos personalizados, generados con propiedades @Output.

En el pasado artículo del [Manual de Angular](#) conocimos las [propiedades @Input](#), que permitían comunicar datos desde el componente padre hacia el componente hijo.

También comentamos que existe la posibilidad de implementar la otra dirección en la comunicación entre componentes: desde los hijos hacia los padres. Ésta se realiza mediante la emisión de un evento personalizado, pero no llegamos a explicar cómo se implementaba.

Así que vamos a poner manos a la obra para conocer las propiedades @Output, que nos permitirán la emisión de esos eventos personalizados, con los que avisar a los padres de cualquier situación que ocurra en los hijos y que deban conocer.



Para aclarar las ideas, resumimos el esquema de trabajo de la comunicación de hijos a padres, en el que están involucrados dos componentes:

- **El componente hijo** será el encargado de escalar el evento hacia el padre, para avisarle de un suceso. Al avisarle, el hijo podrá comunicar un dato que el padre deba conocer, relacionado lógicamente con ese suceso.
- **El componente padre** será capaz de capturar el evento personalizado emitido por el hijo y recuperar aquel dato que fue enviado.

Implementar el evento personalizado en el componente hijo

Nuestro trabajo en la comunicación de hijos a padres comienza en el componente hijo. El proceso de trabajo no es trivial y su realización implica el uso de varios actores. Para explicarlo comenzaremos presentando las partes implicadas en el proceso.

Clase EventEmitter

La clase de Angular que implementa objetos capaces de emitir un evento se llama "EventEmitter". Pertenece al "core" de Angular, por lo que necesitamos asegurarnos de importarla debidamente.

```
import { Component, OnInit, EventEmitter } from  
'@angular/core';
```

Para poder emitir eventos personalizados necesitaremos crear una propiedad en el componente, donde instanciar un objeto de esta clase (new EventEmitter). Solo que este proceso tiene un detalle que lo puede complicar un poco, al menos al principio, ya que hace uso de los "generics" de TypeScript.

Básicamente, el genérico nos sirve para decirle a TypeScript el tipo del dato que nuestro evento personalizado escalará hacia el padre en su comunicación. Este es el código que podríamos utilizar para crear nuestra propiedad emisora de eventos, haciendo uso del generics.

```
propagar = new EventEmitter<string>();
```

Así le estamos diciendo que nuestro emisor de eventos será capaz de emitir datos de tipo "string". Obviamente, este tipo de dato dependerá del componente y podrá ser no solamente un tipo primitivo, sino una clase o una interfaz.

Nota: Si no entiendes mucho esta parte, te recomendamos leer este artículo con mayor información de los [genéricos de TypeScript](#).

El nombre de la propiedad, "propagar" en este caso, es importante, porque a la hora de capturar el evento en el padre tendremos que usarlo tal cual.

Decorador @Output

La propiedad de tipo "EventEmitter", necesaria para emitir el evento personalizado, debe ser decorada con @Output. Esto le dice al framework que va a existir una vía de comunicación desde el hijo al padre.

Para usar ese decorador tenemos que importarlo también. Igual que EventEmitter, la declaración de la función decoradora "Output" está dentro de "@angular/core". Nuestro import, quedará por tanto así.

```
import { Component, OnInit, Output, EventEmitter } from  
'@angular/core';
```

Para usar el decorador colocaremos la llamada con los paréntesis vacíos, lo que será suficiente en la mayoría de los casos. El código de la declaración de la propiedad, usando el correspondiente decorador @Output() es el siguiente:

```
@Output ()
```

```
propagar = new EventEmitter<string>();
```

Método emit()

Cuando queramos disparar el evento personalizado invocaremos el método `emit()` del objeto `EventEmitter`.

Este método recibe como parámetro aquel dato que queramos hacer llegar al padre. Lógicamente, el tipo del dato que enviemos hacia el padre debe concordar con el que hayamos declarado en el genérico al instanciar la el objeto `EventEmitter`.

```
this.propagar.emit('Este dato viajará hacia el padre');
```

Ejemplo de componente que envía eventos al padre

Con lo que sabemos ahora ya somos capaces de enfrentarnos a nuestro primer ejemplo de componente con propiedades `@Output`.

Vamos a realizar un sencillo componente que tiene un campo de texto y un botón. Al pulsar el botón emitirá un evento hacia el padre, pasando la cadena escrita en el campo de texto.

Comenzaré mostrando el template del componente, que es la parte que nos ayudará a entender el comportamiento del componente.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo

multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben

procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva

aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable

que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo

multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está

escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben

procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva

aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable

que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo

multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen

desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

1. Node Package Manager (npm)

Esta es una herramienta necesaria para todos los desarrolladores web en estos días. Todo en Angular se distribuye en varios paquetes npm, por lo que es muy probable que en algún momento necesites instalar algunos paquetes útiles que no sean Angular, por lo que es conveniente saber cómo hacerlo para cuando lo requieras.

2. CLI angular

El primer paquete Angular que debes instalar con npm es Angular CLI (Interfaz de línea de comandos). Aunque ciertamente es posible instalar paquetes Angular individualmente y escribir todo el código a mano para configurar su nueva aplicación, la CLI hace que ese proceso sea mucho más fácil y asegura que su aplicación cumplirá con las mejores prácticas aceptadas.

3. HTML y CSS

Tres familias de lenguajes de programación forman las herramientas básicas involucradas en prácticamente todos los aspectos del desarrollo web: HTML, CSS y JavaScript. Los dos primeros son de vital conocimiento para todo desarrollador Angular, si bien este framework proporciona los componentes básicos que necesitas

para crear aplicaciones rápidas y funcionales, esas aplicaciones aún deben procesarse en un navegador y eso significa crear interfaces de usuario con HTML y CSS.

4. Angular

Es tan obvia esta herramienta que parece casi innecesario colocarla en esta lista, pero te podrías sorprender las veces que omiten esto. Para ser un buen desarrollador Angular, deber tener un conocimiento sólido de este framework. Te tomará algún tiempo lograrlo debido a su amplitud, pero valdrá la pena.

5. TypeScript

Las aplicaciones web del lado del cliente se han escrito tradicionalmente con JavaScript. TypeScript es un superconjunto de JavaScript que incluye soporte para escritura fuerte. Angular está escrito en TypeScript. Es el lenguaje recomendado para crear aplicaciones con Angular.

Angular es uno de los frameworks más versátiles en la actualidad y en Rootstack lo hemos utilizado para dar solución a varios problemas y necesidades tecnológicas que presentan nuestros clientes internacionales. Tu puedes formar parte de un equipo multicultural de expertos, enfocados en crear hoy las aplicaciones y websites que dominarán el futuro.

Manual de HTML HTML es el lenguaje utilizado como base para crear las páginas web. En este manual explicamos en profundidad cómo utilizarlo, desde lo más básico a los temas más avanzados. Prólogo al manual de HTML Bienvenidos al manual de HTML de DesarrolloWeb. A través de todos estos capítulos vamos a descubrir el lenguaje utilizado para la creación de páginas web: el Hyper Text Markup Language, más conocido como HTML. Puede que en un principio, el hecho de hablar de un lenguaje informático pare los pies a más de uno. No os asustéis, el HTML no deja de ser más que una forma un tanto peculiar de dar formato a los textos e imágenes que pretendemos ver por medio de un navegador. Antes de entrar en materia, lo cual haremos de una forma directa y practica, os recomendamos fervorosamente la lectura previa de nuestro manual Publicar en Internet. A partir de esta guía, aprenderéis los

conceptos más básicos necesarios para creación de un sitio web. También os permitirá acceder a este manual con unos conocimientos de base sobre HTML imprescindibles y os dejara bien claro lo que su conocimiento aporta con respecto al simple uso de editores de HTML. El público al que va enfocado este manual es a todos aquellos que, con conocimientos mínimos de informática, desean hacer mundialmente público un mensaje, una idea o una información usando para ello el medio más práctico, económico y actual: Internet. Lo que necesitáis como base para llevar a buen término el aprendizaje (aparte de leer el manual Publicar en Internet) es:

- Saber escribir con un teclado
- Saber manejar un ratón
- Tener ganas de aprender

Lo que obtendréis después de haber pasado por estos capítulos:

- Capacidad para crear y publicar vuestro propio sitio web con un mínimo de calidad
- Conocimientos de todo tipo sobre las tecnologías y herramientas empleadas en el ámbito de la Red
- Posiblemente una afición que puede convertirse en pasión y terminar, en algunos casos, siendo un vicio o un oficio.

Os recordamos que estamos a vuestra entera disposición para resolveros todo tipo de dudas referentes a este manual. Contactarnos es tan fácil como pinchar sobre el mail del autor del artículo (situado al pie de la página). También podéis formular vuestras cuestiones y, esperamos que en un futuro ayudar a otros compañeros, en el foro sobre HTML o bien en la lista de correo de DesarrolloWeb.

Manual de HTML HTML es el lenguaje utilizado como base para crear las páginas web. En este manual explicamos en profundidad cómo utilizarlo, desde lo más básico a los temas más avanzados.

Prólogo al manual de HTML Bienvenidos al manual de HTML de DesarrolloWeb. A través de todos estos capítulos vamos a descubrir el lenguaje utilizado para la creación de páginas web: el Hyper Text Markup Language, más conocido como HTML. Puede que en un principio, el hecho de hablar de un lenguaje informático pare los pies a más de uno. No os asustéis, el HTML no deja de ser más que una forma un tanto peculiar de dar formato a los textos e imágenes que pretendemos ver por medio de un navegador. Antes de entrar en materia, lo cual haremos de una forma directa y practica, os recomendamos fervorosamente la lectura previa de nuestro manual Publicar en Internet. A partir de esta guía, aprenderéis los conceptos más básicos necesarios para creación de un sitio web. También os permitirá acceder a este manual con unos conocimientos de base sobre HTML imprescindibles y os dejara bien claro lo que su conocimiento aporta con respecto al simple uso de editores de HTML. El público al que va enfocado este manual es a todos aquellos que, con conocimientos mínimos de informática, desean hacer mundialmente público un mensaje, una idea o una información usando para ello el medio más práctico, económico y actual: Internet. Lo que necesitáis como base para llevar a buen término el aprendizaje (aparte de leer el manual Publicar en Internet) es:

- Saber escribir con un teclado
- Saber manejar un ratón
- Tener ganas de aprender

Lo que obtendréis después de haber pasado por estos capítulos:

- Capacidad para crear y publicar vuestro propio sitio web con un mínimo de calidad
- Conocimientos de todo tipo sobre las tecnologías y herramientas empleadas en el ámbito de la Red
- Posiblemente una afición que puede convertirse en pasión y terminar, en algunos casos, siendo un vicio o un oficio.

Os recordamos que estamos a vuestra entera disposición para resolveros todo tipo de dudas referentes a este manual. Contactarnos es tan fácil como pinchar sobre el mail del autor del

artículo (situado al pie de la página). También podéis formular vuestras cuestiones y, esperamos que en un futuro ayudar a otros compañeros, en el foro sobre HTML o bien en la lista de correo de DesarrolloWeb. Pasemos pues sin más preámbulos a ver de qué se trata el HTML... Introducción al HTML HTML es el lenguaje con el que se escriben las páginas web. Las páginas web pueden ser vistas por el usuario mediante un tipo de aplicación llamada navegador. Podemos decir por lo tanto que el HTML es el lenguaje usado por los navegadores para mostrar las páginas webs al usuario, siendo hoy en día la interface más extendida en la red. Este lenguaje nos permite aglutinar textos, sonidos e imágenes y combinarlos a nuestro gusto. Además, y es aquí donde reside su ventaja con respecto a libros o revistas, el HTML nos permite la introducción de referencias a otras páginas por medio de los enlaces hipertexto. El HTML se creó en un principio con objetivos divulgativos. No se pensó que la web llegara a ser un área de ocio con carácter multimedia, de modo que, el HTML se creó sin dar respuesta a todos los posibles usos que se le iba a dar y a todos los colectivos de gente que lo utilizarían en un futuro. Sin embargo, pese a esta deficiente planificación, si que se han ido incorporando modificaciones con el tiempo, estos son los estándares del HTML. Numerosos estándares se han presentado ya. El HTML 4.01 es el último estándar a septiembre de 2001. Esta evolución tan anárquica del HTML ha supuesto toda una serie de inconvenientes y deficiencias que han debido ser superados con la introducción de otras tecnologías accesorias capaces de organizar, optimizar y automatizar el funcionamiento de las webs. Ejemplos que pueden sonaros son las CSS, JavaScript u otros. Veremos más adelante en qué consisten algunas de ellas. Otros de los problemas que han acompañado al HTML es la diversidad de navegadores presentes en el mercado los cuales no son capaces de interpretar un mismo código de una manera unificada. Esto obliga al webmáster a, una vez creada su página, comprobar que esta puede ser leída satisfactoriamente por todos los navegadores, o al menos, los más utilizados. Además del navegador necesario para ver los resultados

de nuestro trabajo, necesitamos evidentemente otra herramienta capaz de crear la página en si. Un archivo HTML (una página) no es más que un texto. Es por ello que para programar en HTML necesitamos un editor de textos. Es recomendable usar el Bloc de notas que viene con windows, u otro editor de textos sencillo. Hay que tener cuidado con algunos editores más complejos como Wordpad o Microsoft Word, pues colocan su propio código especial al guardar las páginas y HTML es únicamente texto plano, con lo que podremos tener problemas. Existen otro tipo de editores específicos para la creación de páginas web los cuales ofrecen muchas facilidades que nos permiten aumentar nuestra productividad. No obstante, es aconsejable en un principio utilizar una herramienta lo más sencilla posible para poder prestar la máxima atención a nuestro código y familiarizarnos lo antes posible con él. Siempre tendremos tiempo más delante de pasarnos a editores más versátiles con la consiguiente ganancia de tiempo. Para tener más claro todo el tema de editores y los tipos que existen, visita los artículos: • Editores de HTML. • Bloc de notas. • También puedes acceder a descripciones editores más complejos que el Block de Notas, pero más potentes como Homesite o UltraEdit. Es importante tener claro todo ello puesto que en función de vuestros objetivos puede que, más que aprender HTML, resulte más interesante aprender el uso de una aplicación para la creación de páginas. Así pues, una página es un archivo donde está contenido el código HTML en forma de texto. Estos archivos tienen extensión .html o .htm (es indiferente cuál utilizar). De modo que cuando programemos en HTML lo haremos con un editor de textos y guardaremos nuestros trabajos con extensión .html, por ejemplo mipágina.html Consejo: Utiliza siempre la misma extensión en tus archivos HTML. Eso evitará que te confundas al escribir los nombres de tus archivos unas veces con .htm y otras con .html. Si trabajas con un equipo en un proyecto todavía más importante que os pongáis todos de acuerdo en la extensión. Sintaxis del HTML El HTML es un lenguaje que basa su sintaxis en un elemento de base al que llamamos etiqueta. La etiqueta presenta frecuentemente dos

partes: Una apertura de forma general Un cierre de tipo Todo lo incluido en el interior de esa etiqueta sufrirá las modificaciones que caracterizan a esta etiqueta. Así por ejemplo: Las etiquetas `` y `` definen un texto en negrita. Si en nuestro documento HTML escribimos una frase con el siguiente código: `Esto esta en negrita` El resultado Será: Esto esta en negrita Las etiquetas

y

definen un párrafo. Si en nuestro documento HTML escribiéramos:

Hola, estamos en el párrafo 1

Ahora hemos cambiado de párrafo

El resultado sería: Hola, estamos en el párrafo 1 Ahora hemos cambiado de párrafo Partes de un documento HTML Además de todo esto, un documento HTML ha de estar delimitado por la etiqueta `<html>` y `</html>`. Dentro de este documento, podemos asimismo distinguir dos partes principales: El encabezado, delimitado por `<head>` y `</head>` donde colocaremos etiquetas de índole informativo como por ejemplo el título de nuestra página. El cuerpo, flanqueado por las etiquetas `<body>` y `</body>`, que será donde colocaremos nuestro texto e imágenes delimitados a su vez por otras etiquetas como las que hemos visto. El resultado es un documento con la siguiente estructura: Etiquetas y contenidos del encabezado Datos que no aparecen en nuestra página pero que son importantes para catalogarla: Título, palabras clave,... Etiquetas y contenidos del cuerpo Parte del documento que será mostrada por el navegador: Texto e imágenes Las mayúsculas o minúsculas son indiferentes al escribir etiquetas A notar que las etiquetas pueden ser escritas con cualquier tipo de combinación de mayúsculas y minúsculas. , o son la misma etiqueta. Resulta sin embargo aconsejable acostumbrarse a escribirlas en minúscula ya que otras tecnologías que pueden convivir con nuestro HTML (XML por ejemplo) no son tan permisivas y nunca viene mal coger buenas costumbres desde el principio para evitar fallos triviales en un futuro. Tu primera página Podemos ya con estos conocimientos, y

alguno que otro más, crear nuestra primera página. Para ello, abre tu editor de textos y copia y pega el siguiente texto en un nuevo documento.

Bienvenido,

Estás en la página **Comida para Todos**.

Aquí aprenderás recetas fáciles y deliciosas.

Ahora guarda ese archivo con extensión .html o .htm en tu disco duro. Para ello accedemos al menú Archivo y seleccionamos la opción Guardar como. En la ventana elegimos el directorio donde deseamos guardarlo y colocaremos su nombre, por ejemplo mi_pagina.html Consejo: Utiliza nombres en tus archivos que tengan algunas normas básicas para ahorrarte disgustos y lios. Nuestro consejo es que no utilices acentos ni espacios ni otros caracteres raros. También te ayudará escribir siempre las letras en minúsculas. Esto no quiere decir que debes hacer nombres de archivos cortos, es mejor hacerlos descriptivos para que te aclaren lo que hay dentro. Algún caracter como el guión "-" o el guión bajo "_" te puede ayudar a separar las palabras. Por ejemplo quienes_somos.html Con el documento HTML creado, podemos ver el resultado obtenido a partir de un navegador. Es conveniente, llegado a este punto, hacer hincapié en el hecho de que no todos los navegadores son idénticos. Desgraciadamente, los resultados de nuestro código pueden cambiar de uno a otro por lo que resulta aconsejable visualizar la página en varios. Generalmente se usan Internet Explorer y Netscape como referencias ya que son los más extendidos. A decir verdad, en el momento que estas líneas son escritas, Internet Explorer acapara la inmensa mayoría de usuarios (90% más o menos) y Netscape esta relegado a un segundo plano. Esto no quiere decir que lo debemos dejar totalmente de lado ya que el 10% de visitas que puede proporcionarnos puede resultar muy importante para nosotros. Por otra parte, parece que se ha hecho publica la intención de Netscape de desviar un poco su temática de negocios hacia otros derroteros y abandonar esta llamada "lucha de

navegadores" en la cual estaba recibiendo la peor parte. Pues bien, volviendo al tema, una vez creado el archivo .html o .htm, podemos visualizar el resultado de nuestra labor abriendo dicha página con un navegador. Para hacerlo, la forma resulta diferente dependiendo del navegador: Si estamos empleando el Explorer, hemos de ir al barra de menú, elegir Archivo y seleccionar Abrir. Una ventana se abrirá. Pulsamos sobre el botón Examinar y accederemos a una ventana a partir de la cual podremos movernos por el interior de nuestro disco duro hasta dar con el archivo que deseamos abrir. La cosa no resulta más difícil para Netscape. En este caso, nos dirigimos también a la barra de menú principal y elegimos File y a continuación Open File. La misma ventana de búsqueda nos permitirá escudriñar el contenido de nuestro PC hasta dar con el archivo buscado. Nota: También puedes abrir el archivo si accedes al directorio donde lo guardaste. En él podrás encontrar tu archivo HTML y verás que tiene como icono el logotipo de Netscape o el de Internet Explorer. Para abrirlo simplemente hacemos un doble click sobre él. Una vez abierto el archivo podréis ver vuestra primera página web. Algo sencillita pero por algo se empieza. Ya veréis como en poco tiempo seremos capaces de mejorar sensiblemente. Fijaos en la parte superior izquierda de la ventana del navegador. Podréis comprobar la presencia del texto delimitado por la etiqueta

Encabezado de nivel 1

Encabezado de nivel 2

Encabezado de nivel 3

Encabezado de nivel 4

Encabezado de nivel 5

Encabezado de nivel 6

Se puede ver el ejercicio en una página aparte. Consejo: No debemos utilizar las etiquetas de encabezado para formatear el texto, es decir, si queremos colocar un tipo de letra más grande y en negrita

debemos utilizar las etiquetas que existen para ello (que veremos en seguida). Los encabezados son para colocar titulares en páginas web y es el navegador el responsable de formatear el texto de manera que parezca un titular. Cada navegador, pues, puede formatear el texto a su gusto con tal de que parezca un titular. formateando el texto Además de todo lo relativo a la organización de los párrafos, uno de los aspectos primordiales del formateo de un texto es el de la propia letra. Resulta muy común y práctico presentar texto resaltado en negrita, itálica y otros. Paralelamente el uso de índices, subíndices resulta vital para la publicación de textos científicos. Todo esto y mucho más es posible por medio del HTML a partir de multitud de etiquetas entre las cuales vamos a destacar algunas. Negrita Podemos escribir texto en negrita incluyéndolo dentro de las etiquetas **y (bold)**. Esta misma tarea es desempeñada por **y siendo ambas equivalentes**. Nosotros nos inclinamos por la primera por simple razón de esfuerzo. Escribiendo un código de este tipo: **Texto en negrita** Obtenemos este resultado: **Texto en negrita** Nota: ¿Qué diferencia hay entre **y** ? Aunque las dos etiquetas hacen el mismo efecto, tienen una peculiaridad que las hace distintas. La etiqueta indica negrita, mientras que la etiqueta indica que se debe escribir resaltado. El HTML lo interpretan los navegadores según su criterio, es por eso que las páginas se pueden ver de distinta manera en unos browsers y en otros. La etiqueta

quiere decir "encabezado de nivel 1", es el navegador el responsable de formatear el texto de manera que parezca un encabezado de primer nivel. En la práctica los encabezados de Internet Explorer y Netscape son muy

parecidos (tamaño de letra grande y en negrita), pero otro navegador podría colocar los encabezados con subrayado si le pareciese oportuno. La diferencia entre y se podrá entender ahora.

Mientras que significa simplemente negrita y todos los navegadores la interpretarán como negrita, es una etiqueta que significa que se tiene que resaltar fuertemente el texto y cada navegador es el responsable de resaltarlo como desee. En la práctica coloca el texto en negrilla, pero podría ser que un navegador decidiese resaltar colocando negrilla, subrayado y color rojo en el texto. Itálica También en este caso existen dos posibilidades, una corta: *e* (italic) y otra un poco más larga: *y* . En este manual, y en la mayoría de las páginas que veréis por ahí, os encontraréis con la primera

forma sin duda más sencilla a escribir y a acordarse. He aquí un ejemplo de texto en itálica: *Texto en itálica* Que da el siguiente efecto: *Texto en itálica*

Subrayado El HTML nos propone también para el subrayado el par de etiquetas: y (underlined). Sin embargo, el uso de subrayados ha de ser aplicado con mucha precaución dado que los enlaces hipertexto van, a no ser que se indique lo contrario, subrayados con lo que podemos confundir al lector y apartarlo del verdadero interés de nuestro texto. Subíndices y supraíndices Este tipo de formato resulta de extremada utilidad para textos científicos. Las etiquetas empleadas son: ^y para los supraíndices y _y para los subíndices Aquí tenéis un ejemplo: La ¹³CC₃H₄ClNOS es un heterociclo alergeno enriquecido El

resultado: La 13CC₃H₄ClNOS es un heterociclo alergeno enriquecido

Anidar etiquetas Todas estas etiquetas y por supuesto el resto de las vistas y que veremos más adelante pueden ser anidadas unas dentro de otras de manera a conseguir resultados diferentes. Así, podemos sin ningún problema crear texto en negrita e itálica embebiendo una etiqueta dentro de la otra: Esto sólo está en negrita y *esto en negrita e itálica* Esto nos daría: Esto sólo está en negrita y esto en negrita e itálica Consejo: Cuando anides etiquetas HTML hazlo correctamente. Nos referimos a que si abres etiquetas dentro de otra más principal, antes de cerrar la etiqueta principal cierras las etiquetas que hayas abierto dentro de ella. Debemos evitar códigos como el siguiente: Esto

está en negrita e *itálica* En favor de
códigos con etiquetas correctamente
anidadas: Esto está en negrita e *itálica*
Esto es muy aconsejable, aunque los
navegadores entiendan bien las
etiquetas mal anidadas, por dos
razones: 1. Sistemas como XML no son
tan permisivos con estos errores y
puede que en el futuro nuestras
páginas no funcionen correctamente. 2.
A los navegadores les cuesta mucho
tiempo de procesamiento resolver este
tipo de errores, incluso más que
construir la propia página y debemos
evitarles que sufran por una mala
codificación. Color, tamaño y tipo de
letra A pesar de que por razones de
homogeneidad y sencillez de código
este tipo de formatos son controlados
actualmente por hojas de estilo en
cascada (de las cuales ya tendremos

tiempo de hablar), existe una forma clásica y directa de definir color tamaño y tipo de letra de un texto determinado. Esto se hace a partir de la etiqueta y su cierre correspondiente. Dentro de esta etiqueta deberemos especificar los atributos correspondientes a cada uno de estos parámetros que deseamos definir. A continuación os comentamos los atributos principales de esta etiqueta: Atributo face Define el tipo de letra. Este atributo es interpretado por versiones de Netscape a partir de la 3 y de MSIE 3 o superiores. Otros navegadores las ignoran completamente y muestran el texto con la fuente que utilizan. Hay que tener cuidado con este atributo ya que cada usuario, dependiendo de la plataforma que utilice, puede no

disponer de los mismos tipos de letra que nosotros con lo que, si nosotros elegimos un tipo del que no dispone, el navegador se verá forzado a mostrar el texto con la fuente que utiliza por defecto (suele ser Times New Roman). Para evitar esto, dentro del atributo suelen seleccionarse varios tipos de letra separados por comas. En este caso el navegador comprobará que dispone del primer tipo enumerado y si no es así, pasará al segundo y así sucesivamente hasta encontrar un tipo que posea o bien acabar la lista y poner la fuente por defecto. Veamos un ejemplo. Este texto tiene otra tipografía Que se visualizaría así en una página web. Este texto tiene otra tipografía Nota: Aquí tenemos un ejemplo de atributo cuyo valor debe

estar limitado por comillas (").

Habíamos dicho que las comillas eran opcionales en los atributos, sin embargo esto no es así siempre. Si el valor del atributo contiene espacios, como es el caso de: `face="Comic Sans MS,arial,verdana"` debemos colocar las comillas para limitarlo. En caso de no tener comillas `face=Comic Sans MS,arial,verdana` se entendería que `face=Comic`, pero no se tendría en cuenta todo lo que sigue, porque HTML no lo asociaría al valor del atributo. En este caso HTML pensaría que las siguientes palabras (después del espacio) son otros atributos, pero como no los conoce como atributos simplemente los desestimaré. Atributo `size` Define el tamaño de la letra. Este tamaño puede ser absoluto o relativo. Si hablamos en términos absolutos,

existen 7 niveles de tamaño distintos numerados de 1 a 7 por orden creciente. Elegiremos por tanto un valor `size="1"` para la letra más pequeña o `size="7"` para la más grande. Este texto es más grande Que se visualizaría así en una página web. Este texto es más grande Podemos asimismo modificar el tamaño de nuestra letra con respecto al del texto mostrado precedentemente definiendo el número de niveles que queremos subir o bajar en esta escala de tamaños por medio de un signo + o -. De este modo, si definimos nuestro atributo como `size="+1"` lo que queremos decir es que aumentamos de un nivel el tamaño de la letra. Si estabamos escribiendo previamente en 3, pasaremos automáticamente a 4. Los tamaños reales que veremos en

pantalla dependerán de la definición y del tamaño de fuente elegido por el usuario en el navegador. Este tamaño de fuente puede ser definido en el Explorer yendo al menu superior, Ver/Tamaño de la fuente. En Netscape elegiremos View/Text Size. Esta flexibilidad puede en más de una ocasión resultarnos embarazosa ya que en muchos casos desearemos que el tamaño del texto permanezca constante para que éste quepa en un determinado espacio. Veremos en su momento que esta prefijación del tamaño puede ser llevada a cabo por las hojas de estilo en cascada.

Atributo color El color del texto puede ser definido mediante el atributo color. Cada color es a su vez definido por un número hexadecimal que esta compuesto a su vez de tres partes. Cada

una de estas partes representa la contribución del rojo, verde y azul al color en cuestión. Podéis entender cómo funciona esta numeración y cuáles son los colores que resultan más compatibles a partir de este artículo: Los colores y HTML. Por otra parte, es posible definir de una manera inmediata algunos de los colores más frecuentemente usados para los que se ha creado un nombre más memotécnico: Nombre Color Aqua Black Blue Fuchsia Gray Green Lime Maroon Navy Olive Purple Red Silver Teal White Yellow **Este texto está en rojo** Que se visualizaría así en una página web. Este texto está en rojo Con todo esto estamos ya en disposición de crear un texto formateado de una forma realmente elaborada. Pongamos pues en práctica todo lo que hemos

aprendido en estos capitulos haciendo un ejercicio consistente en una página que tenga las siguientes características:

- Un titular con encabezado de nivel 1, en *itálica* y color verde oliva.
- Un segundo titular con encabezado de nivel 2, también de color verde oliva.
- Todo el texto de la página deberá presentarse con una fuente distinta de la fuente por defecto. Por ejemplo "Comic Sans MS" y en caso de que ésta no esté en el sistema que se coloque la fuente "Arial". Se puede ver una posible solución del ejercicio en este enlace. (Ver el código fuente de la página para ver cómo lo hemos resuelto)

Atributos para páginas

Las páginas HTML pueden construirse con variedad de atributos que le pueden dar un aspecto a la página muy personalizado. Podemos

definir atributos como el color de fondo, el color del texto o de los enlaces. Estos atributos se definen en la etiqueta y, como decíamos son generales a toda la página. Lo mejor para explicar su funcionamiento es verlos uno por uno. Atributos para fondos bgcolor: especificamos un color de fondo para la página. En el capítulo anterior y en el taller de los colores y HTML hemos aprendido a construir cualquier color, con su nombre o su valor RGB. El color de fondo que podemos asignar con bgcolor es un color plano, es decir el mismo para toda la superficie del navegador. background: sirve para indicar la colocación de una imagen como fondo de la página. La imagen se coloca haciendo un mosaico, es decir, se repite muchas veces hasta ocupar

todo el espacio del fondo de la página. En capítulos más adelante veremos como se insertan imágenes con HTML y los tipos de imágenes que se pueden utilizar. Ejemplo de fondo Vamos a colocar esta imagen como fondo en la página. La imagen se llama fondo.jpg y suponemos que se encuentra en el mismo directorio que la página. En este caso se colocaría la siguiente etiqueta Se puede ver el efecto de colocar ese fondo en una página a parte. Consejo: siempre que coloquemos una imagen de fondo, debemos poner también un color de fondo cercano al color de la imagen. Esto se debe a que, al colocar una imagen de fondo, el texto de la página debemos colocarlo en un color que contraste suficientemente con dicho fondo. Si el visitante no puede ver el

fondo por cualquier cuestión (Por ejemplo tener deshabilitada la carga de imágenes) puede que el texto no contraste lo suficiente con el color de fondo por defecto de la web. Creo que lo mejor será poner un ejemplo. Si la imagen de fondo es oscura, tendremos que poner un texto claro para que se pueda leer. Si el visitante que accede a la página no ve la imagen de fondo, le saldrá el fondo por defecto, que generalmente es blanco, de modo que al tener un texto con color claro sobre un fondo blanco, nos pasará que no podremos leer el texto convenientemente. Ocurre parecido cuando se está cargando la página. Si todavía no ha llegado a nuestro sistema la imagen de fondo, se verá el fondo que hayamos seleccionado con bgcolor y es interesante que sea

parecido al color de la imagen para que se pueda leer el texto mientras se carga la imagen de fondo. Color del texto

text: este atributo sirve para asignar el color del texto de la página. Por defecto es el negro. Además del color del texto, tenemos tres atributos para asignar el color de los enlaces de la página. Ya debemos saber que los enlaces deben diferenciarse del resto del texto de la página para que los usuarios puedan identificarlos fácilmente. Para ello suelen aparecer subrayados y con un color más vivo que el texto. Los tres atributos son los siguientes: **link:** el color de los enlaces que no han sido visitados. (por defecto es azul clarito) **vlink:** el color de los enlaces visitados. La "v" viene justamente de la palabra visitado. Es el color que tendrán los enlaces que ya hemos

visitado. Por defecto su color es morado. Este color debería ser un poco menos vivo que el color de los enlaces normales. `alink`: es el color de los enlaces activos. Un enlace está activo en el preciso instante que se pulsa. A veces es difícil darse cuenta cuando un enlace está activo porque en el momento en el que se activa es porque lo estamos pulsando y en ese caso el navegador abandonará la página rápidamente y no podremos ver el enlace activo más que por unos instantes mínimos. Ejemplo de color del texto Vamos a ver una página donde el color de fondo sea negro, y los colores del texto y los enlaces sean claros. Pondremos el color de texto blanco y los enlaces amarillos, más resaltados los que no estén visitados y menos resaltados lo que ya están

visitados. Para ello escribiríamos la etiqueta body así: El efecto se puede ver en una página a parte. Márgenes

Con otros atributos de la etiqueta se pueden asignar espacios de margen en las páginas, lo que es muy útil para eliminar los márgenes en blanco que aparecen a los lados, arriba y debajo de la página. Estos atributos son distintos para Internet Explorer y para Netscape Navigator, por lo que debemos utilizarlos todos si queremos que todos los navegadores los interpreten perfectamente.

`leftmargin:` para indicar el margen a los lados de la página. Válido para iexplorer.

`topmargin:` para indicar el margen arriba y debajo de la página. Para iexplorer.

`marginwidth:` la contrapartida de `leftmargin` para Netscape. (Margen a los lados)

marginheight: igual que topmargin, pero para Netscape. (Margen arriba y abajo) Tenemos un artículo sobre la utilización de estos atributos para hacer diseños avanzados con tablas en distintas definiciones de pantalla, que puede ser interesante de leer. Un ejemplo de página sin margen es la propia página de DesarrolloWeb.com, que estás visitando actualmente. (Por lo menos a la hora de escribir este artículo) Además, vamos a ver otra página sin márgenes, por si alguien necesita ver el ejemplo en estas líneas.

Hola amigos

Gracias por visitarme!

Esta página tiene el fondo blanco y dentro una tabla con el fondo rojo. En la página podremos ver que la tabla ocupa el espacio en la página sin dejar sitio para ningún tipo de margen. Puede verse el ejemplo en una página a parte. Listas I Las posibilidades que nos ofrece el HTML en cuestión de tratamiento de texto son realmente notables. No se limitan a lo visto hasta ahora, sino que van más lejos todavía. Varios ejemplos de ello son las listas, que sirven para enumerar y definir elementos, los textos preformateados y las cabeceras o títulos. Las listas son utilizadas para citar, numerar y definir objetos. También son utilizadas corrientemente para desplazar el comienzo de línea hacia la derecha. Podemos distinguir tres tipos de listas:

- Listas desordenadas • Listas**

**ordenadas • Listas de definición Las
veremos detenidamente una a una.
Listas desordenadas Son delimitadas
por las etiquetas**

**y (unordered list). Cada uno de los
elementos de la lista es citado por
medio de una etiqueta (sin cierre,
aunque no hay inconveniente en
colocarlo). La cosa queda así:**

Países del mundo

- Argentina**
- Perú**
- Chile**

**El resultado: Países del mundo •
Argentina • Perú • Chile Podemos
definir el tipo de viñeta empleada para
cada elemento. Para ello debemos**

especificarlo por medio del atributo type incluido dentro de la etiqueta de apertura

- **, si queremos que el estilo sea válido para toda la lista, o dentro de la etiqueta si queremos hacerlo específico de un solo elemento. La sintaxis es del siguiente tipo:**
 - **donde tipo de viñeta puede ser uno de los siguientes: circle disc square Nota: En algunos navegadores no funciona la opción de cambiar el tipo de viñeta a mostrar y por mucho que nos empeñemos, siempre saldrá el redondel negro. En caso de que no funcione siempre podemos construir la lista a mano con la viñeta que queramos utilizando las tablas**

de HTML. Veremos más adelante cómo trabajar con tablas. Vamos a ver un ejemplo de lista con un cuadrado en lugar de un redondel, y en el último elemento colocaremos un círculo. Para ello vamos a colocar el atributo type en la etiqueta

- , con lo que afectará a todos los elementos de la lista.

1.Elemento 1

2. Elemento 2

3. Elemento 3

4. Elemento 4

- Que tiene como resultado f
Elemento 1 f Elemento 2 f
Elemento 3 o Elemento 4
Listas II Continuamos
estudiando las listas de
HTML, con las que crear

**estructuras atractivas para
presentar la información.
Listas ordenadas En este caso
usaremos las etiquetas**

1.(ordered list) y su cierre.

**Cada elemento sera
igualmente precedido de
su etiqueta . Pongamos un
ejemplo:**

**Reglas de
comportamiento en el
trabajo**

**1.El jefe siempre tiene la
razón**

**2. En caso de duda
aplicar regla 1**

**2. El resultado es: Reglas
de comportamiento en el
trabajo 1. El jefe siempre
tiene la razón 2. En caso**

de duda aplicar regla 1 Del mismo modo que para las listas desordenadas, las listas ordenadas ofrecen la posibilidad de modificar el estilo. En concreto nos es posible especificar el tipo de numeración empleado eligiendo entre números (1, 2, 3...), letras (a, b, c...) y sus mayúsculas (A, B, C,...) y números romanos en sus versiones mayúsculas (I, II, III,...) y minúsculas (i, ii, iii,...). Para realizar dicha selección hemos de utilizar, como para el caso precedente, el atributo type, el cual será situado dentro de la etiqueta

- 1. . Los valores que puede tomar el atributo en este caso son: 1 Para ordenar por números a Por letras del alfabeto A Por letras mayúsculas del alfabeto i Ordenación por números romanos en minúsculas I Ordenación por números romanos en mayúsculas Nota: Recordamos que en algunos navegadores no funciona la opción de cambiar el tipo de viñeta a mostrar Puede que en algún caso deseemos comenzar nuestra enumeración**

por un número o letra que no tiene por qué ser necesariamente el primero de todos. Para solventar esta situación, podemos utilizar un segundo atributo, `start`, que tendrá como valor un número. Este número, que por defecto es 1, corresponde al valor a partir del cual comenzamos a definir nuestra lista. Para el caso de las letras o los números romanos, el navegador se encarga de hacer la traducción del número a la letra correspondiente. Os

**proponemos un
ejemplo usando este
tipo de atributos:
Ordenamos por
numeros**

- **Elemento 1**

- **Elemento 2**

2. Ordenamos por letras

- **Elemento a**

- **Elemento b**

**3. Ordenamos por
números romanos
empezando por el 10**

- **Elemento x**

- **Elemento xi**

4. El resultado:

**Ordenamos por
números 1. Elemento 1**

2. Elemento 2

Ordenamos por letras

a. Elemento a b.

**Elemento b Ordenamos
por numeros romanos
empezando por el 10 x.**

Elemento x xi.

Elemento xi Listas III

**Terminamos el tema
de listas estudiando las
listas de definición.**

**Veremos también la
anidación de listas.**

Listas de definición

**Cada elemento es
presentado junto con
su definición. La**

etiqueta principal es

y

**(definition list). La
etiquetas del elemento**

y su definición son
(definition term) y
(definition definition)
respectivamente. Aquí
os proponemos un
código que podrá
aclarar este sistema:
Diccionario de la Real
Academia

Brujula
Señórula montada en
una escóbula
Oreja
Sesenta minutejos
El efecto producido:
Diccionario de la Real
Academia Brujula
Señórula montada en
una escóbula Oreja

Sesenta minutejos
Fijaos en que cada línea
esta desplazada hacia
la izquierda. Este tipo
de etiquetas son usadas
a menudo con el
propósito de crear
textos más o menos
desplazados hacia la
izquierda. El código:

Primer nivel de
desplazamiento

Segundo nivel de
desplazamiento

Tercer nivel de
desplazamiento

**El resultado: Primer
nivel de
desplazamiento
Segundo nivel de
desplazamiento Tercer
nivel de
desplazamiento
Anidando listas Nada
nos impide utilizar
todas estas etiquetas
de forma anidada como
hemos visto en otros
casos. De esta forma,
podemos conseguir
listas mixtas como por
ejemplo:
Ciudades del mundo**

■ Argentina

1. Buenos Aires

2. Bariloche

■ Uruguay

1. Montevideo

2. Punta del Este

5. De esta forma creamos una lista como esta: Ciudades del mundo • Argentina 1. Buenos Aires 2. Bariloche • Uruguay 1. Montevideo 2. Punta del Este Caracteres especiales Una página web se ha de ver en países distintos, que usan conjuntos de caracteres distintos. El lenguaje HTML nos ofrece un mecanismo por el que podemos estar seguros que una serie de caracteres raros se van a ver bien en todos los ordenadores del mundo, independientemente de su juego de caracteres. Este conjunto son los caracteres especiales. Cuando queremos poner uno de estos caracteres en una página, debemos sustituirlo por su código. Por ejemplo,

la "á" (a minúscula acentuada) se escribe "á" de modo que la palabra página se escribiría en una página HTML de este modo:

páaacute;gina Caracteres especiales básicos En realidad estos caracteres se usan en HTML para no confundir un principio o final de etiqueta, unas comillas o un & con su correspondiente caracter. < < > > & & "

" Caracteres especiales del HTML 2.0 Á Á À À É É È È Í Í Ì Ì Ó Ó Ò Ò Ú Ú Ù Ù á á à à é é è è í í ì ì ó ó ò ò ú ú ù ù Ä Ä Â Â Ë Ë Ê Ê Ï Ï Î Î Ö Ö Ô Ô Û Û Û Û ä ä â â ë ë ê ê ì ì î î ö ö ô ô ü ü û û Ã Ã å å Ñ Ñ Å Å Õ Õ Ç Ç ã ã ç ç ñ ñ Ý Ý ã ã ý ý Ø Ø ÿ ÿ ø ø Þ Þ Ð Ð þ þ ð ð Æ Æ ß ß æ æ Caracteres especiales del HTML 3.2 1/4 1/4 1/2 1/2 ¡ ¡ 3/4 3/4 £ £ © © ¥ ¥ ® ® § § ª ª ¯ ¯ ² ² | | ³ ³ « « ¹ ¹ ¬ ¬ ¯ ¯ µ µ ° ° ¶ ¶ ´ ´ · · ¨ ¨ ° ° ± ± , , » » ¿ ¿ Otros caracteres especiales × × ¢ ¢ ÷ ÷

€ € “ “ ™ ™ ” ” ‰ ‰ Œ Œ f f ‡ ‡ † †

Enlaces en HTML Hasta aquí, hemos podido ver que una página web es un archivo HTML en el que podemos incluir, entre otras cosas, textos formateados a nuestro gusto e imágenes (las veremos enseguida). Del mismo modo, un sitio web podrá ser considerado como el conjunto de archivos, principalmente páginas HTML e imágenes, que constituyen el contenido al que el navegante tiene acceso. Sin embargo, no podríamos hablar de navegante o de navegación si estos archivos HTML no estuviesen debidamente conectados entre ellos y con el exterior de nuestro sitio por medio de enlaces hipertexto. En efecto, el atractivo original del HTML radica en la posible puesta en relación de los contenidos de los archivos

introduciendo referencias bajo forma de enlaces que permitan un acceso rápido a la información deseada. De poco serviría en la red tener páginas aisladas a las que la gente no puede acceder y desde las que la gente no puede saltar a otras. Un enlace puede ser fácilmente detectado en una página. Basta con deslizar el puntero del ratón sobre las imágenes o el texto y ver como cambia de su forma original transformándose por regla general en una mano con un dedo señalador. Adicionalmente, estos enlaces suelen ir, en el caso de los textos, coloreados y subrayados para que el usuario no tenga dificultad en reconocerlos. Si no especificamos lo contrario (ya tendremos ocasión de explicar como), estos enlaces texto estarán subrayados y coloreados en

azul. En el caso de las imágenes que sirvan de enlace, veremos que están delimitadas por un marco azul por defecto. Para colocar un enlace, nos serviremos de las etiquetas `a` y `img`. Dentro de la etiqueta de apertura deberemos especificar asimismo el destino del enlace. Este destino será introducido bajo forma de atributo, el cual lleva por nombre `href`. La sintaxis general de un enlace es por tanto de la forma:

`contenido`

Siendo el contenido un texto o una imagen. Es la parte de la página que se colocará activa y donde deberemos pulsar para acceder al enlace. Por su parte, destino será una página, un correo electrónico o un archivo. En función del destino los enlaces son clásicamente agrupados del siguiente modo:

- Enlaces internos: los que se dirigen a otras partes dentro

de la misma página. • Enlaces locales: los que se dirigen a otras páginas del mismo sitio web. • Enlaces remotos: los dirigidos hacia páginas de otros sitios web. • Enlaces con direcciones de correo: para crear un mensaje de correo dirigido a una dirección. • Enlaces con archivos: para que los usuarios puedan hacer download de ficheros. Enlaces internos Son los enlaces que apuntan a un lugar diferente dentro de la misma página. Este tipo de enlaces son esencialmente utilizados en páginas donde el acceso a los contenidos puede verse dificultado debido al gran tamaño de la misma. Mediante estos enlaces podemos ofrecer al visitante la posibilidad de acceder rápidamente al principio o final de la página o bien a diferentes párrafos o secciones. Para crear un

enlace de este tipo es necesario, aparte del enlace de origen propiamente dicho, un segundo enlace que será colocado en el destino. Veamos más claramente como funcionan estos enlaces con un ejemplo sencillo: Enlace con final de este documento, para que probéis su funcionamiento: Ir abajo Supongamos que queremos crear un enlace que apunte al final de la página. Lo primero será colocar nuestro enlace origen. Lo pondremos aquí mismo y lo escribiremos del siguiente modo: Ir abajo Como podéis ver, el contenido del enlace es el texto "Ir abajo" y el destino, abajo, es un punto de la misma página que todavía no hemos definido. Ojo al símbolo #; es él quien especifica al navegador que el enlace apunta a una sección en particular. En segundo lugar, hay que

generar un enlace en el destino. Este enlace llevara por nombre abajo para poder distinguirlo de los otros posibles enlaces realizados dentro de la misma página. En este caso, la etiqueta que escribiremos será ésta: A decir verdad, estos enlaces, aunque útiles, no son los más extendidos de cuantos hay. La tendencia general es la de crear páginas (archivos) independientes con tamaños más reducidos enlazados entre ellos por enlaces locales (los veremos enseguida). De esta forma evitamos el exceso de tiempo de carga de un archivo y la introducción de exceso de información que pueda desviar la atención del usuario. Una aplicación corriente de estos enlaces consiste en poner un pequeño índice al principio de nuestro documento donde introducimos enlaces origen a las

diferentes secciones. Paralelamente, al final de cada sección introducimos un enlace que apunta al índice de manera que podamos guiar al navegante en la búsqueda de la información útil para él. Enlaces locales Como hemos dicho, un sitio web esta constituido de páginas interconexas. En el capitulo anterior hemos visto como enlazar distintas secciones dentro de una misma página. Nos queda pues estudiar la manera de relacionar los distintos documentos HTML que componen nuestro sitio web. Para crear este tipo de enlaces, hemos de crear una etiqueta de la siguiente forma: contenido Por regla general, para una mejor organización, los sitios suelen estar ordenados por directorios. Estos directorios suelen contener diferentes secciones de la página,

imágenes, sonidos...Es por ello que en muchos casos no nos valdrá con especificar el nombre del archivo, sino que tendremos que especificar además el directorio en el que nuestro archivo.html esta alojado. Si habéis trabajado con MS-DOS no tendréis ningún problema para comprender el modo de funcionamiento. Tan solo hay que tener cuidado en usar la barra "/" en lugar de la contrabarra "\". Para aquellos que no saben como mostrar un camino de un archivo, aquí van una serie de indicaciones que os ayudaran a comprender la forma de expresarlos. No resulta difícil en absoluto y con un poco de practica lo haréis prácticamente sin pensar. 1. Hay que situarse mentalmente en el directorio en el que se encuentra la página con el enlace. 2. Si la página destino esta en

un directorio incluido dentro del directorio en el que nos encontramos, hemos de marcar el camino enumerando cada uno de los directorios por los que pasamos hasta llegar al archivo y separándolos por el símbolo barra "/". Al final obviamente, escribimos el archivo. 3. Si la página destino se encuentra en un directorio que incluye el de la página con el enlace, hemos de escribir dos puntos y una barra "../" tantas veces como niveles subamos en la arborescencia hasta dar con el directorio donde esta emplazado el archivo destino. 4. Si la página se encuentra en otro directorio no incluido ni incluyente del archivo origen, tendremos que subir como en la regla 3 por medio de "../" hasta encontrar un directorio que englobe el directorio que contiene a la página

destino. A continuación haremos como en la regla 2. Escribiremos todos los directorios por los que pasamos hasta llegar al archivo. Ejemplo: Para clarificar este punto podemos hacer un ejemplo a partir de la estructura de directorios de la imagen. Para hacer un enlace desde index.html hacia yyy.html: contenido Para hacer un enlace desde xxx.html hacia yyy.html: contenido Para hacer un enlace desde yyy.html hacia xxx.html: contenido Los enlaces locales pueden a su vez apuntar ya no a la página en general sino más precisamente a una sección concreta. Este tipo de enlaces resultan ser un híbrido de interno y local. La sintaxis es de este tipo: contenido Como para los enlaces internos, en este caso hemos de marcar la sección con otro enlace del tipo: Como

ejemplo, he aquí un enlace que apunta al capítulo anterior al final de la página. Enlaces externos, de correo y hacia archivos. Para acabar con los enlaces vamos a ver los últimos 3 tipos de enlaces que habíamos señalado.

Enlaces remotos Son los enlaces que se dirigen hacia páginas que se encuentran fuera de nuestro sitio web, es decir, cualquier otro documento que no forma parte de nuestro sitio. Este tipo de enlaces es muy común y no representa ninguna dificultad.

Simplemente colocamos en el atributo HREF de nuestra etiqueta la URL o dirección de la página con la que queremos enlazar. Será algo parecido a esto. [ir a guiarte.com](http://ir.guiarte.com) Sólo cabe destacar que todas las direcciones web (URLs) empiezan por http://. Esto indica que el protocolo por el que se

accede es HTTP, el utilizado en la web. No debemos olvidarnos de colocarlas, porque si no los enlaces serán tratados como enlaces locales a nuestro sitio. Otra cosa interesante es que no tenemos que enlazar con una página web con el protocolo HTTP necesariamente. También podemos acceder a recursos a través de otros protocolos como el FTP. En tal caso, las direcciones de los recursos no comenzarán por http:// sino por ftp://.

Enlaces a direcciones de correo Los enlaces a direcciones de correo son aquellos que al pincharlos nos abre un nuevo mensaje de correo electrónico dirigido a una dirección de mail determinada. Estos enlaces son muy habituales en las páginas web y resultan la manera más rápida de ofrecer al visitante una vía para el

contacto con el propietario de la página. Para colocar un enlace dirigido hacia una dirección de correo colocamos mailto: en el atributo href del enlace, seguido de la dirección de correo a la que se debe dirigir el enlace. eugim@desarrolloweb.com Este enlace se puede ver en funcionamiento aquí: eugim@desarrolloweb.com Consejo: Cuando coloques enlaces a direcciones de correo procura indicar en el contenido del enlace (lo que hay entre y) la dirección de correo a la que se debe escribir. Esto es porque si un usuario no tiene configurado un programa de correo en su ordenador no podrá enviar mensajes, pero por lo menos podrá copiar la dirección de mail y escribir el correo a través de otro ordenador o un sistema web-mail. Además de la dirección de

correo del destinatario, también podemos colocar en el enlace el asunto del mensaje. Esto se consigue colocando después de la dirección de correo un interrogante, la palabra subject, un signo igual (=) y el asunto en concreto.

eugim@desarrolloweb.com Podemos colocar otros atributos del mensaje con una sintaxis parecida. En este caso indicamos también que el correo debe ir con copia a

colabora@desarrolloweb.com.

eugim@desarrolloweb.com Nota: El visitante de la página necesitará tener configurada una cuenta de correo electrónico en su sistema para enviar los mensajes. Lógicamente, si no tiene servicio de correo en el ordenador no se podrán enviar los mensajes y este sistema de contacto con el visitante no

funcionará. Tenemos un artículo en desarrolloweb que habla sobre el contacto con el navegante. Enlaces con archivos Este no es un tipo de enlace propiamente dicho, pero lo señalamos aquí porque son un tipo de enlaces muy habitual y que presenta alguna complicación para el usuario novato. El mecanismo es el mismo que hemos conocido en los enlaces locales y los enlaces remotos, con la única particularidad de que en vez de estar dirigidos hacia una página web está dirigido hacia un archivo de otro tipo. Si queremos enlazar con un archivo `mi_fichero.zip` que se encuentra en el mismo directorio que la página se escribiría un enlace así. Descarga `mi_fichero.zip` Si pinchamos un enlace de este tipo nuestro navegador descargará el fichero, haciendo la

pregunta típica de "Qué queremos hacer con el archivo. Abrirlo o guardarlo en disco". Podemos ver un ejemplo de enlace a archivo con su consiguiente ventana de descarga de un archivo. Consejo: No colocar en Internet archivos ejecutables directamente sino archivos comprimidos. Por dos razones: 1. El archivo ocupará menos, con lo que será más rápida su transferencia. 2. Al preguntar al usuario lo que desea hacer con el fichero le ofrece la opción de abrirlo y guardarlo en disco. Nosotros generalmente desearemos que el usuario lo guarde en disco y no lo ejecute hasta que lo tenga en su disco duro. Si se decide a abrirlo en vez de guardarlo simplemente lo pondrá en marcha y cuando lo pare no se quedará guardado en su sistema. Si los archivos


están comprimidos obligaremos al usuario a descomprimirlos en su disco duro antes de ponerlos en marcha, con lo que nos aseguramos que el usuario lo guarde en su ordenador antes de ejecutarlo. Si queremos enlazar hacia otro tipo de archivo como un PDF o un mundo VRML (Realidad virtual para Internet) lo seguimos haciendo de la misma manera. El navegador, si reconoce el tipo de archivo, es el responsable de abrirlo utilizando el conector adecuado para ello. Así, si por ejemplo enlazamos con un PDF pondrá el programa Acrobat Reader en funcionamiento para mostrar los contenidos. Si enlazamos con un mundo VRML pondrá en marcha el plug-in que el usuario tenga instalado para ver los mundos virtuales (Cosmo Player por ejemplo). Este sería un

ejemplo de enlace a un documento PDF. Descarga el PDF Imágenes en HTML Sin duda uno de los aspectos más vistosos y atractivos de las páginas web es el grafismo. La introducción en nuestro texto de imágenes puede ayudarnos a explicar más fácilmente nuestra información y darle un aire mucho más estético. El abuso no obstante puede conducirnos a una sobrecarga que se traduce en una distracción para el navegante, quien tendrá más dificultad en encontrar la información necesaria, y un mayor tiempo de carga de la página lo que puede ser de un efecto nefasto si nuestro visitante no tiene una buena conexión o si es un poco impaciente. En este capítulo no explicaremos como crear ni tratar las imágenes, únicamente diremos que para ello se

utilizan aplicaciones como Paint Shop Pro, Photoshop o Corel Draw.

Tampoco explicaremos las particularidades de cada tipo de archivo GIF o JPG y la forma de optimizar nuestras imágenes. Un capítulo posterior al respecto será dedicado a este menester: Formatos gráficos para páginas web. Las imágenes son almacenadas en forma de archivos, principalmente GIF (para dibujos) o JPG (para fotos). Estos archivos pueden ser creados por nosotros mismos o pueden ser descargados gratuitamente en sitios web especializados. En desarrolloweb contamos con la mayor base de datos de gifs animados e imágenes de todo tipo en castellano, que nos provee el sitio internacional GOgraph. Así pues, en estos primeros capítulos nos

limitaremos a explicar como insertar y alinear debidamente en nuestra página una imagen ya creada. La etiqueta que utilizaremos para insertar una imagen es (image). Esta etiqueta no posee su cierre correspondiente y en ella hemos de especificar obligatoriamente el paradero de nuestro archivo grafico mediante el atributo src (source). La sintaxis queda entonces de la siguiente

forma:  Para expresar el camino, lo haremos de la misma forma que vimos para los enlaces. Las reglas siguen siendo las mismás, lo único que cambia es que, en lugar de una página destino, el destino es un archivo grafico. Aparte de este atributo, indispensable obviamente para la visualización de la imagen, la etiqueta nos propone otra serie de atributos de

mayor o menor utilidad: Atributo alt
Dentro de las comillas de este atributo
colocaremos una brevísima
descripción de la imagen. Esta etiqueta
no es indispensable pero presenta
varias utilidades. Primeramente,
durante el proceso de carga de la
página, cuando la imagen no ha sido
todavía cargada, el navegador
mostrara esta descripción, con lo que
el navegante se puede hacer una idea
de lo que va en ese lugar. Esto no es tan
trivial si tenemos en cuenta que
algunos usuarios navegan por la red
con una opción del navegador que
desactiva el muestreo de imágenes,
con lo que tales personas podrán
siempre saber de qué se trata el grafico
y eventualmente cambiar a modo con
imágenes para visualizarla. Además,
determinadas aplicaciones para


discapacitados o teléfonos vocales que no muestran imágenes ofrecen la posibilidad de leerlas por lo que nunca esta de más pensar en estos colectivos. En general podemos considerar como aconsejable el uso de este atributo salvo para imágenes de poca importancia y absolutamente indispensable si la imagen en cuestión sirve de enlace. Atributos height y width Definen la altura y anchura respectivamente de la imagen en pixels. Todos los archivos gráficos poseen unas dimensiones de ancho y alto. Estas dimensiones pueden obtenerse a partir del propio diseñador grafico o bien haciendo clic con el botón derecho sobre la imagen vista por el navegador para luego elegir propiedades sobre el menú que se despliega. El hecho de explicitar en


nuestro código las dimensiones de nuestras imágenes ayuda al navegador a confeccionar la página de la forma que nosotros deseamos antes incluso de que las imágenes hayan sido descargadas. Así, si las dimensiones de las imágenes han sido proporcionadas, durante el proceso de carga, el navegador reservara el espacio correspondiente a cada imagen creando una maquetación correcta. El usuario podrá comenzar a leer tranquilamente el texto sin que este se mueva de un lado a otro cada vez que una imagen se cargue. Además de esta utilidad, el alterar los valores de estos dos atributos, es una forma inmediata de redimensionar nuestra imagen. Este tipo de utilidad no es aconsejable dado que, si lo que pretendemos es aumentar el tamaño, la pérdida de

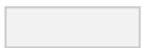
calidad de la imagen será muy sensible. Inversamente, si deseamos disminuir su tamaño, estaremos usando un archivo más grande de lo necesario para la imagen que estamos mostrando con lo que aumentamos el tiempo de descarga de nuestro documento innecesariamente. Es importante hacer hincapié en este punto ya que muchos debutantes tienen esa mala costumbre de crear gráficos pequeños redimensionando la imagen por medio de estos atributos a partir de archivos de tamaño descomunal. Hay que pensar que el tamaño de una imagen con unas dimensiones de la mitad no se reduce a la mitad, sino que resulta ser aproximadamente 4 veces inferior. Atributo border Definen el tamaño en pixels del cuadro que rodea la imagen.

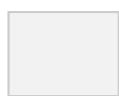
De esta forma podemos recuadrar nuestra imagen si lo deseamos. Es particularmente útil cuando deseamos eliminar el borde que aparece cuando la imagen sirve de enlace. En dicho caso tendremos que especificar `border="0"`. Atributos `vspace` y `hspace` Sirven para indicar el espacio libre, en pixeles, que tiene que colocarse entre la imagen y los otros elementos que la rodean, como texto, otras imágenes, etc. Atributo `lowsrc` Con este atributo podemos indicar un archivo de la imagen de baja resolución. Cuando el navegador detecta que la imagen tiene este atributo primero descarga y muestra la imagen de baja resolución (que ocupa muy poco y que se transfiere muy rápido). Posteriormente descarga y muestra la imagen de resolución adecuada

(señalada con el atributo src, que se supone que ocupará más y será más lenta de transferir). Este atributo está en desuso, aunque supone una ventaja considerable para que la descarga inicial de la web se realice más rápido y que un visitante pueda ver una muestra de la imagen mientras se descarga la imagen real. Truco: Utilizar imagenes como enlaces Ni que decir tiene que una imagen, lo mismo que un texto, puede servir de enlace. Vista la estructura de los enlaces podemos muy fácilmente adivinar el tipo de código

necesario:  Ejemplo práctico
Resultará obvio para los lectores hacer ahora una página que contenga una imagen varias veces repetida pero con distintos atributos. • Una de las veces que salga debe mostrarse con su

tamaño original y con un borde de 3 pixeles. • En otra ocasión la imagen aparecerá sin borde, con su misma altura y con una anchura superior a la original • También mostraremos la imagen sin borde, con su misma anchura y con una altura superior a la original • Por último, mostraremos la imagen con una altura y anchura mayores que las originales, pero proporcionalmente igual que antes. Vamos a utilizar esta imagen para hacer el ejercicio: Las dimensiones originales de la imagen son 28x21, así que este sería el código fuente: 





Se puede ver el ejemplo en una página aparte. Alineación de imágenes con HTML Vimos en su momento el atributo align que nos permitía alinear el texto a derecha, izquierda o centro de nuestra página. Dijimos que este atributo no era exclusivo de la etiqueta sino que podía ser encontrado en otro tipo de etiquetas. Pues bien, resulta ser una de esas etiquetas que aceptan este atributo aunque en este caso el funcionamiento resulta ser diferente. Para alinear una imagen horizontalmente podemos hacerlo de la misma forma que el texto, es decir, utilizando el atributo align dentro de una etiqueta

o

. En este caso, lo que incluiremos dentro de esa etiqueta será la imagen

en lugar del texto: Este código
mostrará la imagen en el centro:



Quedaría así: Sin embargo, ya hemos dicho que la etiqueta puede aceptar el atributo align. En este caso, la utilidad que le damos difiere de la anterior. El hecho de utilizar el atributo align dentro de la etiqueta nos permite, en el caso de darle los valores left o right, justificar la imagen del lado que deseamos a la vez que rellenamos con texto el lado opuesto. De esta forma embebemos nuestras imágenes dentro del texto de una manera sencilla. Aquí podéis ver el tipo de código a crear para obtener dicho efecto:



Texto tan extenso como
queramos que cubrirá la parte

izquierda de la imagen. Sigo poniendo texto para que se vea el efecto, Bla bla bla bla bla bla...

Quedaría así: Texto tan extenso como queramos que cubrirá la parte izquierda de la imagen. Sigo poniendo texto para que se vea el efecto, Bla bla bla bla bla bla...



Texto tan extenso como queramos que cubrirá la parte derecha de la imagen. Sigo poniendo texto para que se vea el efecto, Bla bla bla bla bla bla bla...

Quedaría así: Texto tan extenso como queramos que cubrirá la parte izquierda de la imagen. Sigo poniendo texto para que se vea el efecto, Bla bla bla bla bla bla... Si en algún momento deseásemos dejar de rellenar ese espacio lateral, podemos pasar a

una zona libre introduciendo un salto de línea

dentro del cual añadiremos un atributo: clear Así, etiquetas del tipo: Saltara verticalmente hasta encontrar el lateral izquierdo libre.

Saltara verticalmente hasta encontrar el lateral derecho libre.

Saltará verticalmente hasta encontrar ambos laterales libres. Ejemplo de clear: Texto tan extenso como queramos que cubrirá la parte izquierda. Esto está debajo de la imagen. Existen otro tipo de valores que puede adoptar el atributo align dentro de la etiqueta . Estos son relativos a la alineación vertical de la imagen. Supongamos que escribimos una línea al lado de nuestra imagen. Esta línea puede quedar por ejemplo arriba, abajo o al medio de la imagen.

Asimismo, puede que en una misma línea tengamos varias imágenes de alturas diferentes que pueden ser alineadas de distintas formas. Estos valores adicionales del atributo align son: top Ajusta la imagen a la parte más alta de la línea. Esto quiere decir que, si hay una imagen más alta, ambas imágenes presentarán el borde superior a la misma altura. bottom Ajusta el bajo de la imagen al texto. Absbottom Colocará el borde inferior de la imagen a nivel del elemento más bajo de la línea. middle Hace coincidir la base de la línea de texto con el medio vertical de la imagen. absmiddle Ajusta la imagen al medio absoluto de la línea. Estas explicaciones, que pueden resultar un poco complicadas, pueden ser más fácilmente asimiladas a partir con un poco de práctica. Nos queda

explicar como introducir debajo de la imagen un pie de foto o explicación. Para ello tendremos que ver antes de nada las tablas, en el próximos capítulos... Formátos gráficos para páginas web El componente gráfico de las páginas web tiene mucha importancia, es el que hace que estas sean vistosas y el que nos permite aplicar nuestra creatividad para hacer del diseño de sitios una tarea agradable. Es también una herramienta para acercar los sitios al mundo donde vivimos, si embargo, es también el causante de errores graves en las páginas y hacer de estas, en algunos casos, un martirio para el visitante. Las nociones básicas para el uso de archivos gráficos son sencillas, conocerlas, aunque sea ligeramente, nos ayudará a crear sitios agradables y

rápidos. No cometer errores en el uso de las imágenes es fundamental, aunque no seas un diseñador y las imágenes que utilices sean feas, utilízalas bien y así estarás haciendo más agradable la visita a tus páginas.

Tipos de archivos En Internet se utilizan principalmente dos tipos de archivos gráficos GIF y JPG, pensados especialmente para optimizar el tamaño que ocupan en disco, ya que los archivos pequeños se transmiten más rápidamente por la Red. El formato de archivo GIF se usa para las imágenes que tengan dibujos, mientras que el formato JPG se usa para las fotografías. Los dos comprimen las imágenes para guardarlas. La forma de comprimir la imagen que utiliza cada formato es lo que los hace ideales para unos u otros

propósitos. Adicionalmente, se puede usar un tercer formato gráfico en las páginas web, el PNG. Este formato no tiene tanta aceptación como el GIF o JPG por varias razones, entre las que destacan el desconocimiento del formato por parte de los desarrolladores, que las herramientas habituales para tratar gráficos (como por ejemplo Photoshop) generalmente no lo soportan y que los navegadores antiguos también tienen problemas para visualizarlas. Sin embargo, el formato se comporta muy bien en cuanto a compresión y calidad del gráfico conseguido, por lo que resultaría útil si se llega a extender su uso. GIF A parte de ser un archivo ideal para las imágenes que estén dibujadas tiene muchas otras características que son importantes y útiles. Compresión:

Es muy buena para dibujos, como ya hemos dicho. Incluso puede ser interesante si la imagen es muy pequeña, aunque sea una foto.

Transparencia: es una utilidad para definir ciertas partes del dibujo como transparentes. De este modo podemos colocar las imágenes sobre distintos fondos sin que se vea el cuadrado donde está inscrito la el dibujo, viendose en cambio la silueta del dibujo en cuestión. Para crear un gif transparente debemos utilizar un programa de diseño gráfico, con el podemos indicar qué colores del dibujo queremos que sean transparentes.

Generalmente, definimos la transparencia cuando vamos a guardar el gráfico. Colores: Con este formato gráfico podemos utilizar paletas, conjuntos, de 256 colores o menos.

Este es un detalle muy importante, puesto que cuantos menos colores utilicemos en la imagen, por lo general, menos ocupará el archivo. En ocasiones, aunque utilicemos menos colores en un gráfico, este no pierde mucho en calidad, llegando a ser inapreciable a la vista. En algunos programas podemos modificar la cantidad de colores al guardar el archivo, en otros lo hacemos mientras creamos el gráfico. Un logotipo es un ejemplo claro de imagen GIF Parte de esta imagen es transparente 32 colores 16 colores 8 colores Imagen tomada con distintas paletas de colores. Se puede apreciar como con pocos colores se ve bien el gráfico y como pierde un poco a medida que le restamos colores. JPG Veamos ahora cuales son las características fundamentales del

formato JPG: Compresión: Tal como hemos dicho anteriormente, su algoritmo de compresión hace ideal este formato para guardar fotografías. Además, con JPG podemos definir la calidad de la imagen, con calidad baja el fichero ocupará menos, y viceversa.

Transparencia: Este formato no tiene posibilidad de crear áreas transparentes. Si deseamos colocar una imagen con un área que parezca transparente procederemos así: con nuestro programa de diseño gráfico haremos que el fondo de la imagen sea el mismo que el de la página donde queremos colocarla. En muchos casos los fondos de la imagen y la página parecerán el mismo. **Colores:** JPG trabaja siempre con 16 millones de colores, ideal para fotografías.

Optimizar ficheros Para que las

imágenes ocupen lo menos posible y se transfieran rápidamente por la Red debemos aprender a optimizar los ficheros gráficos. Para ello debemos hacer lo siguiente: Una fotografía con formato JPG Intento de transparencia en JPG. Pulsar para ampliar Para los archivos GIF: Reduciremos el número de colores de nuestra paleta. Esto se hace con nuestro editor gráfico, en muchos casos podremos hacerlo al guardar el archivo. GIF 256 colores – 10,8 KB GIF 32 colores – 5,5 KB GIF 4 colores – 2 KB Para los archivos JPG: Ajustaremos la calidad del archivo cuando lo estemos guardando. Este formato nos permite bajar mucho la calidad de la imagen sin que esta pierda mucho en su aspecto visual. JPG calidad 0 3 KB JPG calidad 20 5,9 KB JPG calidad 50 10 KB Es imprescindible

disponer para optimizar la imagen de una herramienta buena que nos permita configurar estas características de la imagen con libertad y fácilmente. Photoshop 5.5 o 6 es un programa bastante recomendable, pues incorpora una opción que se llama "Guardar para el Web" con la que podemos definir los colores del gif, calidad del JPG y otras opciones en varias muestras a la vez. Así con todas las opciones configurables, viendo los resultados a la vez que el tamaño del archivo podemos optimizar la imagen de una manera precisa con los resultados que deseamos. También existen en el mercado otros programas que nos permiten optimizar estas imágenes de manera sorprendente. Una vez hemos creado la imagen la pasamos por estos

programas y nos comprimen aun más el archivo, haciéndolo rápido de transferir y, por tanto, más optimo para Internet. Al ser estas utilidades tan especializadas los resultados suelen ser mejores que con los programas de edición gráfica.

Ejemplos de optimizadores gráficos: – WebGraphics Optimizer – ProJPG, GIF Imantion Y con versiones Online: – JPG – GIF Crunchers – GIF Wizard

Photoshop es una herramienta excelente para optimizar ficheros.

Viendo varias copias podemos elegir la más adecuada. Tablas en HTML Una tabla en un conjunto de celdas organizadas dentro de las cuales podemos alojar distintos contenidos.

En un principio nos podría parecer que las tablas son raramente útiles y que pueden ser utilizadas principalmente

para listar datos como agendas, resultados y otros datos de una forma organizada. Nada más lejos de la realidad. Hoy, gran parte de los diseñadores de páginas basan su maquetación en este tipo de artilugios. En efecto, una tabla nos permite organizar y distribuir los espacios de la manera más optima. Nos puede ayudar a generar texto en columnas como los periódicos, prefijar los tamaños ocupados por distintas secciones de la página o poner de una manera sencilla un pie de foto a una imagen. Puede que en un principio nos resulte un poco complicado trabajar con estas estructuras pero, si deseamos crear una página de calidad, tarde o temprano tendremos que vérnoslas con ellas y nos daremos cuenta de las posibilidades nos ofrecen. Para

empezar, nada más sencillo que por el principio: las tablas son definidas por las etiquetas y

6. . Dentro de estas dos etiquetas colocaremos todas las otras etiquetas, textos e imágenes que darán forma y contenido a la tabla. Las tablas son descritas por líneas de izquierda a derecha. Cada una de estas líneas es definida por otra etiqueta y su cierre: y Asimismo, dentro de cada línea, habrá diferentes celdas. Cada una de estas celdas será definida por otro par de etiquetas: y . Dentro de estas etiquetas será donde coloquemos nuestro contenido. Aquí tenéis un ejemplo de estructura de tabla:

Celda	Celda
1,	2,
linea 1	linea 1

Celda	Celda
1,	2,
linea	linea 2
2	

7. El resultado: Celda 1, linea 1 Celda 2,
linea 1 Celda 1, linea 2 Celda 2, linea 2

Nota: Hasta aquí hemos visto todas las etiquetas que necesitamos conocer para crear tablas. Existen otras etiquetas, pero lo que podemos conseguir con ellas se puede conseguir también usando las que hemos visto. Por poner un ejemplo, señalamos la etiqueta , que sirve para crear una celda cuyo contenido esté formateado como un título o cabecera de la tabla. En la práctica, lo que hace es poner en negrita y centrado el contenido de esa celda, lo que se puede conseguir aplicando las correspondientes

**etiquetas dentro de la celda. Así:
contenido de la celda. A partir de esta
idea simple y sencilla, las tablas
adquieren otra magnitud cuando les
incorporamos toda una batería de
atributos aplicados sobre cada tipo de
etiquetas que las componen. A lo largo
de los siguientes capítulos nos
adentraremos en el estudio de estos
atributos de manera a proporcionarnos
los útiles indispensables para una
buena puesta en página. Tablas en
HTML. Atributos para filas y celdas.
Hemos visto en el capítulo anterior que
las tablas están compuestas de líneas
que, a su vez, contienen celdas. Las
celdas son delimitadas por las
etiquetas o por las etiquetas (si
queremos texto en negrita y centrado)
y constituyen un entorno
independiente del resto del**

documento. Esto quiere decir que: • Podemos usar prácticamente cualquier tipo de etiqueta dentro de la etiqueta para, de esta forma, dar forma a su contenido. • Las etiquetas situadas en el interior de la celda no modifican el resto del documento. • Las etiquetas de fuera de la celda no son tenidas en cuenta por ésta. Así pues, podemos especificar el formato de nuestras celdas a partir de etiquetas introducidas en su interior o mediante atributos colocados dentro de la etiqueta de celda o bien, en algunos casos, dentro de la etiqueta , si deseamos que el atributo sea válido para toda la línea. La forma más útil y actual de dar forma a las celdas es a partir de las hojas de estilo en cascada que ya tendréis la oportunidad de abordar más adelante. Veamos a

continuación algunos atributos útiles para la construcción de nuestras tablas. Empecemos viendo atributos que nos permiten modificar una celda en concreto o toda una línea: **align** Justifica el texto de la celda del mismo modo que si fuese el de un párrafo. **valign** Podemos elegir si queremos que el texto aparezca arriba (top), en el centro (middle) o abajo (bottom) de la celda. **bgcolor** Da color a la celda o línea elegida. **bordercolor** Define el color del borde. Otros atributos que pueden ser únicamente asignados a una celda y no al conjunto de celdas de una línea son: **background** Nos permite colocar un fondo para la celda a partir de un enlace a una imagen. **height** Define la altura de la celda en pixels o porcentaje. **width** Define la anchura de la celda en pixels o

porcentaje. `colspan` Expande una celda horizontalmente. `rowspan` Expande una celda verticalmente. Nota: El atributo `height` no funciona en todos los navegadores, además, su uso no está muy extendido. Las celdas por lo general tienen el alto que necesitan para que quepa todo el contenido que se le haya insertado, es decir, crecen lo suficiente para que quepa lo que hemos colocado dentro. El atributo `width` si que funciona en todos los navegadores y lo tendréis que utilizar constantemente. Si le asignamos un ancho a la celda, el ancho será respetado y si dicha celda tiene mucho texto o cualquier otro contenido, la celda crecerá hacia abajo todo lo necesario para que quepa lo que hemos colocado. Un matiz al último párrafo. Se trata de que si definimos una celda de

un ancho 100 por ejemplo, y colocamos en la celda un contenido como una imagen que mida más de 100 pixeles, la celda crecerá en horizontal todo lo necesario para que la imagen quepa. Si el elemento, aunque más ancho, fuera divisible (como un texto) el ancho sería respetado y el texto crecería hacia abajo o lo que es lo mismo, en altura, como señalábamos en el anterior párrafo. Estos últimos cuatro atributos descritos son de gran utilidad. Concretamente, height y width nos ayudan a definir las dimensiones de nuestras celdas de una forma absoluta (en pixels o puntos de pantalla) o de una forma relativa, es decir por porcentajes referidos al tamaño total de la tabla. Podéis leer un artículo interesante a propósito de estas dos modalidades de diseño en

nuestro manual de usabilidad. A título de ejemplo: Dará una anchura de 80 pixels a la celda. Sin embargo, Dará una anchura a la celda del 80% de la anchura de la tabla. Hay que tener en cuenta que, definidas las dimensiones de las celdas, el navegador va a hacer lo que buenamente pueda para satisfacer al programador. Esto quiere decir que puede que en algunas ocasiones el resultado que obtengamos no sea el esperado. Concretamente, si el texto presenta una palabra excesivamente larga, puede que la anchura de la celda se vea aumentada para mantener la palabra en la misma línea. Por otra parte, si el texto resulta muy largo, la celda aumentara su altura para poder mostrar todo su contenido.

Análogamente, si por ejemplo definimos dos anchuras distintas a

celdas de una misma columna, el navegador no sabrá a cual hacer caso. Es por ello que resulta conveniente tener bien claro desde un principio como es la tabla que queremos diseñar. No esta de más si la prediseñamos en papel si la complejidad es importante. El HTML resulta en general fácil pero las tablas pueden convertirse en un verdadero quebradero de cabeza si no llegamos a comprenderlas debidamente. Los atributos rowspan y colspan son también utilizados frecuentemente. Gracias a ellos es posible expandir celdas fusionando éstas con sus vecinas. El valor que pueden tomar estas etiquetas es numérico. El número representa la cantidad de celdas fusionadas. Así, Fusionara la celda en cuestión con su vecina derecha. Esta celda tiene un

`colspan="2"` Celda normal Otra celda
Del mismo modo, Celda Normal Esta
celda tiene `rowspan="2"`, por eso tiene
fusionada la celda de abajo. Otra celda
normal Expandirá la celda hacia abajo
fusionándose con la celda inferior. El
resto de los atributos presentados
presentan una utilidad y uso bastante
obvios. Los dejamos a vuestra propia
investigación. Tablas en HTML.

Atributos de la tabla y conclusión.

Además de los atributos específicos de
cada celda o línea, las tablas pueden
ser adicionalmente formateadas a
partir de los atributos que nos ofrece la
propia etiqueta . He aquí aquellos que
pueden parecernos en un principio
importantes: `align` Alinea

horizontalmente la tabla con respecto
a su entorno. `background` Nos permite
colocar un fondo para la tabla a partir

de un enlace a una imagen. bgcolor Da color de fondo a la tabla. border Define el número de pixels del borde principal. bordercolor Define el color del borde. cellpadding Define, en pixels, el espacio entre los bordes de la celda y el contenido de la misma. cellspacing Define el espacio entre los bordes (en pixels). height Define la altura de la tabla en pixels o porcentaje. width Define la anchura de la tabla en pixels o porcentaje. Los atributos que definen las dimensiones, height y width, funcionan de una manera análoga a la de las celdas tal y como hemos visto en el capítulo anterior. Contrariamente, el atributo align no nos permite justificar el texto de cada una de las celdas que componen la tabla, sino más bien, justificar la propia tabla con respecto a

su entorno. Vamos a poner tres ejemplos de alineado de tablas, centradas, alineadas a la derecha y a la izquierda. Ejemplo de tabla centrada Esta tabla está centrada (`aling="center"`). Solo tiene una celda. Este sería un texto cualquiera colocado al lado de una tabla centrada Ejemplo de tabla alineada a la derecha Para que se vea el efecto de alineado a la tabla debemos colocar un texto al lado y el texto rodeará la tabla, igual que ocurría con las imágenes alineadas a un lado. Esta tabla está alineada a la derecha (`aling="right"`). Solo tiene una celda. Ejemplo de tabla alineada a la izquierda Para que se vea el efecto de alineado a la tabla debemos colocar un texto al lado y el texto rodeará la tabla, igual que ocurría con las imágenes alineadas a un lado. Esta tabla está

alineada a la izquierda (aling="left"). Solo tiene una celda. Los atributos cellpadding y cellspacing nos ayudaran a dar a nuestra tabla un aspecto más estético. En un principio puede parecernos un poco confuso su uso pero un poco de practica será suficiente para hacerse con ellos. En la siguiente imagen podemos ver gráficamente el significado de estos atributos. Podéis comprobar vosotros mismos que los atributos definidos para una celda tienen prioridad con respecto a los definidos para una tabla. Podemos definir, por ejemplo, una tabla con color de fondo rojo y una de las celdas de color de fondo verde y se verá toda la tabla de color rojo menos la celda verde. Del mismo modo, podemos definir un color azul para los bordes de la tabla y hacer que una celda

particular sea mostrada con un borde rojo. (Aunque esto no funcionará en todos los navegadores debido a que algunos no reconocen el atributo bordercolor. Tabla de color rojo de fondo El atributo bgcolor de la tabla está en rojo. Celda normal Esta celda está en verde. tiene el atributo bgcolor en color verde Tablas anidadas Muy útil también es el uso de tablas anidadas. De la misma forma que podíamos incluir listas dentro de otras listas, las tablas pueden ser incluidas dentro de otras. Así, podemos incluir una tabla dentro de la celda de otra. El modo de funcionamiento sigue siendo el mismo aunque la situación puede complicarse si el número de tablas embebidas dentro de otras es elevado. Consejo: Páginas como DesarrolloWeb.com y muchas otras (La

mayoría de las páginas avanzadas) que basan su diseño en tablas, realizan anidaciones de tablas constantemente para meter unos elementos de la página dentro de otros. Se pueden anidar tablas sin límite, sin embargo, en el caso de Netscape 4 hay que tener cuidado con el número de tablas que anidamos, porque a medida que metemos una tabla dentro de otra y otra dentro de esta y otra más, aumentando el grado de anidación sucesivamente... podemos encontrar problemas en su visualización y puede que la página tarde un poco de tiempo más en mostrarse en pantalla. Vamos a ver un código de anidación de tablas. Veamos primero el resultado y luego el código, así conseguiremos entenderlo mejor. Celda de la tabla principal Tabla anidada, celda 1 Tabla anidada, celda 2

Tabla anidada, celda 3 Tabla anidada, celda 4 Este sería el código:

8.

Celda de la tabla principal	Tabla anidada, celda 1	Tabla anidada, celda 2
	Tabla anidada, celda 3	Tabla anidada, celda 4

9. **Ejemplos prácticos Hasta aquí la información que pretendíamos**

transmitiros sobre las tablas en HTML. Sería importante ahora realizar algún ejemplo de realización de una tabla un poco compleja. Por ejemplo la siguiente: Animales en peligro de extinción

Nombre	Cabezas 2010	Previsión 2020
Ballena	6000	4000
Oso Pardo	1500	50
Lince	10	0
Tigre	300	210

Se puede ver esta tabla en otra ventana, donde también podremos examinar su código fuente. Otro ejemplo de tabla con el que podemos practicar: Climas de América del Sur

Parte de arriba de América del Sur	Países como:
Venezuela	Colombia
Ecuador	Perú

Parte de abajo de América del Sur. Países como:

Argentina	Chile	Uruguay	Paraguay
Bosque tropical,	clima de sabana,	clima marítimo con inviernos secos.	Climas marítimos con veranos secos,

con inviernos secos, climas frios, clima de estepa, clima desértico. También la podemos ver en una ventana a parte para extraer su código fuente. Formularios HTML Hasta ahora hemos visto la forma en la que el HTML gestiona y muestra la información, esencialmente mediante texto, imágenes y enlaces. Nos queda por ver de qué forma podemos intercambiar información con nuestro visitante. Desde luego, este nuevo aspecto resulta primordial para gran cantidad de acciones que se pueden llevar a cabo mediante el Web: Comprar un artículo, rellenar una encuesta, enviar un comentario al autor... Hemos visto anteriormente que podíamos, mediante los enlaces, contactar directamente con un correo electrónico. Sin embargo, esta opción

puede resultar en algunos casos poco versátil si lo que deseamos es que el navegante nos envíe una información bien precisa. Es por ello que el HTML propone otra solución mucho más amplia: Los formularios. Los formularios son esas famosas cajas de texto y botones que podemos encontrar en muchas páginas web. Son muy utilizados para realizar búsquedas o bien para introducir datos personales por ejemplo en sitios de comercio electrónico. Los datos que el usuario introduce en estos campos son enviados al correo electrónico del administrador del formulario o bien a un programa que se encarga de procesarlo automáticamente. Usando HTML podemos únicamente enviar el formulario a un correo electrónico. Si queremos procesar el formulario

mediante un programa la cosa puede resultar un poco más compleja ya que tendremos que emplear otros lenguajes más sofisticados. En este caso, la solución más sencilla es utilizar los programás prediseñados que nos proponen un gran número de servidores de alojamiento y que nos permiten almacenar y procesar los datos en forma de archivos u otros formatos. Si vuestras páginas están alojadas en un servidor que no os propone este tipo de ventajas, siempre podéis recurrir a servidores de terceros que ofrecen este u otro tipo de servicios gratuitos para webs. Por supuesto, existe otra alternativa que es la de aprender lenguajes como ASP o PHP que nos permitirán, entre otras cosas, el tratamiento de formularios. Los formularios son definidos por

medio de las etiquetas

y

. Entre estas dos etiquetas colocaremos todos los campos y botones que componen el formulario.

Dentro de esta etiqueta

debemos especificar algunos atributos: action Define el tipo de acción a llevar a cabo con el formulario. Como ya hemos dicho, existen dos posibilidades: • El formulario es enviado a una dirección de correo electrónico • El formulario es enviado a un programa o script que procesa su contenido En el primer caso, el contenido del formulario es enviado a la dirección de correo electrónico especificada por medio de una sintaxis de este tipo: Si lo que queremos es que el formulario sea procesado por un programa, hemos de

especificar la dirección del archivo que contiene dicho programa. La etiqueta quedaría en este caso de la siguiente forma: La forma en la que se expresa la localización del archivo que contiene el programa es la misma que la vista para los enlaces. method Este atributo se encarga de especificar la forma en la que el formulario es enviado. Los dos valores posibles que puede tomar esta atributo son post y get. A efectos prácticos y, salvo que se os diga lo contrario, daremos siempre el valor post. enctype Se utiliza para indicar la forma en la que viajará la información que se mande por el formulario. En el caso más corriente, enviar el formulario por correo electrónico, el valor de este atributo debe de ser "text/plain". Así conseguimos que se envíe el contenido del formulario

como texto plano dentro del email. Si queremos que el formulario se procese automáticamente por un programa, generalmente no utilizaremos este atributo, de modo que tome su valor por defecto, es decir, no incluiremos enctype dentro de la etiqueta. Ejemplo de etiqueta completa. Así, para el caso más habitual -el envío del formulario por correo- la etiqueta de creación del formulario tendrá el siguiente aspecto: Entre esta etiqueta y su cierre colocaremos el resto de etiquetas que darán forma a nuestro formulario, las cuales serán vistas en capítulos siguientes.

Referencia: Mandar formulario por correo electrónico

Los formularios se utilizan habitualmente para implementar un tipo de contacto con el navegante, que consiste en que éste pueda mandarnos sus

comentarios por correo electrónico a nuestro buzón. Para este tipo de utilización de los formularios hemos publicado hace tiempo en DesarrolloWeb.com un artículo que puede resultar muy interesante para los que deseen una referencia extremadamente rápida para construir un formulario que envíe los datos por correo electrónico al desarrollador de la página. El artículo en cuestión se llama contacto con el navegante.

Elementos de Formularios. Campos de texto El HTML nos propone una gran diversidad de alternativas a la hora de crear nuestros formularios. Estas van desde la clásica caja de texto hasta la lista de opciones pasando por las cajas de validación. Veamos en qué consiste cada una de estas modalidades y como podemos implementarlas en nuestro

formulario. Texto corto Las cajas de texto son colocadas por medio de la etiqueta . Dentro de esta etiqueta hemos de especificar el valor de dos atributos: type y name. La etiqueta es de la siguiente forma: De este modo expresamos nuestro deseo de crear una caja de texto cuyo contenido será llamado nombre (por ejemplo). El aspecto de este tipo de cajas es de sobra conocido, aquí lo podéis ver: El nombre del elemento del formulario es de gran importancia para poder identificarlo en nuestro programa de procesamiento o en el mail recibido. Por otra parte, es importante indicar el atributo type, ya que, como veremos, existen otras modalidades de formulario que usan esta misma etiqueta. El empleo de estas cajas esta fundamentalmente destinado a la

toma de datos breves: palabras o conjuntos de palabras de longitud relativamente corta. Veremos más adelante que existe otra forma de tomar textos más largos a partir de otra etiqueta. Además de estos dos atributos, esenciales para el correcto funcionamiento de nuestra etiqueta, existen otra serie de atributos que pueden resultarnos de utilidad pero que no son imprescindibles: **size** Define el tamaño de la caja en número de caracteres. Si al escribir el usuario llega al final de la caja, el texto ira desfilando a medida que se escribe haciendo desaparecer la parte de texto que queda a la izquierda. **maxlength** Indica el tamaño máximo del texto que puede ser tomado por el formulario. Es importante no confundirlo con el atributo **size**. Mientras el primero

define el tamaño aparente de la caja de texto, maxlength indica el tamaño máximo real del texto que se puede escribir. Podemos tener una caja de texto con un tamaño aparente (size) que es menor que el tamaño máximo (maxlength). Lo que ocurrirá en este caso es que, al escribir, el texto ira desfilando dentro de la caja hasta que lleguemos a su tamaño máximo definido por maxlength, momento en el cual nos será imposible continuar escribiendo. value En algunos casos puede resultarnos interesante asignar un valor definido al campo en cuestión. Esto puede ayudar al usuario a rellenar más rápidamente el formulario o darle alguna idea sobre la naturaleza de datos que se requieren. Este valor inicial del campo puede ser expresado mediante el atributo value. Veamos su

efecto con un ejemplo sencillo: Genera un campo de este tipo: Perico Palote

Nota: estamos obligados a utilizar la etiqueta Aunque de lo que se lee en estos capítulos sobre formularios se puede entender bien esto, hemos querido remarcarlo para que quede muy claro: Cuando queremos utilizar en cualquier situación elementos de formulario debemos escribirlos siempre entre las etiquetas y

. De lo contrario, los elementos se verán perfectamente en Explorer pero no en Netscape. Dicho de otra forma, en Netscape no se visualizan los elementos de formulario a no ser que esten colocados entre las correspondientes etiquetas de inicio y fin de formulario. Es por ello que para mostrar un campo de texto no vale con poner la etiqueta , sino que habrá que

ponerla dentro de un formulario. Así:

Veremos posteriormente que este atributo puede resultar relevante en determinadas situaciones. Texto oculto Podemos esconder el texto escrito por medio asteriscos de manera a aportar una cierta confidencialidad. Este tipo de campos son análogos a los de texto con una sola diferencia: remplazamos el atributo `type="text"` por `type="password"`: En este caso, podéis comprobar que al escribir dentro del campo en lugar de texto veréis asteriscos. Estos campos son ideales para la introducción de datos confidenciales, principalmente códigos de acceso. Se ve en funcionamiento a continuación. Texto largo Si deseamos poner a la disposición de usuario un campo de

texto donde pueda escribir cómodamente sobre un espacio compuesto de varias líneas, hemos de invocar una nueva etiqueta: y su cierre correspondiente. Este tipo de campos son prácticos cuando el contenido a enviar no es un nombre teléfono o cualquier otro dato breve, sino más bien, un comentario, opinión, etc.

Dentro de la etiqueta textarea deberemos indicar, como para el caso visto anteriormente, el atributo name para asociar el contenido a un nombre que será asemejado a una variable en los programas de proceso. Además, podemos definir las dimensiones del campo a partir de los atributos siguientes: rows Define el número de líneas del campo de texto. cols Define el número de columnas del campo de texto. La etiqueta queda por tanto de

esta forma: `<textarea
name="comentario" rows="10"
cols="40">` El resultado es el
siguiente: Asimismo, es posible
predefinir el contenido del campo.
Para ello, no usaremos el atributo
value sino que escribiremos dentro de
la etiqueta el contenido que deseamos
atribuirle. Veámoslo: Escribe tu
comentario.... Dará como resultado:
Escribe tu comentario.... Otros
elementos de formulario
Efectivamente, los textos son un
manera muy practica de hacernos
llegar la información del navegante.
No obstante, en muchos casos, los
textos son difícilmente adaptables a
programás que puedan procesarlos
debidamente o bien, puede que su
contenido no se ajuste al tipo de
información que requerimos. Es por

ello que, en determinados casos, puede resultar más efectivo proponer una elección al navegante a partir del planteamiento de una serie de opciones. Este es el caso de, por ejemplo, ofrecer una lista de países, el tipo de tarjeta de crédito para un pago,... Este tipo de opciones pueden ser expresadas de diferentes formás.

Veamos a continuación cuales son:

Listas de opciones Las listas de opciones son ese tipo de menús desplegables que nos permiten elegir una (o varias) de las múltiples opciones que nos proponen. Para construirlas emplearemos una etiqueta con su respectivo cierre: Como para los casos ya vistos, dentro de esta etiqueta definiremos su nombre por medio del atributo name. Cada opción será incluida en una línea precedida de la

etiqueta . Podemos ver, a partir de estas directivas, la forma más típica y sencilla de esta etiqueta: Primavera Verano Otoño Invierno El resultado es: Primavera Esta estructura puede verse modificada principalmente a partir de otros dos atributos: size Indica el número de valores mostrados de la lista. El resto pueden ser vistos por medio de la barra lateral de desplazamiento. multiple Permite la selección de más varios elementos de la lista. La elección de más de un elemento se hace como con el explorador de Windows, a partir de las teclas ctrl o shift. Este atributo se expresa sin valor alguno, es decir, no se utiliza con el igual: simplemente se pone para conseguir el efecto, o no se pone si queremos una lista desplegable común. Consejo: Si es posible, no uses

multiple No recomendamos especialmente la puesta en practica de esta opción ya que el manejo de las teclas ctrl o shift para elegir varias opciones puede ser desconocido para el navegante. Evidentemente, siempre cabe la posibilidad de explicarle como funciona aunque no dejara de ser una complicación para más para el visitante. Veamos cual es el efecto producido por estos dos atributos cambiando la línea: por: La lista quedara de esta forma: Primavera Verano Otoño La etiqueta puede asimismo ser matizada por medio de otros atributos selected Del mismo modo que multiple, este atributo no toma ningún valor sino que simplemente indica que la opción que lo presenta esta elegida por defecto. Así, si cambiamos la línea del código

anterior: Otoño por: Otoño El resultado será: Otoño value Define el valor de la opción que será enviado al programa o correo electrónico si el usuario elige esa opción. Este atributo puede resultar muy útil si el formulario es enviado a un programa puesto que a cada opción se le puede asociar un número o letra, lo cual es más fácilmente manipulable que una palabra o texto. podríamos así escribir líneas del tipo: Primavera De este modo, si el usuario elige primavera, lo que le llegara al programa (o correo) es una variable llamada estacion que tendrá com valor 1. En el correo electrónico recibiríamos: estacion=1

Botones de radio Existe otra alternativa para plantear una elección, en este caso, obligamos al internauta a elegir únicamente una de las opciones

que se le proponen. La etiqueta empleada en este caso es en la cual tendremos el atributo type ha de tomar el valor radio. Veamos un ejemplo:

Primavera

Verano

Otoño

Invierno Nota: Hay que fijarse que la etiqueta sólo coloca la casilla pinchable en la página. Los textos que aparecen al lado, así como los saltos de línea los colocamos con el correspondiente texto en el código de la página y las etiquetas HTML que necesitamos. El resultado es el siguiente: Primavera Verano Otoño Invierno Como puede verse, a cada una de las opciones se le atribuye una etiqueta input dentro de la cual asignamos el mismo nombre (name) para todas las opciones y un valor (value) distinto. Si el usuario

elige supuestamente Otoño,
recibiremos en nuestro correo una
línea tal que esta: estacion=3 Cabe
señalar que es posible preseleccionar
por defecto una de las opciones. Esto
puede ser conseguido por medio del
atributo checked: Verano Veamos el
efecto: Primavera Verano Otoño
Invierno Cajas de validación Este tipo
de elementos pueden ser activados o
desactivados por el visitante por un
simple clic sobre la caja en cuestión. La
sintaxis utilizada es muy similar a las
vistas anteriormente: Me gusta la
paella El efecto: Me gusta la paella La
única diferencia fundamental es el
valor adoptado por el atributo type. Del
mismo modo que para los botones de
radio, podemos activar la caja por
medio del atributo checked. El tipo de
información que llegara a nuestro

correo (o al programa) será del tipo: paella=on (u off dependiendo si ha sido activada o no) Envío, borrado y demás en formularios HTML Los formularios han de dar plaza no solamente a la información a tomar del usuario sino también a otra serie de funciones. Concretamente, han de permitirnos su envío mediante un botón. También puede resultar práctico poder proponer un botón de borrado o bien acompañarlo de datos ocultos que puedan ayudarnos en su procesamiento. En este capítulo, para terminar la saga de formularios, daremos a conocer los medios de instalar todas estas funciones. botón de envío Para dar por finalizado el proceso de relleno del formulario y hacerlo llegar a su gestor, el navegante ha de validarlo por medio de un botón

previsto a tal efecto. La construcción de dicho botón no reviste ninguna dificultad una vez familiarizados con las etiquetas input ya vistas: Con este código generamos un botón como este: Enviar Como puede verse, tan solo hemos de especificar que se trata de un botón de envío (`type="submit"`) y hemos de definir el mensaje del botón por medio del atributo value. botón de borrado Este botón nos permitirá borrar el formulario por completo en el caso de que el usuario desee rehacerlo desde el principio. Su estructura sintáctica es análoga a la anterior: A diferencia del botón de envío, indispensable en cualquier formulario, el botón de borrado resulta meramente optativo y no es utilizado frecuentemente. Hay que tener cuidado de no ponerlo muy cerca del botón de

envío y de distinguir claramente el uno del otro. Datos ocultos En algunos casos, aparte de los propios datos enviados por el usuario, puede resultar práctico enviar datos definidos por nosotros mismos que ayuden al programa en su procesamiento del formulario. Este tipo de datos, que no se muestran en la página pero si pueden ser detectados solicitando el código fuente, no son frecuentemente utilizados por páginas construidas en HTML, son más bien usados por páginas que emplean tecnologías de servidor. No os asustéis, veremos más adelante qué quiere decir esto. Tan solo queremos dar constancia de su existencia y de su modo creación. He aquí un ejemplo: Esta etiqueta, incluida dentro de nuestro formulario, enviara un dato adicional al correo o

programa encargado de la gestión del formulario. podríamos, a partir de este dato, dar a conocer al programa el origen del formulario o algún tipo de acción a llevar a cabo (una redirección por ejemplo). Botones normales

Dentro de los formularios también podemos colocar botones normales, pulsables como cualquier otro botón. Igual que ocurre con los campos hidden, estos botones por si solos no tienen mucha utilidad pero podremos necesitarlos para realizar acciones en el futuro. Su sintaxis es la siguiente. Quedaría de esta manera: El uso más frecuente de un botón es en la programación en el cliente. Utilizando lenguajes como Javascript podemos definir acciones a tomar cuando un visitante pulse el botón de una página web. Ejemplo de formulario Con este

capitulo finalizamos nuestro tema de formularios. Pasemos ahora a ejemplificar todo lo aprendido a partir de la creación de un formulario que consulta el grado de satisfacción de los usuarios de una línea de autobuses ficticia. El formulario está construido para que envíe los datos por correo electrónico a un buzón determinado. Vemos el formulario en esta página. Vosotros tratar de construirlo para ver si habéis entendido bien los temas sobre formularios. Nombre Email @ Población Sexo Hombre Mujer Frecuencia de los viajes Varias veces al dia Comentarios sobre su satisfacción personal Deseo recibir notificación de las novedades en las líneas de autobuses. Enviar formulario Borrar todo El formulario se puede ver también en una página a parte.

Recordad que podéis ver el código fuente de cualquier página web utilizando los menús de vuestro navegador, así podréis revisar el código que hemos utilizado para construir el formulario. A continuación también mostraremos el código fuente de este formulario, que es importante que todos le echemos un vistazo, aunque sea rápidamente.

Nombre

Email

Población

Sexo

Hombre

Mujer

Frecuencia de los viajes

Varias veces al dia Una vez al dia

Varias veces a la semana varias veces al mes

Comentarios sobre su satisfacción personal

Deseo recibir notificación de las novedades en las líneas de autobuses.

Para acabar, vamos a ver lo que recibirían por correo electrónico en la empresa de autobuses cuando un usuario cualquiera rellenase este formulario y pulsase sobre el botón de envío. nombre=Federico Mijo Silvestre email=fede@terramix.com poblacion=Astorga, León sexo=Varon utilizacion=2 comentarios=No creo que sea una buena linea. Poner más

autobuses. recibir_info=on

Referencia: Taller con formularios

Hemos publicado un taller de HTML con un formulario para valorar la página web. Muy sencillo y práctico.

Puede ser interesante para afianzar estos conocimientos. Entrar Mapas de imágenes con HTML En capítulos anteriores hemos podido adentrarnos en el elemento básico de navegación del web: El enlace hipertexto. Hemos visto que estos enlaces son palabras, textos o imágenes que, al pinchar sobre ellos, nos envían a otras páginas o zonas. Los mapas de imágenes es un nuevo planteamiento de navegación que incorpora una serie de enlaces dentro de una misma imagen. Estos enlaces son definidos por figuras geométricas y funcionan exactamente del mismo modo que los otros enlaces.

Podéis ver el funcionamiento de uno en este enlace. En un principio, estos mapas no eran directamente reconocidos por los navegadores y recurrían a tecnologías de lado del servidor para ser visualizados. Hoy en día pueden ser implementados por medio de código HTML tal y como veremos en este capítulo. Podemos utilizar estos mapas, por ejemplo, en portadas donde damos a conocer cada una de las secciones del sitio por medio de una imagen. También puede ser muy práctico en mapas geográficos donde cada ciudad, provincia o punto cualquiera representa un enlace a una página. En cualquier caso, el uso de estos mapas ha de estar sistemáticamente acompañado de un texto explicativo que dé a conocer al usuario la posibilidad de hacer clic

sobre los distintos puntos de la imagen. Frases como "Haz clic sobre tal icono para acceder a tal información" resultan muy indicativas a la hora de hacer intuitiva la navegación por los mapas de imágenes. Por otro lado, no esta de más introducir esa misma explicación en el atributo alt de la imagen. Así pues, un mapa de imagen esta compuesto de dos partes:

- La imagen propiamente dicha que estará situada como de costumbre dentro de la etiqueta de nuestro documento HTML.
- Un código, situado en el interior de la etiqueta , que delimitara por medio de líneas geométricas imaginarias cada una de las áreas de los enlaces presentados en la imagen. Las líneas geométricas que delimitan los enlaces, es decir, las áreas de los enlaces, han

de ser definidas por medio de coordenadas. Cada imagen es definida por unas dimensiones de ancho (X) y alto (Y) y cada punto de la imagen puede ser definido por tanto diciendo a que altura (x) y anchura (y) nos encontramos. De este modo, la esquina superior izquierda corresponde a la posición 0,0 y la esquina inferior derecha corresponde a las coordenadas X,Y. Si deseamos saber qué coordenadas corresponden a un punto concreto de nuestra imagen, lo mejor es utilizar un programa de diseño grafico como Photoshop o Paint Shop Pro. La mejor forma de explicar el funcionamiento de este tipo de mapas es a partir de un ejemplo práctico. Supongamos que tenemos una imagen con un mapa como esta: Pulsa en los círculos para acceder a las secciones!

Dentro de ella queremos introducir un enlace a cada uno de los elementos que la componen. Para ello, definiremos nuestros enlaces como zonas circulares de pequeño tamaño que serán distribuidas a lo largo y ancho de la imagen. Veamos a continuación el código que utilizaremos:



Pulsa en los círculos para acceder a las secciones!

3. Nota: Los href de las áreas van a # Este es un ejemplo parcial de utilización de los mapas, faltaría colocar los href con valores reales y no con la #. Cada uno de los enlaces de las áreas

- atributo href de la etiqueta - deberían llevar a una página web. El ejemplo quedaría completo si creasemos todas las páginas donde enlazar las áreas y colocasemos los href dirigidos hacia dichas páginas. Como no hemos hecho las páginas "destino" hemos colocado enlaces que no llevan a ningún sitio, que, como puedes ver, se indica con el caracter "#". Podéis observar, tal y como hemos explicado antes, que nuestro mapa consta de dos partes principales: la imagen y la etiqueta que

define las áreas de cada enlace. Cada área se indica con una etiqueta , que tiene los siguientes atributos: alt Para indicar un texto que se mostrará cuando situemos el ratón en el área. shape Indica el tipo de área. coords Las coordenadas que definen el área. Serán un grupo de valores numéricos distintos dependiendo del tipo de área (shape) que estemos definiendo. href Para indicar el destino del enlace correspondiente al área. En este caso hemos utilizado unas áreas circulares (shape="CIRCLE"), que se

definen indicando el centro del círculo -una coordenada (X,Y) y el radio, que es un número entero que se corresponde con el número de pixels desde el centro hasta el borde del círculo. Tipos de áreas: shape distintas.

Existen tres tipos de áreas distintas, suficientes para hacer casi cualquier tipo de figura. En el dibujo que acompaña estas líneas se puede ver una representación de las áreas, que detallamos a continuación.

`shape="RECT"` Crea un área rectangular. Para definirla se utilizan las

coordenadas de los puntos de la esquina superior izquierda y la esquina inferior derecha. Tal como están nombradas dichas coordenadas en nuestro dibujo, el área tendría la siguiente etiqueta:

`shape="CIRCLE"` Crea un área circular, que se indica con la coordenada del centro del círculo y el radio. A la vista de nuestro dibujo, la etiqueta de un área circular tendría esta forma: `shape="POLY"`

Este tipo de área, poligonal, es la más compleja de todas. Un polígono queda definido indicando todos sus

puntos, pero atención, los tenemos que indicar en orden, siguiendo el camino marcado por el perímetro del polígono. A la vista del dibujo y los nombres que hemos dado a los puntos del polígono, la etiqueta quedaría de esta forma. Frames en HTML Una de las más modernas características de HTML son los frames, que se añadieron, tanto en Netscape Navigator como en Internet Explorer, a partir de sus versiones 2.0. Los frames -que significan en castellano marcos- son una manera de partir la página en

distintos espacios independientes los unos de los otros, de modo que en cada espacio se coloca una página distinta que se codifica en un fichero HTML distinto. Al principio se crearon como etiquetas propietarias del navegador Netscape y rápidamente la potencia del recurso hizo que el uso de frames se extendiera por toda la web. Poco tardaría Internet Explorer en incluirlos, para que no se le escapase una novedad tan popular de su competidor. Finalmente, como respuesta a la popularidad entre los

desarrolladores de los frames, el estándar HTML 4.0 incluyó estas etiquetas dentro de las permitidas. Los frames, como decíamos, nos permiten partir la ventana del navegador en diferentes áreas. Cada una de estas áreas son independientes y han de ser codificadas con archivos HTML también independientes. Como resultado, cada frame o marco contiene las propiedades específicas que le indiquemos en el código HTML a presentar en ese espacio. Así mismo, y dado que cada marco es

independiente, tendrán sus propias barras de desplazamiento, horizontales y verticales, por separado. Existen en la web muchas páginas que contienen frames y seguro que todos hemos tenido la ocasión de conocer algunas. Se suelen utilizar para colocar en una parte de la ventana una barra de navegación, que generalmente se encuentra fija y permite el acceso a cualquier zona de la página web. Una de las principales ventajas de la programación con frames viene derivada de la

independencia de los distintos frames, pues podemos navegar por los contenidos de nuestro web con la barra de navegación siempre visible, y sin que se tenga que recargar en cada una de las páginas que vamos visitando. Un ejemplo de las áreas que se pueden construir en una construcción de frames se puede ver en las imágenes siguientes. Para el que no haya tenido oportunidad de conocer los frames y su funcionamiento, ofrecemos un enlace a una página cualquiera de Internet que los utiliza.

Además, podemos ver uno de los ejemplos del tema de frames que simula la página de una carnicería.

Frames - Explicación básica
Las páginas web que están hechas con frames se componen de una declaración de los marcos y tantas páginas en formato HTML corriente como distintas divisiones hemos definido. La declaración o definición de frames es la única página que realmente debemos aprender, puesto que las páginas que se van a visualizar en cada uno de los cuadros son ficheros

HTML de los que venimos aprendiendo anteriormente en este manual. Dicha definición está compuesta por etiquetas y , con las que se indicamos la disposición de todos los cuadros. La etiqueta indica las particiones de la ventana del navegador y la etiqueta indica cada uno de los cuadros donde colocaremos una página independiente. Las particiones que se pueden hacer con un son en filas o columnas. Por ejemplo, podríamos indicar que deseamos hacer una división de la página en

dos filas, o dos columnas, tres filas, etc. Para indicar tanto la forma de partir la ventana -en filas o columnas- como el número de particiones que pretendemos hacer, se ha de utilizar el atributo COLS o ROWS. El primero sirve para indicar una partición en columnas y el segundo para una partición en filas. Nota: Es importante indicar que no se puede hacer una partición en filas y columnas a la vez, sino que debemos escoger en partir la ventana en una de las dos disposiciones. Más adelante indicaremos

cómo partir la ventana tanto en filas como en columnas, que se hace con la anidación de frames. En el atributo COLS o ROWS -sólo podemos elegir uno de los dos- colocamos entre comillas el número de particiones que deseamos realizar, indicando de paso el tamaño que va a asignarse a cada una. Un valor típico de estos atributos sería el siguiente:

cols="20%,80%" Indica que se deben colocar dos columnas, la de la izquierda tendría un 20% del espacio total de la ventana y la de la derecha

un 80%.

rows="15%,60%,25%"

Así indicamos que deseamos tres filas, la de arriba con un 15% del espacio total, la del medio con un espacio correspondiente al 60% del total y la de abajo con un 25%. En total suman el 100% del espacio de la ventana. Además del porcentaje para indicar el espacio de cada una de las casillas, también podemos indicarlo en pixeles. De esta manera.

cols="200,600" Para indicar que la columna de la izquierda debe tener 200 pixels de ancho y la de

la derecha 600. Esto está bien si nuestra ventana tiene 800 pixels de ancho, pero esto no tiene porque ser así en todos los monitores de los usuarios, por lo que este modo de expresar los marcos es importante que se indique de la siguiente manera.

`cols="200,*"` Así indicamos que la primera columna ha de medir 200 pixels y que el resto del espacio disponible -que será mayor o menor dependiendo de la definición de la pantalla del usuario- se le asignará a segunda columna. En la práctica podemos mezclar

todos estos métodos para definir los marcos de la manera que deseemos, con porcentaje, con pixels o con el comodín (*). No importa cómo se definan, la única recomendación es que uno de los valores que indiquemos sea un asterisco, para que el área correspondiente a dicho asterisco o comodín sea más o menos grande dependiendo del espacio que tenga la ventana de nuestro navegador. Otros métodos de definir filas y columnas, atendiendo a este consejo, serían los siguientes:

rows="100,*,12%"

Definimos tres filas, la primera con 100 pixels de ancho, la segunda con el espacio que sobre de las otras dos, y la tercera con un 12% del espacio total.

`cols="10%,50%,120,*"`

Estamos indicando cuatro columnas. La primera del 10% del espacio de la ventana, la segunda con la mitad justa de la ventana, la tercera con un espacio de 120 pixels y la última con la cantidad de espacio que sobre al asignar espacio a las demás particiones. Una vez hemos indicado el número de filas o columnas y el espacio reservado a cada

una con la etiqueta ,
debemos especificar con
la etiqueta la procedencia
de cada uno de los frames
en los que hemos partido
la ventana. Para ello,
disponemos del atributo
SRC, que se ha de definir
para cada una de las filas o
columnas. De esta
manera. Así queda
indicado que el frame que
estamos definiendo debe
mostrar la página
marco1.html en su
interior. Frames –
Creación de una
estructura simple Para
ilustrar todo lo que
venimos explicando
podemos ver el ejemplo

sobre cómo se crearía la definición de frames de la imagen que podemos ver a continuación. Se puede ver esta partición de frames en una página a parte. Además tenemos algunas consideraciones que hacer para terminar de comprender este ejemplo: • El título de la definición de frames es el que hereda toda la página web, por ello, no es buena idea titular como "definición de frames" por ejemplo, ya que entonces toda nuestra página se titularía así y seguramente no sea muy descriptivo. Si

estuviésemos haciendo una página para la carnicería pepe sería mejor titular a la definición de frames algo como "Carnicería Pepe, las mejores carnes en Madrid". • La página que define los frames no tiene body. HTML puede arrojarnos un error si lo incluimos. • Las páginas "pagina1.html", "pagina2.html"y "pagina3.html" han de escribirse en archivos independientes con el nombre indicado. En este ejemplo, dichas páginas deberían encontrarse en el mismo directorio que la

declaración de frames. Si especificamos una ruta para acceder al archivo podemos colocarlo en el directorio que deseemos. • Los colores de cada uno de los frames los hemos colocado con el atributo bgcolor colocado en la etiqueta de cada una de las páginas que se muestran en los marcos. Frames – Una página en cada marco Las páginas que mostraremos en cada marco son documentos HTML iguales a los que venimos creando anteriormente. Podemos colocar cualquier elemento HTML de los

estudiados en este manual, como etiquetas de párrafo, imágenes, colores de fondo, etc. Cada documento, como ya hemos indicado, se escribe por separado en su propio archivo HTML. Para el ejemplo del capítulo anterior podemos definir los archivos HTML de la siguiente manera. pagina1.html Es la página que contiene el titular de la web. Simplemente se trata de una etiqueta

de titular. La página tiene su propio título, con la etiqueta

Carnicería PEPE

1.pagina2.html Es la página que se presentará en el área principal de la definición de frames, es decir, la página que tiene más espacio para visualizarse y donde pondremos los contenidos de la web. En este caso muestra un mensaje de bienvenida a la web, que hará las veces de portada.

Bienvenidos a nuestra web

1.

La carnicería PEPE, con más de 100 años de experiencia, es la mejor fuente de carnes de vacuno y cerdo de la comunidad.

Tanto en invierno como en verano puede encontrar nuestras ofertas de temporada de primera calidad.
pagina3.html En esta página se mostrará la barra de navegación por los contenidos del sitio. Contiene enlaces que deberían actualizar el contenido del área principal de la declaración de frames, para mostrar los distintos contenidos del sitio, por ejemplo, la portada, los productos, la página de contacto, etc.

Portada | Productos | Contacto

Podemos ver cómo queda la página de frames con estos contenidos, que simulan la página de una carnicería.

Frames – Dirigir los enlaces La única particularidad destacable en el ejemplo del capítulo anterior, y en el manejo de frames en general, se trata de que cada uno de los enlaces que colocamos en

las páginas actualizan el frame donde está colocado este enlace. Por ejemplo, si tenemos enlaces en la parte inferior de la ventana, en el espacio correspondiente al tercer marco, actualizarán los contenidos del tercer frame, que es donde están situados los enlaces. Este efecto que comentamos se puede observar en el ejemplo de la página de la carnicería, tal como quedaría al incluir los códigos de las distintas páginas. Lo lógico es que al pulsar sobre un enlace de la barra de navegación actualicemos el frame principal, que es donde habíamos planeado colocar los contenidos, en lugar del frame donde colocamos la barra de navegación, que debería mantenerse fija. Para conseguir este efecto debemos hacer un par de cosas:

1. Darle un nombre al frame que

deseamos actualizar Dicho nombre se indica en la etiqueta de la definición de frames. Para ello utilizamos el atributo name, igualado al nombre que le queremos dar a dicho marco. 2. Dirigir los enlaces hacia ese frame Para ello debemos colocar en el atributo target de los enlaces -etiqueta - el nombre del frame que deseamos actualizar al pulsar el enlace. Después de darle un nombre al frame principal, nuestra declaración de frames quedaría de la siguiente manera. Además, deberíamos colocar el atributo target a los enlaces, tal como sigue. Portada | Productos | Contacto Una vez realizados este par de cambios podemos ver como los enlaces de la barra de navegación sí actualizan la página que deben. Valores para el atributo target Como hemos visto, con

el atributo target de la etiqueta podemos indicar el nombre del frame que deseamos que actualice ese enlace. Sin embargo, no es este el único valor que podemos aplicarle al atributo. Tenemos algunos valores adicionales que podemos asignar a cualquier enlace en general. `_blank` Para hacer que ese enlace se abra en una ventana a parte. Nuestros ejemplos en este manual se suelen abrir en una ventana a parte, colocando este valor en el target de los enlaces que llevan a los ejemplos. `_self` Se actualiza el frame donde está situado el enlace. Es el valor por defecto. `_parent` El enlace se actualiza sobre su padre o sobre la ventana que estamos trabajando, si es que no hay un padre. `_top` La página se carga a pantalla completa, es decir, eliminando todos los frames que

pudiera haber. Este atributo es muy importante porque si colocamos en nuestra página con frames un enlace a una página externa, se abriría en uno de los frames y se mantendrían visibles otros frames de la página, haciendo un efecto que suele ser poco agradable, porque parece que están evitando que nos escapemos. La sintaxis de uno de estos valores de atributos colocados en un enlace sería la siguiente. [Acceder a guiarte.com](http://guiarte.com)

Frames – Anidar frames Para crear estructuras de marcos en las que se mezclen las filas y las columnas debemos anidar etiquetas . Empezando por la partición de frames más general, debemos colocar dentro las particiones de frames más pequeñas. La manera de indicar esto se puede ver fácilmente con un ejemplo. Los pasos para definir

la anidación se pueden encontrar a la derecha. Los distintos frames vienen numerados con el orden en el que se escriben en el código. En la imagen se puede ver el resultado final acompañada de la representación sobre la manera de definirlos. En primer lugar definimos una estructura de frames en dos columnas y dentro de la primera columna colocamos otra partición de frames en dos filas. El código necesario es el siguiente. Podemos ver el ejemplo en una página a parte. Nota: hemos colocado un margen en cada una de las líneas de esta definición de frames para conseguir un código más entendible visualmente. Estos márgenes no son en absoluto necesarios, simplemente nos sirven para ver en qué nivel de anidación nos encontramos. El

ejemplo anterior se puede complicar un poco más si incluimos más particiones. Vamos a ver algo un poco más complicado para practicar más con las anidaciones de frames. Los pasos para definir la anidación se pueden encontrar a la derecha. Los distintos frames vienen numerados con el orden en el que se escriben en el código. En la imagen se observa que el primer frameset a definir se compone de dos filas. Posteriormente, dentro de la segunda fila del primer frameset, tenemos otra partición en dos columnas, dentro de las que colocamos un tercer nivel de frameset con una definición en filas en los dos casos. El código se puede ver a continuación. Podemos ver el ejemplo en una página a parte. Hasta aquí hemos visto la parte más básica de la creación de frames. En

los siguientes capítulos podremos aprender a configurar los marcos para variar su apariencia y, entre otras cosas, eliminar las barras que separan cada uno de los distintos frames.

Frames - Atributos avanzados Aparte de la creación de los marcos propiamente dicha, existen muchos atributos con los que configurar su apariencia. Para ello, tanto la etiqueta como admiten diversos atributos que permiten especificar la forma de elementos como los bordes de los frames, el margen, la existencia o no de barras de desplazamiento, etc.

Atributos para la etiqueta Ya hemos conocido el atributo cols y rows, que sirven para indicar si la distribución en marcos se hará horizontalmente o verticalmente. Sólo se puede utilizar uno de ellos y se iguala a las

dimensiones de cada uno de las divisiones, separadas por comas.

border="número de pixels" Permite especificar de manera global para todo el frameset el número de pixels que ha de tener el borde de los frames.

bordercolor="#rrggbb" Con este atributo podemos modificar el color del borde de los frames, también de manera global a todo el frameset.

frameborder="yes|no|0" Sirve para mostrar o no el borde del frame. Sus posibles valores son "yes" (para que se vean los bordes) y "no" o "0" (para que no se vean). En la práctica elimina el borde, pero permanece una línea de separación de los frames.

framespacing="número de pixels"
Para determinar la anchura de la línea de separación de los frames. Se puede utilizar en Internet Explorer y junto

con el atributo `frameborder="0"` sirve para eliminar los bordes de los marcos.

Atributos para la etiqueta Para esta etiqueta hemos señalado en capítulos anteriores los atributos `src`, que sirve para indicar el archivo que contiene el marco y `name`, para darle un nombre al marco y luego dirigir los enlaces hacia el. Veamos ahora otros atributos disponibles.

`marginwidth="número de pixels"` Define el número de pixels que tiene el margen del frame donde se indica. Este margen se aplica a la página que pretendemos ver en ese marco, de modo que si colocamos 0, los contenidos de la página en ese marco estarán pegados por completo al borde del margen y si indicamos un valor de 10, los contenidos de la página estarían separados del borde 10 pixels.

`marginheight="número de pixels"` Lo

mismo que el anterior atributo, pero para el margen vertical.

scrolling="yes|no|auto" Sirve para indicar si queremos que haya barras de desplazamiento en los distintos marcos. Si indicamos "yes" siempre saldrán las barras, si indicamos "no" no saldrán nunca y si colocamos "auto" saldrán sólo si son necesarias.

Auto es el valor por defecto. Consejo: hay que tener cuidado si eliminamos los bordes de los frames, puesto que la página web puede tener dimensiones distintas dependiendo de la definición de pantalla del visitante. Si el espacio de la ventana se ve reducido, podría verse reducido el espacio para el frame y puede que no quepan los elementos que antes si que cabían y si hemos eliminado las barras de desplazamiento puede que el visitante

no pueda ver todo el contenido del marco. Este mismo consejo se puede aplicar al redimensionamiento de frames, que veremos en el siguiente atributo. Si hacemos que los marcos no sean redimensionables probablemente tengamos una declaración de frames demasiado rígida, que puede verse mal en algún tipo de pantalla. `noresize` Este atributo no tiene valores, simplemente se pone o no se pone. En caso de que esté presente indica que el frame no se puede redimensionar. Como hemos podido ver, al colocar el ratón sobre el borde de los marcos sale un cursor que nos señala que podemos mover dicho borde y redimensionar así los frames. Por defecto, si no colocamos nada, los marcos si se pueden redimensionar. `frameborder="yes|no|0"` Este atributo permite controlar la aparición de los

bordes de los frames. Con este atributo igualado a "0" o "no" los bordes se eliminan. Sin embargo, quedan los feos márgenes en el borde. Por lo que hemos podido comprobar funciona mejor en Netscape que en Internet Explorer. De todos modos, tenemos una nota un poco más adelante para explicar los frames sin bordes. Nota: los atributos de frames no funcionan siempre bien en todos los navegadores. Es recomendable que hagamos un test sobre lo que estamos diseñando en varios navegadores para comprobar que nuestros frames se ven bien en todas las plataformas.

`bordercolor="#rrggbb"` Permite especificar el color del borde del marco. Referencia: Tenemos un taller de HTML en el que explicamos como realizar un frame sin bordes que se vea

bien en los navegadores más habituales. Ventajas e inconvenientes del uso de frames El diseño con frames es un asunto bastante controvertido, ya que distintos diseñadores tendrán unas u otras opiniones. Referencia: Si deseas saber qué son los frames y cómo crearlos consulta los capítulos de Frames de nuestro manual de HTML. En mi caso, pienso que es preferible no utilizarlos, aunque eso depende del tipo de sitio web que estés construyendo, ya que en algunos casos sí que sería muy adecuado su uso. Voy a colocar unas ventajas e inconvenientes del uso de marcos (frames). Siempre es a mi entender, otros pueden tener otras opiniones.

Ventajas de usar frames

- La navegación de la página será más rápida. Aunque la primera carga de la

página sería igual, en sucesivas impresiones de páginas ya tendremos algunos marcos guardados , que no tendrían que volverse a descargar. • Crear páginas del sitio sería más rápido. Como no tenemos que incluir partes de código como la barra de navegación, título, etc. crear nuevas páginas sería un proceso mucho más rápido. • Partes de la página (como la barra de navegación) se mantienen fijas y eso puede ser bueno, para que el usuario no las pierda nunca de vista. • Estas mismas partes visibles constantemente, si contienen enlaces, pueden servir muy bien para mejorar la navegación por el sitio. • Mantienen una identidad del sitio donde se navega, pues los elementos fijos conservan la imagen siempre visible. Inconvenientes de usar frames •

Quitan espacio en la pantalla. El espacio ocupado por los frames fijos se pierde a la hora de hacer páginas nuevas, porque ya está utilizado. En definiciones de pantalla pequeña o dispositivos como Palms, este problema se hace más patente. •

Fuerzan al visitante a entrar por la declaración de frames. Si no lo hacen así, sólo se vería una página interior sin los recudros. Estos recuadros podrían ser insuficientes para una buena navegación por los contenidos y podrían no conservar una buena imagen corporativa. • La promoción de la página sería, en principio, más limitada. Esto es debido a que sólo se debería promocionar la portada, pues si se promocionan páginas interiores, podría darse en caso de que los visitantes entrasen por ellas en lugar

de por la portada, creandose el problema descrito en el punto anterior.

- A mucha gente les disgustan pues no se sienten libres en la navegación, pues entienden que esas partes fijas están limitando su movilidad por la web.

Este efecto se hace más patente si la página con frames tiene enlaces a otras páginas web fuera del sitio y, al pulsar un enlace, se muestra la página nueva con los marcos de la página que tiene frames.

- Algunos navegadores no los soportan. Esto no es muy habitual, pero si estamos haciendo una página que queramos que sea totalmente accesible deberíamos considerarlo importante.
- Los bookmarks o favoritos no funcionan correctamente en muchos casos. Si queremos incluir un favorito a una página de un frame que no sea la portada podemos

encontrar problemas. • Puede que el botón de atrás del navegador no se comporte como deseamos. • Si quieres actualizar más de un frame con la pulsación de un enlace deberás utilizar Javascript. Además los scripts se pueden complicar bastante cuando se tienen que comunicar varios frames entre si.

Conclusión

El trabajo con frames puede ser más o menos indicado dependiendo de las características de la página a desarrollar, es tu tarea saber si en tu caso debes utilizarlos o no.

Las nuevas etiquetas de HTML 4.0

Introducción.

Cuando Internet empezaba su imparable escalada, la versión del estándar HTML que circulaba era la 2.0, el cuál siguen soportando los navegadores más actuales. Pero las herramientas de que se disponía no

ofrecían un control preciso de los documentos. Pero como por aquel entonces el objetivo de Internet estaba fundamentalmente orientado al ámbito académico y no al de diseño, no se le dio demasiada importancia a la cuestión de lanzar una versión mejorada del estándar hasta que Netscape, que por aquel entonces era la empresa líder en el sector, tomó la iniciativa de incluir nuevas etiquetas pensadas para mejorar el aspecto visual de las páginas web. Por este motivo el IETF (Internet Engineering Task Force)

www.ietf.cnri.reston.va.us, o lo que es lo mismo, Grupo de Trabajo en Ingeniería de Internet, comenzó a elaborar nuevos estándares, los cuales dieron como fruto el HTML 3.0, que resultó ser demasiado grande para las

infraestructuras que había en ese momento, lo cual dificultó su aceptación. Así pues, una serie de compañías (entre las que estaban Netscape, Sun Microsystems o Microsoft, entre otras), se unieron para crear lo que hoy se denomina W3C (o lo que es lo mismo, Consorcio para la World Wide Web), que fue fundado en octubre de 1.994 para conducir a la World Wide Web a su máximo potencial, desarrollando protocolos de uso común, para normalizar el uso de la web en todo el mundo. El compromiso del W3C de encaminar a la Web a su máximo potencial incluye promover un alto grado de accesibilidad para las personas con discapacidades. El grupo de trabajo permanente Web Accessibility Initiative (WAI, Iniciativa para la

Accesibilidad de la Red), en coordinación con organizaciones alrededor de todo el mundo, persigue la accesibilidad de la Web a través de cinco áreas de trabajo principales: Tecnología, directrices, herramientas, formación, difusión, e investigación y desarrollo. De esta iniciativa nació el borrador de HTML 3.2 y en su versión definitiva se introdujeron cambios esenciales para las posibilidades que empezaban a ofrecer los navegadores, estas inclusiones fueron las tablas, los applets, etc. En julio de 1.997 nace el borrador del HTML 4.0 y finalmente se aprueba en diciembre de 1.997 este estándar incluía como mejoras los marcos (frames), las hojas de estilo y la inclusión de scripts en páginas web, entre otras cosas. Las nuevas etiquetas de HTML 4.0 (1) Entre el estándar del

HTML 3.2 al 4.0 se introdujeron ocho nuevas etiquetas de las cuales daremos una breve explicación. ... Las etiquetas y actúan de forma muy parecida a pero con la particularidad de que añade un sangrado en párrafos más pequeños y sin necesidad de romper el párrafo. Según el W3C, la etiqueta es para añadir sangrados largos y , para sangrados más pequeños, sin necesidad de romper el párrafo. Nota: En el HTML 4.0 es imprescindible poner la etiqueta de apertura y la de clausura Las etiquetas ... , indican que hay un acrónimo en el texto. Un acrónimo es un pequeño texto que ayuda a explicar la estructura del texto una frase. ... y ... Utilice < INS>... para marcar las partes de un documento que se han agregado desde la versión pasada del documento. ...

marca de manera similar un texto de un documento que se ha suprimido desde la versión anterior. ... Se utiliza para tener un mejor control sobre el formato de las tablas especificando las características que comparten como: anchura, altura y alineación. Cada tabla debe tener por lo menos un ; sin especificar ninguna característica de < COLGROUP >. HTML 4.0 asume que una tabla contiene un solo grupo de columnas y que este contiene todas las columnas de una tabla. Por ejemplo, esto nos serviría para crear una tabla con una celda en la que puede incluirse una descripción y después seguido de check boxes para seleccionar las opciones deseadas. Código: ... De esta forma, proporciona un formato más agradable a los check boxes sin necesidad de especificar, propiedades

identicas para cada fila. La etiqueta de inicio < COLGROUP >, requiere otra de cierre. Con el que obtenemos: (en Netscape sólo se verá la tabla, no el botón). Las nuevas etiquetas de HTML 4.0 (2) ... Hasta ahora, no disponíamos de ninguna manera de agrupar visualmente varios controles, si no echábamos mano de elementos que no son del formulario, como tablas o imagenes. Ahora, si encerramos una parte de un formulario dentro de la etiqueta FIELDSET se mostrara un rectángulo alrededor de los mismos. Además, podemos indicar un título por medio de la etiqueta LEGEND, que admite el parámetro align="left / center / right / top / bottom", lo que nos permite alinear el título horizontal y verticalmente. La única pega es que deberemos introducir el conjunto en

una celda de tabla con un ancho determinado, ya que si no lo hacemos así el recuadro abarcará todo el ancho de pantalla disponible. Ejemplo.- (Sólo para I. Explorer)

2.

Caja de
texto pon tu
nombre:

3. ... Hasta no hace mucho los campos de entrada no estaban asociados a ellos mismos. Por ejemplo; a la hora de pulsar sobre un campo de confirmación, ¡no sucedía nada! Pero ahora, sí lo pulsamos el control cambiará de estado. Ejemplo: Le deseamos un feliz año nuevo ... A partir de la implementación de los estándares HTML 4.0 contamos con varias etiquetas nuevas para construir

formularios, siendo BUTTON una de ellas, bastante útil por cierto. La pega es que las versiones de 4 de Netscape se lanzaron antes de estas implementaciones, por lo que estas nuevas etiquetas sólo se pueden visualizar correctamente con Internet Explorer 4 y superiores. Esta etiqueta proporciona un método único para la implementación de cualquier tipo de botón de formulario. Sus principales atributos son:

- `type= " tipo "`, que puede tomar los ya conocidos valores `submit` (por defecto), `reset` y `button`.
- `name= " nombre "`, que asigna un nombre identificador único al botón.
- `value= " texto "`, que define el texto que va a aparecer en el botón.

La principal ventaja que aporta estas etiquetas es que ahora vamos a poder introducir dentro de ellas cualquier

elemento de HTML, como imagenes y tablas. Ejemplos.

4.

u n o	do s
tr e s	cu atr o

**2. Sonido en HTML I,
introducción En su corta
pero rápida vida, las
páginas web han pasado a
ser no ya unos meros
documentos textuales a
los que se puede acceder
por Internet, sino unas
verdaderas
presentaciones**

multimedia, que combinan textos con imágenes, sonidos, videos y elementos de realidad virtual. Si el primer paso que se dio fue añadir imágenes a las páginas web, tanto estáticas como dinámicas GIF animados), el siguiente paso consistió en introducir sonidos en las mismas, consiguiendo con ellos el apelativo de “multimedia”. Y nos referiremos en lo sucesivo cuando hablemos de sonido tanto a sonido sintetizado como a verdaderas grabaciones de audio, de calidad muy elevada. Ahora bien,

aunque los navegadores han sido capaces de interpretar los ficheros de sonido adecuados desde hace ya algunas versiones, es cierto que la aplicación de sonidos a las páginas web ha estado limitada desde siempre por el ancho de banda necesario en las conexiones a Internet para poder descargar de forma adecuada dichos ficheros, debido al tamaño “excesivo” de los mismos. Otra de las limitaciones importantes que encontramos a la hora de incluir ficheros de sonido en nuestras páginas es la

diferentes
implementación que
hacen de ellos los
navegadores web más
usados. En efecto, no sólo
deberemos usar etiquetas
HTML distintas para
Internet Explorer que para
Netscape Navigator, sino
que a veces la forma
misma de interpretar el
sonido puede diferir de
uno a otro navegador. Por
último, hay que destacar
que a la hora de incluir
ficheros de audio en
nuestras páginas debemos
ser conscientes que
muchos de los formatos
usados, sobre todo en
grabaciones de calidad,

precisan un plugin o programa especial para su reproducción en el navegador cliente. Y si es cierto que actualmente hay ciertos plugins se han transformado casi en un estándar en Internet (como el de Real Audio o el de MP3), hay otros posibles que no es normal tener instalados, por lo que si incluimos ficheros de esos tipos obligaremos al usuario a tener que instalarlos, cosa a la que suele ser reacio. Sonido en HTML II, características del sonido digital Vamos a estudiar algunos de los conceptos básicos del

sonido digital, aunque sin entrar en demasiadas consideraciones técnicas. Para aquellos que deseen más información, existen multitud de sitios web que estudian específicamente el sonido digital y el hardware necesario para su captura y reproducción. El sonido tiene una naturaleza ondulante, es decir, se propaga en forma de ondas analógicas desde el objeto que lo produce. Las características propias de cualquier sonido (desde el producido por un automóvil hasta una bella canción), sus diferentes tonos y notas dependen

precisamente de las propiedades físicas de las ondas que lo forman. Para poder viajar desde el emisor al receptor, las ondas de sonido precisan de un medio físico de soporte, ya sea el aire de la atmósfera, al agua, etc. Tanto es así que en el espacio exterior, donde no hay medio físico soporte, no se pueden transmitir sonidos. Si representamos en un gráfico un sonido complejo, obtendremos la siguiente figura: En la que podemos apreciar los diferentes valores de onda que va tomando el sonido. Todos sabemos que los

equipos informáticos no trabajan con datos analógicos, sino que lo hacen con datos digitales, formados por estados binarios. Por lo tanto, para representar un sonido, desde el punto de vista informático, es preciso capturarlo en una naturaleza binaria, para lo que se hace un muestreo del mismo, tomando determinados valores de las ondas y representando dichos valores en formato digital. En cada captura obtendremos un punto de la gráfica anterior. Pero, ¿Cuántas muestras deberemos tomar?. Este es

el verdadero meollo de la cuestión, ya que cuantas más muestras tomemos, más fiel será el sonido capturado respecto al original, con lo que tendrá más calidad. Para medir el número de capturas utilizamos la frecuencia del muestreo. Como un Herzio es un ciclo por segundo, la frecuencia de una captura en Herzios representa el número de capturas que realizamos en un segundo. Así, una frecuencia de muestreo de 20 KHz (20 Kilo Herzios = 20000 Herzios) realizará 20000 capturas de puntos cada segundo. El oído

humano es captar de captar la asombrosa cantidad de 44000 sonidos por segundo, es decir, 44 KHz. Por lo tanto, para que un sonido digital tenga suficiente calidad deberá estar basado en una frecuencia similar a ésta. En general, el valor estándar de captura de sonidos de calidad es de 44,1 KHz (calidad CD), aunque hay capturadoras de sonido profesionales que llegan hasta los 100 KHz, con objeto de obtener un mayor número de puntos sobre la muestra, consiguiendo una calidad

máxima. Otro concepto del que habréis oído hablar en torno al sonido digital es el número de bits de una tarjeta de sonido. El origen de esta magnitud es que, a la hora de capturar el sonido, no sólo es importante el número de muestreos tomados, sino también la cantidad de información capturada en cada uno de esos muestreos. Una vez capturado el sonido, para su posterior reproducción en un equipo informático es necesario mandar una serie de impulsos o posiciones a los altavoces para que creen el sonido a

partir de ellos. ¿Cómo?.
Bien, produciendo a partir
de esas posiciones
movimientos de las
membranas de los
altavoces, movimientos
que transforman de nuevo
el sonido digital en
analógico, estado en el
que es capaz de viajar por
el aire y producir los
estímulos necesarios en
nuestros tímpanos, con lo
que somos capaces de
percibir el sonido
“original”. Cuantas más
posiciones de información
se envíen a los altavoces,
mejor calidad tendrá el
sonido reproducido. Con
estas bases, se define el

número de bits de un sonido digital como el número de impulsos de información (posiciones) que se envían a los altavoces para su transformación en ondas analógicas. Las tarjetas de sonido actuales trabajan normalmente con 8 bits de información, con los que se pueden obtener $2^8=256$ posiciones (ceros y unos binarios), aunque hay algunas de mayor calidad que son capaces de trabajar con capturas de 16 bits, que originan $2^{16} = 65536$ posiciones de información. Como dato de referencia, los CDs

**actuales están basados en
sonido grabado a 44 Khz y
con un tamaño de muestra
de 16 bits. Estas medidas
se conocen con el nombre
de sonido de calidad CD.
Por último, una vez que el
sonido digital llega a
nuestros oídos, impactan
contra los tímpanos,
verdaderas membranas
especializadas que
vuelven a transformar las
ondas analógicas en
impulsos eléctricos, que
viajan hasta nuestro
cerebro, donde son
interpretados y producen
las sensaciones auditivas
que todos conocemos. Una
excepción al sonido**

**anteriormente descrito,
que podemos denominar
"de datos de sonido", es el
sonido sintetizado, en el
que no se realiza ninguna
captura de ondas sonoras
reales, sino que es sonido
totalmente digital,
generado directamente en
el equipo informático por
un reproductor digital
conocido con el nombre
de MIDI (Music
Instrument Digital
Interface). Cuando se
desea reproducir una nota
musical concreta, se envía
un comando MIDI al chip
sintetizador, que se
encarga de traducir ese
comando en una vibración**

especial que produce la nota. Mediante este sistema es posible crear melodías bastante aceptables, aunque nunca tendrán la calidad ni riqueza de una onda sonora natural capturada.

Sonido en HTML (III)

Formatos de sonido A la hora de incluir ficheros de sonido en nuestras páginas web debemos distinguir entre los que pueden ser directamente ejecutados por el navegador y aquellos que deben ser abiertos por un programa propio, que deberá tener el usuario instalado en su equipo

para poder reproducir el fichero. De forma general, podemos incluir en la web los siguientes tipos de ficheros de audio. • WAV (Wave form Audio File format): formato típico de la casa Windows, de elevada calidad, usado en las grabaciones de CDs, que trabaja a 44 KHz y a 16 bits. Consta básicamente de tres bloques: el de identificación, el que especifica los parámetros del formato y el que contiene las muestras. Su principal inconveniente es el elevado peso de los ficheros, por lo que su uso queda limitado en

Internet a la reproducción de ruidos o frases cortas.

La extensión de estos ficheros es .wav. Es soportado por Internet Explorer y Netscape 4x. •

AU (Audio File format): formato creado por la casa Apple para plataformas MAC, cuyos ficheros se guardan con la extensión .au • MIDI formato de tabla de ondas, que no guardan el sonido a reproducir, sino un código que nuestra tarjeta de sonido tendrá que interpretar. Por ello, este tipo de ficheros no puede almacenar sonidos reales, como voces o música rela

**grabada; sólo puede
contener sonidos
almacenables en tablas de
ondas. Como
contrapartida, los ficheros
MIDI, que se guardan con
extensión .mid, son de
pequeño tamaño, lo que
los hace idóneos para la
web. Es soportado por
Internet Explorer y
Netscape 4x. • MP3 (MPEG
1 Layer 3): desarrollado
por el MPEG (Moving
Picture Expert Group),
obtiene una alta
compresión del sonido y
una muy buena calidad
basándose en la
eliminación de los
componentes del sonido**

que no estén entre 20 hz y 16 Kh (los que puede oír el ser humano normal).

Tiene en cuenta el sonido envolvente (surround) y la extensión multilingüe, y guarda los ficheros con la extensión .mp3, y permite configurar el nivel de compresión, consiguiéndose calidades similares a las del formato WAVE pero con hasta 10 veces menos tamaño de fichero. Es soportado directamente sólo por Internet Explorer 5.5 y superiores. • MOD especie de mezcla entre el formato MIDI y el formato WAV, ya que por un lado almacena

el sonido en forma de instrucciones para la tarjeta de sonido, pero por otro puede almacenar también sonidos de instrumentos musicales digitalizados, pudiendo ser interpretados por cualquier tarjeta de sonido de 8 bits. No es un formato estándar de Windows, por lo que su uso es más indicado para sistemas Mac, Amiga o Linux. La extensión de los ficheros es .mod • μ -Law Format de calidad similar al formato WAV, es original de las máquinas NeXt, y guarda sus ficheros con la extensión

.au • Real Audio de calidad media, aunque permite ficheros muy comprimidos, que guarda con extensión .rmp o .ra. Para su reproducción hace falta tener instalado el plugin Real Audio. A la hora de trabajar con estos formatos de sonido, deberemos tener en cuenta las limitaciones en su uso, ya que muchos de ellos no pueden ser reproducidos más que en sistemas operativos concretos, y aún así, con plugins o programas específicos. En busca de la compatibilidad, si usamos Windows como sistema

operativo conviene usar para ficheros musicales a reproducir directamente en el navegador los formatos WAV y MIDI, que son los más compatibles. En cambio, si lo que deseamos es poder brindar a nuestros visitantes la opción de navegar con música ejecutable desde un programa externo, lo mejor es usar ficheros en formato MP3, ya que en la actualidad la mayoría de los navegantes tienen instalado en su equipo algún programa reproductor adecuado, pudiendo valer desde

software incluido en Windows, como Windows Media Player, hasta aplicaciones externas, como Winamp. En este caso, basta colocar un enlace normal en nuestras páginas, apuntando al fichero de sonido. Como ejemplo, si queremos enlazar en nuestra página un fichero MP3, bastaría con escribir: Pincha aquí para oír la música. Que nos da: Pincha aquí para oír la música Con esto, al pinchar el usuario el enlace, se lanzará la aplicación que tenga asociada con el tipo de fichero MP3, que

dependerá de la configuración interna de cada navegador y usuario. Un caso especial es Netscape 6x. Casi no admite directamente ningún tipo de formato de sonido incrustado en la página, al no venir configuradas por defecto las aplicaciones o plugins necesarios. Y en el caso de ficheros enlazados, Netscape 6x suele lanzar su propio reproductor, que suele ser de la casa AOL, precisando para la ejecución una serie de pasos para darse de alta en esa compañía como usuario del software.

Resumiendo: cada usuario tendrá configurada su máquina de forma particular, soliendo prevalecer el último software de sonido instalado, ya que estos programas suelen adueñarse de ciertos tipos de ficheros para su ejecución automática. Entre las aplicaciones posibles de ejecución de ficheros de audio, bien de forma directa o en forma de plugina para los navegadores, destacan Windos Media Player, Real Player, Winamp, Quick time, etc. Sonido en HTML (IV) Incluir sonidos en la

web. Una vez elegidos nuestros ficheros de sonido, es hora de incluirlos en nuestra página web. Lógicamente, para que un fichero de audio pueda ser reproducido por un navegador es necesario que su máquina tenga incluida una tarjeta de sonido y un par de altavoces. Existen diversas formas de incluir un fichero de audio en una página, formas que dependen del tipo de fichero y del navegador usado, y podemos usar diferentes etiquetas para cada una de ellas.

**BGSOUND La etiqueta
bgsound incorpora
sonidos de fondo en una
página web, sonidos que
se ejecutan
automáticamente al
cargarse la página. Es una
etiqueta propietaria de
Microsoft, por lo que sólo
es interpretada por
Internet Explorer,
admitiendo los formatos
de audio MID y WAV,
aunque generalmente
también acepta AU y MP3,
en versiones actuales del
navegador o mediante
plugins de uso general. Su
sintaxis general, con sus
atributos más
importantes, es del tipo:**

Donde: •

src="ruta_fichero" fija la ruta en la que se encuentra el fichero de audio a reproducir. La ruta puede ser relativa a nuestro sistema de carpetas local, absoluta respecto el sistema de carpetas del servidor web o una URL completa que localice el fichero en Internet. •

loop="1" determina el número de veces (1) que se debe ejecutar el fichero de audio. Si le damos el valor infinite, el fichero se reproducirá indefinidamente. •

balance="b" determina el balance del sonido entre

los dos altavoces del equipo, es decir, la potencia o intensidad con que se oirá en cada uno de ellos (derecho e izquierdo). Sus valores pueden estar entre -10,000 y +10,000, correspondiendo el valor 0 a un balance equilibrado entre los dos altavoces. • `volume="v"` fija el volumen al que se oirá el sonido, y sus valores pueden variar entre -10,000 (mínimo) y 0 (máximo). No es soportado por los equipos MAC. Ejemplo: Que podéis ver funcionando en esta ventana (sólo Internet

Explorer). La etiqueta bgsound admite muchas más propiedades (disabled, delay, id, class, controls, etc.). Asimismo, esta etiqueta es accesible en Internet Explorer mediante código JavaScript, pudiendo modificar en tiempo real sus propiedades balance, loop, src, y volume, aunque ésta última sólo es accesible en plataformas PC. Para una información completa sobre todas las propiedades y funcionalidades de este etiqueta podéis visitar la página correspondiente de Microsoft:

<http://msdn.microsoft.com/library/default.asp?url=/workshop/author/dhtml/reference/objects/backgroundsound.asp> EMBED

Nestcape Navigator implementó la etiqueta embed para incorporar ficheros de audio. Es ésta una etiqueta de carácter general, que se usa para la inclusión en las páginas web de todos aquellos archivos ajenos al navegador y que necesitan por lo tanto la ejecución de algún plugin para su interpretación.

Paradójicamente, Internet Explorer asumió después el uso de esta etiqueta

**para la inclusión de
ficheros de audio, para
llegar a interpretarla
mejor y ampliarla con más
atributos y propiedades,
de tal forma que la
ejecución de sonidos con
embed es actualmente
más cómoda con este
navegador, al incorporar
la suite de Microsoft sus
propios plugins para la
interpretación de los
diferentes formatos de
audio. En cambio, si
usamos Netscape
Navigator nos
encontraremos en
muchos casos con un fallo
en la reproducción o con
un engorroso mensaje de**

necesidad de algún plugin especial (sobre todo en las versiones 6x), lo que nos obligará a visitar la página de Netscape para su descarga e instalación, que muchas veces no será efectiva. Sea como sea, hay que indicar que esta etiqueta nos va a incluir en la página web un objeto especial, una especie de consola de mando, denominada Crescendo, que consta de tres botones, similares al de cualquier reproductor de audio: un botón Play, para comenzar la reproducción (si no está establecida a automática), un botón

Pause, para detenerla momentáneamente y un botón Stop, para detenerla definitivamente (puesta a cero). Esta consola es diferente según el navegador usado; en el caso de Internet Explorer se muestra la típica consola de Windows Media, cuyo tamaño podemos configurar, mientras que en Netscape se muestra una consola propia, de tamaño fijo definido. La sintaxis general de la etiqueta embed es del tipo: Y en el caso que nos ocupa, de la inclusión de ficheros de audio, los atributos

podemos dividirlos en dos tipos: 1. Atributos

referentes al sonido: •

src="ruta_fichero", que fija la ruta en la que se encuentra el fichero de audio a reproducir. La ruta puede ser relativa a nuestro sistema de carpetas local, absoluta respecto el sistema de carpetas del servidor web o una URL completa que localice le fichero en Internet. •

loop="1/true/false", que determina el número de veces que se debe ejecutar el fichero de audio. Los valores admitidos son 1 (número entero de veces),

**true (infinitas veces) y
false (sólo una vez). Sólo
es reconocida por
Netscape Navigator. •
playcount="n", que define
el número de veces (n) que
se debe ejecutar en fichero
de audio en el caso de
Internet Explorer. •
type="tipo_fichero",
atributo importante, que
declara el tipo de fichero
de audio que estamos
usando, con lo que el
navegador web puede
ejecutar el programa o
plugin adecuado para la
reproducción del fichero.
Puede ser audio/midi,
audio/wav, etc. •
autostart="true/false",**

que determina si el fichero de audio debe empezar a reproducirse por sí sólo al cargarse la página o si por el contrario será preciso la actuación del usuario (o de código de script) para que comience la audición.

- `pluginspage="URL"`, que establece, en caso de ser necesario un plugin especial para reproducir el fichero, la página web donde se puede descargar el mismo. Sólo se activa en el caso de que el navegador no sea capaz de reproducir el fichero por sí mismo, y es soportada tan sólo por Netscape Navigator. •

**name="nombre", que
asigna un nombre
identificador (debe ser
único en la página) a una
etiqueta embed
determinada, con objeto
de ser accedida luego por
lenguajes de script. •
volume="v", que
determina el volumen de
reproducción del sonido, y
que puede variar entre 0 y
100. Es sólo soportada por
Netscape Navigator, que
en la consola muestra el
valor establecido en su
indicador de volumen,
siendo su valor por
defecto 50. En en caso de
Internet Explorer, el valor
del volumen por defecto**

es 50 en plataformas PC, y 75 en MAC, siendo necesario actuar sobre el control de volumen de la consola para modificarlo.

2. Atributos referentes a la consola: •

hidden="true/false", que establece si la consola va a ser visible (false) o no (true). Es éste un aspecto polémico, ya que si ocultamos la consola obligamos al usuario a oír nuestro fichero, sin posibilidad de detenerlo ni de modificar el volumen, y si la mostramos estaremos incrustando en la pantalla un objeto que muchas veces nos

romperá el esquema de diseño de nuestra página. Queda determinar su uso en cada caso concreto. •

`width="w"`, que determina el ancho visible de la consola, en pixels.

`height="h"`, que determina el alto visible de la consola, en pixels.

Estos atributos son también muy importantes, caso de que hayamos establecido `hidden="false"`, ya que de su valor va a depender la correcta visualización de la consola. En el caso de Internet Explorer, que muestra un logo de Windows Media sobre los

controles, el tamaño mínimo aceptable debe ser de 140x100 pixels, ya que si no la consola saldrá deformada en exceso o recortada. Y en el caso de Netscape Navigator, deberemos asignar unos valores de 145x60 pixels, que es lo que ocupa la consola; si ponemos un tamaño menor, la consola será recortada, perdiendo funcionalidades, y si asignamos un tamaño mayor, aparecerán espacios grises alrededor de la consola, afeando el aspecto de la página. Si no especificamos estos atributos y tampoco

hidden, nos aparecerán en la página tan sólo los mandos de la consola, sin logotipos añadidos (Internet Explorer) o la consola recortada (Netscape Navigator). •

align="top/bottom/center/baseline/left/right/texttop/middle/absmiddle/absbottom", análogo al de la etiqueta IMG, define la alineación horizontal o vertical de la consola respecto de los elementos de la página. •

hspace="hs", que establece la separación horizontal, vspace="vs", que establece la separación vertical, en

pixels, entre la consola y los elementos de la página que la redean. Análoga a sus equivalentes de la etiqueta IMG. Estos son los atributos principales, aunque podemos encontrar referencias de otros admitidos, aunque no suelen ser operativos en la realidad, ya que no suelen funcionar de forma correcta o son específicos de Netscape (como toda la serie de atributos que configuran los controles de la consola. Ejemplo sin consola: Que podemos ver en funcionamiento en esta ventana. Ejemplo con consola: Que tenemos

visible (y audible) en esta otra ventana. Sonido en HTML (V) La etiqueta OBJECT. Con objeto de normalizar la inclusión de ficheros no nativos en los navegadores web se decidió sustituir las diferentes etiquetas que realizaban este papel (APPLET, BGSOUND, EMBED, etc.), y que no pertenecían a los estándares web, por una etiqueta general, que fuera capaz de incrustar en el navegador todo tipo de ficheros. La etiqueta elegida en el estándar HTML 4.0 fué OBJECT, a la que se dotó de suficientes

atributos y flexibilidad para poder realizar correctamente su trabajo. Debido a esto, la propuesta ha sido usar la etiqueta object también para incluir ficheros de audio de todo tipo en las páginas web. Ahora bien, la aceptación e implementación que la misma a tenido varía según el navegador en particular, así como en función del objeto a incrustar. De este forma, Internet Explorer a realizado su propia implementación de la etiqueta object, incluyendo en ella

referencias a filtros y componentes ActiveX específicos para los ficheros de audio. Por su lado, los navegadores Netscape no soportan correctamente esta etiqueta para ficheros de este tipo.

Restringiéndonos a Internet Explorer, la polémica sigue, ya que en diferentes manuales nos encontraremos diferentes formas de incrustar sonidos mediante object, unas que funcionan bien, y otras que no. ¿Porqué sucede esto?. Yo creo que porque Microsoft ha ido usando la etiqueta object

**para implementar todo un
grán conjunto de
componentes propios, que
además han ido
adaptándose a las
diferentes versiones de
Internet Explorer. Como
regla general, válida no
sólo para incrustar
ficheros de sonido, sino
también para otros tipos,
la etiqueta object va a
definir un objeto o
componente externo
encargado de la
reproducción del fichero,
que en el caso de Internet
Explorer suele ser algún
tipo de control ActiveX.
Mediante object se
instancia el objeto, se**

declara su URL y sus principales propiedades generales, y mediante un conjunto de etiquetas especiales, PARAM, se le van pasando los valores que necesita para su correcto funcionamiento o para su configuración deseada. La sintaxis general de la etiqueta object, para el caso de ficheros de sonido, es del tipo: ... Los principales atributos de object, en referencia a ficheros de audio, son: •

classid="identificador_objeto", que fija la URL del objeto o componente externo

necesario para reproducir el fichero de audio, y la implementación CLSID de los controles ActiveX necesarios. •

type="tipo_fichero", atributo importante, que declara el tipo de fichero de audio que estamos usando. • width="w", que determina el ancho visible de la consola, en pixels. • height="h", que determina el alto visible de la consola, en pixels. • align="top/bottom/center/baseline/left/right/texttop/middle/absmiddle/absbottom", análogo al de la etiqueta IMG, define la alineación horizontal o

**vertical de la consola
respecto de los elementos
de la página. •**

**hspace="hs", que
establece la separación
horizontal, vspace="vs",
que establece la
separación vertical, en
pixels, entre la consola y
los elementos de la página
que la redean. Análoga a
sus equivalentes de la
etiqueta IMG. •**

**autostart="true/false",
que determina si el fichero
de audio debe empezar a
reproducirse por sí sólo al
cargarse la página o si por
el contrario será preciso la
actuación del usuario (o
de código de script) para**

que comience la audición.

- standby="mensaje", que presenta en pantalla un mensaje al usuario mientras el fichero se carga. En cuanto a los elementos param, los más importantes son:**
- param name="FileName" value="ruta_fichero", determina la ruta o URL del fichero de audio a reproducir. No es necesario utilizar sólo ficheros WAV o MID, pudiendo reproducirse también ficheros MP3 o Real Audio. El reproductor del primero lo incluye Explorer en ActiveMovie (componente de Windows**

Media). • param
name="autostart"
value="true/false", indica
al navegador si se debe
empezar a reproducir el
sonido automáticamente
al cargar la página o si por
el contrario será preciso
que el usuario pulse el
botón Play para ello. No
son estos todos los
atributos y parámetros
posibles. Es más, en
cuanto nos metemos en
componentes Microsoft,
podemos encontrarnos
multitud de
configuraciones posibles,
que nos van a permitir
fijar muchos aspectos de
los mismos. Dejo a cada

uno la posibilidad de profundizar en el estudio de aquellos componentes y propiedades que necesite, pero sabiendo que con los elementos vistos arriba tenemos más que suficiente para presentar un fichero de audio en nuestra página web. xEjemplo: > Que podéis ver funcionando en esta ventana (sólo Internet Explorer). La etiqueta A. Si hasta ahora hemos visto cómo podemos incluir en nuestras páginas sonidos de fondo o inicializados por el usuario mediante interacción con la consola

Crescendo, vamos a ver ahora cómo podemos implementar audio mediante el uso de una de las etiquetas más polivalentes en HTML: la etiqueta A. Efectivamente, los enlaces son la base del hipertexto, base a su vez de la web, y dentro de sus múltiples usos podemos considerar el enlace a ficheros de audio. El fichero enlazado puede ser interpretado directamente por el navegador (porque sea de reproducción directa o se tenga instalado el plugin adecuado) o puede ser ejecutado por un

**programa independiente
que se abra
automáticamente
(Winamp, Real Audio,
etc.), siendo este el caso
más común. Si el usuario
no dispone del programa o
plugin adecuado, se le
abrirá una ventana de
descarga del fichero, con
lo que podrá guardarlo
hasta disponer de la
aplicación necesaria para
su reproducción. La
sintaxis general en este
caso será del tipo:
Mensaje Ejemplo de
fichero MID: Música para
tí Que podemos ver en
funcionamiento en esta
ventana. Ejemplo de**

fichero MP3: Madonna

Que tenemos en esta otra ventana.

Los Desastres y el Medio Ambiente Los desastres y el medio ambiente están intrínsecamente vinculados. Lo que llamamos "desastres de origen natural" son eventos extremos que ocurren naturalmente dentro de un ecosistema. Estos eventos naturales extremos son el resultado de un cambio en las condiciones dentro de un ecosistema. Algunas veces el cambio puede ser un aumento repentino de la temperatura que causa el derretimiento rápido de la nieve de la montaña; desbordando arroyos y ríos y provocando inundaciones. A veces un evento extremo ocurre como resultado de cambio lento a través de un largo período de tiempo, tales como la desertificación. En algún caso el evento extremo puede ser un proceso que ocurre regularmente, tales como la inundación de tierras semiáridas que sirve para recargar los sistemas de agua subterránea y proporciona nutrientes al suelo. Igualmente importante es el rol que desempeñan los ecosistemas en la prevención o mitigación de daños causados por estos eventos extremos. Las dunas de arena, manglares y arrecifes de coral absorben la energía de olas poderosas inducidas por los ciclones tropicales. Los bosques costeros pueden servir como barreras contra el viento protegiendo las zonas del interior

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de

protección. • El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el

delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales

peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección.
- El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a

levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos

humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección.
- El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la

ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad.

Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006). Desastres de Origen Natural Dañan Valiosos Ecosistemas Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y

deslizamientos sepultando a barrios en los niveles inferiores. • La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección. • El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de

sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los

ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección.
- El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta

para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros

fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de

protección. • El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el

delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales

peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección.
- El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a

levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos

humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

- La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores.
- La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección.
- El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la

ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad.

Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades .

• La tala de las laderas boscosas ha disminuido la estabilización del suelo y ha dado lugar a numerosos desprendimientos y deslizamientos sepultando a barrios en los niveles inferiores. • La excavación de las dunas para el desarrollo del turismo y de los materiales de construcción, ha eliminado las barreras naturales que anteriormente protegían los medio ambientes costeros interiores, y los asentamientos humanos, de la fuerza directa de las olas de tormenta y vientos huracanados. Extracción de arena de las dunas para la construcción puede debilitar aún más su capacidad de protección. • El drenaje de humedales para la agricultura y los asentamientos humanos ha resultado en graves inundaciones a lo largo de los lagos, ríos y otros cuerpos de agua. Dichas inundaciones, pueden robar a los suelos de nutrientes (disminuyendo la producción agrícola) y contaminar cuerpos de agua con pesticidas y fertilizantes químicos. Los seres humanos han intentado también, de forma sistemática, controlar la ocurrencia de ciertos eventos peligrosos tales como inundaciones. Sin embargo, sin una comprensión adecuada de las potenciales consecuencias directas e indirectas a lo largo y en todos los ecosistemas, muchas de estas intervenciones sólo han agravado el problema, y en muchos casos provocado una cadena de otros nuevos. Un conmovedor ejemplo de ello es la serie de intervenciones para aprovechar el sistema del río Misisipi y el delta para la producción, que en última instancia contribuyó a la devastación de la ciudad de Nueva Orleans, tras el huracán Katrina en el 2005. El delta del Misisipi, hogar de 2,2 millones, representa el peor de los escenarios imaginados. Se está hundiendo y perdiendo humedales más rápido que casi cualquier lugar en la tierra y se enfrenta a la mayor cantidad de huracanes al año. El record de oleaje marino que indujo a los Países Bajos y Gran Bretaña a levantar barreras fue de 15 pies de altura; el de Katrina alcanzó un máximo de 28 pies. Es fundamental para el problema que durante el último siglo el [Ejército] Cuerpo [de Ingenieros], con la bendición del Congreso, construyeron diques en el río Misisipi para prevenir su inundación anual, de manera que las granjas y las industrias

pudieran extenderse a lo largo de sus orillas. Sin embargo, los diques han privado a la región de enormes cantidades de sedimentos, nutrientes y de agua dulce. Inundación natural en la desembocadura del río ha enviado también volúmenes de sedimentos al oeste y al este a una cadena de islas de barrera que reducen las crecidas y las olas, reconstruyendo cada año, lo que la erosión regular del mar ha robado. Pero debido a que ahora la desembocadura está dragada para vías de navegación, el sedimento simplemente fluye hacia las profundidades del océano, dejando el delta-y a Nueva Orleans en su interior - desnudas contra el mar. El Cuerpo y la industria también destruyeron el pantano por el dragado de cientos de kilómetros de canales para que las tuberías pudieran ser colocadas. Canales de navegación aun más grandes fueron excavados, y la erosión por las olas de los barcos transformaron esos cortes en hendiduras profundas que permiten a las crecidas del agua inducidas por los huracanes penetrar la ciudad. Prácticas similares están en uso en muchos de los deltas del mundo, que podrían beneficiarse de planes como los que ahora se están considerando en Luisiana (Fischetti, 2006).

Desastres de Origen Natural Dañan Valiosos Ecosistemas

Los ecosistemas en zonas propensas a desastres suelen ser muy resilientes. Sin embargo, la degradación ambiental extensa expone a los ecosistemas a un daño mayor frente a huracanes, maremotos, inundaciones u otros fenómenos extremos. Este ciclo de degradación ambiental y daños por desastres, destruirán eventualmente la capacidad de un ecosistema para proporcionar servicios críticos de producción (tales como las tierras cultivables y agua potable) y servicios de protección (estabilización del suelo o barreras costeras). Estudios de los impactos del maremoto del océano Índico del 2004 sobre los ecosistemas costeros indican que donde los asentamientos humanos invadieron la costa, las tierras agrícolas sufrieron daños significativos debido al anegamiento del agua. En algunas áreas el agua nunca se retiró, mientras que otras áreas experimentan continuo anegamiento desde el maremoto. Esto ha dejado la tierra estéril y obligó a muchos a encontrar nuevos medios de

subsistencia (DEWGA, 2008). El Cuadro 3 proporciona ejemplos adicionales de los daños que los desastres pueden causar sobre los ecosistemas y los factores de origen humano que han exacerbado los daños. Los desastres también pueden dañar los ecosistemas de forma indirecta. Daños al medio ambiente construido, puede resultar en la liberación y dispersión de los desechos y residuos peligrosos. Residuos municipales, bloqueando desagües y canales, pueden causar inundaciones, propagando enfermedades y exponiendo a las personas y a los ecosistemas a materiales peligrosos. Daños a las instalaciones industriales pueden liberar sustancias tóxicas, contaminando el aire, los suelos y las fuentes de agua. Estos tipos de daños al medio ambiente pueden tener graves efectos a corto y largo plazo sobre la salud y los medios de subsistencia de las comunidades afectadas.

PABLO ARGUETA