

# Implantación de contenido multimedia



# Contenidos

- Imágenes
- Audio
- Video



# Imágenes

Las imágenes representan de media más del 60% de los bytes necesarios par cargar una página web completa 😬.

En este escenario, integrar imágenes que se adapten a los diferentes tamaños del *viewport* y *casos de uso* de forma suave y optimizando tiempos de carga es una habilidad imprescindible para el diseño moderno de páginas web.

## Imagenes en HTML5


Si al potente elemento 💪 `img` 💪, que descarga, decodifica y renderiza contenido le sumamos que los navegadores modernos soportan prácticamente cualquier formato de imagen, tenemos un *win win* de manual.

Pero como dijo alguien alguna vez en un anuncio de ruedas...

“

***La potencia sin control no sirve de nada***

”

así que por aquí  dejo una lista de consejos zen 🙏

## Usa tamaños relativos para las imagenes

Tienes para elegir (% , vh , vw , em , ex , rem , ch , vmin o vmax... 🙄) así que déjate de *px* para todo sino quieres acabar en 🤖 **stack overflow** 🤖 como este figura, al que la imagen le desborda el contenedor



y por si las 🐜 🐜 es buena idea 💡 hacer algo así

```
img, embed, object, video {  
  max-width: 100%;  
}
```

limitando la anchura máxima que pueden tomar los elementos multimedia.

## Usa las propiedades **srcset** y **sizes** del tag **img** o... el elemento **picture**

Así ayudarás al navegador a elegir la mejor imagen para utilizar en función de las condiciones del dispositivo.

- ¿**srcset**, **sizes**, **picture** ... ayudar YO al NAVEGADOR 🤔...? No entiendo nada *hulio*  
😞
- Son la solución **HTML** al *art direction problem*...
  - ¿*art direc...* whaaaat?  
- 👁️ ⬇️

## ***Art direction problem***

🐺 en la pantalla de mi escritorio





En la pantalla de mi 🇪🇺 Iphone 🇪🇺 ... bonitos 🌲 🌲 🌲



🚀 Houston, tenemos un *art direction problem*... ¿qué hacemos?



Pues hacemos algo así

```
@media (max-width: 320px) {  
  .img-lobo {  
    transform: scale(1.5)  
  }  
}
```

y arreglado...

⇒ mi perro tiene problemas más graves y sin nombre propio en inglés

🙄 Prueba con el gato, listillo 📄



😅 puedo contar los pixeles con los dedos de la mano ([aquí](#) 🔗)



baia baia



Con un escalado no podemos decidir en que zona de la imagen hacer foco y la imagen se ve muy pixelada cuando la ampliamos.

La solución al *art direction problem* consiste en disponer de **varias versiones de la misma imagen** en las que nos centramos en lo que queremos mostrar.

Veamos esto con ***srcset***, ***sizes*** y los gatos

```

```

Explicación 

```
srcset="
  gato-ciudad-280w.jpg 240w,
  gato-ciudad-680w.jpg 680w
"
```

📶 le digo al navegador

“ Tienes 2 imágenes de gatos disponibles una de 280w de ancho y otra de 680w. Elige la mejor en función de las características del dispositivo ”

⚠ Los *widths* ( 240w y 680w ) son los anchos intrínsecos de la imagen ([mas info](#) 🔗)

En lugar de indicar los anchos intrínsecos de las imagenes ( 240w , 680w , .. ) puedes establecer las densidades de pixel que aplican a cada imagen ( 2x , 3x ... )

```
sizes="
  (min-width: 960px) 720px,
  100vw
```

📶 le digo al navegador

“ Si el ancho del dispositivo es  $\geq 960\text{px}$  entonces elige la imagen que mejor se ajuste a un ancho de  $720\text{px}$ . Si ocurre lo contrario haz que la imagen se ajuste al ancho completo del *viewport* (y elige la mejor imagen para ello) ”

[Aquí](#) 🔗 tenéis una explicación mucho mejor que la mía.

¿Se puede hacer mejor?

Por supuesto, con el elemento 🧐 `picture` 🧐 (cuidado que es su primer día en navegadores)

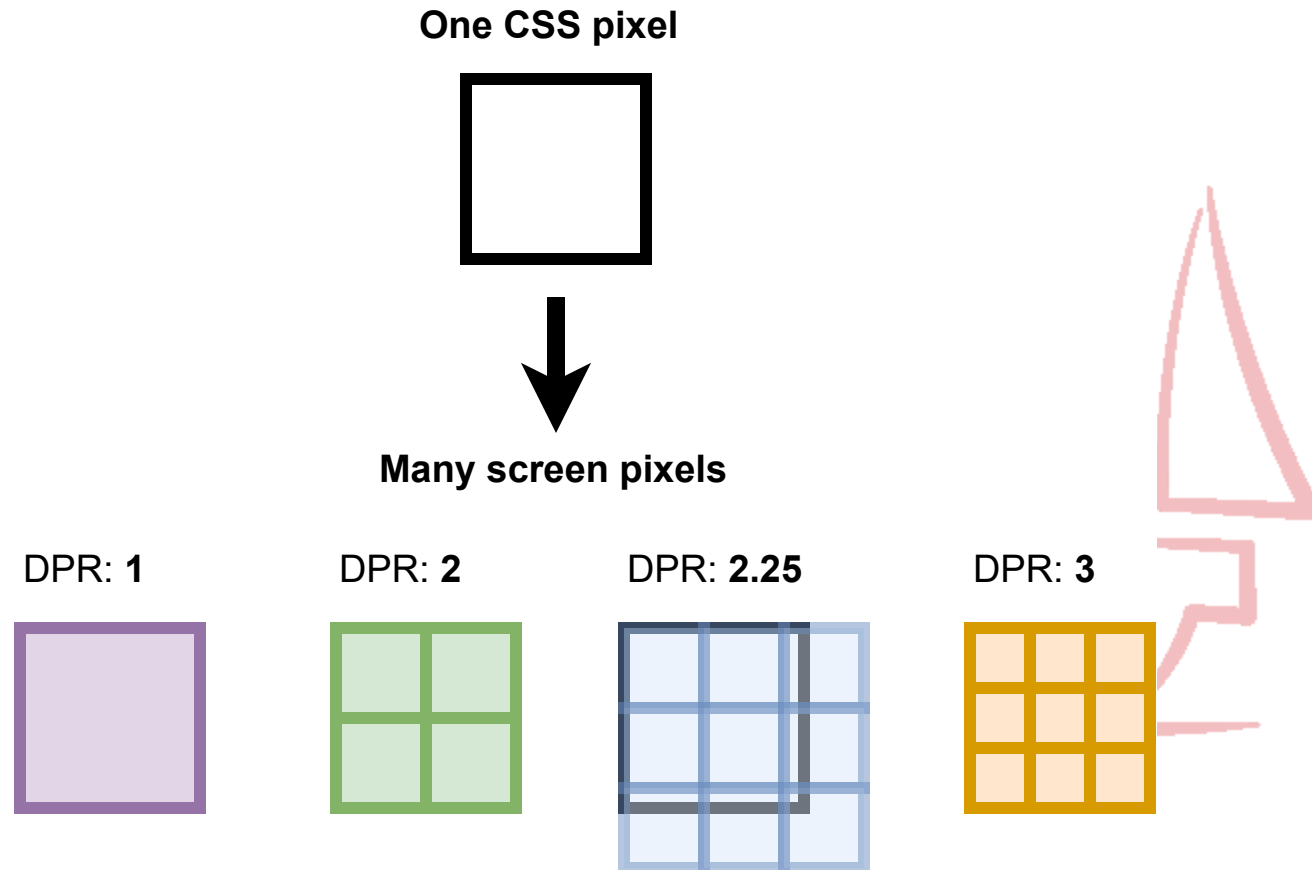
```
<picture>
  <source
    media="(max-width: 960px)"
    srcset="gato-ciudad-280w.jpg"
  >
  
</picture>
```

de todos modos, [aquí](#) 🗄️ hay un ejemplo de alguien mucho más listo

¿Y si os digo que también se hace todo lo de arriba no solo en función del ancho de pantalla del dispositivo sino también de su densidad de píxeles? 📶



Una tabla (y un [ejemplo](#)) vale más que mil palabras... y un figura preguntando en [stack overflow](#) ya ni os digo. [Aquí](#) también está muy bien explicado.



TL;DR... ¿una ⚡ actividad ⚡ y tal...?

## Actividad 1

1. Elegid una imagen que os guste (⚠️ con una resolución decente pero sin pasarse)
2. Haced 2 versiones más de ella (una para dispositivos móviles y otra para *tablets*) usando 🖌️ *paint* 🖌️ o el editor que querais.
3. Usad **srcset** y **sizes** para cargarlas de forma condicional en función del ancho de pantalla.
4. Usad 😎 `picture` 😎 para hacer lo de arriba ⬆️

⚡ 20 MINUTOS ⚡



## ImageMagick

En la ⚡ actividad ⚡ anterior, os pedía que versionarais una imagen usando un editor de vuestra elección, así, a la buena de Dios.

Redimensionar, rotar, difuminar, cambiar la opacidad, recortar... y sobre todo **CONVERTIR** entre formatos de imagen son habilidades a dominar en este mundillo... y para dominar algo hacen falta buenas herramientas 🔧🔩🔨

✨ Os presento [ImageMagick](#) 🔗 ✨ ⬇️

Espera, espera... eso es una herramienta de consola de comandos 🤖🤖🤖

Efectivamente... bienvenidos al mundo real 🤖🤖

La parte buena es que os podéis hacer los *hackers* delante de vuestras abuelas mientras redimensionais vuestro avatar de *Fortnite*.

Pero lo mas importante es que una vez dominada el flujo de trabajo se agiliza y además os permite automatizar tareas comunes con *scripting*.

Me has convencido, dale 🤖🤖

1. Instala desde [aquí](#) 🔗
2. Verifica que todo va bien haciendo esto en la terminal ⬇️

```
magick logo: logo.gif  
magick identify logo.gif  
magick logo.gif win:
```

3. Si estas en Windows y no te funciona 😞😞 instala este [dll](#) 🔗 y si todo va bien...



## Vale, ahora unos ejemplos

1. Convertimos el 🐾 de antes de .jpg a .webp 🔗

```
magick lobo-en-el-bosque.jpg -quality 50 -define webp:lossless=true lobo-en-el-bosque.webp
```

2. Redimensionamos los 🐱🐱 de antes para que se queden a la mitad

```
magick gato-ciudad-680w.jpg -resize 50% gato-ciudad-340w.png
```

3. Ahora quiero el logo de ✨ImageMagick✨ en b/n

```
magick imagemagick.jpg -monochrome imagemagick-bn-mal.png
```

4. Y para acabar, las tres primeras de golpe con la calculadora humana

```
magick actividad-calculo.jpg -resize 50% -monochrome actividad-calculo-magick.png
```

Por [aquí](#) 🔗 tenéis la referencia para montar bien los comandos de terminal 💪...

y a continuación... 😬😬

⚡ actividad ⚡ 📄

## Actividad 2

1. Instala ✨ImageMagick✨ como vimos ⬆️
2. Busca una imagen cualquiera en alta resolución
3. Redimensionala al 75%, cambia su orientación y aplica un desenfoque (*blur*) a tu elección
4. Conviertela a formato *webp* sin pérdidas (*lossless*)
5. Compara el tamaño en disco de la imagen original con la nueva...  
y me contáis el resultado 😊😊

⚡ **20 MINUTOS** ⚡





## Imagenes en CSS

HTML dominado 👍👍 pero... ¿CSS no tiene nada que decir 🤔?

Bueeeh, menos de lo habitual pero algo hay... siempre de la mano de la propiedad `background` 🔗) ⬇️

## ...exacto, las típicas @media queries

```
.img-gato {  
  width: 720px;  
  background-image: url(assets/gato-ciudad-680w.jpg);  
  background-repeat: no-repeat;  
  background-size: contain;  
  background-position-x: center;  
}  
  
@media (max-width: 960px) {  
  .img-gato {  
    width: 100vw;  
    background-image: url(assets/gato-ciudad-280w.jpg);  
  }  
}
```

🏆 Esta jugada seguro que se os ocurría sin ayuda 🏆

## La función *image-set()* (para tema *DPRs*)

Con esta 👁️ ⚠️ que esta recién salida del horno y tiene mal soporte (sólo *Chrome* y *Safari*, añadir `-webkit`). Pero nosotros siempre al limite ⬇️

```
.img-gato {  
  width: 720px;  
  background-image: url(assets/gato-ciudad-680w.jpg);  
  background-image: -webkit-image-set(  
    url(assets/gato-ciudad-280w.jpg) 1x,  
    url(assets/gato-ciudad-680w) 2x  
  );  
  background-image: image-set(  
    url(assets/gato-ciudad-280w) 1x,  
    url(assets/gato-ciudad-680w) 2x  
  );  
  background-repeat: no-repeat;  
  background-size: contain;  
  background-position-x: center;  
}  
  
@media (max-width: 960px) {  
  .img-gato {  
    width: 100vw;  
  }  
}
```

...aunque podemos hacer lo mismo con `@media` queries (otra vez 🧐)

```
.img-gato {  
  width: 720px;  
  background-image: url(assets/gato-ciudad-680w.jpg);  
  background-repeat: no-repeat;  
  background-size: contain;  
  background-position-x: center;  
}  
  
@media (max-width: 960px) and (min-resolution: 2dppx), /* Sintaxis estandar */  
(-webkit-min-device-pixel-ratio: 2) /* Navegador Safari y Android */  
{  
  .img-gato {  
    width: 100vw;  
    background-image: url(assets/gato-ciudad-680w.jpg);  
  }  
}
```

y hasta aquí

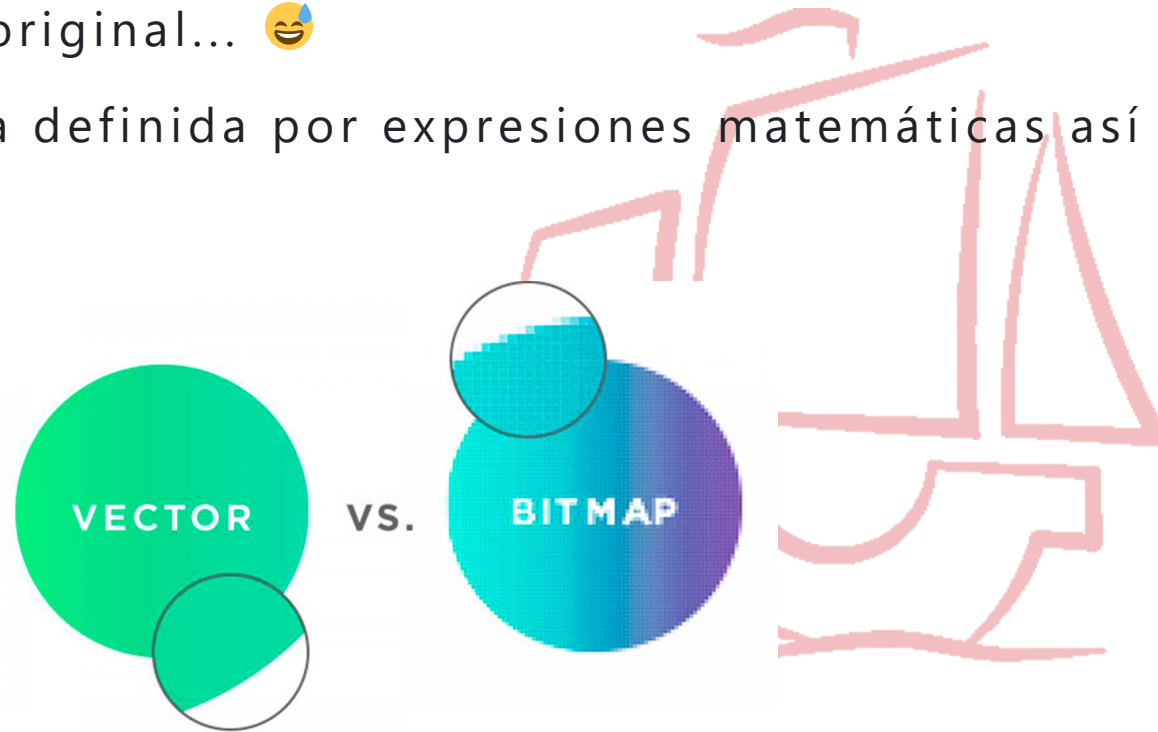
# Formatos de imagen y cuando usarlos para *dummies*

Aquí os dejo una imagen con *info* sobre esto



La diferencia entre imagen rasterizada y vectorial es difícil de ex...nahh 😊

- Una imagen ráster (o *bitmap*) es una cuadrícula de píxeles que si escala por encima de su resolución original... 😊
- Una imagen vectorial esta definida por expresiones matemáticas así que escala sin perdida de calidad



y es todo lo que tenéis que saber 🙄🙄

## Imágenes ráster o *bitmaps*

- Antes de que veamos los principales tipos de imágenes ráster y cuando utilizarlos... ●



## ***lossless y lossy (podría ser el nombre de mis 🐶🐶)***

Las imágenes son 1s y 0s - como cualquier entidad digital 🧐🧐 - y el formato es sólo la manera en que estos bits estan ordenados y estructurados.

👉👉... ¿pero que me estas contando?

Pues que hay gente muy lista que hace cosas de gente muy lista con estos bits para que una imagen en lugar de ocupar 1 MB ocupe 500 KB, es decir comprime la imagen, y esto es lo que hacen los formatos de imagen de dos formas diferentes 📶



- *lossless* - Comprime la imagen de forma que podemos reconstruirla a su estado original sin perdida de información.
- *lossy* - Comprime la imagen a costa de determinadas propiedades de la original, p. ej. reduce el rango de colores. Si esto se hace de forma inteligente 🧐 al reconstruir la imagen el ojo humano no lo percibe.

Como era de esperar las *lossy* tienen mejores *ratios* de compresión y son ideales para *web*.

Aclarado 👍. Ahora resumen de formatos ⚡️ ⬇️  
Primero las ráster



## JPEG

*Lossy en acción*



*No compression*



*High compression*

## Casos de uso ⬇️

- 👍 Ofrecen mucha flexibilidad en la edición y compresión, así que son ideales en entornos *web* donde es importante la velocidad de descarga.
- 👍 Imágenes para imprimir. ⬆️ Resolución ⬇️ Compresión... y al revés
- 👍 Ideales para imágenes de tamaño muy pequeño en plan *preview* para enviar por *email* o \*avatares del *fortnite*
- 👎 Si necesitas transparencia... olvídate, no tienen canal de transparencia, usa mejor PNG o GIF.
- 👎 Necesitas hacer ediciones muy sofisticadas por capas y demás... nada, son imágenes planas donde todas las modificaciones se guardan en una sola capa. Usa PSD (Photoshop)

## GIF


¿A quién no le gustan los GIFs de gatos? 🥹🥹... son *lossless* por cierto.









## Casos de uso ⬇️

- 👍 Para animaciones web
- 👍 Necesitas transparencia. Los GIFs tienen un canal *alfa* transparente
- 👍 ¿Quiéres una imagen muy pequeña? GIF es tu imagen.
- 👎 Para imagenes de calidad fotográfica, no gracias 🙅. GIF tiene un limite de 256 colores, las fotos tienen miles... 🤔
- 👎 Si necesitas imprimir la imagen, por lo mismo que arriba ⬆️. Usa TIFF, PSD o JPG
- 👎 Ediciones por capas y bla bla... le pasa como a JPG

## PNG

La niña bonita del mundo *web*, prácticamente un estándar y... *lossless*. Vamos a ver cuando usarlo y cuando no 

-  Transparencia *over 9000*. Tienen un canal *alfa* variable que permite grados de transparencia (GIF es todo/nada) y con una buena profundidad de color
-  Imágenes con un rango limitado de colores
-  Imágenes de tamaño pequeño
-  Fotos. Aunque la resolución y la profundidad de color esta bien resuelta... es *lossless* y para *web* va a ocupar demasiado espacio.
-  ¿Tienes que imprimir?. PNG esta optimizado para pantallas ... puedes imprimir pero... mejor JPEG o TIFF

## TIFF

*Lossless* de altísima calidad.

- 👍 Para imprimir fotos en alta calidad, sobre todo en formatos grandes
- 👍 Para escanear fotos o documentos con la mejor calidad posible
- 👎 Demasiado bueno... y pesado para *web*. JPEG o PNG para imágenes de alta calidad en entornos *web*

y ahora las vectoriales 



## PDF

“

El *santo grial* de las copisterías...

”

- 👍 La mayoría de imprentas y copisterías te haran descuento 💰 si les llevas tus documentos en PDF... 😊. Es universal.
- 👍 Para mostrsa un documento en la *web* van muy bien ya que mantienen todo el diseño en un elemento, lo que lo hace más fácil de ver, descargar o imprimir.
- 👎 Si tienes que editar el diseño...mejor pasa de PDF, son buenos contenedores pero nada más.

## EPS

El más triste de los formatos vectoriales (aunque puede incluir datos rasterizados).  
Sirve para hacer logos y... para hacer logos.

- 👍 ¿Necesitas un logo vectorial para tu empresa? EPS es lo que estás buscando.
- ⚠️ No es recomendable usar EPS para cualquier cosa que no sea hacer logos



## SVG

🏆 SVG 🏆 puede ser buscado, indexado y comprimido en ficheros mas pequeños que otros formatos de imagen, vamos un *win-win* para *web*. Además se puede editar hasta con el 🧑🏫 bloc de notas 🧑🏫 al ser un formato basado en *XML*

- 👍 Gráficos y diagramas para web
- 👍 Imprimir entra en la ecuación

## Tenemos un nuevo vecino en el barrio... WEBP

Soporta compresión *lossless* y *lossy* con mejores ratios en ambos casos que PNG y JPEG y calidades equivalentes. ¡Y transparencia! 🙌🙌

Demasiado bueno para ser cierto 🤔🤔... tiene truco...

No lo tiene, sólo que es nuevo y en consecuencia no está tan bien soportado por los navegadores como otros formatos. [Aquí](#) 🔗 una web con info detallada.

Os dejo por [aquí](#) 🔗 la tabla del MDN con un resumen de todo lo de arriba (más o menos) pero en inglés.

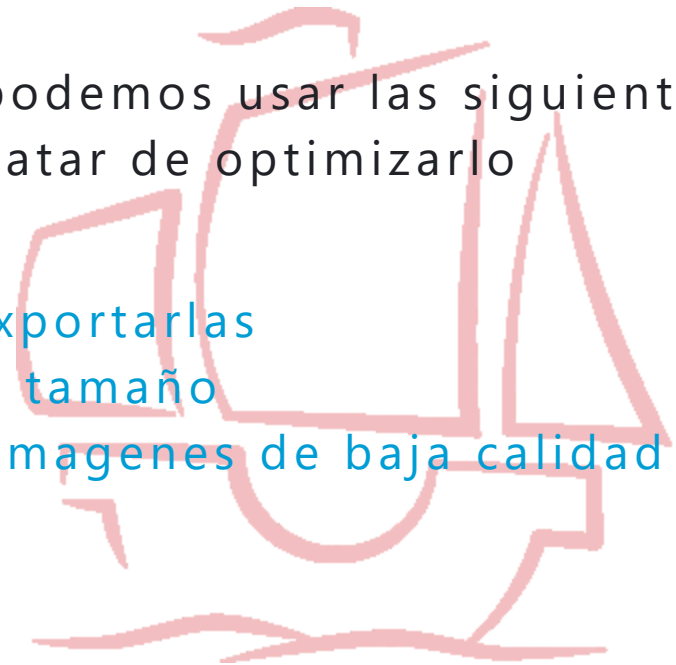
## Optimización de imágenes

Antes de optimizar una *web* es buena idea 💡 hacer un *benchmark* del sitio para afinar mejor el tiro 🎯. Os dejo por aquí unas cuantas herramientas para hacerlo 📄

- [Google PageSpeed Insights](#) 🔗
- [Pingdom Tools](#) 🔗
- [GTMetrix](#) 🔗
- [WebPageTest](#) 🔗

Cuando tengamos claro el estado de la web podemos usar las siguientes técnicas aplicadas a imagenes para tratar de optimizarlo

- Elegir el formato adecuado de imagen
- Redimensionar las imagenes antes de exportarlas
- Comprimir las imagenes para reducir su tamaño
- Usar la técnica del *blur up* para cargar imagenes de baja calidad primero
- Usar *lazy loading*
- CSS Sprites



## Elegir el formato adecuado de imagen

No hay mucho que decir 😊... *WEBP* y *SVG* cuando se pueda y para todo lo demás...  
mirad [aquí](#) ↑



## **Redimensionar las imagenes antes de exportarlas**

Esta técnica tampoco tiene mucho misterio... Si teneis imágenes de 1024 x 1024 px pero os basta con 500 x 500 ya sabeis, recortad ✂ o redimensionad, cada pixel que os cargueis son menos bits que almacenar y descargar.



## Comprimir las imagenes para reducir su tamaño

Otra técnica para genios 🧐 Podeis usar herramientas de escritorio como [RIOT](#) 🔗 o alternativas online como [tinypng](#) 🔗.

Os prometo que lo que viene a continuación es más sofisticado 😊

## Usar la técnica del *blur up* para cargar imagenes de baja calidad primero

Esta técnica nace en el equipo de ingeniería de *Facebook*, que tiene que resolver los tiempos de carga lentos que se producen en sus apps debido al gran tamaño de las imagenes de fondo de los perfiles de sus usuarios... y es un ejemplo de pensamiento lateral e ingenio 🧐, así que atentos ⬇️

1. Devolvemos una imagen muy pequeña, de unos 40 px de ancho
2. La escalamos mientras aplicamos un desenfoque gaussiano

En este punto el usuario vé una imagen de fondo que, debido al desenfoque, resulta agradable a la vista y da una idea de como se verá la imagen final.

3. Mientras todo esto ocurre, cargamos en el *background* la imagen de fondo definitiva y cuando este le damos el cambio *smoothly* 😎😎



# A walk along the coast

In July, we went for a walk from the Brösarp hills and out to the coast near Kivik. It was one of the most beautiful days of the summer.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Quibusdam laudantium eos minima dolor asperiores maxime voluptas itaque iure nostrum quos ipsam aliquam quasi amet repudiandae totam, vel, veniam sit repellat.

## Usar *lazy loading*

Es una idea similar a la de .

Cargamos las imagenes que están en la parte baja de la pagina a medida que el usuario hace *scrolling*. Esta técnica ya es nativa en HTML.

Os dejo [info](#)  detallada de como utilizarla.



## CSS Sprites

Otra 💡 idea top por aqui. Unimos varias imagenes en una imagen más grande y luego posicionando con CSS mostramos lo que queremos mostrar. Ya esta 😎.

Yo no lo veo claro 🤔 porque esa imagen *Frankenstein* va a pesar lo mismo o mas que sus partes por separado...

Ya, eso es cierto peeeeero... el cliente se descarga el *sprite* en una sola petición en lugar de tener que hacer varias (tantas como imagenes). Veamos un ejemplo ⬇️

Tenemos estas tres banderas ➡ y  
hacemos esto ⬇

```
.flags-canada, .flags-mexico, .flags-usa {  
  background-image: url('../images/flags.png');  
  background-repeat: no-repeat;  
}  
  
.flags-canada {  
  height: 128px;  
  background-position: -5px -5px;  
}  
  
.flags-usa {  
  height: 135px;  
  background-position: -5px -143px;  
}  
  
.flags-mexico {  
  height: 147px;  
  background-position: -5px -288px;  
}
```

Y aqui un enlace al [pen](#) 🔗





Ufff preparar estos estilos a mano... 😓😓

Haya calma... primero crear el sprite con... 🧙‍♂️ ImageMagick 🧙‍♂️ así

```
convert *.png -append sprites.png # append vertically  
convert *.png +append sprites.png # append horizontally
```

y luego tenemos este servicio online para generar los estilos [spritecow](#) 🔗 🐮

Por cierto, si os preguntais si un sprite es mejor horizontal o vertical la respuesta es... ninguno... Compactalo en un grid lo mas pequeño dimensionalmente que puedas... así... a ojo 👁

Venga ¿porque no? ⚡ actividad ⚡ 📄


## Actividad 3

Busca las imágenes con los logotipos de 3 empresas y usando la receta de arriba

1. Crea el sprite con 🧙 ImageMagick 🧙
2. Usando [spritecow](#) 🔗 🐮 genera los estilos
3. Reutiliza el HTML del [pen](#) 🔗 anterior para mostrar los 3 logos por separado.



## DataURLs

Hay otra técnica sencilla que nos ahorrará peticiones y que consiste en embeber las imágenes como texto codificado, generalmente en [base64](#) , directamente en el HTML o en el CSS. 👁👁 Veamos dos ejemplos

### CSS

```
div {  
  background:  
    url(data:image/gif;base64,R0lGODlhEAAQAMQAAORHHOVSKudfOulrSOp3W0yDZu6QdvCchPGolf00o/XBs/fNwfjZ0fr13/zy7///wAAAAAAAAAAAAAAAAAA  
    no-repeat  
    left center;  
  padding: 5px 0 5px 25px;  
}
```

### HTML

```
<img src="data:image/gif;base64,R0lGODlhEAAQAMQAAORHHOVSKudfOulrSOp3W0yDZu6QdvCchPGolf00o/XBs/fNwfjZ0fr13/zy7///wAAAAAAAAAAAAAAAAAA
```

El formato general es

```
data:[<mime type>][;charset=<charset>][;base64],<encoded data>
```

Podeis usar esta herramienta para [convertir a base64](#) 🔗...

... y aquí el [pen](#) 🔗...

y ahora una ⚡ actividad ⚡

## Actividad 4

Repíte la [actividad anterior](#) usando *DataURLs* en lugar de *sprites*.



