01

Imagens Responsivas

Transcrição

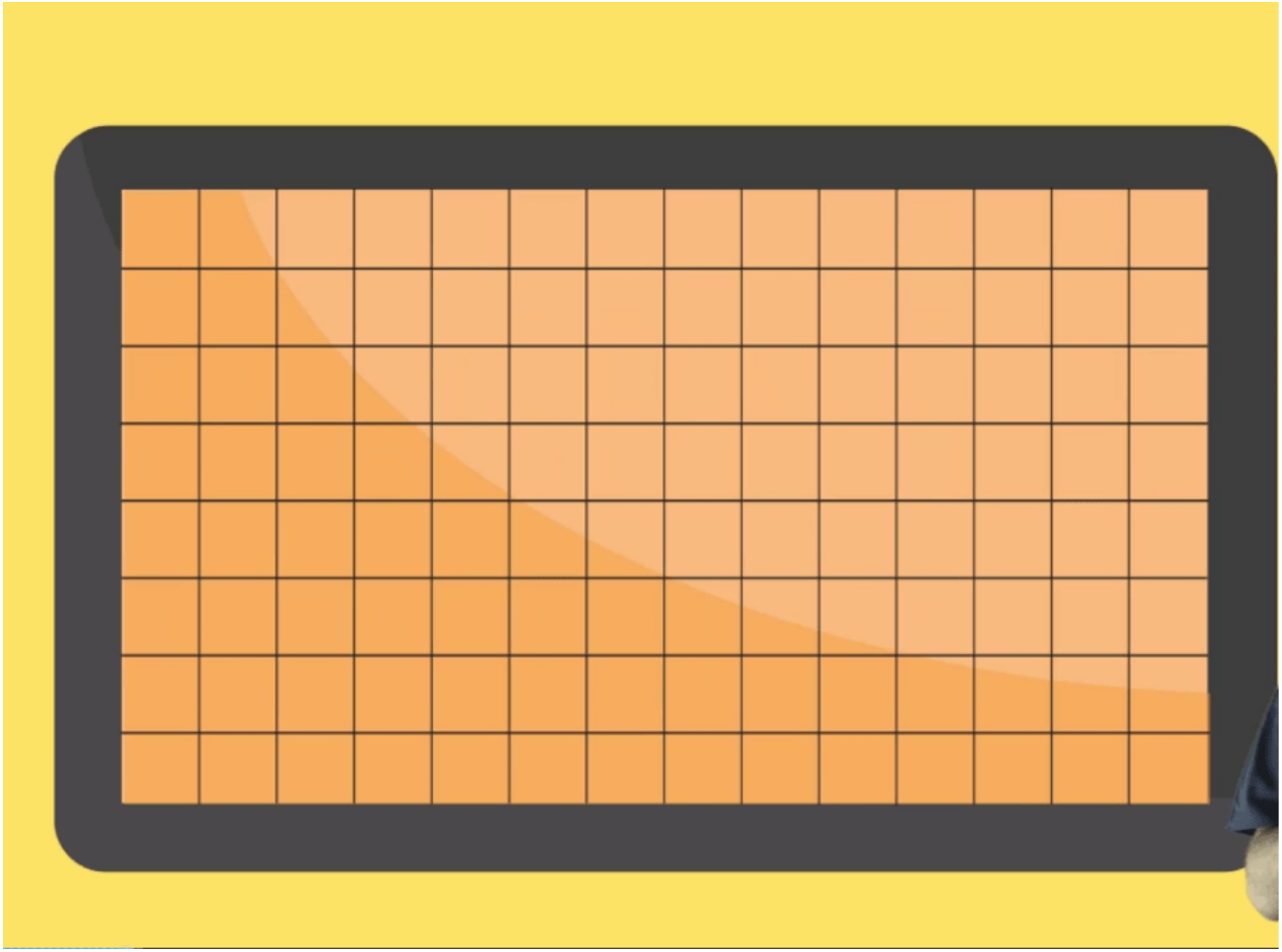
Nesta aula iremos falar sobre ***Imagens Responsivas***. As imagens são parte essencial de todas as páginas da *Web* e quando falamos de *Design Responsivo,* elas costumam trazer algumas dificuldades.

As imagens nada mais são do que pixeis, e já falamos como o Responsivo trabalha com medidas flexiveis, porcentagens e com adaptações, os quais podem trazer alguns problemas.

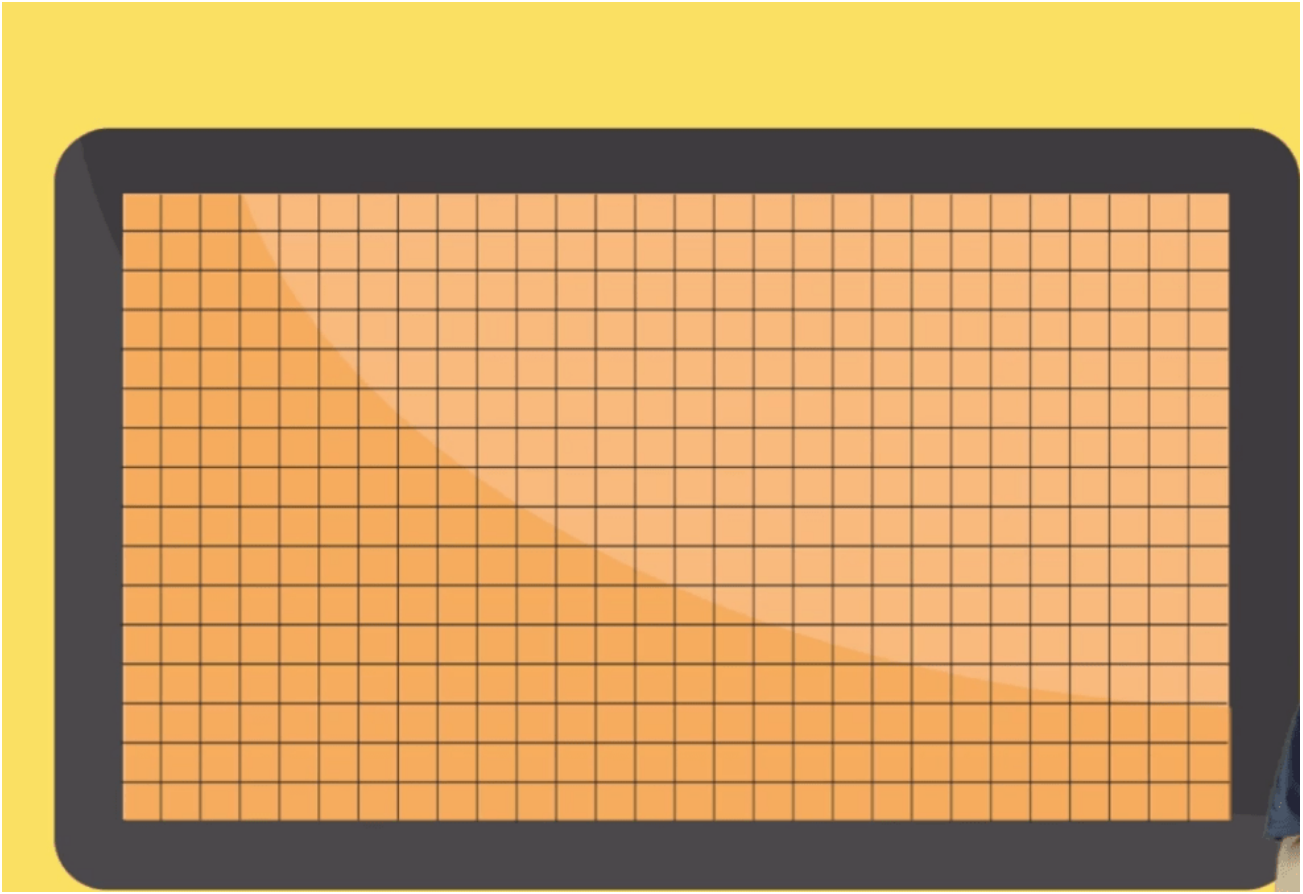
Iremos tratar tanto nesta quanto na próxima aula dos casos de problemas e das soluções para tratar de *Imagens Responsivas*. Um dos pontos em que as pessoas costumam ter dúvidas é em relação às *telas de alta resolução*. Com o advento dos *smartphones,* começamos a trabalhar com telas pequenas de alta resolução.

Resoluções

Vamos primeiramente entender o que significa uma tela de alta resolução. Imaginemos que temos uma tela e sua resolução, ou seja, com sua quantidade de pixeis (o menor quadrado possível que recebe uma cor):



Uma tela de alta resolução nada mais é do que uma tela com muitos pixeis:

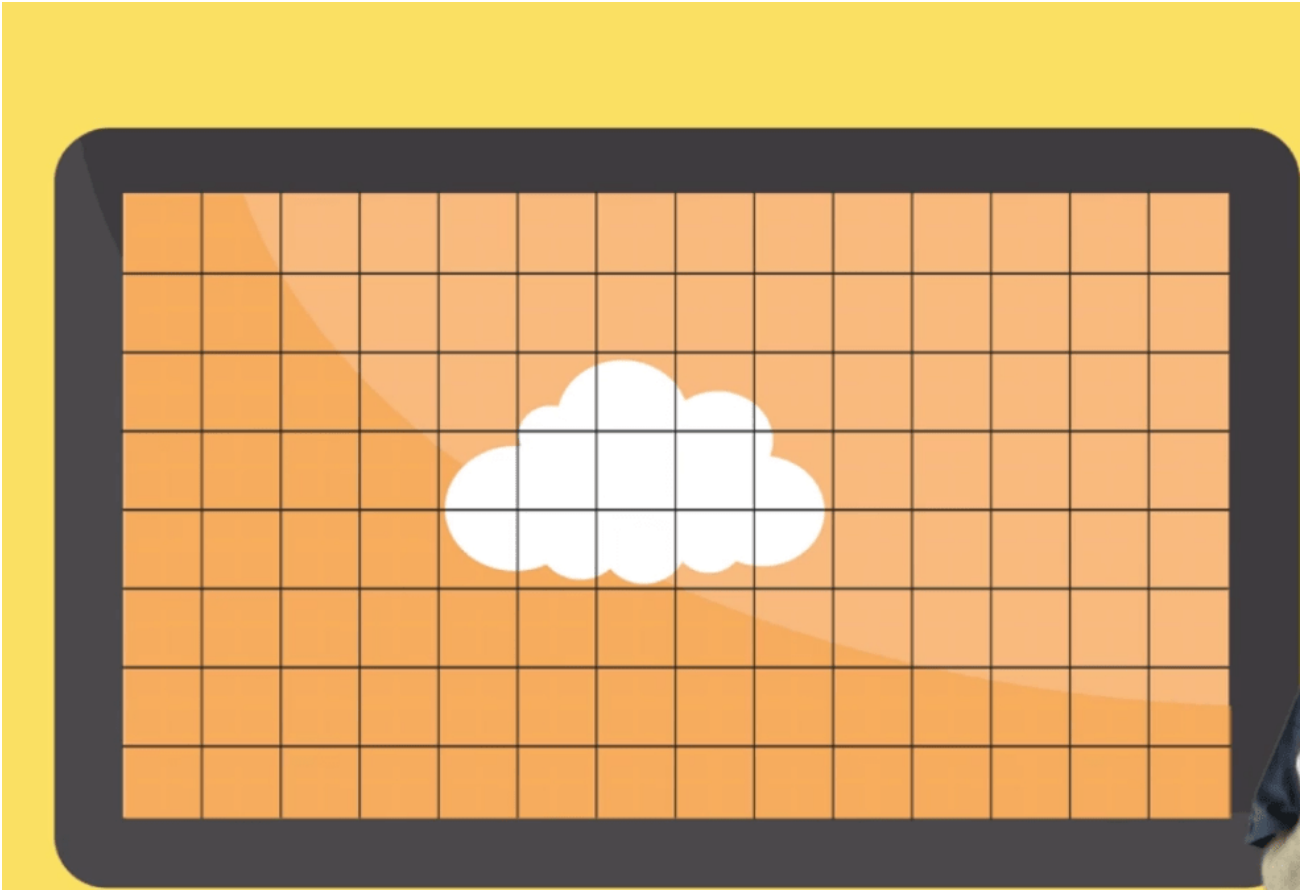


Se mantivermos o tamanho físico da tela aumentando sua resolução, os pixeis ficam menores.

Se tivéssemos uma imagem numa tela com resolução baixa e essa imagem ocupasse um certo número de pixeis, ao passarmos para uma tela de resolução maior, como os pixeis serão menores, a imagem também diminuirá de tamanho.

Tal mudança não seria um problema muito grande se apenas estivéssemos lidando com telas grandes. Porém, em telas de alta resolução e pequenas, como vistas em *smartphones*, tais imagens ficariam muito pequenas, o que causaria problemas de usabilidade.

Uma solução automática que essas novas plataformas encontraram é aumentar o tamanho físico das imagens, dando a aparência de que elas têm o mesmo tamanho que tinham em uma tela comum. Veja um exemplo:



No CSS, para uma tela comum:

```
.nuvem { width: 100px; }
```

Nas duas telas a imagem tem o mesmo tamanho físico. Mas na segunda tela a imagem ocupa um maior número (o dobro) de pixeis. Mas como que o número de pixeis que usamos no código não é o mesmo que é desenhado na tela?

Acontece que estamos falando de dois tipos diferentes de pixeis:

- Pixel físico: como vimos, é o menor quadrado na tela;
- *Device Independent Pixel* (DIP): na documentação do Android, é o menor ponto da tela dependendo do aparelho em que estamos trabalhando.

No CSS, na realidade, os 100 pixeis escritos no código estão em DIP e não em pixeis físicos. Desta maneira os pixeis físicos variam de tela para tela. Para uma tela de alta resolução eles se transformam em 200 físicos.

Quem faz essa mudança nos valores é o próprio navegador do sistema operacional de um aparelho de alta resolução, o que é uma boa prática pois não precisamos nos preocupar em fazer contas ou escrever dois CSS diferentes.

Device Pixel Ratio

Se prestarmos atenção, existe uma proporção entre o número de pixeis no CSS e o pixel que é usado efetivamente nas imagens na tela. Pode ser 1:1, no caso da tela comum, ou 2:1, no exemplo visto a cima. Tal proporção tem o nome de "device pixel ratio" ou dPR e sua fórmula é bem simples:

```
dPR = pixeis físicos / DIPs
```

Dividimos a quantidade de pixeis físicos da tela pela quantidade de DIPs. No caso da tela de alta resolução com a imagem de 200 pixeis físicos:

```
dPR = 200 / 100 = 2
```

Então a tela de alta resolução tem um *device pixel ratio* de 2:1. Por padrão damos a essa proporção específica o nome de *Retina*.

Mas, na prática, como lidamos com isso no nosso código? Primeiro precisamos saber que podemos usar uma *media query* para poder diferenciar os dispositivos a partir dos seus dPRs. Tal *media query* é a *resolution*. Veja dois exemplos:

```
@media (resolution: 1dppx) {
}

@media (resolution: 2dppx) {
}
```

O "dppx" significa *dots per pixel*, ou seja, pontos por pixel. Tais pontos são os pixeis físicos. Essas especificações são importantes para não acontecer casos como este abaixo:


Caelum
Ensino e Inovação
Caelum
Ensino e Inovação

Quando colocamos numa tela de alta resolução (2:1) uma imagem feita para tela comum (1:1), essa imagem fica com qualidade baixa, "pixelada". Isso acontece porque se uma imagem possui 100 pixeis de largura, numa tela de retina ele ocupará o dobro e o SO esticará a imagem.

Para resolvermos o problema basta trocar a imagem por uma de maior resolução usando as *media queries*:

```
.logo {
    background-image: url(logo.png);
}

@media (resolution: 2dppx) {
    .logo {
        background-image: url(logo@2x.png);
    }
}
```

Perceba que para telas de resolução comum, o logotipo é uma imagem comum *.png*. Para telas com 2 dppx a imagem é outra, com o dobro da resolução. Sempre lembrando que podemos utilizar outros comandos já vistos para as *media queries* e dppx quebrados:

```
@media (min-resolution: 1.5dppx) {
}
```

Uma observação que deve ser feita é que para aparelhos Android mais antigos não poderemos usar a *media query* "*resolution*". Então, para atingirmos a maior parte dos dispositivos fazemos

```
@media (min-resolution: 1.5dppx),
    (-webkit-min-device-pixel-ratio: 1.5) {
}
```

Qualidade visual, Performance e Direção de arte

Tudo isso que vimos até agora está ligado à nossa necessidade de melhorarmos a qualidade visual de nossas páginas. Quando pensamos em *Imagens Responsivas*, são vários os seus casos de uso:

- **Qualidade visual**: como já vimos, queremos boas imagens para todos os dispositivos.
- **Performance**: queremos que as imagens de maior resolução só sejam baixadas se estivermos em um dispositivo com capacidade de mostrar tal imagem. Não queremos desperdiçar downloads.
- **Direção de arte**: muitas vezes a melhor solução não é apenas aumentar ou diminuir o tamanho e a resolução das imagens. É interessante pensarmos em imagens diferentes para cada dispositivo para melhorar a experiência do usuário.

Com as *media queries* conseguimos resolver todos esses casos. Já vimos um em relação à qualidade de imagem. No caso de substituição de imagem, temos:

```
.foto {
    background-image: url(foto-recorte.png);
}

@media (min-width: 800px) {
    .foto {
        background-image: url(foto-full.png);
    }
}
```

Temos uma imagem para os dispositivos pequenos (foto-recorte.png) e outra para a versão desktop (foto-full.png).

Outras alternativas em *Imagens Responsivas*

Nosso próximo desafio é aprender a trabalhar com a *tag* "img". Ela recebe um *src* direto com o nome do arquivo que queremos carregar. Diferentemente do que fizemos com as *media queries*, aqui não conseguimos chamar ora uma imagem, ora outra.

Existem alguns *frameworks* que permitem implementar uma lógica parecida com a das *media queries* para imagens responsivas, que nada mais fazem do que substituir o *src*. Porém, para a performance, eles são muito ruins: dependem de javascript e de downloads duplicados. Então evite utilizá-los.

Veremos, então, algumas alternativas aos *frameworks*. Nos navegadores mais modernos já existem algumas soluções para tais cenários, como por exemplo:

- *srcset*: nos permite passar um conjunto de *src*'s

```
<img src="logo.png"
     srcset="logohd.png 2x, logo.png 1x">
```

  É um conjunto de imagens, separadas por vírgulas, onde o "2x" e o "1x" significam o dPR da tela. Por ser um recurso novo, consulte se os navegadores para os quais quer dar suporte já o possuem.

- *picture*: uma nova ideia para o futuro (ainda com pouco suporte dos navegadores), que nos permite lógicas mais complexas que o *src*:

```
<picture>
  <source media="(min-width: 600px)" src="completa.jpg">
  <source media="(min-width: 1000px)" src="grande.jpg">

  <source src="recorte.jpg">

  <img src="fallbackjpg">
</picture>
```

Com a *tag* "*picture*" podemos passar várias imagens por meio da *tag* "*source*" e tais imagens recebem uma *media query* e, com isso, podemos usar os recursos já conhecidos dentro do HTML para carregar ora uma imagem, ora outra.

Com ambas as soluções conseguimos fazer as mesmas coisas que antes eram feitas com o CSS, só que agora para imagens no HTML: carregar imagem dependendo da resolução, do tamanho da tela e da direção de arte. Nunca esquecendo de consultar a compatibilidade dos navegadores para utilizar a solução mais adequada. Sendo as *medias queries* a que tem maior suporte.

Otimizando com uma imagem

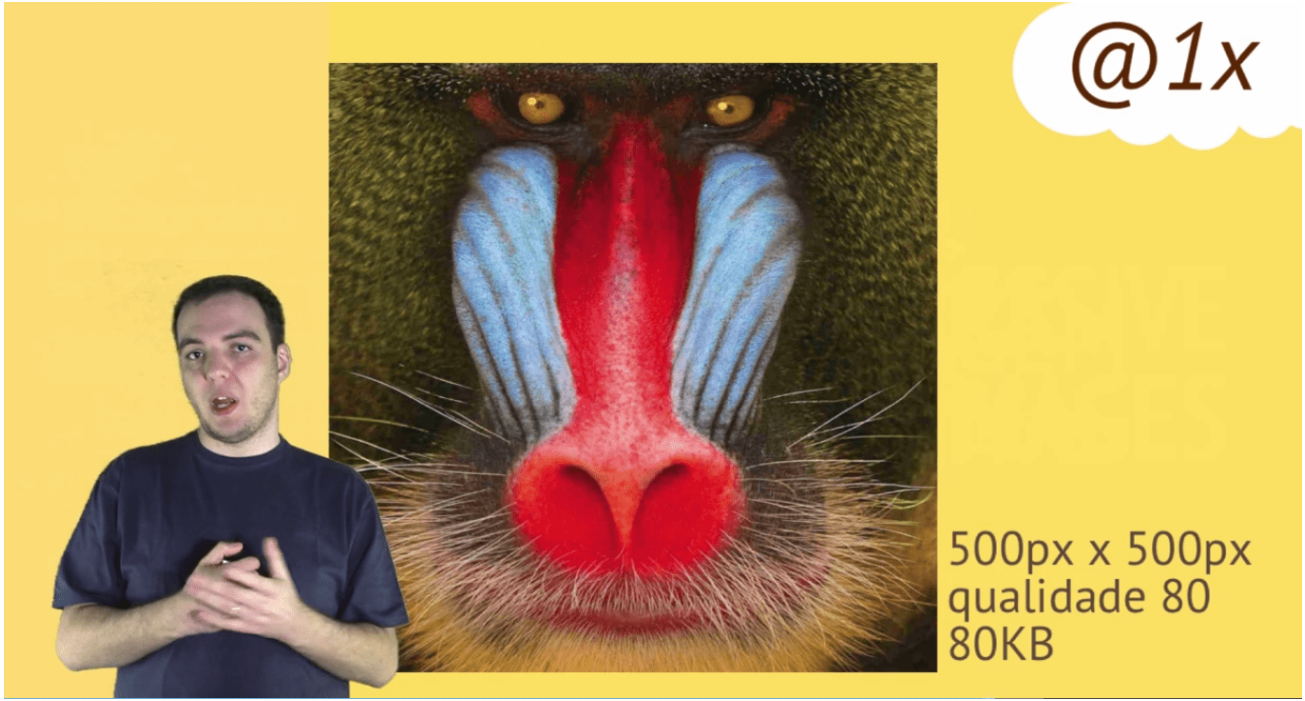
Embora os códigos fiquem cada vez mais complexos, no dia-a-dia o maior trabalho está em criar diversas versões de uma mesma imagem. Em um cenário ideal, usaríamos apenas uma que fosse boa para todas as resoluções e dispositivos. A primeira razão para pensarmos isso no âmbito de *imagens responsivas* é em relação à performance.

Não queremos, como já falamos, carregar uma imagem grande em um dispositivo de tela pequena. Achar uma solução em que utilizamos apenas uma imagem sem esquecer a performance, não precisaremos escrever uma série de linhas de código para trocar imagens a todo tempo. Vamos ver alguns casos.

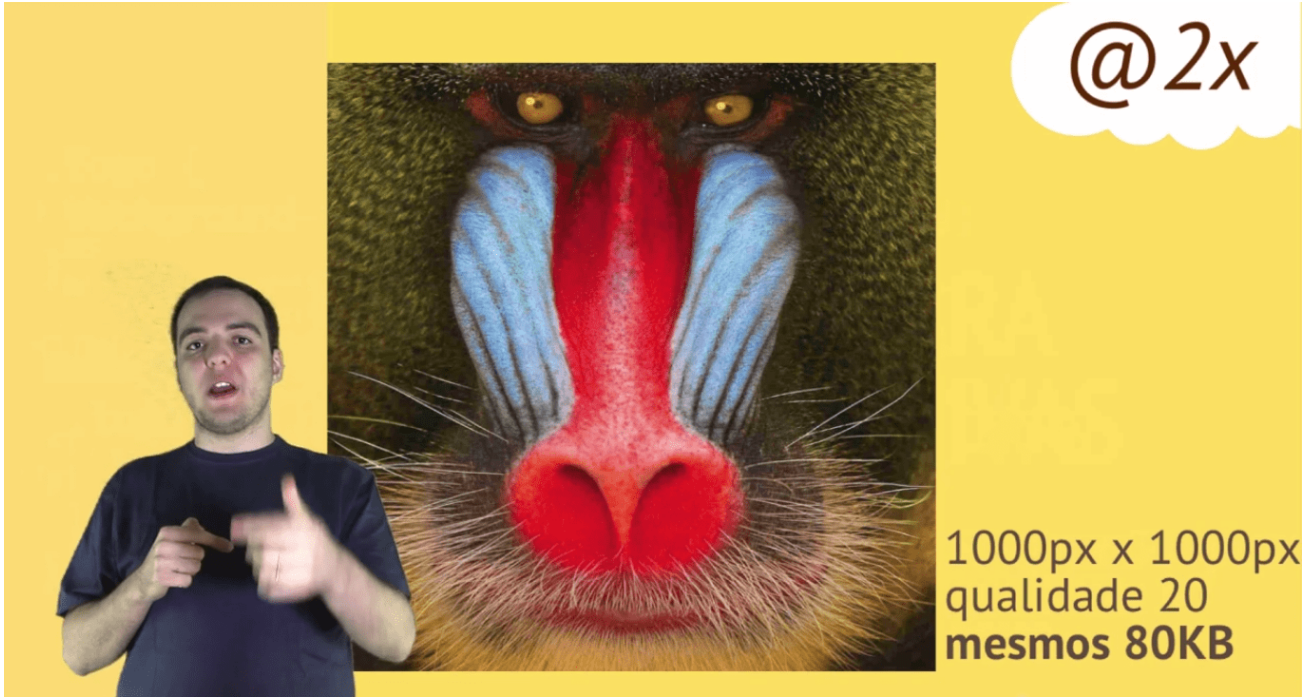
Compressive Images

Pegamos uma imagem de tamanho *500px* x *500px* - relativamente grande para a *Web* - e a exportamos com uma qualidade de 80, com *80KB*, tudo para telas comuns, de 1:1. Queremos que essa imagem tenha a mesma qualidade numa

tela de retina, ou seja, precisaremos do dobro da resolução (em ambas as dimensões)



Em uma resolução de *1000px* x *1000px* a imagem já está muito pesada. Veja essa segunda imagem, ela possui o mesmo tamanho de *80KB*, porém é quatro vezes maior:



A diferença é que, na hora de exportarmos, diminuímos a qualidade de 80 para 20. E perceba que a imagem não fica pixelada ou esteticamente pior necessariamente.

Se utilizarmos uma imagem para telas de retina de baixa resolução, ela será usada em dois cenários diferentes:

- Em uma tela de retina propriamente dita, ocupando *1000px* x *1000px*. Porém, lembremos que os pixeis físicos são muito pequenos nessa tela e, ao usarmos uma imagem de qualidade baixa, mas com pixeis muito pequenos, o resultado visual é muito bom.
- Em uma telas normais, ocupando *1000px* x *1000px*. Como tais telas possuem menos pixeis, o navegador compactará a imagem para *500px* x *500px*, ou seja, teremos mais pixeis do que o necessário. O resultado final é que a imagem também fica boa.

Perceba que com a técnica de *Imagens Compressivas* conseguimos utilizar apenas uma imagem em dois casos. Uma imagem grande de qualidade baixa. Nos exercícios veremos outros exemplos.

Mudança de formato

Uma outra técnica de otimização é reexportar a imagem com outro formato. Uma imagem pode ser otimizada ao mudarmos seu formato de PNG para PNG-8, por exemplo, que é muito bom para imagens em preto e branco ou com menos de 256 cores.

Nos exercícios iremos treinar todas as técnicas que vimos nesta aula.