### Lógica da Programação 2

## Canvas será nossa tela

Nosso primeiro passo nesta aula será criarmos nosso programa.html.

Abriremos nosso editor de texto e salvaremos o arquivo em nossa área de trabalho, com o nome programa.html.

Aqui estamos utilizando o Sublime 2, mas você pode utilizar qualquer editor de sua preferência.

Lembrando que, no curso anterior de Lógica de Programação, vimos que quando queremos escrever um código em JavaScript utilizamos as tags <meta charset=UTF-8"> e a <script> para podermos escrever nosso código.

Conforme vimos, tudo que está dentro da tag <script> pertence ao mundo JavaScript, e tudo que está fora - seja antes ou depois -, pertence ao mundo HTML.

Aprendemos no mundo JavaScript, a utilizar funções do mundo HTML por meio do document, por exemplo o document.write() que nos permite imprimir algo:

<meta charset="UTF-8">

<script>

document.write("");

</script>

Podemos, por exemplo, escrever a mensagem "Oi":

<meta charset="UTF-8">

<script>

document.write("<h1>Oi</h1>");

</script>

Salvaremos o programa e retornaremos ao navegador. Nele, escolheremos a opção de abrir arquivo, e selecionaremos o programa.html, assim, temos a seguinte exibição:

Oi

Entretanto, nosso objetivo não é escrever HTML no mundo HTML, em vez disso, queremos desenhar. Sendo assim, não utilizaremos o document.write().

Para desenhar precisamos: de uma tela e de um pincel.

Vamos acessar a página do [Google](http://www.google.com.br/) e digitar "*empty canvas*". A palavra "*empty*" significa "vazio", em Português.

Observando os resultados, vemos que são exibidas imagens de telas em branco. Os mundos HTML e JavaScript utilizam o Inglês, por isso, as tags estão neste idioma, "*canvas*" também é uma tag do mundo HTML.

Se inserirmos em nosso código a tag <canvas>, ela é válida do mundo HTML e tem a mesma finalidade de uma tela em branco, ou seja, uma área em que podemos desenhar.

Como vamos trabalhar com desenhos, não iremos utilizar textos, não será necessária a inclusão da tag <meta charset="UTF-8">, que serve para resolvermos problemas de acentuação.

Assim, temos o mínimo para podermos começar a programar:

<canvas></canvas>

<script>

</script>

Recapitulando: O *canvas* é uma área da tela onde podemos desenhar, escrever com um pincel.

Precisamos informar, em nosso programa, quanto o <canvas> ocupa de espaço. Para isso, utilizaremos dois atributos, o width, que em português é "largura", diremos que é 600, e o height, ou "altura", que será de 400.

Se salvarmos e recarregarmos a página, nada acontece para nós, visualmente. Isso porque o <canvas> é branco, por padrão. Como o fundo do navegador também é, não conseguimos ver o que acabamos de criar.

Para que possamos visualizar, faremos com que ele ganhe cor. É o que veremos adiante.

## Desenhando com um pincel

Nesta aula, aprenderemos a desenhar utilizando um pincel.

Nas aulas anteriores, aprendemos que o document possui uma função que nos permite utilizar algo do mundo HTML dentro do mundo JavaScript.

Para isso, criaremos a variável tela, e pediremos ao document.querySelector() o parâmetro "canvas" - note que está entre aspas:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

</script>

JavaScript pode utilizar tanto as aspas ("), quanto apóstrofo ('), que nos referimos como **aspas simples**. É o que utilizaremos até o final do treinamento, apenas para simplificar a digitação.

Assim, o documento tem o <canvas> e, dentro do JS, pedimos para o document executar a função querySelector() para pegar este <canvas> e passá-lo como valor na variável tela.

Isso nos dá a área para escrever, mas e o pincel? Criaremos uma variável pincel, que receberá como valor a tela:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.

</script>

A tela, assim como o document sabe buscar um elemento da página por meio da tag, e fornecer um retorno para a variável, poderá passar o pincel.

No mundo da programação podemos ter gráficos em 2D, 3D, 4D, enfim, portanto, precisamos informar à tela qual o tipo de pincel, ou seja, o contexto no qual escreveremos na tela. Para isso, utilizaremos o getContext() e passaremos o 2d como parâmetro:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

</script>

Já temos a tela e o pincel, portanto podemos escrever na tela.

Se o pincel foi dado pelo <canvas>, qualquer operação feita nele atuará sobre a tela.

Nossa primeira operação será desenhar um retângulo de dimensões 600x400, para preencher todo o <canvas>. Para fazer isso, chamaremos o pincel, e pediremos para ele, com o operador ., preencher o retângulo.

Preencher em Inglês é "fill", enquanto "rect" é a abreviação de retângulo:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillRect();

</script>

Ou seja, estamos pedindo ao JavaScript para preencher um retângulo. Entretanto, precisamos informar ao programa o ponto da tela onde desejamos começar a preencher.

Assim como um pintor inicia sua pintura em um ponto específico da tela, nós também precisamos posicionar nosso pincel no <canvas> para podermos iniciar nosso preenchimento.

Como sabemos, na matemática trabalhamos com a ideia de **planos cartesianos**, onde temos um eixo x, na horizontal, e um eixo y, na vertical.

De acordo com as dimensões que especificamos, começaremos nosso preenchimento no canto superior esquerdo, isso significa que iniciaremos no ponto 0 tanto no eixo x quanto y.

No código, estas coordenadas são inseridas entre os parênteses da instrução pincel.fillRect():

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillRect(0, 0);

</script>

Dessa coordenada, queremos criar uma forma que tenha 600 de largura, e 400 de altura, portanto, inseriremos isto em nosso código:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillRect(0, 0, 600, 400);

</script>

Salvaremos o programa e recarregaremos a página, e vemos na tela um retângulo preto, com as dimensões 600 (largura) x 400 (altura), posicionando o pincel a partir do ponto mais extremo no canto superior esquerdo.

Como estamos utilizando o fill, ou "preencher", ele completa o restante da área com preto.

Só que, a cor preta é muito forte, colocaremos uma cor mais suave. Para indicar qual a "tinta", temos de indicar o fillStyle:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle

</script>

Diferentemente de fillRect(), de getContext(), ou de querySelector(), o fillStyle **não é uma função**, e sim o que chamamos de **propriedade**. É equivalente a uma variável, por receber um valor, que é a cor. No caso, utilizaremos o 'lightgrey':

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'lightgrey';

</script>

Salvaremos e recarregaremos a página. Entretanto o retângulo continua preto, por quê?

Pensando em uma situação real, se pedimos para alguém pintar algo, a pessoa precisará da tinta. No caso, estamos passando as informações de cor após termos pintado, sem termos fornecido a tinta.

Sendo assim, o fillStyle deve vir antes do fillRect:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

</script>

Salvaremos e recarregaremos. Deu certo! Temos nosso <canvas> em cinza claro. Qualquer desenho que fizermos respeitará a área que delimitamos nas dimensões.

Aumentaremos as dimensões, para 800 de largura e 600 de altura:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector("canvas");

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 800, 600);

</script>

Salvaremos e recarregaremos a página. Não aconteceu nada. Isso porque, por mais que tenhamos informado novas dimensões, o máximo que caberá em nosso <canvas> é 600x400, portanto, respeitaremos isso.

## Nossa Primeira Obra de Arte. Será?

Nesta aula, daremos continuidade ao projeto da aula anterior, onde fizemos nosso primeiro desenho.

Nossa visualização será, no lado esquerdo, do <canvas> em sua totalidade, e, no lado direito, veremos nosso código.

Criaremos um novo retângulo, que terá 200 de largura e 400 de altura, na cor verde. Primeiro, selecionaremos a nova cor, utilizando o fillStyle:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

</script>

Em seguida, pediremos ao pincel que se movimente em nossa tela, por meio do fillRect na posição (0, 0, 200, 400):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

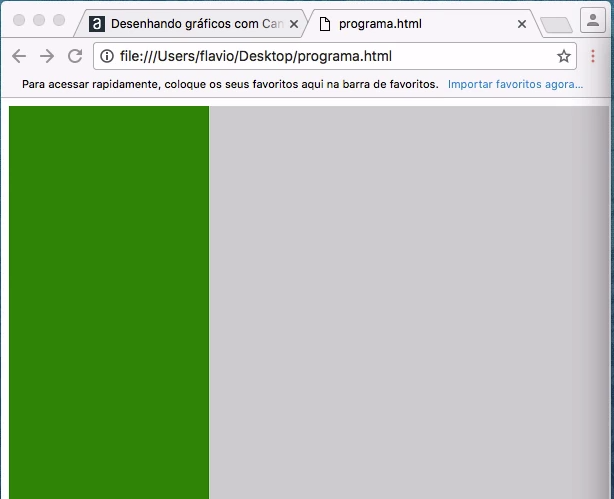
pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

</script>

Salvaremos e recarregaremos a página. Temos em nossa tela a exibição de um retângulo menor, ao lado esquerdo, na cor verde, e o resto do <canvas> preenchido em cinza, algo análogo à representação abaixo:



O próximo passo será criar um novo retângulo, de mesmo tamanho, no lado direito, na cor vermelha. Para isso, criaremos um novo pincel.fillStyle e um novo pincel.fillRect.

Como queremos que ele comece em determinado ponto do retângulo, ou seja, do eixo x, precisamos representar isso no fillRect. Já que este lado mede 600, e teremos três formas, cada uma delas terá 200 de largura, sendo assim, a terceira forma iniciará a partir do ponto 400:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

</script>

Salvaremos e recarregaremos, teremos na tela um retângulo verde, como na imagem anterior, mas agora temos também um retângulo cinza ao seu lado direito, de mesmo tamanho e, a sua direita, um retângulo vermelho com as mesmas dimensões.

Comentaremos o bloco que contém as informações com a cor cinza do <canvas>, em JavaScript isso pode ser feito utilizando o comando /\* indicando o ponto inicial do comentário, seguido do \*/ para finalizá-lo:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

/\*

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

\*/

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

</script>

Salvaremos e recarregaremos a página. Como podemos observar, não há mais a área cinza do <canvas>. Temos portanto a bandeira da Itália. Retornaremos com o código, para termos tudo funcionando:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

</script>

Salvaremos e recarregaremos.

Nosso próximo objetivo será desenhar um triângulo no centro do retângulo cinza, é o que veremos adiante.

## A vida não é só retângulo

Nesta aula, aprenderemos a criar um **triângulo**.

Para a criação desta forma geométrica, não utilizaremos o fillRect, em vez disso, queremos começar um caminho. O que é isso?

Começar, em inglês, é begin, e caminho é path. Portanto, utilizaremos um begin path para dizermos qual será a direção seguida por nosso pincel. O primeiro passo é utilizar o fillStyle, com a cor amarela:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

</script>

Com isso, indicaremos que nosso pincel iniciará seu caminho com o pincel.beginPath():

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

</script>

Para começarmos a escrever, indicaremos onde está o ponto inicial do nosso triângulo. Nossa primeira coordenada será centralizada na tela, para fazermos isso, utilizaremos a função moveTo:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo();

</script>

Pensando em nossas coordenadas, temos que posicionar nosso pincel exatamente no ponto 300, da coordenada X (que mede 600), e 200 na Y (que mede 400):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

</script>

Em seguida, moveremos o pincel para uma nova posição, com a função lineTo(). Esta posição será o ponto de encontro entre o verde e o cinza, e a linha inferior de nosso <canvas>, portanto, 200 no eixo X, e 400 no eixo Y:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

pincel.lineTo(200, 400);

</script>

O próximo movimento criará uma linha horizontal, do ponto que acabamos de criar até o ponto em que o cinza encontra o vermelho, de coordenadas 400 (X), e 400 (Y):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

pincel.lineTo(200, 400);

pincel.lineTo(400, 400);

</script>

Por fim, basta inserirmos um comando para que o programa preencha esta forma, que é o pincel.fill():

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

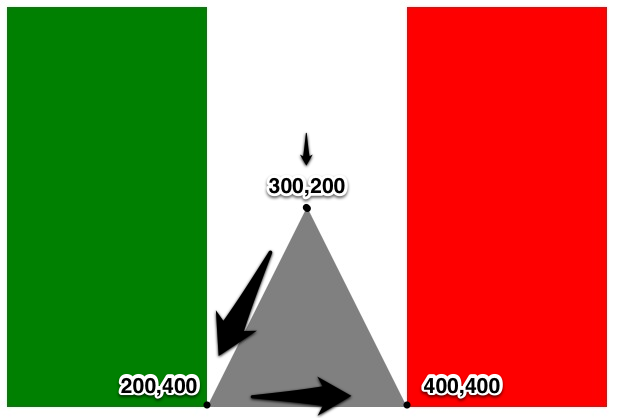
pincel.lineTo(200, 400);

pincel.lineTo(400, 400);

pincel.fill();

</script>

Salvaremos e recarregaremos a página. Temos nossa imagem, assim como antes, só que agora há um triângulo de cor cinza (apenas para ilustrar), com sua base no final do <canvas> e pico centralizado em relação a forma como um todo:

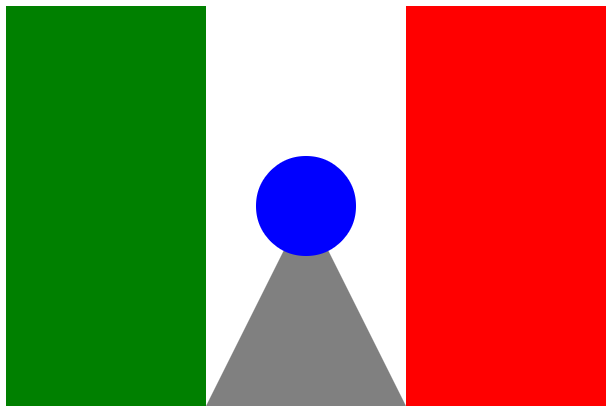


Como vimos, a construção do triângulo foi trabalhada a partir de um caminho, criamos o primeiro ponto, ou vértice, e em seguida os demais. Por último, fazemos com que o programa preencha esta forma com a cor que já escolhemos.

Nosso próximo desafio será criar uma esfera.

## Tem espaço para um circulo?

Podemos fazer qualquer figura que tenha lados usando essa fórmula, não apenas triângulos. Mas e se precisarmos de algo arredondado? Vamos colocar uma circunferência azul no meio da nossa imagem, pra ficar assim:



Para começarmos, indicaremos um novo pincel, com o fillStyle na cor azul, e um novo caminho, com o beginPath:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillSrtyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

pincel.lineTo(200, 400);

pincel.lineTo(400, 400);

pincel.fill();

pincel.fillStyle = 'blue';

pincel.beginPath();

</script>

Temos de iniciar um novo caminho, caso contrário, o programa entenderá que deve continuar o caminho anterior.

Com esta etapa concluída, partiremos para a utilização da função arc(), para traçarmos nossa esfera. Nela, incluiremos as seguintes informações:

* Posicionamento da esfera, que definiremos como 300 (X) e 200 (Y);
* Tamanho, ou seja, o raio - que definiremos como 50;
* O ângulo inicial, e o ângulo final, em radianos (multiplicado por PI - 3,14);

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

pincel.lineTo(200, 400);

pincel.lineTo(400, 400);

pincel.fill();

pincel.fillStyle = 'blue';

pincel.beginPath();

pincel.arc(300, 200, 50, 0, 2 \* 3.14);

</script>

Por enquanto, nosso foco não serão as questões matemáticas.

Em seguida, chamaremos a função fill para que a esfera seja preenchida:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgrey';

pincel.fillRect(0, 0, 600, 400);

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 200, 400);

pincel.fillStyle = 'red';

pincel.fillRect(400, 0, 200, 400);

pincel.fillStyle = 'yellow';

pincel.beginPath();

pincel.moveTo(300, 200);

pincel.lineTo(200, 400);

pincel.lineTo(400, 400);

pincel.fill();

pincel.fillStyle = 'blue';

pincel.beginPath();

pincel.arc(300, 200, 50, 0, 2 \* 3.14);

pincel.fill();

</script>

Salvaremos o programa e recarregaremos a página. Temos nossa esfera azul, exatamente no pico de nosso triângulo.

Para que pudéssemos construir esta imagem, precisamos conhecer a função getContext(), fillRect(), lineTo(), beginPath(), a propriedade fillStyle, a esse conjunto de propriedades damos o nome de **APIs**, ou bibliotecas. Aqui, estamos manipulando a API específica para a criação de gráficos.

Para descobrirmos estas bibliotecas, temos que pesquisar pela documentação da linguagem na internet, utilizando os termos "canvas 2d api".

O resultado escolhido está [disponível](https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D).

Clicamos no link das Web APIs, há materiais disponíveis sobre fillRect, fillStyle, entre outros. O que faremos é melhorar certos aspectos do nosso programa nas próximas aulas e organizar nosso código da melhor forma pois, se alguém que está iniciando na programação ler ele, não conseguirá identificar com clareza qual a sua função.

Adiante, organizaremos nosso código, para facilitar sua manutenção e legibilidade.

## Repetir código não rola, não é mesmo?

Dando continuidade ao nosso trabalho com formas, criaremos um novo programa. Podemos salvá-lo como programa2.html.

Primeiramente, ele terá nossas tags <script> e <canvas>:

<canvas width="600" height="400"></canvas>

<script>

</script>

Como sabemos, temos que transportar o <canvas> para o JavaScript, e faremos isso por meio do document.querySelector():

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

</script>

Importante ressaltar que estamos utilizando as "aspas simples" (apóstrofo) para escrevermos menos, e que isso **só pode acontecer no mundo JavaScript**, que aceita tanto as aspas simples quanto as duplas (").

Para podermos escrever na tela, precisaremos da variável pincel, que receberá um getContext para nos fornecer um pincel 2D:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

</script>

Nosso próximo objetivo será imprimir um quadrado pequeno.

O primeiro passo é determinar a cor, com o fillStyle, que será verde:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

</script>

O segundo passo será indicar **o que** será preenchido em verde, ou seja, nosso quadrado. Para isso, utilizaremos ofillRect e passaremos as coordenadas da forma como **parâmetros**:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

</script>

Salvaremos nosso programa. Abriremos o navegador e selecionaremos nosso arquivo, a partir do menu "Arquivo > Abrir arquivo...". Ao fazermos isso, temos a imagem do nosso quadrado.

E se quisermos colocar um outro quadrado ao lado deste que já existe? Podemos utilizar a mesma fórmula, mas precisamos ter cuidado com seu posicionamento no eixo X, para que não coincida com o que já temos.

Se nosso quadrado tem 50 de largura, isso significa que ele ocupa 50 pixels no eixo X, assim, nosso novo quadrado deve começar a partir do ponto 50 no eixo X:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

</script>

Criaremos, ainda, um terceiro, que iniciará no ponto 100 do eixo X:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>

Salvaremos e recarregaremos a página, e o que temos são os três quadrados, verdes, um ao lado do outros. Entretanto, para quem vê, parece ser um único retângulo posicionado horizontalmente. Colocaremos as bordas, para separar cada um dos quadrados.

Após fazermos o fillRect, podemos indicar ao pincel o tipo de strokeStyle, qual será a cor da borda, que no caso definiremos como preta:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>

Em seguida, indicaremos ao pincel que ele deve desenhar a borda, utilizando o strokeRect(). A borda deve ser inserida na mesma posição do quadrado:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

</script>

Salvaremos o programa e recarregaremos a página. O primeiro quadrado agora tem borda, enquanto os outros dois permanecem sem divisões ou delimitações. Repetiremos o strokeRect para os demais quadrados, lembrando que devemos replicar as respectivas posições:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

pincel.fillRect(50, 0, 50, 50);

pincel.strokeRect(50, 0, 50, 50);

pincel.fillRect(100, 0, 50, 50);

pincel.strokeRect(100, 0, 50, 50);

</script>

Salvaremos e recarregaremos o programa. Temos a imagem de três quadrados verdes, com bordas pretas, um ao lado do outro.

Vamos imaginar que nossa intenção fosse e criar apenas um quadrado, nosso código ficaria então da seguinte forma:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

</script>

É bastante código para desenhar apenas uma forma simples. Um iniciante em programação nem seria capaz de identificar, com facilidade, a finalidade deste código.

Idealmente, seria interessante se pudéssemos, com apenas uma linha de código, atingir esta mesma funcionalidade, por exemplo:

<script>

//outras propriedades...

desenhaQuadradoVerde();

</script>

Qualquer um que analisasse nosso código, saberia identificar o que esta linha faz - ela **desenha um quadrado verde**. Como aprendemos no módulo anterior, é possível fazermos isso por meio do uso de **funções**.

A **função** é um código que foi guardado para ser chamado posteriormente. Criaremos portanto a função desenhaQuadradoVerde():

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde() {

}

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

desenhaQuadradoVerde();

</script>

O conteúdo da nossa função será todo o bloco que cria nosso quadrado:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde() {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

}

desenhaQuadradoVerde();

</script>

Como já havíamos pré-definido, chamamos a função, sem esquecer dos parênteses (()).

Salvaremos e recarregaremos o programa. Ele continua funcionando, ainda temos nosso quadrado verde com borda preta. Entretanto, se agora quisermos criar um novo quadrado basta chamarmos a função novamente, e assim por diante, para quantos quadrados quisermos criar.

Por exemplo, para criarmos três quadrados chamaremos a função três vezes:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde() {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(0, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(0, 0, 50, 50);

}

desenhaQuadradoVerde();

desenhaQuadradoVerde();

desenhaQuadradoVerde();

</script>

Salvaremos e recarregaremos o programa. Entretanto, continuamos vendo apenas um quadrado! Por que isso aconteceu?

Isso ocorre porque todas as vezes que chamamos a mesma função, teremos os mesmos parâmetros, assim, os quadrados serão criados exatamente no mesmo lugar! Não é essa a intenção, queremos desenhar os quadrados lado a lado. É o que veremos adiante.

## Uma função mais genérica

Nesta aula, veremos como podemos deixar nossa função genérica, para que possa funcionar para diferentes objetos.

O fillRect e o strokeRect não poderão ser o mesmo para todos, deverá receber parâmetros diferentes para cada objeto que o usuário deseje criar. Faremos com que ela seja capaz de aceitar o parâmetro x, que será aceito em fillRect() e strokeRect():

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde();

desenhaQuadradoVerde();

desenhaQuadradoVerde();

</script>

Assim, da próxima vez que chamarmos a função desenhaQuadradoVerde() faremos com que o primeiro inicie em 0, o segundo em 50, e o terceiro em 100:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde(0);

desenhaQuadradoVerde(50);

desenhaQuadradoVerde(100);

</script>

Salvaremos e recarregaremos. Temos assim os três quadrados, lado a lado.

Recapitulando:

Ao chamarmos a função desenhaQuadradoVerde() e passarmos 0 como parâmetro, este número ocupa o parâmetro x e é utilizado na coordenada X do fillRect e do strokeRect. Na próxima chamada da função, o parâmetro passado será 50, e este será o número considerado no fillRect e no strokeRect, e assim sucessivamente.

Com isso, conseguimos escrever um código genérico, para desenhar o quadrado verde.

Mas e se quisermos alinhar os quadrados não mais no eixo X, e sim no eixo Y? Nesta hipótese, além do x, teremos também o parâmetro y:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, 0, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, 0, 50, 50);

}

desenhaQuadradoVerde(0);

desenhaQuadradoVerde(50);

desenhaQuadradoVerde(100);

</script>

Portanto, substituiremos os 0s por ys em nosso código construtor e, no chamamento da função, teremos que passar dois parâmetros, da seguinte forma:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadradoVerde(0, 0);

desenhaQuadradoVerde(0, 50);

desenhaQuadradoVerde(0, 100);

</script>

Salvaremos e recarregaremos o programa. Temos todos os quadrados alinhados verticalmente, no lado esquerdo da tela.

E se agora quisermos criar um quadrado vermelho? Podemos copiar o código da função que já temos e realizar algumas alterações.

Primeiro, alteraremos o nome dela, para desenhaQuadradoVermelho, e onde temos 'green' alteraremos para 'red':

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

function desenhaQuadradoVermelho(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'red';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadradoVerde(0, 0);

desenhaQuadradoVerde(0, 50);

desenhaQuadradoVerde(0, 100);

</script>

Em seguida, alteraremos um dos chamados da função para desenhaQuadradoVermelho:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadradoVerde(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

function desenhaQuadradoVermelho(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'red';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadradoVerde(0, 0);

desenhaQuadradoVermelho(0, 50);

desenhaQuadradoVerde(0, 100);

</script>

Salvaremos e recarregaremos a página. Assim, temos dois quadrados verdes nas pontas, e um vermelho ao meio.

E se quisermos quadrados de outras cores? Podemos passar a cor também como um parâmetro, assim não precisaremos alterar todas as vezes que quisermos um quadrado com uma nova cor.

Assim, a função passará a se chamar apenas desenhaQuadrado:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'green';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0);

desenhaQuadrado(0, 50);

desenhaQuadrado(0, 100);

</script>

Primeiro, teremos que passar um parâmetro a mais, que será a cor:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'red';

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'blue');

desenhaQuadrado(0, 50, 'red');

desenhaQuadrado(0, 100, 'yellow');

</script>

Em seguida, teremos que informar à função que ela receberá mais um parâmetro, na própria construção da função, que chamaremos de cor, e em vez de termos a cor no fillStyle, substituiremos também por cor:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'blue');

desenhaQuadrado(0, 50, 'red');

desenhaQuadrado(0, 100, 'yellow');

</script>

Salvaremos e recarregaremos a página. Temos a exibição dos três quadrados alinhados verticalmente, o primeiro na cor azul, em seguida na cor vermelha, e por último na cor amarela.

Como podemos observar, a ideia de função é importante para economizarmos linhas de código, e deixá-lo mais legível.

Retomaremos o desenho com os quadrados verdes, alinhados horizontalmente, no eixo X:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

desenhaQuadrado(0, 0, 'green');

desenhaQuadrado(50, 0, 'green');

desenhaQuadrado(100, 0, 'green');

</script>

Salvaremos e recarregaremos, confirmando a linha horizontal com três quadrados verdes.

## Como eu amo esse tal de loop!

Agora que aprendemos a criar uma função genérica, proponho uma pergunta: e se quisermos preencher <canvas> com quadrados?

Como nosso tela tem 600 de largura, teríamos que colar linha por linha, de 50 em 50, até completarmos 600 - seriam 30 linhas de código. Mas nós temos um padrão, em que a coordenada X deve ser repetida de 50 em 50 no código.

Como sabemos, podemos trabalhar com estruturas de repetição, chamadas de **laços**. Um deles é o **while**, que funciona enquanto alguma condição for verdadeira. A sua função será inserida em um bloco, entre chaves ({}):

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

while() {

desenhaQuadrado(0, 0, 'green');

}

</script>

Enquanto a condição do while for verdadeira, o desenhaQuadrado será repetido.

Primeiro, declararemos o a variável x, que iniciará em 0:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

var x = 0;

while() {

desenhaQuadrado(0, 0, 'green');

}

</script>

A condição de validade será que o while deve ser repetido enquanto x for menor que 600. Então, passaremos como parâmetro a própria variável x:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

}

</script>

Para podermos passar para o próximo quadrado, temos que avançar 50 pixels, assim, diremos que, ao ser executada a primeira função, x será acrescido de 50:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

x = x + 50;

}

</script>

Todas as vezes que o *loop* for executado, x será acrescido de 50, até o limite de 600.

Para criarmos, abaixo da linha de quadrados verdes, uma linha de quadrados vermelhos, basta declararmos a função desenhaQuadrado() novamente, com um parâmetro y correspondente à nova posição:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

desenhaQuadrado(x, 50, 'red');

x = x + 50;

}

</script>

Salvaremos e recarregaremos a página. Podemos ver uma linha de quadrados verdes, e abaixo, uma linha de quadrados vermelhos. Podemos fazer o mesmo processo com a cor azul:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

desenhaQuadrado(x, 50, 'red');

desenhaQuadrado(x, 100, 'blue');

x = x + 50;

}

</script>

Ao salvarmos e recarregarmos as páginas, veremos as três linhas de quadrados. Temos que prestar atenção aos valores do eixo Y, pois se houver dois iguais, prevalecerá o que foi desenhado por último.

Podemos obter este mesmo resultado utilizando um for. Deixaremos o método while() comentado. O trecho será destacado com \*/:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

/\*

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

x = x + 50;

}

\*/

</script>

O for nos permite criar uma variável internamente, portanto nossa var x = 0 estará inserida nele, assim como a condição x < 600, e o incremento x = x + 50:

<canvas width="600" height="400"></canvas>

<script>

function desenhaQuadrado(x, y, cor) {

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = cor;

pincel.fillRect(x, y, 50, 50);

pincel.strokeStyle = 'black';

pincel.strokeRect(x, y, 50, 50);

}

/\*

var x = 0;

while(x < 600) {

desenhaQuadrado(x, 0, 'green');

x = x + 50;

}

\*/

for(var x = 0; x < 600; x = x + 50) {

desenhaQuadrado(x, 0, 'green');

}

</script>

Salvaremos e recarregaremos a página. Temos o mesmo resultado que tivemos com o while, uma linha com quadrados verdes. Portanto, utilizamos as duas estruturas de repetição possíveis para conseguir um mesmo resultado, com foco organizacional para o nosso código.

Nesta aula, iniciaremos um novo programa, ao qual demos o título programa3.html. É um <canvas>, de dimensão 600x400, no qual utilizamos o pincel e preenchemos em cinza:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'grey';

pincel.fillRect(0, 0, 600, 400);

</script>

Até então, nós havíamos trabalhado com formas geométricas, como esferas e quadrados, pensando também na organização de nosso código. Nosso objetivo a partir de agora será fazer com que o nosso <canvas> interaja com o usuário.

Se você pensa em seguir e construir jogos, temos que ter uma maneira de interagir com o usuário. Ele deve ser capaz de clicar no <canvas> e isso dar início a alguma funcionalidade.

De início, faremos com que, ao clicar no <canvas>, seja exibido um alerta.

Para começar, criaremos uma função chamada exibeAlerta(), que exibirá a mensagem Cliquei:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'grey';

pincel.fillRect(0, 0, 600, 400);

function exibeAlerta() {

alert('Cliquei');

}

</script>

Salvaremos e recarregaremos a página, porém, nada acontecerá se clicarmos nela. Para que o alerta seja exibido, temos que chamar a função:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'grey';

pincel.fillRect(0, 0, 600, 400);

function exibeAlerta() {

alert('Cliquei');

}

exibeAlerta();

</script>

Se recarregarmos a página agora, o exibeAlerta é exibido automaticamente. Não é este o objetivo, queremos que o alerta seja exibido somente quando a página for clicada.

Como vimos, o JavaScript nos permite trabalhar com **eventos**. No caso, utilizaremos a propriedade onclick, tudo que atribuirmos a ela - se for uma função - esta será chamada pelo clique. Assim, conectaremos o exibeAlerta ao onClick:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'grey';

pincel.fillRect(0, 0, 600, 400);

function exibeAlerta() {

alert('Cliquei');

}

tela.onclick = exibeAlerta;

</script>

Lembrando que não podemos utilizar os parênteses neste último caso pois, assim executaríamos a função automaticamente, em vez de guardá-la para ser executada ao clique.

Salvaremos o programa e recarregaremos a página. Ao clicarmos na tela, é exibido um alerta com a mensagem "Cliquei". Funcionou!

Neste caso, quem chama a função exibeAlerta? O próprio navegador, ao clicarmos sobre o <canvas>. Além de exibirmos um alerta, é importante que nosso programa seja capaz, também, de identificar em qual posição a tela foi clicada.

Como podemos fazer isso?

Ao clicarmos na tela, como dito anteriormente, o próprio navegador é quem chama a função exibeAlerta. Toda vez que faz isso, ele passa um parâmetro especial para a função - até então, não havíamos feito isso em nosso código.

Nós criaremos este parâmetro, que chamaremos de evento, e utilizarmos o console.log(evento) para que ele seja exibido no navegador:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'grey';

pincel.fillRect(0, 0, 600, 400);

function exibeAlerta(evento) {

alert('Cliquei');

console.log(evento);

}

tela.onclick = exibeAlerta;

</script>

Isso fará com que tenhamos acesso a este parâmetro passado pelo navegador, e assim podemos descobrir a posição exata do clique.

Após recarregaremos o programa, clicaremos na tela, onde será exibido o alerta. Em seguida, abriremos o console - utilizando o atalho "F12" -, e teremos o MouseEvent, com todos os detalhes do evento.

O nome do parâmetro que nós passamos poderia ser qualquer um, o importante é **haver um parâmetro**.

Se expandirmos o MouseEvent, clicando na seta antes de seu nome, há diversos detalhes sobre ele, inclusive a posição do mouse dentro do <canvas>.

Antes de aprendermos a obter estas informações, vamos recapitular:

* Criamos uma função chama exibeAlerta;
* Ela recebe como parâmetro um evento;
* Em seu bloco, ela exibe o alerta 'Cliquei' apenas, e faz um console.log(evento);

Se chamarmos o exibeAlerta, temos que passar um parâmetro, mas não temos como saber de antemão qual ponto da tela será clicado. Assim, quem chama essa função é, exclusivamente, o navegador, ele quem tem o parâmetro que trará para nós as coordenadas da posição do cursor no momento do clique.

Adiante, veremos como obter tais coordenadas.

## Alvo aleatório

Para darmos início à elaboração de nosso jogo, temos que começar pelo desenho do alvo.

Este objeto é construído pela função desenhaCirculo(), que já temos em nosso código:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio) {

pincel.fillStyle = 'blue';

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

</script>

Ela deverá receber mais um parâmetro(a cor) e será utilizado no fillStyle. Temos isso porque haverá círculos na cor branca e vermelha:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

</script>

Além disso, temos que saber que nosso raio padrão terá o valor inicial de 10, os demais serão maiores, para criar a imagem de um alvo.

Para efetivamente desenharmos o círculos, temos que chamar a função desenhaCirculo(), com os parâmetros x e y ambos com valor 100, raio 10 e a cor vermelha (red), já que estamos lidando com o círculo central:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, 10, 'red');

</script>

O próximo círculo será construído na mesma posição, entretanto, o seu raio será maior, equivalente a 30, e sua cor será branca white:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, 10, 'red');

desenhaCirculo(100, 100, 30, 'white');

</script>

Por fim, desenharemos um círculo ainda maior, com raio 40, e novamente na cor vermelha (red):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, 10, 'red');

desenhaCirculo(100, 100, 30, 'white');

desenhaCirculo(100, 100, 40, 'red');

</script>

Salvaremos e recarregaremos a página. Por que será exibido somente um círculo vermelho? Porque a última função que passamos foi para a criação de um círculo maior que todos os demais, assim, ela cobriu todas as outras. Como consertaremos isso?

Basta invertermos a ordem de chamamento das funções, se chamarmos o círculo vermelho maior primeiro, e o círculo vermelho menor por último, da seguinte forma:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, 40, 'red');

desenhaCirculo(100, 100, 30, 'white');

desenhaCirculo(100, 100, 10, 'red');

</script>

Aprimoraremos nosso código, usando o raio inicial de 10 como parâmetro, guardando-o em uma variável. Em seguida, acrescentaremos os valores necessários para compor os raios das demais circunferências. Criaremos a variável raio = 10 a seguir:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, 40, 'red');

desenhaCirculo(100, 100, 30, 'white');

desenhaCirculo(100, 100, 10, 'red');

</script>

Nos chamamentos das funções, trocaremos os valores pela variável:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

desenhaCirculo(100, 100, raio+20, 'red');

desenhaCirculo(100, 100, raio+10, 'white');

desenhaCirculo(100, 100, raio, 'red');

</script>

Salvaremos e recarregaremos a página. Pronto! Temos nosso alvo.

Com o alvo criado, precisamos fazer com que seu aparecimento seja aleatório, ou seja, que sejam criadas coordenadas arbitrárias. Para isso, vamos inserir as instruções para a criação do alvo dentro de uma nova função chamada desenhaAlvo(), e a chamaremos logo em seguida:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo() {

desenhaCirculo(100, 100, raio+20, 'red');

desenhaCirculo(100, 100, raio+10, 'white');

desenhaCirculo(100, 100, raio, 'red');

}

desenhaAlvo();

</script>

Salvaremos e recarregaremos e podemos ver que o alvo permanece inalterado.

Como queremos que o alvo seja aleatório, a função desenhaAlvo() receberá parâmetros referentes às coordenadas X (até o limite de 600) e Y (até o limite de 400), com relação ao desenhaCirculo(), elas receberão estes parâmetros, respectivamente:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

desenhaAlvo();

</script>

Para chamar o desenhaAlvo(), passaremos os parâmetros (200, 200):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

desenhaAlvo(200, 200);

</script>

A posição do alvo variará de acordo com os valores que passarmos como parâmetros em desenhaAlvo(), passaremos, por exemplo (50, 200), e teremos uma posição diferente.

O próximo passo será fazer com que os valores das coordenadas sejam gerados aleatoriamente.

Criaremos uma função chamada sorteiaPosicao(), que receberá um valor maximo que pode sortear entre 0 a 600 (para o eixo X), e 0 a 400 (para o eixo Y):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

}

desenhaAlvo(50, 200);

</script>

Criaremos uma variável xAleatorio que recebe sorteiaPosicao, já que estamos trabalhando com a variável X, então o valor máximo será 600. Do mesmo modo, criaremos uma variável yAleatorio que receberá sorteiaPosicao, cujo máximo será 400:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

}

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(50, 200);

</script>

O próximo passo é passarmos estes valores para a função desenhaAlvo():

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

}

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(xAleatorio, yAleatorio);

</script>

Ainda precisamos programar a função sorteiaPosicao(). Ela nos dará um retorno - return - de Math.floor, que arredonda o número para baixo, diferente do Math.round, que o arredonda para cima. Em seguida temos `Math.random() \* maximo):

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

return Math.floor(Math.random() \* maximo);

}

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(xAleatorio, yAleatorio);

</script>

Sempre que salvarmos e recarregaremos a página, o alvo aparecerá em um novo ponto do nosso <canvas>.

Nosso próximo objetivo será fazer com que o alvo seja redesenhado em certos intervalos de tempo, pré-determinados.

Para isso, criaremos uma função atualizaTela(), que receberá as variáveis referentes ao X e Y aleatórios, bem como o chamamento de desenhaAlvo, da seguinte forma:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

return Math.floor(Math.random() \* maximo);

}

function atualizaTela() {

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(xAleatorio, yAleatorio);

}

</script>

Em seguida, utilizaremos o setInterval(), com os parâmetros atualizaTela e o intervalo de tempo 500, sem esquecer de incluir na função o limpaTela:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

return Math.floor(Math.random() \* maximo);

}

function atualizaTela() {

limpaTela();

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(xAleatorio, yAleatorio);

}

setInterval(atualizaTela, 500);

</script>

Salvaremos e recarregaremos. Pronto! Temos nosso alvo viajando pela tela, em pontos aleatórios. Como ainda está rápido, trocaremos o intervalo de tempo para 1000:

<canvas width="600" height="400"></canvas>

<script>

var tela = document.querySelector('canvas');

var pincel = tela.getContext('2d');

pincel.fillStyle = 'lightgray';

pincel.fillRect(0, 0, 600, 400);

var raio = 10;

function desenhaCirculo(x, y, raio, cor) {

pincel.fillStyle = cor;

pincel.beginPath();

pincel.arc(x, y, raio, 0, 2 \* Math.PI);

pincel.fill();

}

function limpaTela() {

pincel.clearRect(0, 0, 600, 400);

}

function desenhaAlvo(x, y) {

desenhaCirculo(x, y, raio+20, 'red');

desenhaCirculo(x, y, raio+10, 'white');

desenhaCirculo(x, y, raio, 'red');

}

function sorteiaPosicao(maximo) {

return Math.floor(Math.random() \* maximo);

}

function atualizaTela() {

limpaTela();

var xAleatorio = sorteiaPosicao(600);

var yAleatorio = sorteiaPosicao(400);

desenhaAlvo(xAleatorio, yAleatorio);

}

setInterval(atualizaTela, 1000);

</script>

Nosso próximo objetivo é tornar o objeto responsivo ao clique, com o alerta de que o usuário acertou ao clicar sobre o alvo.