

Habilidades trabalhadas nesta aula:

- (EM13CO02)** Explorar e construir a solução de problemas por meio de refinamentos, utilizando diversos níveis de abstração desde a especificação até a implementação.
- (EM13CO18)** Planejar e gerenciar projetos integrados às áreas de conhecimento de forma colaborativa, solucionando problemas, usando diversos artefatos computacionais.
- (EMIFMAT04)** Reconhecer produtos e/ou processos criativos por meio de fruição, vivências e reflexão crítica na produção do conhecimento matemático e sua aplicação no desenvolvimento de processos tecnológicos diversos.

## Aula 1

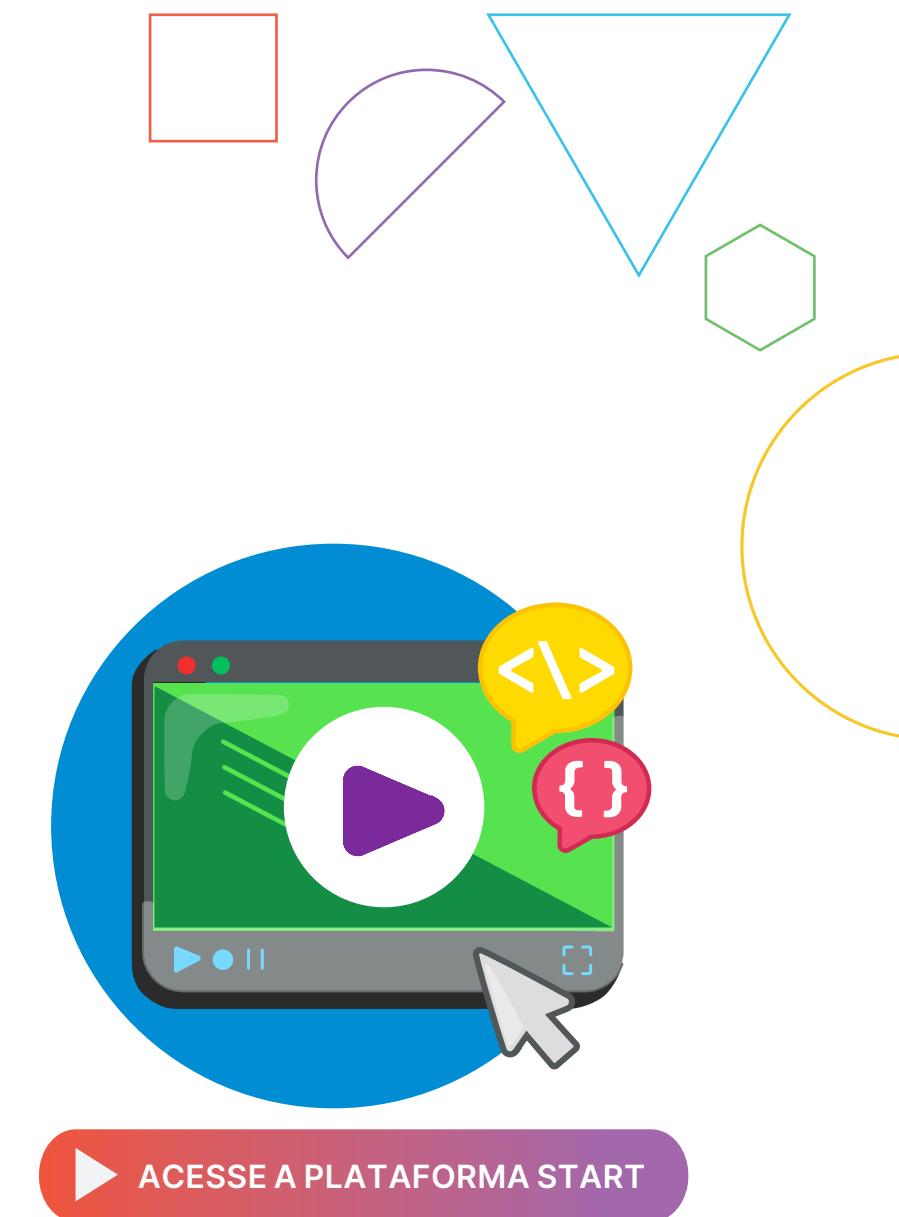
# Iniciando nossa jornada

► Unidade

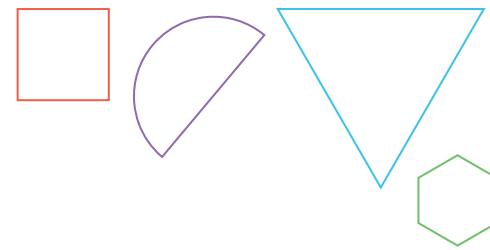
Lógica de programação:  
criando arte interativa com P5.js

# O que vamos aprender?

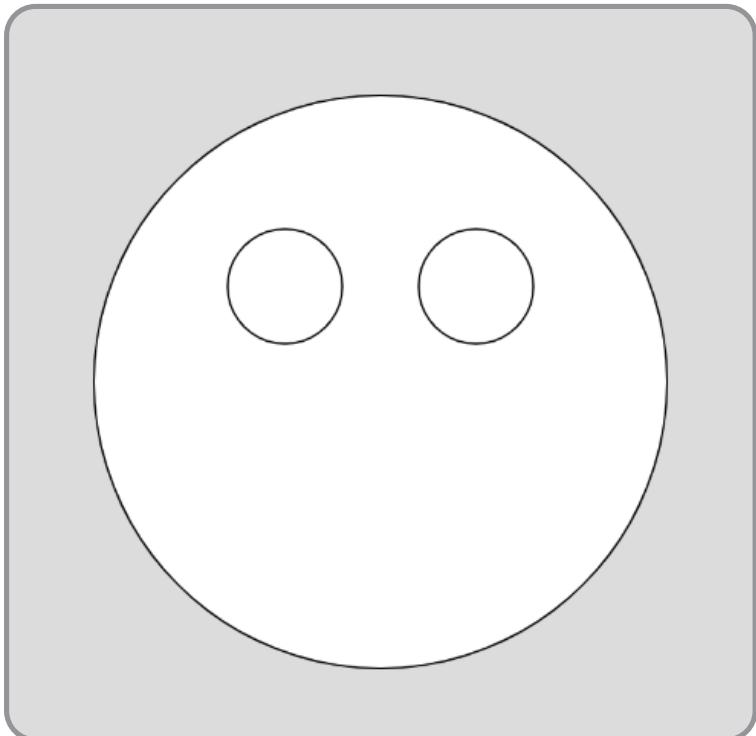
- Compreender os conceitos básicos de funções em JavaScript.
- Aplicar comandos básicos de desenho no P5.js, como `createCanvas()` e `circle()`.
- Modificar parâmetros de comandos em P5.js para ajustar a posição e o tamanho de formas geométricas.



ACESSE A PLATAFORMA START

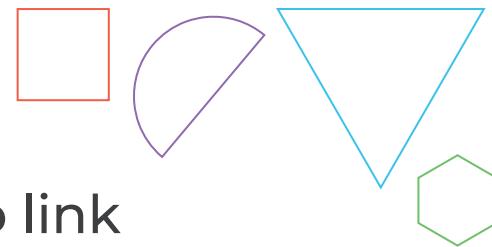


# Primeiras linhas de código



Nesta unidade, construiremos dois projetos: o Monalisa e o Quente e frio. O projeto Monalisa trabalhará com a ideia de uma forma que segue o ponteiro do mouse, enquanto o projeto Quente e frio abordará números aleatórios. Para isso, utilizaremos a ferramenta P5.js e a linguagem JavaScript. Nesta primeira aula, trabalharemos com o desenho e o posicionamento de formas geométricas, iniciando o projeto Monalisa.

Professor, nesta unidade, os estudantes iniciarão a construção do projeto Monalisa. A ideia é que eles criem uma obra de arte contendo olhos que seguem o ponteiro do mouse para simular o quadro Monalisa, que nos passa a impressão de que os olhos da personagem “seguem” quem está olhando para ela. Como forma de trazer contexto para o projeto, você pode iniciar a aula apresentando a obra aos estudantes e conversar sobre esse efeito que a pintura possui, além de apresentar sua história.



Começaremos o projeto acessando o site do P5.js através do link <https://editor.p5js.org/>. Observe que, antes de desenvolver o projeto, é necessário criar uma conta para que você possa salvar os códigos criados e retornar a eles sempre que necessário.

Para isso, acesse o site indicado e, para facilitar o processo, altere o idioma para português no menu de idiomas, localizado no canto superior direito da tela:

The screenshot shows the p5.js editor interface. At the top, there's a navigation bar with 'File ▾', 'Edit ▾', 'Sketch ▾', 'Help ▾', and a user profile icon. Below the navigation bar, there are buttons for play/pause, auto-refresh, and a text input field 'Cookie pantydraco'. The main area has a code editor titled 'sketch.js' with the following code:

```
1 function setup() {
2   createCanvas(400, 400);
3 }
4
5 function draw() {
6   background(220);
7 }
```

To the right of the code editor is a 'Preview' window showing a blank white canvas. On the far right, there's a language selection dropdown menu. The menu includes 'English ▾' (which is currently selected), 'Log in or Sign up', and a list of languages: বাংলা, Deutsch, English, Español, Français, हिन्दी, Italiano, 日本語, 한국어, Português (which is highlighted in red), Svenska, Türkçe, and Українська.



Ao fazer isso, você verá, ao lado do menu de idiomas, as opções *Entrar* ou *Registrar-se*. Clique em *Registrar-se*.



Em seguida, uma nova tela se abrirá solicitando informações para a criação de uma conta. Forneça essas informações e clique em *Entrar*, ou entre com sua conta Google ou do GitHub, se preferir.

Criar Conta

Nome de Usuário

Email

Senha

Confirmar Senha

Entrar

Ou

Entrar com GitHub

Entrar com Google



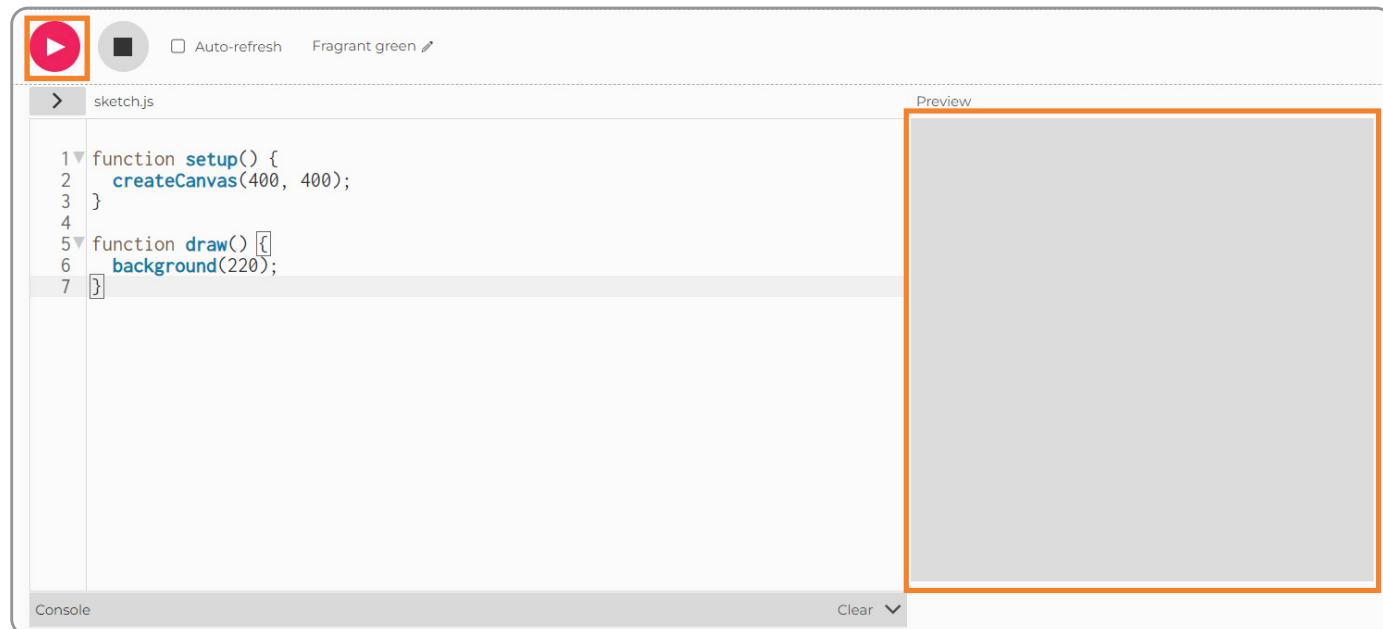
Logo de inicio, ao acessar a ferramenta, ela já mostra algumas linhas de código. Vamos compreendê-las!

A função **setup** serve para inicializar alguma coisa e, nesse caso, estamos inicializando um canvas, ou seja, uma tela de tamanho 400 de largura por 400 de altura.

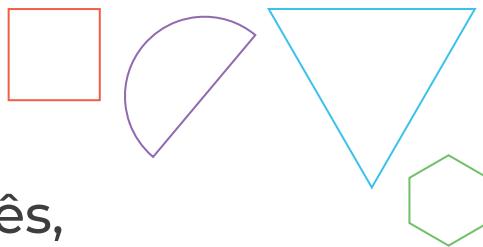
```
function setup() {  
    createCanvas(400, 400);  
}  
  
function draw() {  
    background(220);  
}
```



Podemos clicar no botão de executar, no canto superior esquerdo da tela, para ver como esse canvas aparece. Confira na imagem:



O quadrado cinza na área de Preview é o nosso canvas. Podemos alterar os valores dentro da função **createCanvas** para torná-lo maior ou menor.



Abaixo da função **setup**, temos a função **draw** (em português, desenhar). Todos os desenhos que criaremos precisam ficar dentro dessa função. Observe que já está sendo passada a função **background**, que é responsável por alterar a cor de fundo do canvas. Nesse caso, o valor **220** representa a cor cinza e, assim como na função anterior, podemos alterar esse valor e ver as cores que aparecem.

```
function setup() {  
  createCanvas(400, 400);  
}  
  
function draw() {  
  background(220);  
}
```

! Professor, recomenda-se verificar se os estudantes compreenderam essas duas funções e incentivá-los a testar diferentes tamanhos de canvas e diversas cores de fundo, de modo que eles fixem bem esses comandos iniciais.



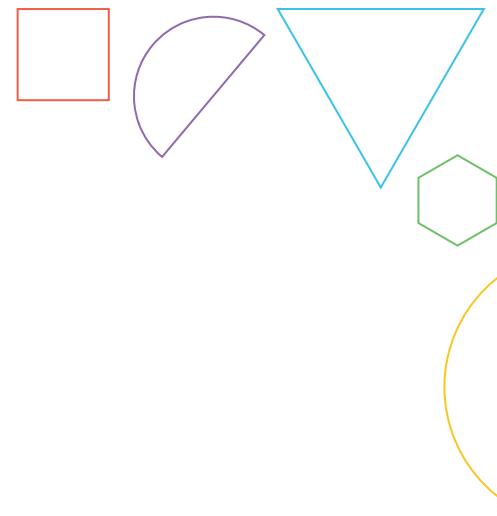
Tendo compreendido esses comandos iniciais, o primeiro desenho que criaremos será um círculo, que representará o rosto da nossa personagem. Para isso, utilizaremos a função **circle** (em português, círculo). Como você deve ter percebido, ao criar uma função, precisamos adicionar parênteses logo após seu nome e, dentro desses parênteses, vamos inserir alguns valores.

Para que a função **circle** funcione, precisamos de três valores: sua posição no eixo X, sua posição no eixo Y e o diâmetro do círculo. Confira o código:

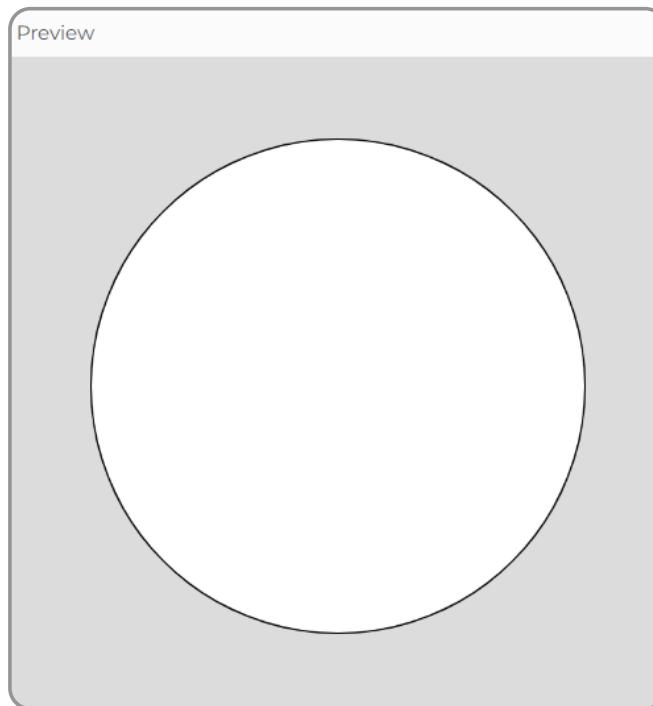
```
function setup() {
  createCanvas(400, 400);
}

function draw() {
  background(220);
  circle (200,200,300);
}
```

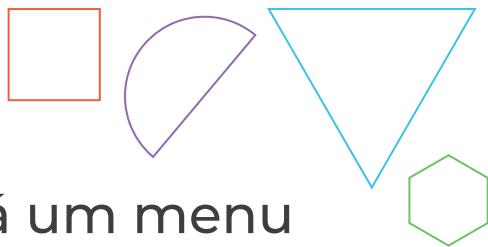
Os valores **200** e **200**, para os eixos X e Y, respectivamente, representam o centro do nosso canvas. O valor **300** representa o diâmetro do círculo.



Ao executar o código, o resultado será o seguinte:



O círculo aparece corretamente. Mas, e se quisermos adicionar outras formas? Como sabemos quais funções precisamos usar? Vamos descobrir!



Observe que, no canto superior direito da página do P5.js, há um menu chamado *Ajuda*. Nele, acessaremos a opção *Referência*, conforme imagem abaixo:



Ao clicar nessa opção, seremos redirecionados para a seguinte página:

p5.js ^

Reference  
Tutorials  
Examples  
Contribute  
Community  
About  
(</> Start Coding)  
Donate

English Accessibility Search

## Reference

Find easy explanations for every piece of p5.js code.

Filter by keyword

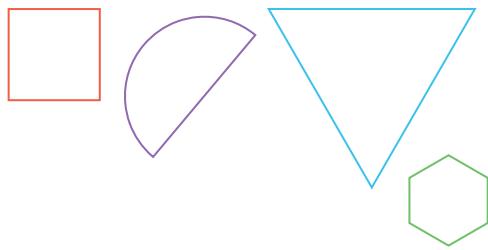
Jump to ▾  
Shape  
Color  
Typography  
Image  
Transform  
Environment  
3D  
Rendering

Looking for p5.sound? Go to the [p5.sound reference!](#)

### Shape

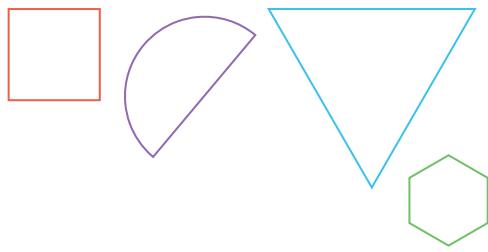
#### 2D Primitives

Looking for the old p5.js site? Find it [here!](#)



Veja que a página está toda em inglês e, nas opções de alterar idioma, o português não está disponível. Nesse caso, podemos usar o recurso de tradução do próprio navegador. Caso você esteja usando o Google Chrome, esse recurso estará disponível no final da barra de endereço, no lado direito da tela:

The screenshot shows a web browser window with the URL `p5js.org/reference/` in the address bar. The page content is in English, with the word "Referência" prominently displayed. At the top of the page, there are language selection dropdowns: one set to "Inglês" and another to "Acessibilidade". To the right of these, a Google Translate overlay is visible, with the "português" button highlighted by a red box. The overlay also displays the text "Google Translate". The browser's toolbar at the top includes various tabs and icons.



Ao selecionar essa opção, a página inteira será traduzida. Porém, precisamos tomar cuidado, pois o recurso de tradução do navegador traduz até mesmo o nome das nossas funções, como você pode ver na imagem abaixo:

## Primitivos 2D

`arco()`

Desenha um arco.

`círculo()`

Desenha um círculo.

`elipse()`

Desenha uma elipse (oval).

`linha()`

Desenha uma linha reta entre dois pontos.

`apontar()`

Desenha um único ponto no espaço.

`quadrângulo()`

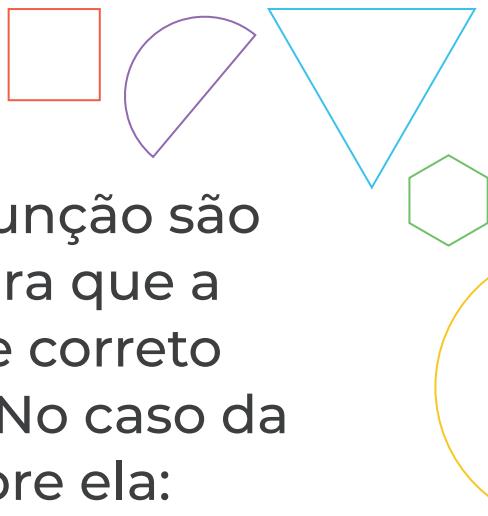
Desenha um quadrilátero (forma de quatro lados).

`reto()`

Desenha um retângulo.

`quadrado()`

Desenha um quadrado.



A função **círculo()** não existe no P5.js, pois os nomes de função são todos em inglês. Desse modo, precisamos tomar cuidado para que a tradução não gere erros ao programar. Para verificar o nome correto de cada função, basta acessá-la clicando sobre o seu nome. No caso da função **circle**, a seguinte página se abrirá ao clicarmos sobre ela:

Reference > **circle()**

## circle()

Draws a circle.

A circle is a round shape defined by the `x`, `y`, and `d` parameters. `x` and `y` set the location of its center. `d` sets its width and height (diameter). Every point on the circle's edge is the same distance,  $0.5 * d$ , from its center.  $0.5 * d$  (half the diameter) is the circle's radius. See [ellipseMode\(\)](#) for other ways to set its position.

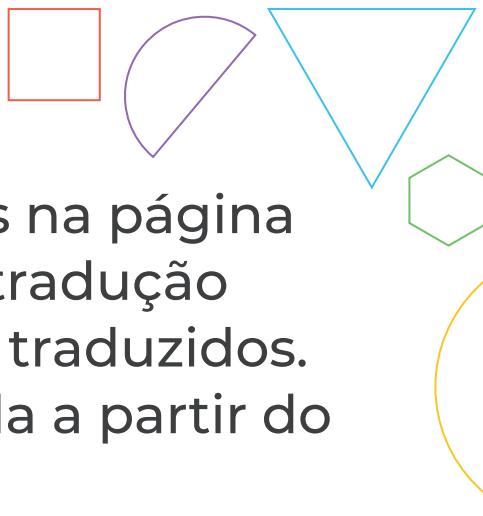
Examples

```
function setup() {  
  createCanvas(100, 100);  
  
  background(200);  
}
```

Rolando a página para baixo, teremos acesso à sintaxe da função, que nada mais é do que a maneira correta de escrevê-la no P5.js, com os valores que precisamos adicionar:

### Syntax

```
circle(x, y, d)
```

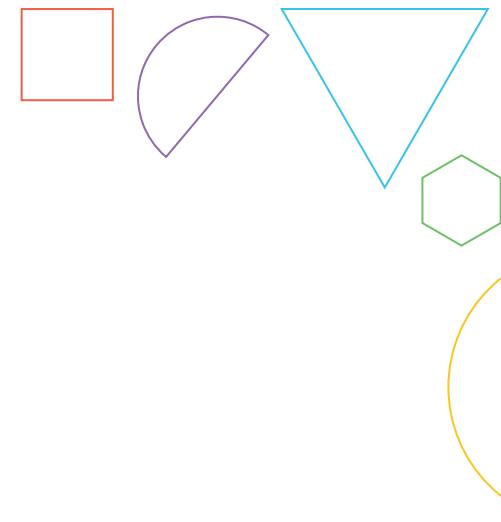


É importante que você explore as outras funções disponíveis na página de referência. Porém, tome cuidado ao utilizar o recurso de tradução do navegador para que os códigos em JavaScript não sejam traduzidos. Lembre-se de que essa linguagem de programação foi criada a partir do inglês e, portanto, não pode ser traduzida.

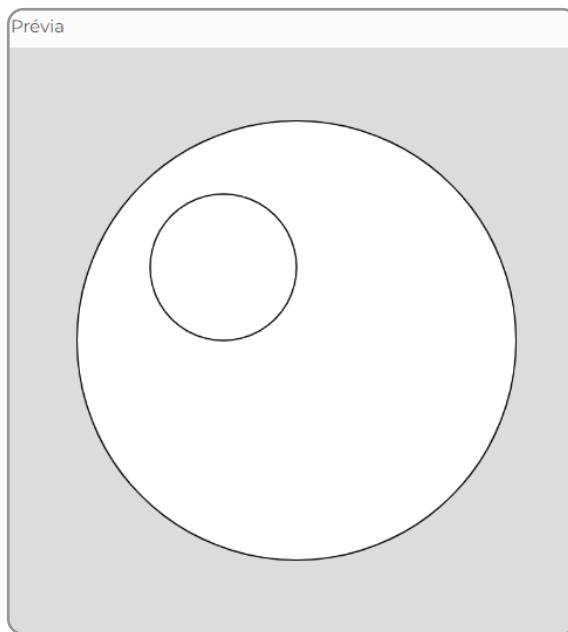
Agora, continuaremos a criar nossa obra de arte adicionando um olho! Para isso, utilizaremos a função **circle** novamente, porém, com valores diferentes. Confira o código:

```
function draw() {  
    background(220);  
    circle (200, 200, 300);  
    circle (150, 150, 100);  
}
```

Repare que modificamos os valores de x e y para **150**, de modo que esse segundo círculo apareça em uma posição diferente do centro.

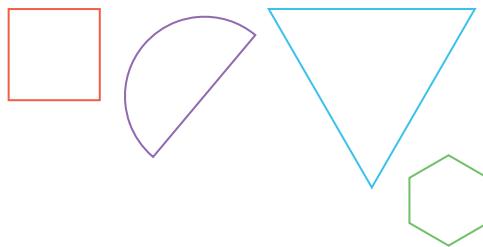


Ao executar o código, temos o seguinte resultado:



Observe que o olho está muito grande! Precisamos mudar o valor do diâmetro do círculo para corrigir esse problema. Para isso, podemos utilizar o valor 60 ou outro que você desejar. Além disso, vamos adicionar novamente a função **circle** para criar o outro olho, do lado direito. Nesse caso, precisamos também mudar o valor no eixo X.

Professor, lembre aos estudantes que eles não precisam seguir os mesmos valores da aula. Eles podem fazer o desenho conforme desejarem, explorando novas possibilidades.

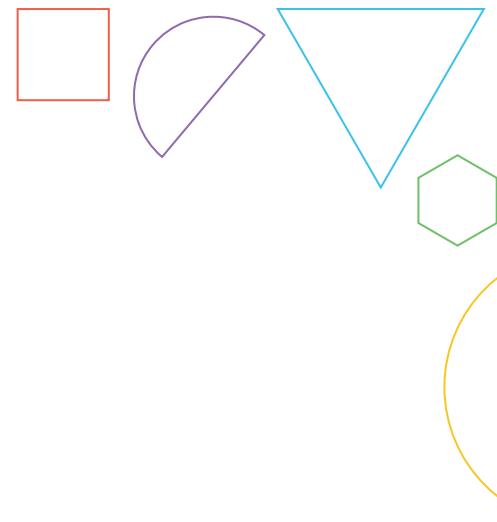


O código para adicionar dois olhos ficará assim:

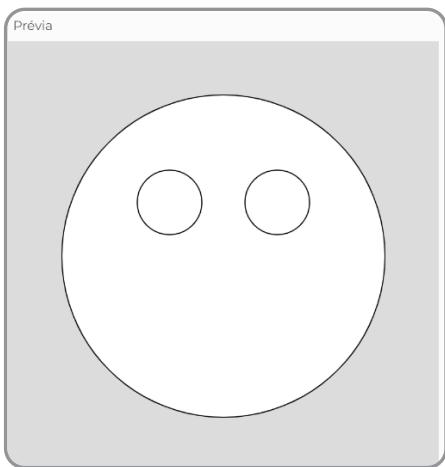
```
function draw() {  
    background(220);  
    circle (200, 200, 300);  
    circle (150, 150, 60);  
    circle (250, 150, 60);  
}
```

Perceba que o valor do eixo Y (**150**) não foi alterado, pois queremos que os dois olhos tenham a mesma altura. Entretanto, fique à vontade para testar valores diferentes e construir sua obra de arte como desejar.

! Professor, neste momento, se necessário, reforce com os estudantes a ideia de plano cartesiano, no qual o eixo X corresponde a um deslocamento horizontal e o eixo Y, a um deslocamento vertical. Essa noção é fundamental em diversos parâmetros de programação, sendo utilizada em muitas linguagens diferentes.



O resultado será o seguinte:



Antes de finalizar, lembre-se de salvar o projeto acessando o botão *Salvar*, no menu *Arquivo*, localizado no canto superior esquerdo da tela, ou utilizando o atalho *Ctrl+S*.



Nas próximas aulas, daremos seguimento ao projeto adicionando novas formas e explorando novas funções!

Professor, é importante lembrar aos estudantes que uma boa prática de programação é salvar regularmente enquanto estamos programando, e não apenas ao final. Lembre-os de, nas próximas aulas, utilizarem o atalho *Ctrl+S* com frequência e salvar constantemente o projeto no qual estão trabalhando.

► CLIQUE AQUI PARA AVALIAR ESTE MATERIAL