start by alura

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 3

Colorindo nosso desenho

▶ Unidade

Lógica de programação: criando arte interativa com P5.js

O que vamos aprender?

- Aplicar conceitos de cores em código usando o P5.js.
- Criar novas formas e elementos visuais (ex.: sobrancelhas, pupilas) na personagem utilizando as coordenadas do plano cartesiano.
- Documentar o código com comentários claros e organizados.

ACESSE A PLATAFORMA START

Cores

Na aula passada, adicionamos mais elementos e formas geométricas ao projeto e, agora, temos um nariz e uma boca compondo nossa personagem. Nesta aula, aprenderemos a adicionar mais algumas formas e a modificar os elementos existentes, deixando nosso projeto mais colorido.

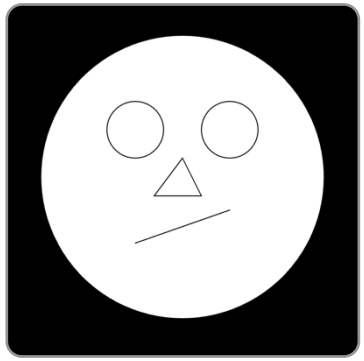
Professor, para iniciar a aula, sugira aos estudantes que tentem descobrir em qual local o código deve ser alterado para que possamos deixar os elementos do desenho coloridos. A seguir, o processo será descrito em detalhes.

Nas aulas anteriores, começamos a programar no P5.js, no qual trabalhamos um pouco com o conceito de **background()**. Por exemplo, dentro da função **draw()**, podemos trocar o valor do **background(220)** de 220 para 0. Observe:

```
function draw() {
  background(0);

  ...
}
```

Ao salvar e executar o código, perceberemos o seguinte resultado:



Veja que o fundo agora é preto. Ou seja, dependendo do valor que passamos nos parênteses do **background()**, conseguimos mudar a cor de fundo.

Também podemos adicionar tons mais claros ao fundo. Tínhamos o valor de 220, porém, se adicionarmos um número maior, como **`background(255)`**, e executarmos o código, o fundo ficará branco. Nesse caso, estamos trabalhando com números entre 0, que é preto, até 255, que é branco.

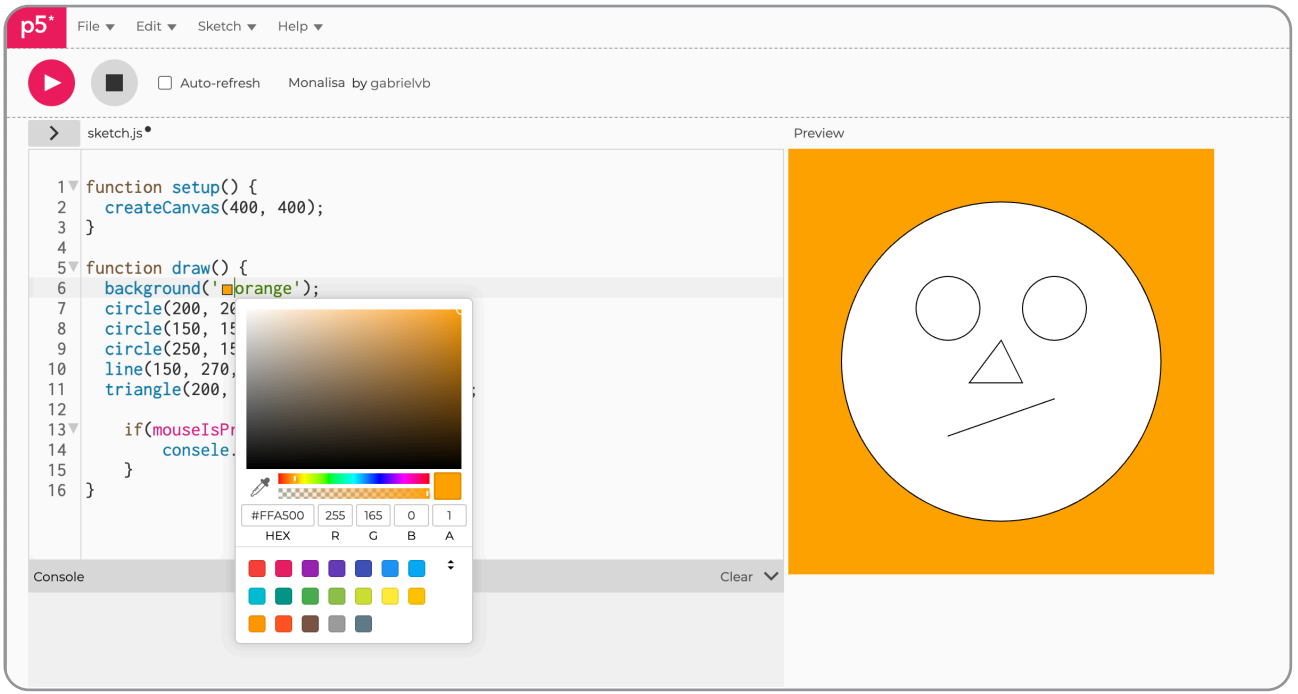
Contudo, se quisermos outras cores, como laranja, rosa ou vermelho, adicionaremos outro tipo de parâmetro. Assim, trabalharemos com palavras, escrevendo-as entre aspas simples. Portanto, escreveremos **`'orange'`** (laranja, em inglês):

```
function draw() {
  background('orange');
   ...
}
```

Ao executarmos o projeto, teremos um fundo laranja.

Contudo, existem tons que podemos ter dificuldade em definir com palavras. Para facilitar esse processo, usamos um código único, que é padronizado em todas as linguagens de programação e se inicia sempre com a hashtag (#). Essa codificação de cores é chamada de hexadecimal.

Assim, é possível definir exatamente os tons das cores que queremos ao clicarmos sobre o quadradinho colorido ao lado da cor. Esse clique abrirá um menu no qual poderemos escolher a cor e o tom desejados; observe:



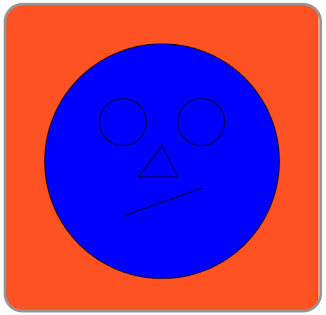
Professor, apresente aos estudantes outras ferramentas que também podem ser úteis para testar códigos hexadecimais e identificar códigos de cores. Ao pesquisar na internet por "cores hexadecimais", várias ferramentas serão indicadas.

Agora, temos um fundo laranja, mas todas as outras formas geométricas continuam na cor branca. Para alterar a cor dos demais elementos, o primeiro passo é iniciar uma nova linha de comando. Para isso, clicaremos no final da linha **`background('#FF5722');`** e pressionaremos *Enter*. Adicionaremos um novo comando chamado **`fill()`**, lembrando de digitar o ponto e vírgula (;) no final da linha.

Dentro dos parênteses, adicionaremos aspas simples e escreveremos, em inglês, o nome da cor para as próximas formas geométricas. Por exemplo, **`fill('blue');`**, já que blue é azul em inglês. Observe:

```
function draw() {
  background('#FF5722');
  fill('blue');
  ...
}
```

Assim, teremos o seguinte resultado:



Contudo, observe que todo o círculo que representa a cabeça do desenho ficou azul. Precisamos alterar a cor dos olhos e dos demais elementos do rosto da nossa personagem para melhorar sua aparência.

Nesse caso, clicaremos no final da linha **`circle(200, 200, 300);`** e pressionaremos *Enter* para adicionar uma nova linha. Nela, escreveremos **`fill('white');`**, já que white é branco em inglês. Observe:

```
function draw() {
  background('#FF5722');
  fill('#03A9F4');
  circle(200, 200, 300);
  fill('white');
  circle(150, 150, 60);
  ...
}
```

Como ficou o seu? Percebeu algum padrão de comportamento do comando **`fill()`**?

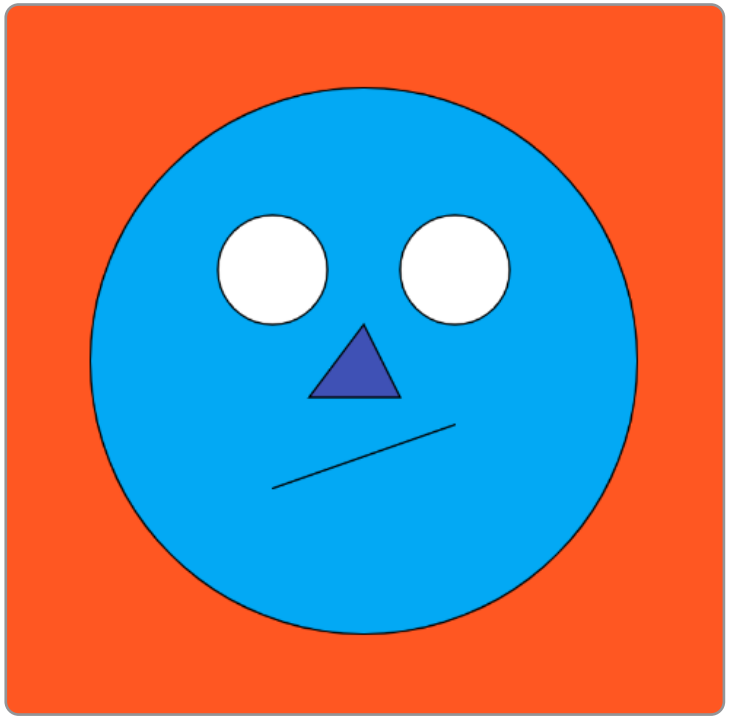
Repare que sempre que temos o **`fill()`** acima do comando de uma forma geométrica, todos os comandos que estão abaixo dele estarão na cor que adicionamos a esse parâmetro. Por exemplo, se escrevermos **`fill('white')`**, todas as formas abaixo dessa linha ficarão na cor branca.

Se quisermos mudar a cor do triângulo (o nariz) para azul, então, acima da linha do **`triangle()`**, escreveremos **`fill('blue');`**. Em seguida, podemos selecionar o tom que queremos, da mesma forma que fizemos para o fundo. Neste caso, escolheremos a cor **`#3F51B5`**, mas você pode escolher a que desejar e que combinar mais com seu projeto:

```
function draw() {
  background('#FF5722');
  fill('#03A9F4');
  circle(200, 200, 300);
  fill('white');
  circle(150, 150, 60);
  circle(250, 150, 60);
  line(150, 270, 250, 235);
  fill('#3F51B5');
  triangle(200, 180, 170, 220, 220, 220);

  ...
}
```

Até o momento, temos uma grande cabeça azul, com um nariz azul-
-escuro e olhos brancos. Se você ainda não estiver totalmente satisfeito,
explore mais essas possibilidades de colorir.



Definidas as cores, adicionaremos alguns detalhes a mais à nossa personagem. Por exemplo, para adicionarmos uma sobrancelha, podemos usar o **`if`**, identificando as coordenadas do plano cartesiano do P5.js e clicando na posição em que queremos que a primeira sobrancelha comece e, depois, no ponto em que queremos que a sobrancelha acabe.

Agora, abaixo da linha **`triangle()`**, escreveremos outra linha usando o comando **`line()`** com os quatro valores que obtivemos no terminal.

```
function draw() {
  ...
  triangle(200, 180, 170, 220, 220, 220);
  line(123,115,178,113);

  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

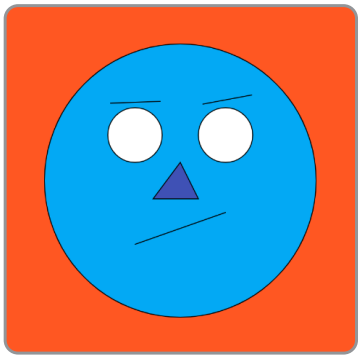
Professor, caso os estudantes encontrem dificuldades em manejar o *if*, revise novamente o uso desse comando com a turma, que foi amplamente estudado na aula anterior. Se necessário, retome também os conceitos do plano cartesiano.

Ao executarmos o projeto, teremos uma linha indo de um ponto a outro, formando uma sobrancelha de acordo com as coordenadas que definimos. Repetiremos esse processo para fazer a segunda sobrancelha, criando mais uma **`line();`**. Observe:

```
function draw() {
  ...
  triangle(200, 180, 170, 220, 220, 220);
  line(123,115,178,113);
  line(225,116,279,106);

  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

Assim, teremos o seguinte resultado:



Em seguida, adicionaremos as pupilas dos olhos, que estão apenas com um fundo branco.

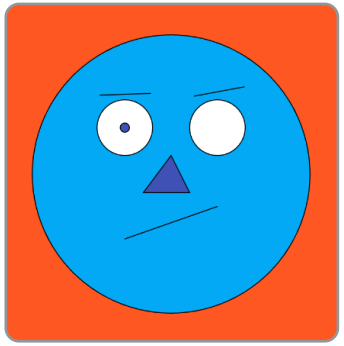
Ao criarmos os olhos da personagem, utilizamos valores aproximados a 150 para as coordenadas. Podemos usar as mesmas coordenadas e mudar apenas o diâmetro do nosso círculo e criar a pupila. Dessa forma, adicionaremos uma nova linha de código com o comando **`circle(150, 150, 10);`**. Observe:

```
function draw() {
  ...
  triangle(200, 180, 170, 220, 220, 220);
  line(123,115,178,113);
  line(225,116,279,106);
  circle(150, 150, 10);

  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

Professor, estimule os estudantes a brincarem com tamanhos, formas e cores neste momento, criando personagens personalizadas. A ideia do projeto é que eles aprendam a manejar diferentes formas e recursos do programa, portanto, explorar esses recursos para além de mera reprodução da videoaula é fundamental.

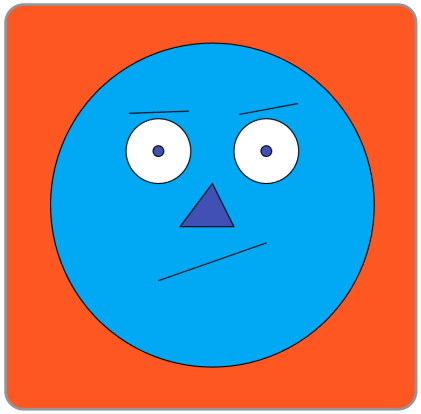
Ao executarmos o programa, teremos uma pupila no centro do olho esquerdo.



Repetindo o processo, usaremos as mesmas coordenadas que já temos para o círculo maior do olho direito. Assim, adicionaremos a linha de código com o comando **`circle(250, 150, 10);`**.

```
function draw() {
  ...
  line(225,116,279,106);
  circle(150, 150, 10);
  circle(250, 150, 10);
  if(mouseIsPressed){
    consele.log(mouseX,mouseY);
  }
}
```

Assim, nossa personagem está concluída!



Vocês podem criar outras formas, cores e elementos em suas próprias personagens, adicionando orelhas, barba, óculos... Explore as diferentes possibilidades!

Ao terminar, observem que teremos várias linhas de código, e é muito importante documentarmos isso.

Dessa forma, nas próximas vezes que visitarmos esse script, entenderemos o que fizemos.

Professor, reforce com os estudantes que documentar um código não o altera, sendo apenas uma organização e documentação do que fizemos. Essa é considerada uma boa prática fundamental na área, e deve se tornar um costume ao programar.

Para criar essa organização no código, faremos comentários que nos ajudem a lembrar o que fizemos ao programar. Para isso, usaremos um comando digitando duas barras (**//**). Podemos adicionar qualquer texto depois do **//**, e ele será lido como comentário pelo programa. Por exemplo, podemos escrever **// Teste de comentário** e executar o código. Observe a organização proposta:

```
function draw() {
  ...
  line(123,115,178,113); // Sobrancelha esquerda
  line(225,116,279,106); // Sobrancelha direita
  circle(150, 150, 10); // Pupila olho esquerdo
  circle(250, 150, 10); // Pupila olho direito
  ...
}
```

Com esse recurso, não teremos nenhum problema de execução ou mudança no comportamento do nosso código, porque qualquer informação depois das duas barras fica oculta, como um comentário. Então, podemos usar as duas barras para identificar cada uma das partes do rosto.

Ao final das alterações, não se esqueça de salvar seu projeto!

start by alura

CLIQUE AQUI PARA AVALIAR ESTE MATERIAL