



Noções de programação: exemplos com manipulação de imagens digitais

Prof. Guilherme Dutra Gonzaga Jaime

| | |
|------------|---|
| Descrição | Conceito de códigos. Combinação de códigos e instruções para computadores. Representação digital de imagens: noções básicas. |
| Propósito | Exercitar importantes habilidades do pensamento computacional, como abstração, automação, reconhecimento de padrões, análise e avaliação, com vistas à compreensão do significado da programação. |
| Preparação | Para este conteúdo será necessário usar alguns códigos-fonte específicos e o código RGB. |

Objetivos

| | |
|---|--|
| <p>Módulo 1</p> <p>Manipulação de dados</p> <p>Definir instruções para manipulação simples de dados.</p> | <p>Módulo 2</p> <p>Repetição <i>for</i></p> <p>Distinguir a estrutura de repetição <i>for</i>.</p> |
| <p>Módulo 3</p> <p>As expressões em código de computador</p> <p>Reconhecer expressões.</p> | <p>Módulo 4</p> <p>A estrutura condicional <i>if</i></p> <p>Distinguir a estrutura condicional <i>if</i>.</p> |

Introdução

Ao utilizarmos um computador, nós manipulamos diversos tipos de software: sistema operacional, aplicativos, editores de texto, entre outros. Para construir esse software, alguém, um programador ou desenvolvedor, escreveu uma sequência de instruções e códigos com a sequência de ações que devem ser desenvolvidas.

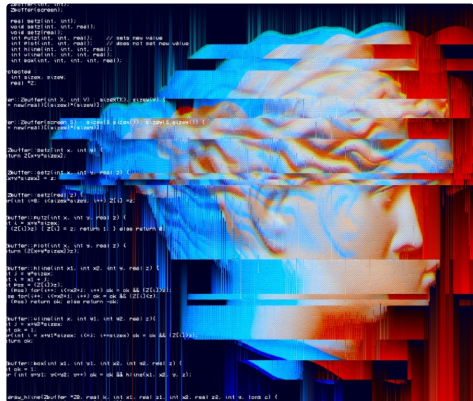
Essa atividade é a chamada programação!

O desenvolvedor utilizou alguma linguagem de programação e, respeitando a sua sintaxe e a sua semântica, codificou o algoritmo a ser utilizado com uma sequência de instruções e comandos dessa linguagem.

Parece meio complexo, não?

Bom, para que você possa compreender de forma mais fácil o significado da programação, vamos ver o conceito aplicado à manipulação de imagens digitais.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



1 - Manipulação de dados

Ao final deste módulo, você será capaz de definir instruções para manipulação simples de dados.

Manipulação singular de dados

Você já ouviu falar em manipulação singular de dados?

Neste módulo, vamos aprender a escrever códigos simples de computador para **manipulação singular de dados**.

Por *singular*, entenderemos a ausência de condições de realizar uma manipulação de múltiplos dados de forma eficiente.

Vamos com calma! Um pixel de cada vez...

Para ilustrar melhor nossa discussão, usaremos como exemplo imagens para aplicação de nossos códigos/instruções.

Como são instruções simples de manipulação de dados, você notará que, nos exemplos, manipularemos apenas um pixel de cada vez.

Instruções para manipulação simples de dados

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Para começar, veja a imagem abaixo, cujo nome é `circulo.bmp`, composta por uma grade de 10x10 pixels:

A imagem possui um fundo branco, com um círculo desenhado.

O **BMP** é um formato de arquivo usado para armazenar imagens digitais.

Não se preocupe, pois o tamanho da figura não é um erro. A imagem realmente está muito pequena.

Figura 1: Pequena imagem de 10 pixels de largura por 10 pixels de altura com fundo branco e círculo preto ao meio.

BMP

– Bitmap image file ou arquivo de imagem bitmap.

Nosso primeiro impulso é realizar uma ampliação ou zoom para vê-la melhor. Esse é o ímpeto que queríamos provocar em você quando a colocamos com esse tamanho.

Como escrever um código que solicita que o computador amplifique a imagem circulo.bmp?

Para isso, usaremos três instruções que fazem parte do conjunto de funções da linguagem de programação JavaScript, veja:

| Passo | Instrução | Significado em português |
|-------|---|--|
| 1 | <code>img = newSimpleImage("circulo.bmp");</code> | Carrega a imagem <code>circulo.bmp</code> na memória e a armazena na variável que, nest exemplo, chamamos de <code>im</code> |
| 2 | <code>img.setZoom(20);</code> | Estabelece ampliação de 20 vezes o tamanho original para a imagem armazenada na variável <code>img</code> . |
| 3 | <code>print(img);</code> | Apresenta a imagem na tela. |

Tabela 1: ampliando e apresentando na tela a imagem `circulo.bmp`.
Guilherme Dutra Gonzaga Jaime.

Agora que você aprendeu os passos que permitem ao computador amplificar a imagem `circulo.bmp`, experimentaremos isso para alcançarmos um nível de amplificação que nos deixe confortáveis.

Vamos praticar?

Prática 1

À esquerda, temos o **Código-Fonte**; à direita, a **Saída** resultante do processamento das instruções fornecidas. Logo abaixo, o botão **Executar**, que é a forma como ordenamos ao computador que execute as instruções fornecidas.

Código-Fonte

Saída

```
img = new SimpleImage("img/circulo.bmp");
img.setZoom(20);
print(img);
```

Rodar/Executar



Após clicar no botão **Executar** e observar o resultado em **Saída**, compare a seguir a solução:

Mostrar solução

Percebeu a diferença?

Note que aparece um círculo de 20x20 pixels, maior do que o apresentado na primeira figura.

Agora altere a instrução setZoom para realizar um zoom de 10 vezes em vez de 20. Clique em Executar e observe o resultado.

Mostrar solução

Note que a execução das instruções resulta em um círculo de 10x10 pixels, agora menor que o anterior.

Considerações

Embora bastante simples, as instruções apresentadas na tabela 1 e na prática que acabamos de realizar tornaram-nos capazes de instruir o computador a realizar ampliação de imagens digitais ao nosso comando.

Dica

Fique à vontade: pratique e experimente com qualquer valor que deseje para a instrução setZoom. Não tenha medo de errar!

Manipulando cada *bit*

Agora vamos partir para um segundo cenário um pouco mais interessante.

Queremos estender o código usado na Prática 1 para sermos capazes de manipular individualmente cada pixel.

Para isso, tudo que precisamos fazer é ajustar os passos da tabela 1 e adicionar duas novas instruções logo antes da instrução *print*, veja a seguir. Essas instruções também fazem parte das funções padronizadas pela linguagem JavaScript.

| Passo | Instrução | Significado em português |
|-------|--|---|
| 1 | <code>img = new SimpleImage("circulo.bmp");</code> | Carrega a imagem <code>circulo.bmp</code> na memória e a armazena na variável chamada <code>img</code> . |
| 2 | <code>img.setZoom(20);</code> | 20 vezes o tamanho original para a imagem armazenada na variável <code>img</code> . Um zoom menor do que 0 (zero) equivale a um afastamento. Por exemplo, um zoom de 0.5 equivale a um afastamento de 2x. |
| 3 | <code>pixel = img.getPixel(4,4)</code> | Obtém a referência ao pixel (4,4) da imagem armazenada na variável <code>img</code> e atribui essa referência à variável <code>pixel</code> . |
| 4 | <code>pixel.setRed(255)</code> | Instrui o computador a ajustar para 255 o nível de vermelho para o pixel em questão. |
| 5 | <code>print(image);</code> | Apresenta a imagem na tela. |

Tabela 2: modificando um dos pixels de `circulo.bmp` para que ele fique vermelho.
Guilherme Dutra Gonzaga Jaime.

Onde aparece a primeira instrução nova da tabela 2?

Resposta

A primeira instrução nova que vemos nela é a do Passo 3, que instrui o computador a obter a referência ao pixel (4,4) da imagem armazenada na variável `img` e, então, a atribuir essa referência à variável `pixel`.

A partir de agora, qualquer operação que fizermos com a variável `img` será efetuada sobre o pixel (4,4).

Em seguida, no Passo 4, a instrução `pixel.setRed(255)` ordena que o computador ajuste a saturação de vermelho do pixel para o nível 255, que é o maior valor possível.

Prática 2

A seguir, assista a uma breve contextualização da Prática 1, seguida do nosso segundo exemplo prático.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Agora, vamos realizar alguns experimentos simples com o código da tabela 2, manipulando o código-fonte a seguir para observarmos o que acontece:

Código-Fonte

```
img = new SimpleImage( "img/circulo.bmp" );
img.setZoom( 20 );
pixel = img.getPixel( 4, 4 );
pixel.setRed( 255 );
print(img);
```

Saída

Rodar/Executar

Após clicar no botão **Executar** e observar o resultado em **Saída**, compare a solução:

Mostrar solução

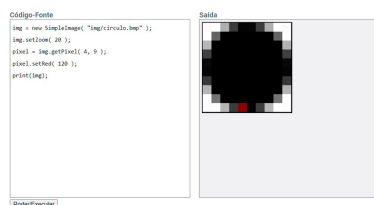
Note que o pixel (4,4), que fica mais ou menos no meio do círculo, ficou vermelho.

Agora, observe as soluções em alterações na mudança de alguns parâmetros:



Alternando o parâmetro da função getPixel para (4, 9)

Observe que, como fornecemos a coordenada de outro pixel (linha 9 em vez de linha 4), o pixel vermelho ficou mais abaixo.



Colocando diferentes valores para o parâmetro da função setRed

Observe que, quanto menor o valor da saturação vermelha do pixel, mais escuro será o vermelho apresentado no pixel em questão.

Considerações

Fique à vontade para experimentar com a Prática 2. Você pode escolher qualquer valor entre 0 e 9 para as coordenadas do pixel (lembre-se: é uma imagem de 10 pixels por 10 pixels) e qualquer valor entre 0 e 255 para o nível de saturação vermelho do pixel (conforme o [esquema de cores RGB](#)).

Assim como a função `setRed()` ajusta o nível de vermelho para o pixel, temos duas outras funções análogas para manipular os níveis de azul e de verde de um pixel. A tabela 3 apresenta as três funções possíveis para essa manipulação, veja:

Esquema de cores RGB

Esquema de codificação de cores, chamado RGB (Red, Blue, Green/Vermelho, Azul, Verde). Trata-se de um esquema simples que permite que possamos instruir (programar) o computador para representar qualquer uma das 16.7 milhões de cores formadas com a combinação dessas três.

| Passo | Instrução | Significado em português |
|-------|-------------------------------------|---|
| 1 | <code>pixel.setRed(número)</code> | Ajusta o nível de vermelho do pixel conforme o número informado. O número deve ser entre 0 e 255. |
| 2 | <code>pixel.setGreen(número)</code> | Ajusta o nível de verde do pixel conforme o número informado. O número deve ser entre 0 e 255. |
| 3 | <code>pixel.setBlue(número)</code> | Ajusta o nível de azul do pixel conforme o número informado. O número deve ser entre 0 e 255. |

Tabela 3: Três funções disponíveis para manipulações de cores de um pixel.
Guilherme Dutra Gonzaga Jaime.

Evoluindo a prática

Prática 3

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Vamos realizar alguns experimentos simples com o código da tabela 3, manipulando o código-fonte a seguir para observar o que acontece:

Código-Fonte

```
img = new SimpleImage( "img/circulo.bmp" );
img.setZoom( 20 );
pixel = img.getPixel( 4, 0 );
pixel.setRed( 255 );
print(img);
```

Saída

Rodar/Executar

Durante essa prática, observe com atenção a proposição em português feita no enunciado, em que o efeito desejado é descrito.

Depois, tente refletir:

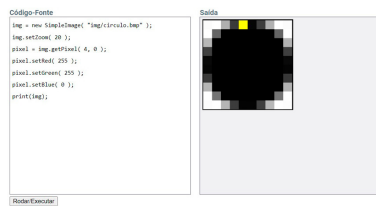
Qual é o código de computador que você deve escrever, ou seja, as instruções e os valores digitados no código-fonte para alcançar o efeito proposto?

Na prática, você aprenderá a traduzir:



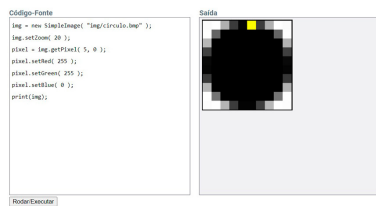
O código fornecido ajusta o pixel (4,0) para vermelho. Ajuste o código para que o pixel (4,0) fique verde. Veja na solução:

Agora, observe as soluções em alterações na mudança de alguns parâmetros:



Manipulação do pixel (4,0) para que fique amarelo

Aqui, o pixel (4,0) estava originalmente preto, ou seja, código RGB (0,0,0). Como sabemos que o amarelo é a combinação de verde e vermelho, a solução é bem simples: basta adicionar o nível máximo (255) dessas cores constituintes.



Manipulação do pixel (5,0) para que fique amarelo

A solução aqui é muito simples: basta alterar o código da solução anterior para que a instrução `getPixel(4,0)` passe a ser `getPixel(5,0)`.

Prática 4

Suponha que, em vez de manipular apenas um pixel por vez, conforme fizemos neste módulo, desejássemos manipular todos os pixels de uma imagem, digamos de 10x10 pixels (exemplo: `circulo.bmp`). Se usássemos o código que aprendemos a escrever, teríamos de escrever 100 vezes a instrução `pixel = img.getPixel()`, em que, a cada vez, passaríamos os valores x,y para cada um dos pixels da imagem.

Isso traz vários problemas.

1. O primeiro deles é que o código ficaria difícil de ler/compreender.

2. O segundo é que, se for uma imagem maior, de 1980x1024 pixels, teríamos cerca de 2 milhões de pixels.

Portanto, a instrução `pixel = img.getPixel()` teria de ser escrita cerca de 2 milhões de vezes no código fonte, a cada vez fazendo referência para um pixel diferente.

Isso seria inviável, não é mesmo?

Mostrar solução

No mundo real, quase sempre desejamos realizar operações com uma quantidade muito grande de dados. Os engenheiros e cientistas de computação pensaram nesta questão décadas atrás.

Felizmente, há técnicas de programação/codificação que nos permitem manipular uma quantidade arbitrária de dados, sem ter de escrever muitos códigos de computador.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Considere a imagem a seguir, chamada "circulo.bmp":

Qual é o código de computador necessário para manipular o pixel do canto superior esquerdo para que fique vermelho?

```
img = new SimpleImage( "img/circulo.bmp" );  
img.setZoom( 20 );
```

A

```
pixel = img.getPixel( 0, 0 );  
pixel.setRed( 255 );  
pixel.setGreen( 0 );  
pixel.setBlue( 0 );
```

```
img = new SimpleImage( "img/circulo.bmp" );  
img.setZoom( 20 );
```

B

```
pixel = img.getPixel( 0, 0 );  
pixel.setRed( 0 );  
pixel.setGreen( 255 );  
pixel.setBlue( 0 );
```

```
img = new SimpleImage( "img/circulo.bmp" );  
img.setZoom( 20 );
```

C

```
pixel = img.getPixel( 0, 0 );  
pixel.setRed( 0 );  
pixel.setGreen( 0 );  
pixel.setBlue( 255 );
```

```
img = new SimpleImage( "img/circulo.bmp" );  
img.setZoom( 20 );
```

- D
- ```
pixel = img.getPixel(1, 1);
pixel.setRed(255);
pixel.setGreen(0);
pixel.setBlue(0);
```
- img = new SimpleImage( "img/circulo.bmp" );  
img.setZoom( 20 );
- E
- ```
pixel = img.getPixel( 1, 1 );
pixel.setRed( 0 );
pixel.setGreen( 255 );
pixel.setBlue( 255 );
```

Parabéns! A alternativa A está correta.

Conforme convenção de coordenadas de pixels, o pixel do canto superior esquerdo será sempre referenciado pela coordenada (0,0). Além disso, segundo o esquema RGB, a cor vermelha pura é representada pelo código RGB (255,0,0).

Veja a solução:

Questão 2

Considerando os experimentos realizados na Prática 1, qual seria o código para que o resultado do zoom fosse um afastamento de 2 x em vez de uma aproximação de 2x?

- A
- ```
img = new SimpleImage("img/circulo.bmp");
img.setZoom(2);
print(img);
```
- B
- ```
img = new SimpleImage( "img/circulo.bmp" );
img.setZoom( -2 );
print(img);
```
- C
- ```
img = new SimpleImage("img/circulo.bmp");
img.setZoom(-0.5);
print(img);
```
- D
- ```
img = new SimpleImage( "img/circulo.bmp" );
img.setZoom( 0.5 );
print(img);
```
- E
- ```
img = new SimpleImage("img/circulo.bmp");
img.setZoom(-20);
print(img);
```

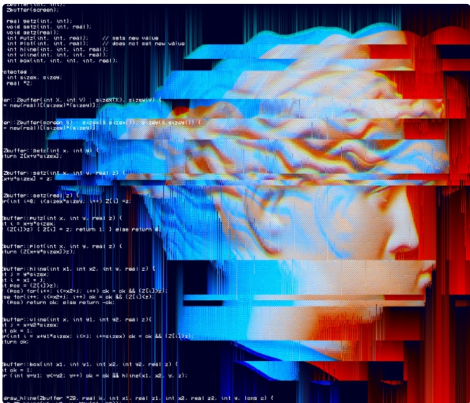
Parabéns! A alternativa D está correta.

Conforme ilustrado na tabela 2, um afastamento de 2x equivale a um zoom de 0.5.

| Passo | Instrução | Significado em português |
|-------|-----------|--------------------------|
|-------|-----------|--------------------------|

|   |                                                    |                                                                                                                                                                                                           |
|---|----------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1 | <code>img = new SimpleImage("circulo.bmp");</code> | Carrega a imagem <code>circulo.bmp</code> na memória e a armazena na variável chamada <code>img</code> .                                                                                                  |
| 2 | <code>img.setZoom(20);</code>                      | 20 vezes o tamanho original para a imagem armazenada na variável <code>img</code> . Um zoom menor do que 0 (zero) equivale a um afastamento. Por exemplo, um zoom de 0.5 equivale a um afastamento de 2x. |
| 3 | <code>pixel = img.getPixel(4,4)</code>             | Obtém a referência ao pixel (4,4) da imagem armazenada na variável <code>img</code> e atribui essa referência à variável <code>pixel</code> .                                                             |
| 4 | <code>pixel.setRed(255)</code>                     | Instrui o computador a ajustar para 255 o nível de vermelho para o pixel em questão.                                                                                                                      |
| 5 | <code>print(image);</code>                         | Apresenta a imagem na tela.                                                                                                                                                                               |

Tabela 2: modificando um dos pixels de `circulo.bmp` para que ele fique vermelho.  
Guilherme Dutra Gonzaga Jaime.



## 2 - Repetição *for*

Ao final deste módulo, você será capaz de distinguir a estrutura de repetição *for*.

## Estruturas de repetição

**Estruturas de repetição** são extremamente importantes, pois representam um grande aumento no poder de quem escreve códigos de computador em comparação com códigos, que são capazes de manipular dados singulares. Neste módulo, veremos os princípios dessas estruturas.

## Estrutura de repetição *for*

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Observe a imagem de um pássaro:

Figura 2: Imagem com 584 pixels de largura e 500 pixels de altura representando um pássaro.

Se multiplicarmos o número de pixels de largura pelo número de pixels de altura, teremos o total de pixel da imagem.

Neste caso:

**$584 \times 500 = 292.000$  pixels ou quase 0.3 megapixels.**

Essa nem é uma imagem muito grande, dado que os telefones celulares atuais são capazes de tirar fotografias de mais de 10 milhões de pixels (10 megapixels).

### Atenção!

Mesmo assim, se o autor de um código de computador, sem usar estruturas de repetição, tentar escrever um código de computador para manipular cada um dos pixels da imagem da figura 2, precisará repetir 292 mil vezes as instruções que apontam para determinado pixel (exemplo: `pixel = img.getPixel( x,y )`), cada uma delas seguidas com as instruções de manipulação desejada (exemplo: `pixel.setRed(255)`).

Claramente, isso não é uma maneira prática de realizar uma operação com grande quantidade de dados. As desvantagens vão desde um código de computador difícil de ler, por ser muito extenso, até a dificuldade de manutenção, atualização ou correção do código, caso haja necessidade futura.

Então, o que queremos é uma construção na qual possamos escrever algumas poucas linhas de código que capturam certas mudanças desejadas. Depois, deixamos o computador executar essas linhas de código repetidamente, uma vez para cada dado que desejamos manipular, ou seja, em nosso exemplo, uma vez para cada pixel da imagem.

Existem diferentes maneiras de realizar isso em códigos de computador, mas, por motivos de simplicidade, vamos estudar a estrutura de **repetição *for***, também chamada de *loop for*.



**Repetição *for***



**Loop *for***

Analise a sintaxe da estrutura de repetição *for* apresentada abaixo:

Figura 3: Sintaxe da estrutura de repetição *for* destacada em negrito.

A linha de código *for* ( pixel: img ), traduzida de código de computador para português, significa: para cada pixel da imagem armazenada na variável *img*, repita uma a uma todas as instruções cercadas por abre chaves “{” e fecha chaves ”}” logo a seguir.

No exemplo da figura 3, temos três instruções destacadas em cinza dentro da estrutura *for*. Nesse caso, estas serão as instruções repetidas para cada um dos 292 mil pixels da imagem “pássaro.jpg”.

**Toda a sintaxe apresentada é requerida para que o computador entenda que se trata de uma estrutura *for* e quais são as instruções a serem repetidas.**

Na prática, o que o computador fará para o código da figura 3 é fixar-se no primeiro pixel da imagem, (0,0), no canto superior esquerdo, e executar os três comandos que ajustam o nível das cores constituintes vermelho, verde e azul todas para 0. Este pixel, então, ficará preto.

1

Observe a seta azul que liga o “}”, no final do *loop for*, no início da estrutura de repetição. Quando o computador percebe que está na linha de instrução que fecha a estrutura de repetição, acontece isso, conforme indicado pela seta azul: o computador volta novamente para o início do *loop*.

2

Em seguida, o computador se fixará no segundo pixel (1,0) da imagem e executará os três comandos em cinza, deixando o segundo pixel preto. O mesmo ocorrerá para o terceiro, o quarto, o quinto e para todos os 292 mil pixels da imagem do pássaro, até que toda a imagem esteja completamente preta.

3

Você perceberá que as três instruções em cinza que formam o corpo do *loop for* estão descoladas para a direita. Isso não é obrigatório para que o computador compreenda as instruções.

4

É, no entanto, uma convenção muito comum mostrar que as linhas de dentro da estrutura de repetição são especiais em relação às demais, porque são executadas várias vezes, até que a condição estabelecida no início da estrutura *for* seja satisfeita.

Neste exemplo, a condição para finalizar é que as três instruções sejam executadas pelo computador para todos os pixels da imagem.

A técnica de recuar a linha de código para deixar claro à pessoa que o estiver lendo que estas são instruções internas à estrutura de repetição é amplamente chamada de **indentação**.

## Indentação

Neologismo derivado da palavra inglesa *indentation*.

# Vamos praticar

## Prática 1

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Vamos experimentar a estrutura *for*. Observe o código-fonte abaixo:

### Código-Fonte

```
img = new SimpleImage("img/passaro.jpg");
for(pixel: img){
 pixel.setRed(0);
 pixel.setGreen(0);
 pixel.setBlue(0);
}
print(img);
```

### Saída

Rodar/Executar

Lembre-se de que, ao clicar em Executar, o computador realizará os seguintes passos, conforme instruções:

#### Instrução 1

Carregar a imagem "passaro.jpg" e armazená-la na variável `img`.

#### Instrução 2

Para cada um dos 292 mil pixels da imagem "passaro.jpg", executar as instruções indicadas:

- Ajustar intensidade de vermelho do pixel para 0 (zero);
- Ajustar a intensidade de verde do pixel para 0 (zero);
- Ajustar a intensidade de azul do pixel para 0 (zero);
- Imprimir/apresentar a imagem na tela.

Note que o computador as executará para cada um dos 292 mil pixels da imagem. Então, temos um total de 876 mil operações a serem realizadas pelo computador. Ao clicar em **Executar**, tente observar quanto tempo o computador leva para ajustar/executar as 876 mil instruções necessárias para colorir de preto cada um dos 292 mil pixels da imagem "passaro.jpg" e, depois, apresentar a imagem na tela.

A imagem preta deve aparecer na tela muito rapidamente:

Ao executar a Prática 1, você obteve um retângulo preto, o que não é muito útil. Entretanto, não deixe de refletir sobre o quão rapidamente o computador foi capaz de seguir suas instruções para manipular os quase 300 mil pixels da imagem.



## Prática 2

Vamos seguir assistindo a um vídeo com uma segunda prática.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## Prática 3

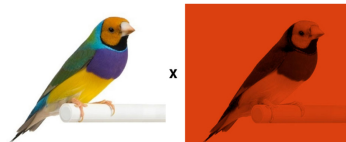
Agora, vamos obter o canal alfa vermelho da imagem do pássaro apresentado na figura 2 com base neste código-fonte:

Execute o código-fonte e observe o canal alfa em vermelho da imagem apresentada:

O que você notou?

Assim como na Prática 1, observe o quão rapidamente o computador foi capaz de remover os componentes azul e verde da imagem em questão. Lembre-se: as instruções de dentro da estrutura *for* foram repetidas 876 mil vezes.

Compare com a imagem original da figura 2, replicada logo a seguir (por comodidade), com a imagem resultante do item "a".



Observe, também, que o lindo colar azul do pássaro praticamente desapareceu na imagem canal alfa vermelho resultante. Isso significa que o colar é constituído principalmente dos componentes verde e azul.

Você compreendeu como o canal alfa em vermelho foi obtido?

Para compreender melhor o código que foi executado, observe na caixa de código-fonte que a instrução `pixel.setRed()` está comentada, ou seja, marcada com `//` no início da linha:

### Código-Fonte

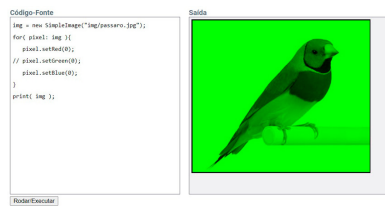
```
img = new SimpleImage("img/passaro.jpg");
for(pixel: img){
 // pixel.setRed(0);
 pixel.setGreen(0);
 pixel.setBlue(0);
}
print(img);
```

Rodar/Executar

### Saída

Comentários de códigos são interessantes por duas razões: A primeira, é permitir que o autor do código-fonte documente seu código com frases que esclarecem o que ele estava pensando quando escreveu aquele trecho; A segunda, é fazer com que o computador ignore uma ou mais instruções.

Obtenha os canais verde e azul respectivamente.



Verde



Azul

## Práctica 4

Afinal para que serve a estrutura de repetição *for*?

### Atenção!

A estrutura de repetição *for* é um recurso muito poderoso, que nos permite escrever algumas poucas linhas de código capazes de ordenar ao computador que processe/manipule uma enorme quantidade de dados.

Neste módulo, usamos exemplos de imagens para demonstrar o enorme potencial das estruturas de repetição. Este foi apenas um exemplo.

Esteja certo de que essa estrutura é amplamente usada em computação e faz parte do dia a dia de todos que, de alguma forma, escrevem códigos para computador. Por fim, devemos mencionar um detalhe da linguagem usada aqui: JavaScript. Essa linguagem não possui uma estrutura de repetição *for* tão compacta e simples como a que estudamos.



Usamos uma versão com sintaxe propositalmente simplificada, pois nosso objetivo era compreender os princípios que motivam o uso de estruturas de repetição. Entendemos que o uso de uma estrutura simplificada ajudaria você a compreender melhor o que se passa.

Não queríamos gastar muito tempo fazendo-o compreender detalhes de sintaxe. Este comentário é importante, pois, se você tentar aproveitar os códigos de computador que escrevemos aqui, provavelmente terá problemas devido ao nosso uso simplificado da estrutura de repetição *for*.

Há outras linguagens de programação que possuem estrutura de repetição bem similar àquela que usamos aqui, mas não JavaScript.

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?



### Questão 1

Dos trechos de código, a seguir, qual gerará o canal azul de uma imagem (toda ela apenas em tons de azul)?

A

```
for(pixel: img){
 pixel.setGreen(0);
 pixel.setBlue(0);
}
print(img);
```

B

```
for(pixel: img){
 pixel.setRed(0);
 pixel.setBlue(0);
}
print(img);
```

C

```
for(pixel: img){
 pixel.setGreen(0);
 pixel.setBlue(0);
 pixel.setRed(0);
}
print(img);
```

D

```
for(pixel: img){
 pixel.setGreen(0);
 pixel.setRed(0);
}
print(img);
```

E

```
for(pixel: img){
 pixel.setGreen(255);
 pixel.setRed(255);
}
print(img);
```

**Parabéns! A alternativa D está correta.**

Conforme vimos na Prática 2, o canal azul de uma imagem é uma modificação à imagem em que apenas as tonalidades de vermelho são mantidas intactas, e os demais componentes (verde e azul) são desconsiderados. Na alternativa D vemos duas: `pixel.setGreen(0);` e `pixel.setRed(0);`, que instruem o computador a retirar toda a intensidade dos componentes verde e vermelho. Essa alternativa ajusta a intensidade de vermelho e de verde de cada pixel para 0, mas deixa a intensidade de azul intacta (não há instrução para ajustar a intensidade de azul). Então, esse componente ficou intacto em todos os pixels.

### Questão 2

Suponha a imagem a seguir, de 1.400 pixels de largura por 932 de altura:

Lembre-se de que o total de pixels de uma imagem é obtido pela multiplicação do número de pixels de altura pelo número de pixels de largura. Suponha que desejemos escrever um código de computador que retire os componentes de azul e de vermelho para cada pixel da imagem.

Para isso, as instruções seriam:

```
img.getPixel(0,0);
pixel.setBlue(0);
pixel.setRed(0);
```

Se escrevêssemos um código para cada um dos pixels da imagem, quantas linhas totais teríamos de escrever?

- A 3.914.400
- B 1.304.800
- C 2.609.600
- D 652.400
- E 2.332

Parabéns! A alternativa A está correta.

O total de pixels da imagem é 1.400x932, ou seja, 1.304.800 pixels. Como são três instruções por pixel, o programador teria de escrever  $3 \times 1.304.800 = 3.914.400$  (3,9 milhões) de linhas de código. Obviamente, isso seria inviável. Então, é fundamental usar a estrutura de repetição for, que nos permite escrever a mesma solução com cerca de cinco linhas de código.



### 3 - As expressões em código de computador

Ao final deste módulo, você será capaz de reconhecer expressões.

## Expressões

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Neste módulo, estudaremos o uso de **expressões** em código de computador. Expressões são construções bastante importantes, pois, em vez de usar apenas valores simples em instruções, podemos combinar diferentes valores para calcular o parâmetro mais personalizado.

Isso nos permite escrever códigos de computador que alcançam resultados mais realistas. Quando apresentarmos exemplos práticos de manipulação de imagens, você

entenderá melhor o assunto.

Usaremos a linguagem de programação JavaScript, com algumas simplificações para facilitar a compreensão. Se analisarmos o código/instrução a seguir, veremos um tipo bem simples de passagem de valor (51):



Dizemos que 51 é o valor passado como parâmetro para a **função print**.

As linguagens de programação compreendem códigos que envolvem expressões, veja:

Exemplo

`print(40+11);`

Então, o termo 40+11, passado como parâmetro para a função print, é o que chamamos de expressão.

Basicamente, em vez de um número fixo conhecido antecipadamente, podemos usar um pouco de expressão aritmética. Assim, temos uma forma mais rica de fornecer parâmetros para funções.



Isso funciona da seguinte maneira: quando o computador executar a linha de código que contém a expressão, o primeiro passo será avaliá-la. Assim, ele lerá e resolverá a expressão para chegar ao valor resultante. Neste exemplo, 40+11 é apenas uma soma. O computador calcula que 40+11 é igual a 51. Uma vez que a expressão foi avaliada e o resultado foi 51, então, o computador continua a execução do código, usando o resultado da expressão como parâmetro para a função. Com efeito, em nosso exemplo o computador simplesmente imprimirá o valor 51 na tela.

De fato, podemos usar expressões em qualquer local do código de computador onde valores numéricos são admitidos. Portanto, é possível sempre embutir expressões aritméticas para que seja computado o valor que desejamos usar de fato. Isso nos possibilita resolver problemas mais realistas.

Antes de continuar, vamos conhecer três funções de manipulações de pixels que fazem parte da linguagem de programação JavaScript:

| Passo | Instrução                     | Significado em português                                        |
|-------|-------------------------------|-----------------------------------------------------------------|
| 1     | <code>pixel.getRed()</code>   | Obtém/Lê o valor atual para o componente RGB vermelho do pixel. |
| 2     | <code>pixel.getGreen()</code> | Obtém/Lê o valor atual para o componente RGB verde do pixel.    |
| 3     | <code>pixel.getBlue()</code>  | Obtém/Lê o valor atual para o componente RGB azul do pixel.     |

Tabela 4: Dado um pixel da imagem, como saber qual é o valor atual para os componentes RGB do pixel?  
Guilherme Dutra Gonzaga Jaime.

Suponha um pixel em amarelo (exemplo: Código RGB 255,255,0). Neste caso, a função `pixel.getRed()` retornará o valor 255, pois este é o valor atual do componente vermelho (R) do pixel.

De forma análoga, a função `pixel.getBlue()` retornaria 0 (zero), pois este é o valor atual do componente azul (B) do pixel em questão.

## Isso ocorre porque o amarelo puro não possui azul em sua composição.

Agora, vamos supor que desejamos duplicar o valor atual do componente vermelho de um pixel. Se esse valor for 50, ajustaremos para 100. Se for 105, ajustaremos para 210. Em resumo, desejamos realizar um ajuste relativo no valor. Então, seja qual for o valor atual, nós o duplicaremos.

Já conhecemos a função `pixel.getRed()` e a possibilidade do uso de expressões. Logo, a tarefa ficou mais fácil. Uma primeira solução seria usar este código:

JavaScript



```
1 ultimo = pixel.getRed();
2 pixel.setRed(ultimo*2);
```

Agora veja as informações sobre a 1ª e a 2ª linhas:

### 1ª linha

Chama `pixel.getRed()` e armazena o valor retornado na **variável ultimo**.

### 2ª linha

Usa a função `pixel.setRed()` para informar que o novo valor deste pixel será o dobro do último valor..

### Variável ultimo

Optamos por usar a variável com nome `ultimo` para representar a ideia de que o que está armazenado ali é o último valor lido do componente vermelho do pixel.

Logo, quando começar a executar a segunda linha:

1

O primeiro passo do computador será avaliar qual é o resultado da expressão **ultimo\*2**. Se imaginarmos que o último valor para o pixel é 60, a expressão multiplicará esse valor por 2, o que resultará em 120.

2

Em seguida, o computador executará a função `pixel.setRed()` com o valor 120 como parâmetro, o que dobrará a intensidade de vermelho do pixel em questão.

3

Com efeito, o resultado das duas linhas de código é, de fato, duplicar a intensidade de vermelho do pixel atual.

Agora que entendemos os princípios do uso de expressões em códigos de computadores, vamos analisar o mesmo exemplo de duplicação do valor atual da intensidade de vermelho de um pixel, porém com uma solução muito mais comum no mundo real.

Veja a linha de código a seguir:

JavaScript



```
1 pixel.setRed(pixel.getRed()*2);
```

Na prática, desejamos mostrar que a solução em questão pode ser condensada em apenas uma linha de código.

Note que a variável `ultimo` não tinha um objetivo muito importante no código. Ela apenas armazenava temporariamente o último valor do componente vermelho do pixel para que este fosse usado na instrução seguinte.

Vamos supor que o pixel em questão esteja com os valores RGB (50,20,30). Então, o componente vermelho possui o valor 50. O computador fará o seguinte:

| 1                                                                                             | 2                                                                                                                                     | 3                                                                                                                                                                                             |
|-----------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Executar a instrução <code>pixel.getRed()</code> para obter o valor atual do pixel, que é 50. | Avaliar a expressão <code>pixel.getRed()*2</code> e resolvê-la, multiplicando o valor atual do pixel por 2 e obtendo o resultado 100. | Executar a instrução <code>pixel.setRed()</code> usando como parâmetro o resultado da operação aritmética do Passo 2 ou seja, o valor dobrado para o componente vermelho do pixel, que é 100. |

Com isso, o novo valor RGB para o pixel em questão será: RGB (100,20,30). Se fizermos o mesmo para todos os pixels de uma imagem, o usuário perceberá essa alteração como uma imagem com os tons de vermelho mais destacados.

Se, no mesmo exemplo, desejássemos reduzir pela metade a intensidade de vermelho do pixel, a linha de código ficaria assim:

JavaScript

```
1 pixel.setRed(pixel.getRed()/2);
```

Note que, em vez de usarmos o asterisco, que denota multiplicação, utilizamos o `/`, que denota divisão.

Outra forma de escrever a mesma solução seria multiplicar o pixel atual por 0.5, o que é o mesmo que dividir por 2. Então, nesse caso, teríamos:

JavaScript

```
1 pixel.setRed(pixel.getRed()*0.5);
```

Para entender melhor o assunto, vamos fazer alguns exercícios.

## Prática 1

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Vamos ajustar a tonalidade de cores de uma imagem por meio de instruções simples. Começemos por esta:



Figura 4: Imagem de flores em amarelo com 587x330 pixels, totalizando quase 194 mil pixels.

Suponha que desejemos ajustar essa imagem para obtermos tonalidades mais para o laranja.

**Para alcançar o objetivo desta prática, basta reduzir um pouco a intensidade do componente verde (G) de cada um dos pixels da imagem.**

JavaScript

```
1 pixel.setGreen(pixel.getGreen()*0.7);
```

O que essa instrução faz é ajustar a intensidade de verde do pixel em questão para 70% do valor atual. Esse é o resultado aritmético que você obtém quando multiplica um valor por 0.7.

Note que o código-fonte fornecido não contém a instrução citada. Apenas disponibilizamos o "esqueleto" do código contendo a solução:

#### Código-Fonte

```
img = new SimpleImage("img/flores.jpg");
for(pixel: img){
 // Insira suas linhas de código abaixo
}
print(img);
```

Rodar/Executar

#### Saída

Se você clicar em **Executar**, o computador apenas apresentará a imagem original com as flores em amarelo. Isso ocorre porque não há instruções de ajuste de cores no código-fonte.

Você mesmo deve inserir a instrução que realiza a redução da tonalidade de verde do pixel. Insira a instrução dentro da estrutura de repetição for logo abaixo do comentário que indica aposição correta. Depois, clique em **Executar** para que o computador execute seu código. Observe o resultado.

Se tiver dúvidas, clique em **Solução**, mas não deixe de tentar implementar sua solução no código-fonte fornecido. Basta inserir uma linha de código!

Mostrar solução

Aqui, vemos lindas flores de cor laranja! Experimente reduzir ainda mais o componente de verde da figura. Observe que, quanto menor o componente de verde, mais as cores das flores se aproximarão do vermelho.

## Considerações

Refleta sobre a quantidade de operações realizada em um piscar de olhos pelo computador, dado o código-fonte que geramos.

Para cada um dos 194 mil pixels da imagem da figura 4, o computador fez as seguintes operações:

- Obteve a intensidade de verde do pixel;
- Multiplicou a intensidade de verde do pixel por 0.7;
- Ajustou a intensidade de verde do pixel para o resultado dos dois passos anteriores.

Então, em português, poderíamos descrever o objetivo deste exercício simplesmente como: tornar a imagem um pouco mais laranja.

Refleta sobre como esse objetivo foi traduzido do português para o código de computador que usamos ao realizarmos a operação. A capacidade de fazer essa tradução e escrever uma solução que computadores são capazes de executar rapidamente é uma habilidade chave para o chamado **pensamento computacional**.



Voltando à imagem da figura 4, suponha, agora, que desejamos convertê-la em uma imagem de escala de cinza.

Recorde, na tabela 5, que uma imagem em escala de cinza possui, para cada pixel, exatamente o mesmo valor para os componentes RGB (vermelho, verde e azul):

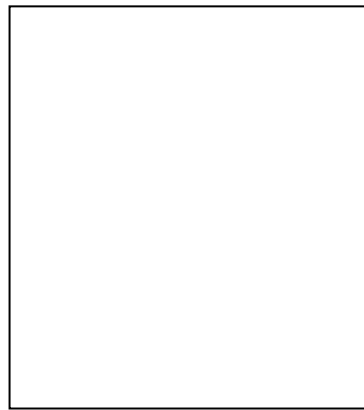
| Esquema RGB     | R - Vermelho                                   | G - Verde | B - Azul |
|-----------------|------------------------------------------------|-----------|----------|
| Branco          | 255                                            | 255       | 255      |
| Azul            | 0                                              | 0         | 255      |
| Vermelho        | 255                                            | 0         | 0        |
| Verde           | 0                                              | 255       | 0        |
| Amarelo         | 255                                            | 255       | 0        |
| Magenta         | 255                                            | 0         | 255      |
| Ciano           | 0                                              | 255       | 255      |
| Preto           | 0                                              | 0         | 0        |
| Escala de cinza | (1,1,1) cinza muito escuro (quase preto)       |           |          |
|                 | (2,2,2)                                        |           |          |
|                 | (3,3,3)                                        |           |          |
|                 | (4,4,4)                                        |           |          |
|                 | (5,5,5)                                        |           |          |
|                 | (6,6,6)                                        |           |          |
|                 | .                                              |           |          |
|                 | .                                              |           |          |
|                 | .                                              |           |          |
|                 | (252,252,252)                                  |           |          |
|                 | (253,253,253)                                  |           |          |
|                 | (254,254,254) cinza muito claro (quase branco) |           |          |

Tabela 5: Escala de cinza do esquema de cores RGB.  
Guilherme Dutra Gonzaga Jaime.

Use os controles deslizantes do visualizador RGB para comprovar este conceito:

Sem cor →→→ Escuro →→→ Mais claro →→→ Saturação total

Vermelho (R - Red):   
Verde (G - Green):   
Azul (B - Blue):  R:0 G:0 B:0  
☐ Mostrar Hexadecimal #000000



## Transformar a imagem para a escala de cinza

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Para alcançar o objetivo de converter a imagem de flores amarelas em uma imagem em escala de cinza, vamos calcular, para cada pixel, o valor médio dos componentes RGB. Depois, ajustaremos cada um dos três componentes para esse valor médio. Veja o trecho de código que calcula a média dos três componentes (R, G e B) para um dado pixel, depois, atribui esse valor médio aos três componentes de cor do pixel:

JavaScript



```
1 soma=(pixel.getGreen()+pixel.getGreen()+pixel.getGreen());
2 media= soma/3;
3 pixel.setRed(media);
4 pixel.setGreen(media);
5 pixel.setBlue(media);
```

Copie e cole estas linhas de código no código-fonte de experimento de programação disponível no início desta prática:

### Código-Fonte

```
img = new SimpleImage("img/flores.jpg");
for(pixel: img){
 // Insira suas linhas de código abaixo
}
print(img);
```

Rodar/Executar

### Saída



Como desejamos tornar todos os bits da imagem em escala de cinza, você deverá colar este trecho dentro da estrutura de repetição *for*. Depois, clique em **Executar**. Observe o computador executar suas instruções e transformar a imagem para a escala de cinza:

Mostrar solução







Fique à vontade para experimentar com o código-fonte, obter diferentes resultados e refletir sobre o resultado obtido com as instruções que você testou. Por exemplo, você pode multiplicar a média por 1.1, 1.5 ou 2 e observar como a imagem vai ficando mais clara. Não tenha medo de errar!



## Prática 2

Assista ao vídeo com uma segunda prática.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## Enigma 5-2-10

### Vamos desvendar o enigma 5-2-10 abordado no vídeo?

Considere a imagem, contendo frutas em seis tons de cores:



Figura 5: Frutas em 6 cores diferentes, com 863 pixels de largura por 535 pixels de altura, totalizando quase 462 mil pixels.

Esta imagem foi modificada, dividindo, para cada pixel, os valores dos componentes RGB (vermelho, verde e azul) por 5, 2 ou 10. Veja o resultado:

Figura 6: Imagem da figura 5, mas com os componentes RGB manipulados.

Note que a imagem está mais escura, além de ter suas cores distorcidas.

Vamos praticar!

Recupere a imagem (Figura 6) para o lindo padrão de cores da figura 5 original. Fique à vontade para experimentar com as multiplicações pelos três valores citados e tente descobrir como retornar para as cores originais, ou seja, desvendar o enigma!

#### Recomendação

Para diminuir o número de combinações que você tentará, tenha em mente que, quando a imagem foi modificada, os valores 5, 10 e 2 foram usados apenas uma vez cada.

Vamos lá!

Código-Fonte

```
img = new SimpleImage("img/frutas-5-2-10.jpg");
for(pixel: img){
 // Insira suas linhas de código abaixo

}
print(img);
```

Rodar/Executar

Saída

Veja a solução a seguir:

Mostrar solução

Como você se saiu?

Mostrar solução

Ao realizar as multiplicações para obter a imagem original de volta, você só precisa saber qual dos três valores foi usado para cada um dos três componentes de cores.

A tabela 6 mostra as seis possibilidades para esta combinação:

| Possibilidade | Tentativa de trazer a imagem da figura 5 de volta ao original                                                                |
|---------------|------------------------------------------------------------------------------------------------------------------------------|
| 1             | <pre>pixel.setRed( pixel.getRed()*5); pixel.setGreen(     pixel.getGreen()*10); pixel.setBlue( pixel.getBlue()*2);</pre>     |
| 2             | <pre>pixel.setRed( pixel.getRed()*5); pixel.setGreen(     pixel.getGreen()*2); pixel.setBlue(     pixel.getBlue()*10);</pre> |
| 3             | <pre>pixel.setRed( pixel.getRed()*2); pixel.setGreen(     pixel.getGreen()*5); pixel.setBlue(     pixel.getBlue()*10);</pre> |
| 4             | <pre>pixel.setRed( pixel.getRed()*2); pixel.setGreen(     pixel.getGreen()*10); pixel.setBlue( pixel.getBlue()*5);</pre>     |
| 5             | <pre>pixel.setRed( pixel.getRed()*10); pixel.setGreen(     pixel.getGreen()*5); pixel.setBlue( pixel.getBlue()*2);</pre>     |
| 6             | <pre>pixel.setRed( pixel.getRed()*10); pixel.setGreen(</pre>                                                                 |

```
pixel.getGreen()*2);
pixel.setBlue(pixel.getBlue()*5);
```

Tabela 6: Possibilidades de combinação dos valores 5, 10 e 2.

Então, o que você precisa fazer é inserir as três instruções no código-fonte, dentro da estrutura de repetição *for*, conforme as possibilidades da tabela 6. Sempre clique em Rodar/Executar para conferir se obteve a imagem original de volta.

No pior dos casos, você só precisará realizar seis tentativas.

## Vamos praticar

Como você deve ter percebido, é muito importante entender que estamos o tempo todo falando de comunicação. Claro que, tratando-se da máquina, não podemos nos comunicar com ela da mesma forma que eu me comunico com você.

## Prática 3

Como você viu, o simples comando "torne a imagem um pouco mais laranja" não poderia ser realizado apenas com estas palavras. Foi necessário traduzir essa mensagem para código de computador, de forma que o comando fosse obedecido.

**Nesse sentido, apenas com a realização de tentativas, com a identificação de erros nos comandos e sua correção, é que podemos nos aprimorar e, pouco a pouco, dar embasamento ao nosso pensamento computacional.**

Você deverá usar as caixas de códigos para realizar as atividades do Verificando o aprendizado apresentadas a seguir. A primeira caixa de código será utilizada para a primeira atividade e a segunda caixa de código para a segunda atividade.

Clique em Executar para ordenar ao computador que execute as instruções escritas por você de acordo com o enunciado de cada atividade.

### Código-Fonte

```
img = new SimpleImage("img/pag-39-2.jpg");
for(pixel: img){
 // Insira suas linhas de código abaixo

}
print(img);
```

Rodar/Executar

### Saída

### Código-Fonte

### Saída

```
img = new SimpleImage("img/pag-41.jpg");
for(pixel: img){
 // Insira suas linhas de código abaixo

}
print(img);
```

Rodar/Executar

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

#### Questão 1

Na imagem a seguir realizamos uma edição de modo que os componentes azul e verde ficaram totalmente embaralhados:

Não há como recuperá-los (canais de cores verde e azul). Toda a informação útil da imagem está apenas em tons de vermelho. Para tentarmos revelar a imagem original, nossa melhor esperança é ordenar ao computador que, para pixel da imagem:

- Ajuste os componentes de cores azul e verde para 0 (zero);
- Realce os tons em vermelho, multiplicando-os por 10.

Escreva o código-fonte que realiza os passos acima para desvendar o que havia na imagem original. Quando conseguir desvendar o enigma, assinale a alternativa que descreve o que apareceu na imagem:

- A Frutas em uma cesta em cima de uma mesa.
- B Um pássaro voando do ninho.
- C Um veículo de passeio abastecendo.
- D A paisagem de mar e montanha.
- E Uma praia.

Parabéns! A alternativa D está correta.

O código-fonte a seguir descreve, em código de computador, o passo a passo necessário para desvendar o enigma:

```
img = new SimpleImage("img/enigma-fig.png");
for(pixel:img){
 // Insira suas linhas de código abaixo
```

```
pixel.setRed(pixel.getRed()*10);
pixel.setGreen(0);
pixel.setBlue(0);
}
print(img);
```

Como resultado, vemos uma linda paisagem do Rio de Janeiro, somente em tons de vermelho.

## Questão 2

A figura a seguir está muito escura:

Desejamos clareá-la, multiplicando os componentes de cores de cada pixel por 30. Clique aqui e escreva, no código-fonte, dentro da estrutura de repetição *for*, as linhas de código que instruirão o computador a realizar essa tarefa:

Quando você tiver realizado isso com sucesso, será possível observar o que há na imagem. Assinale a alternativa que descreve o que apareceu na imagem:

- A Um submarino cercado de tubarões.
- B Um homem observando uma árvore.
- C Um raio caindo em uma árvore.
- D Um castelo cercado de muralhas.
- E Um homem em cima de um cavalo.

**Parabéns! A alternativa B está correta.**

O código-fonte a seguir descreve, em código de computador, o passo a passo necessário para clarear a imagem, conforme solicitado no enunciado, ou seja, para cada pixel da imagem, multiplicando os componentes de cores por 30:

```
img = new SimpleImage("img/muitoescuro.png");
for(pixel: img){
// Insira suas linhas de código abaixo
pixel.setRed(pixel.getRed()*30);
pixel.setGreen(pixel.getGreen()*30);
pixel.setBlue(pixel.getBlue()*30);
}
print(img);
```

Após o clareamento é possível identificar que a imagem mostra um homem observando uma árvore. Basicamente, estamos usando expressões para resolver a questão.



## 4 - A estrutura condicional *if*

Ao final deste módulo, você será capaz de distinguir a estrutura condicional *if*.

## O que são estruturas condicionais?



### Estrutura condicional

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



**Estruturas condicionais** são as linhas de código de computador empregadas para expressar a ideia de lógica ou seletividade. Usando esses tipos de instruções, somos capazes de preparar o computador para realizar um teste, cujo resultado será verdadeiro ou falso, aplicando-o para controlar se determinado pedaço de código será ou não executado.

Todas as linguagens de programação possuem estruturas condicionais. Neste módulo, analisaremos um tipo específico: a **declaração *if*** (do inglês, *if-statement*). Não deixe de manter em mente que *if*, em inglês, significa “se”.

Vamos nos basear em exemplos e práticas na linguagem JavaScript, estando sempre voltados para a resolução de interessantes problemas de manipulação de imagens.

## Estrutura condicional

Veja a sintaxe de uma declaração *if* conforme a linguagem de programação JavaScript:

JavaScript



```
1 if(pixel.getRed() > 160) {
2 pixel.setRed(0);
3 pixel.setGreen(0);
4 Pixel.setBlue(0);
5 }
```

Repare na linha 1. Em português, essa instrução significa:

**Se a intensidade do componente vermelho do pixel for maior do que 160, então execute as instruções que estão entre as chaves esquerda e direita.**

Em outras palavras, as instruções das linhas 2, 3 e 4 só serão executadas pelo computador se o resultado do teste for verdadeiro. Pense nessa instrução como uma forma de selecionar os pixels que serão afetados pelas instruções das linhas 2, 3 e 4.

Veja:

1

Por exemplo, ainda na instrução 1, se o valor do componente vermelho do pixel em questão for 140, então, as instruções 2, 3 e 4 não serão executadas pelo computador, pois o teste `if( 140 > 160)` retornará, é claro, falso.

2

Se o valor do componente vermelho do pixel em questão for 160, então as instruções 2, 3 e 4 também não serão executadas, pois o resultado do teste `if( 160 > 160)` será falso.

3

A execução das instruções 2, 3 e 4 só ocorrerá, por exemplo, se o valor do componente vermelho do pixel em questão, obtido pela função `pixel.getRed()`, for 161 ou qualquer valor maior do que este.

4

Note, na linha 1, onde há a instrução `if`, que o teste a ser realizado pelo computador está entre parênteses. Esta é uma sintaxe requerida pela linguagem JavaScript.

Somente seguindo essa sintaxe o computador será capaz de entender que se trata de um teste que ele deve realizar, para então decidir se irá ou não executar as instruções entre as chaves esquerda "{" e direita "}", que delimitam a declaração `if`.

#### Comentário

Na prática, frequentemente vemos uma declaração `if` ser usada dentro de uma estrutura de repetição, como a estrutura `for`. Esse tipo de combinação permite que problemas bastante interessantes sejam resolvidos.

Para o restante deste módulo, experimentaremos com a estrutura condicional `if`. Abordaremos várias práticas simples que envolvem manipulação de imagens digitais, começando por casos bem rudimentares e chegando a aplicações do mundo bem interessantes, ainda que simples.

Optamos por usar exemplos com imagens digitais, pois esta é uma forma muito simples, intuitiva e rápida para você observar os efeitos da execução de algumas linhas de código de computador. Isso ocorre porque estamos acostumados a observar imagens do mundo real desde que nascemos. Então, este é um processo altamente intuitivo para todos.

**Para analisar e avaliar o efeito de determinadas linhas de código, basta observar a imagem e comparar o antes e o depois da execução do código.**

## Vamos praticar

# Prática 1

Considere a seguinte imagem:

Figura 7: Imagem representativa do esquema de Cores RGB.

Imagine que desejamos escrever um código de computador que altere somente a região em vermelho, transformando-a em cinza.

Para entender como é possível escrever um código de computador a fim de alterar a região em vermelho, considere a tabela 7, que indica como cada cor é representada segundo o esquema RGB:

| Esquema RGB     | R - Vermelho                                   | G - Verde | B - Azul |
|-----------------|------------------------------------------------|-----------|----------|
| Branco          | 255                                            | 255       | 255      |
| Azul            | 0                                              | 0         | 255      |
| Vermelho        | 255                                            | 0         | 0        |
| Verde           | 0                                              | 255       | 0        |
| Amarelo         | 255                                            | 255       | 0        |
| Magenta         | 255                                            | 0         | 255      |
| Ciano           | 0                                              | 255       | 255      |
| Preto           | 0                                              | 0         | 0        |
| Escala de cinza | (1,1,1) cinza muito escuro (quase preto)       |           |          |
|                 | (2,2,2)                                        |           |          |
|                 | (3,3,3)                                        |           |          |
|                 | (4,4,4)                                        |           |          |
|                 | (5,5,5)                                        |           |          |
|                 | (6,6,6)                                        |           |          |
|                 | .                                              |           |          |
|                 | .                                              |           |          |
|                 | .                                              |           |          |
|                 | (252,252,252)                                  |           |          |
|                 | (253,253,253)                                  |           |          |
|                 | (254,254,254) cinza muito claro (quase branco) |           |          |

Tabela 7: Esquema RGB – Exemplos de cores comuns.  
Guilherme Dutra Gonzaga Jaime.

Note que o vermelho equivale ao código RGB(255,0,0). Uma primeira tentativa para transformar a região vermelha em cinza seria pensar nesse código RGB.

Então, podemos escrever um código que ordena ao computador que, para cada pixel da imagem, verifique se o componente vermelho é 255. Em caso positivo, ajuste a cor do pixel para cinza, ou seja, RGB(120,120,120).



## Vamos experimentar?

No código-fonte à esquerda estão listadas as linhas de código que representam, em código de computador, o passo a passo da tentativa do parágrafo anterior. Clique em Rodar/Executar e observe o resultado.

### Código-Fonte

```
img = new SimpleImage("img/RGB.png");
for(pixel: img){

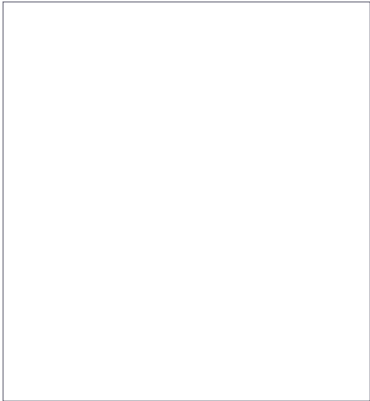
 if(pixel.getRed() == 255){
 pixel.setRed(120);
 pixel.setGreen(120);
 pixel.setBlue(120);
 }

}

print(img);
```

Rodar/Executar

### Saída



Veja a solução a seguir:



Repare que não só a região vermelha foi transformada em cinza, mas também as regiões amarelo, branco e magenta. Colorimos mais do que a região proposta pelo exercício. Observe a tabela 7. Você notará que as cores amarelo, branco e magenta também possuem, assim como o vermelho, o valor 255 para o componente R.

Então, nossa instrução `if` ao computador precisa ser mais específica.

### Relembrando

O computador faz exatamente o que ordenamos. Na prática, precisamos checar três tarefas: para que apenas a região vermelha seja ajustada para cinza, devemos verificar se o Red é igual a 255, se o Green é igual a 0 (zero), e se o Blue é igual a 0 (zero).

Então, você precisará substituir no código-fonte toda a linha da instrução `if` pela instrução a seguir:

JavaScript

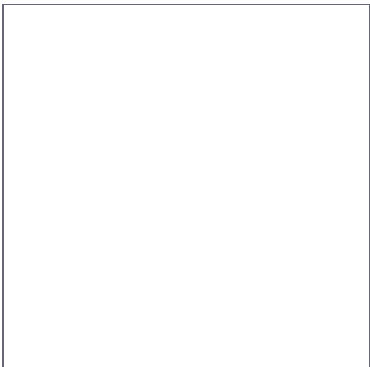


```
1 if(pixel.getRed() == 255 &&pixel.getGreen() == 0
2 &&pixel.getBlue() == 0){
```

Você pode usar o recurso de copiar e colar para isso:

### Código-Fonte

### Saída



```
img = new SimpleImage("img/RGB.png");
for(pixel: img){

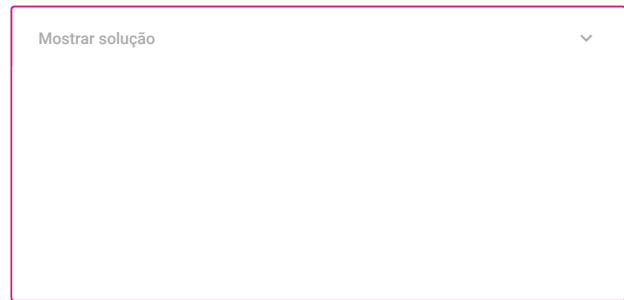
 if(pixel.getRed() == 255){
 pixel.setRed(120);
 pixel.setGreen(120);
 pixel.setBlue(120);
 }

}

print(img);
```

Rodar/Executar

Em seguida, clique em Rodar/Executar e observe que o objetivo desta prática será alcançado.



Observe que, com essa ação, você programou para que a região somente vermelha fosse ajustada para cinza, mas, ao mesmo tempo, permitiu que as regiões de interseção entre as três cores se apresentassem sem alteração.

## Prática 2

Preparamos um vídeo abordando outra prática.

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



## Continuando a praticar

### Que tal praticar um pouco?

## Que tal praticar um pouco? Seleccionando os pixels do meio-fio!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Observe a seguinte imagem:

Figura 8: Uma calçada com o meio-fio pintado de amarelo.

Imagine que desejamos ajustar a imagem para que o meio-fio fique cinza em vez de amarelo.

Pela tabela 7 sabemos que o amarelo é uma cor composta pela combinação de vermelho e verde.

Vamos utilizar o código-fonte a seguir para experimentar com diferentes abordagens de construção para a instrução *if* até alcançarmos uma solução mais refinada que nos pareça suficientemente boa.

#### Código-Fonte

```
img = new SimpleImage("img/calçada.jpg");
for(pixel: img){
 if(pixel.getRed() > 120 && pixel.getGreen() >
 120){
 pixel.setRed(0);
 pixel.setGreen(0);
 pixel.setBlue(0);
 }
}
print(img);
```

#### Saída

Rodar/Executar

No código que já consta no código-fonte, o *if* está testando se os níveis de verde e de vermelho de cada pixel são maiores do que 120. Isso reflete a ideia de que um bom nível de verde e vermelho indica que o pixel **pode** ser amarelo.

**Se o teste do *if* for positivo, o pixel que atende à condição será para a cor preta: RGB (0,0,0). Isso será interessante, pois, ao analisarmos a imagem resultante, basta observarmos o que estiver em preto. Estes terão sido os pixels selecionados pela instrução *if*.**

Clique em **Executar** e observe o resultado:

Mostrar solução

Repare que boa parte do meio-fio amarelo foi pintado de preto. Porém, analisando melhor, podemos detectar um problema: parte importante da calçada e do asfalto foram pintados de preto. Isso ocorre pois essas são áreas da imagem original em que os pixels são mais iluminados/claros.

Então, os três componentes, vermelho, verde e azul, devem estar acima de 120, gerando um tom cinza suficientemente claro. Instrua o computador a selecionar apenas pixels cujos componentes vermelho e verde estejam acima de 150.

#### Código-Fonte

#### Saída

```
img = new SimpleImage("img/calçada.jpg");
for(pixel: img){
 if(pixel.getRed() > 120 && pixel.getGreen() >
 120){
 pixel.setRed(0);
 pixel.setGreen(0);
 pixel.setBlue(0);
 }
}
print(img);
```

Rodar/Executar

Em seguida, clique em **Executar** e observe o resultado:

Mostrar solução

Observe que ainda falta nitidez nas cores, como se a tinta do meio-fio estivesse se espalhado também para a calçada. É necessário mais ajustes.

## Vamos praticar mais um pouco?

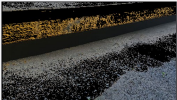
Ajuste a estrutura *if*, a fim de selecionar apenas pixels ainda mais claros e em tom de amarelo. Clique em **Experimentar** e observe o resultado:

Código Fonte

```
img = new SimpleImage("img/calçada.jpg");
for(pixel: img){
 if(pixel.getRed() > 150 && pixel.getGreen() > 150){
 pixel.setRed(0);
 pixel.setGreen(0);
 pixel.setBlue(0);
 }
}
print(img);
```

Rodar/Executar

Saída



## A seleção parece melhor, certo?

Temos menos quantidade pintada de preto na calçada. Porém outro problema tornou-se evidente: os pixels da lateral do meio-fio não foram selecionados pelo *if*. Isso ocorre porque seu tom de amarelo é mais escuro. Certamente, seu nível de luminosidade é menor do que 150. Portanto, esses pixels não passam no teste do *if*.

No último passo desta prática, tentaremos pensar em uma solução mais inteligente para a instrução *if*, de forma que consigamos, também, selecionar os pixels da lateral para realizar o ajuste de cores. Então, vamos construir uma estratégia melhor. Para começar, observe nosso “laboratório RGB” com controles deslizantes para os componentes vermelho, verde e azul:

Sem cor →→→ Escuro →→→ Mais claro →→→ Saturação total

Vermelho  
(R - Red):

Verde (G  
- Green):

Azul (B -  
Blue):

R:0 G:0 B:0

☐ Mostrar Hexadecimal #000000

Repare que você obteve um amarelo-escuro.

Agora, deslize o controle azul de 0(zero) para 84. Se tiver dificuldade em obter o valor exato, use as setas direcionais direita/esquerda no teclado. Observe o resultado:

Observe o amarelo gradativamente aproximando-se do cinza.

## Considerações

Queremos tirar proveito do padrão da natureza que acabamos de observar.

Tons de amarelo são caracterizados por níveis equivalentes de vermelho e verde e um nível de azul muito mais baixo. Como, porém, podemos escrever uma estrutura `if` para instruir o computador a selecionar os pixels que atendam a esse padrão?

Felizmente, isso é muito simples! Observe as duas figuras a seguir:

Na primeira figura, se calcularmos a média dos três componentes `R(97)`, `G(97)` e `B(0)`, obteremos o valor 64.6. A linha tracejada representa onde estaria esse valor nos controles deslizantes. Repare que os componentes vermelho e verde estão acima da média. O mesmo comportamento se repete na segunda figura.



## Que tal praticar um pouco? Alterando a cor do meio-fio para cinza!

Para assistir a um vídeo sobre o assunto, acesse a versão online deste conteúdo.



Então, vamos usar essa observação para construir uma solução mais inteligente para a instrução `if` e observar se, desse modo, selecionamos os pixels em amarelo do meio-fio de forma mais satisfatória. Volte ao código-fonte desta prática e realize os ajustes destacados em sublinhado a seguir:

JavaScript



```
1 for(pixel: img){
```

```

2 media=(pixel.getRed()+pixel.getGreen()+pixel.getBlue()) /
3 if(pixel.getRed() >media&&pixel.getGreen() >media){
4 pixel.setRed(0);
5 pixel.setGreen(0);
6 pixel.setBlue(0);
7 }
8 }
9 print(img);

```

```
img = new SimpleImage("calçada.jpg");
```

#### Código-Fonte

```

img = new SimpleImage("img/calçada.jpg");
for(pixel: img){
if(pixel.getRed() > 120 && pixel.getGreen() >
120){
pixel.setRed(0);
pixel.setGreen(0);
pixel.setBlue(0);
}
}
print(img);

```

#### Saída

Rodar/Executar

Adicionaremos uma linha para o computador calcular a média automaticamente e, na estrutura *if*, testaremos se o valor do vermelho e do verde são maiores do que a média. Depois, clique em **Executar** e observe o resultado:

Mostrar solução

Agora, confirmamos que a nossa estrutura *if* é capaz de selecionar, de forma bem satisfatória, os pixels que desejamos ajustar para a tonalidade cinza. Já pensamos em uma forma de escrever a instrução *if*, de modo que os pixels do meio-fio sejam corretamente selecionados para modificação de cor. O último passo para alcançarmos nosso objetivo é ajustar as instruções internas à estrutura *if*, em que, de fato, as cores dos pixels selecionados são ajustadas.

Durante os passos anteriores, deixamos as instruções alterando o valor dos componentes RGB para 0 (zero), para que pudéssemos observar o que ficou em preto e refletir se a nossa estrutura *if* estava correta.

**Agora, pense um pouco: queremos que o meio-fio, que era amarelo, apareça em escala de cinza.**

Pela tabela 7 já sabemos que a escala de cinza equivale a dizer que os três componentes R, G e B possuem o mesmo valor. Ora, nós já ordenamos que o computador calcule automaticamente a média dos três componentes do pixel.

Então, a variável chamada *media* contém uma estimativa de quanta luminosidade existe no pixel sendo tratado. Uma boa tentativa seria trocar o 0 (zero) das instruções de ajuste de cor internas ao *if* pela variável *media*. Realize esses ajustes no código-fonte, conforme destacado em sublinhado:



```

2 for(mpx=0; mpx< img.getRed()+pixel.getGreen()+pixel.getBlue()) /
3 if(pixel.getRed() > media && pixel.getGreen() > media){
4 pixel.setRed(media);
5 pixel.setGreen(media);
6 pixel.setBlue(media);
7 }
8 }
9 print(img);

```

#### Código-Fonte

```

img = new SimpleImage("img/calçada.jpg");
for(pixel: img){
if(pixel.getRed() > 120 && pixel.getGreen() >
120){
pixel.setRed(0);
pixel.setGreen(0);
pixel.setBlue(0);
}
}
print(img);

```

Rodar/Executar

#### Saída



Observe o resultado:

Mostrar solução



Isso realmente parece bem melhor! Conseguimos alcançar o nosso objetivo.

Escreveremos um código de computador que é capaz de instruí-lo a detectar automaticamente a região em amarelo do meio-fio, para, então, ajustar sua tonalidade de cor de amarelo para cinza.

## Pensamento computacional

O pensamento computacional já é considerado em vários países do mundo como a competência fundamental usada por todas as pessoas nas próximas décadas. Assim como a leitura, a escrita e a aritmética, essa competência será imprescindível para o mercado de trabalho.

Professores e alunos do ensino fundamental, ensino médio e ensino superior, além de cientistas, engenheiros, historiadores, artistas, médicos, advogados, todos precisarão usá-la para desempenhar seu papel de forma competitiva. Este estudo está baseado justamente no pensamento computacional.

Primeiramente, compreendemos que o computador representa qualquer conceito do mundo real por meio de números. No caso de imagens, os

números indicam a posição do pixel e sua cor.

Para isso, precisamos desconsiderar qualquer outro detalhe da imagem que não seja fundamental para sua representação:



Quantos pixels usaremos para representá-la.



Qual é a cor de cada pixel.

Esse tipo de habilidade é denominada abstração e constitui um dos pilares primordiais do pensamento computacional.

Em seguida, escrevemos linhas de códigos (instruções) de computador para que ele fosse capaz de automaticamente manipular esses números e, assim, realizarmos ajustes nas imagens, conforme o objetivo proposto.

### Comentário

Dessa forma, praticamos outra habilidade chave do chamado pensamento computacional, que é a **automação**, ou seja, escrever soluções na forma de instruções que o computador é capaz de seguir automaticamente para chegar ao resultado desejado.

Durante o desenvolvimento das práticas, sempre paramos para observar e analisar os resultados obtidos nas imagens apresentadas. Então, refletimos sobre como poderíamos melhorar os resultados alcançados e realizamos ajustes em nosso código para alcançar resultados mais adequados. Isso constitui outro pilar fundamental do pensamento computacional, denominado **análise/avaliação**.

Observe que, durante as práticas realizadas, o código-fonte que escrevemos sempre decompôs o problema proposto em partes mais simples, que seguiram o seguinte passo a passo padrão:

### Passo 1

Carregar a imagem na memória para que fosse trabalhada – escrevemos uma linha de código que carrega a imagem.

### Passo 2

Automaticamente processar, um a um, todos os pixels (centenas de milhares) de uma imagem – usamos a estrutura `for` para instruir o computador a repetir as instruções para cada pixel da imagem.

### Passo 3

Selecionar em quais pixels desejamos realizar ajuste de cores – escrevemos a estrutura `if` para que o computador realizasse testes e, conforme o resultado (verdadeiro ou falso), executasse ou não as instruções de manipulação de cores do pixel.

### Passo 4

Alterar a cor de um pixel – escrevemos linhas de código para alterar os componentes RGB (vermelho, verde e azul) para alterar a cor de um pixel, conforme nosso objetivo.

### Passo 5

Imprimir o resultado na tela – escrevemos a função `print` para que o resultado das instruções (a



imagem manipulada) fosse apresentado na tela.

A habilidade de decompor um problema em problemas bem menores e mais simples, que podem ser resolvidos isoladamente com uma ou poucas linhas de código de computador, é mais um dos pilares do pensamento computacional, denominado **decomposição**.

Além disso, durante a prática do ajuste de cor do meio-fio de amarelo para escala de cinza, foi preciso observar um padrão natural/orgânico de qualquer pixel de tonalidades amareladas. Eles possuem os componentes verde e vermelho maiores do que o componente azul.

**Usamos esta observação em nosso favor e escrevemos um código de computador que foi capaz de automaticamente selecionar pixels em diversos tons de amarelo, para que pudéssemos ajustar suas cores. Essa habilidade é denominada detecção de padrões.**

A aprendizagem de todos esses pilares permitirá que você desenvolva cada vez mais saberes na área da Computação.

Você deverá usar o explorador RGB e a caixa de códigos para realizar as atividades do **Verificando o aprendizado**.

Sem cor →→ Escuro →→ Mais claro →→ Saturação total

Vermelho (R - Red):

Verde (G - Green):

Azul (B - Blue):

R:0 G:0 B:0

☐ Mostrar Hexadecimal

#000000

Clique em **Executar** para ordenar ao computador que execute as instruções escritas por você.

Código-Fonte

```
img = new SimpleImage("img/RGB.png");
for(pixel: img){

 // Insira seu código aqui

}
print(img);
```

Rodar/Executar

Saída

Falta pouco para atingir seus objetivos.

Vamos praticar alguns conceitos?

Questão 1

Analise a instrução if a seguir:

```
if(pixel.getRed() ==162&&pixel.getGreen() ==162 &&pixel.getBlue() > 200){ }
```

Assinale a alternativa que informa as características dos pixels que serão “selecionados” por este *if*. Se necessário, clique aqui e use os controles deslizantes do simulador RGB para chegar a sua resposta.

- A      Pixels em tonalidades de azul.
- B      Pixels em tonalidades de amarelo.
- C      Pixels em tonalidades de verde.
- D      Pixels em tonalidades de vermelho.
- E      Pixels em tonalidade laranja.

**Parabéns! A alternativa A está correta.**

O *if* em questão seleciona pixels que passem em três testes:

- Nível de vermelho igual a 162;
- Nível de verde igual a 162;
- Nível de azul maior do que 200.

Conforme ilustrado pelo exemplo a seguir, obtido por meio do simulador RGB disponibilizado para esta atividade, o teste em questão equivale a escolher pixels em tonalidades de azul.

## Questão 2

Na Prática 1, usamos como ponto de partida a imagem a seguir para realizar alguns experimentos:

Suponha que desejemos escrever uma linha de instrução *if* que selecione os pixels da região em azul para, então, torná-los amarelos.

Assinale a alternativa que contém o código correto para alcançar essa tarefa.

- A      

```
if(pixel.getRed()==0 &&pixel.getGreen()==0 &&pixel.getBlue()==255)
{
 pixel.setRed(255);
 pixel.setGreen(255);
 pixel.setBlue(0);
}
```
- B      

```
if(pixel.getRed()>0 &&pixel.getGreen()>0 &&pixel.getBlue()==255){
 pixel.setRed(255);
 pixel.setGreen(255);
 pixel.setBlue(0);
}
```

- C
- ```
if( pixel.getRed()==0 &&pixel.getGreen()==0 &&pixel.getBlue()==255 )
{
    pixel.setRed(0);
    pixel.setGreen(255);
    pixel.setBlue(255);
}
```
- D
- ```
if(pixel.getRed()>0 &&pixel.getGreen()>0 &&pixel.getBlue()==255){
 pixel.setRed(0);
 pixel.setGreen(255);
 pixel.setBlue(255);
}
```
- E
- ```
if( pixel.getRed()>0 &&pixel.getGreen()>0 &&pixel.getBlue()==0 ){
    pixel.setRed(0);
    pixel.setGreen(0);
    pixel.setBlue(0);
}
```

Parabéns! A alternativa A está correta.

Conforme estudamos, precisamos usar a estrutura condicional *if* para realizar testes e, se o resultado for verdadeiro, executar algumas instruções.

No caso desta atividade, os pixels da região em azul são os pixels com código RGB(0,0,255). Então, para selecionar corretamente estes pixels, usamos a instrução *if*, conforme observamos a seguir:

```
if( pixel.getRed()==0 &&pixel.getGreen()==0 &&pixel.getBlue()==255 ){
}
```

Depois, precisamos adicionar, dentro da estrutura *if*, instruções que ajustarão a cor dos pixels selecionados para amarelo. O código RGB do amarelo puro é (255,255,0). Então, o código completo ficaria conforme a alternativa A.

Veja a captura a seguir com o código-fonte e o resultado obtido:

Considerações finais

Praticamos aqui conceitos primordiais de pensamento computacional ao exercitarmos noções de programação de computadores com exemplos de manipulação de imagem.

Além disso, realizamos experimentos práticos de programação embutidos no conteúdo digital. Não hesite em voltar ao conteúdo e usar esses experimentos para transmitir adiante o conhecimento adquirido.



Podcast

Para encerrar, ouça um resumo dos principais tópicos deste conteúdo.

Para ouvir o *áudio*, acesse a versão online deste conteúdo.



Referências

CARVALHO, A.; LORENA, A. **Introdução à computação**: hardware, software e Dados. Rio de Janeiro: LTC, 2017.

DALE, N.; LEWIS, J. **Ciência da Computação**. 4. ed. Rio de Janeiro: LTC, 2011.

FEDELI, R. D.; POLLONI, E. G. F.; PERES, F. E. **Introdução à Ciência da Computação**. 2. ed. São Paulo: Cengage, 2010.

FLANAGEN, D. **JavaScript**: o guia definitivo. 6. ed. Porto Alegre: Bookman, 2013.

GLENN, J. **Ciência da Computação**: uma visão abrangente. 11. ed. Porto Alegre: Bookman, 2013.

Explore +

Confira a indicação que separamos especialmente para você!

Pesquise o artigo **Proposta de atividades para o desenvolvimento do pensamento computacional no ensino fundamental**, de Daiane Andrade e outros autores. Campinas: Unicamp, 2013. p. 169-178.