Nome: Pedro Henrique de Araujo Marques

1 – Confira os tipos de dados do Octave/Matlab através do comando:
typeinfo

>> ans =
{
[1,1] = <unknown type>
[2,1] = cell
[3,1] = scalar
[4,1] = complex scalar
[5,1] = matrix
[6,1] = diagonal matrix
[7,1] = complex matrix
[8,1] = complex diagonal matrix
[9,1] = range
[10,1] = bool
[11,1] = bool matrix
[12,1] = string
[13,1] = sq_string
[14,1] = int8 scalar
[15,1] = int16 scalar
[16,1] = int32 scalar
[17,1] = int64 scalar
[18,1] = uint8 scalar
[19,1] = uint16 scalar
[20,1] = uint32 scalar
[21,1] = uint64 scalar
[22,1] = int8 matrix
[23,1] = int16 matrix
[24,1] = int32 matrix
[25,1] = int64 matrix
[26,1] = uint8 matrix
[27,1] = uint16 matrix
[28,1] = uint32 matrix
[29,1] = uint64 matrix
[30,1] = sparse bool matrix
[31,1] = sparse matrix
[32,1] = sparse complex matrix
[33,1] = struct
[34,1] = scalar struct
[35,1] = class
[36,1] = cs-list
[37,1] = magic-colon
[38,1] = built-in function

[39,1] = user-defined function
[40,1] = dynamically-linked function
[41,1] = function handle
[42,1] = inline function
[43,1] = float scalar
[44,1] = float complex scalar
[45,1] = float matrix
[46,1] = float diagonal matrix
[47,1] = float complex matrix
[48,1] = float complex diagonal matrix
[49,1] = permutation matrix
[50,1] = null_matrix
[51,1] = null_string
[52,1] = null_sq_string
[53,1] = lazy_index
[54,1] = onCleanup
[55,1] = octave_java
[56,1] = object
}

2. Tipos de imagem:

RGB

Crie 3 matrizes (RED, GREEN e BLUE), com as mesmas dimensões, contendo valores entre 0 e 1.
Exemplo (três matrizes 3x3):
RED = [1 0 0; 1 0 1; 1 0 0.5];
GREEN = [0 0 1; 1 0 0; 1 1 0.5];
BLUE = [0 0 0; 0 1 1; 1 1 0.5];

Matrizes criadas(3x4):
>> RED: [0 1 1; 1 0.5 1; 0.5 0.75 0; 1 0 0.75]
>> GREEN: [0 0 0; 1 0.75 0.5; 1 1 0.5; 0.75 0.5 0.75]
>> BLUE: [0 0.5 0; 0 0 1; 1 1 0.5; 0.5 0.5 1]

Para ver o conteúdo das matrizes, digite seus nomes e, em seguida Enter. É muito importante notar que o Octave/Matlab
é case sensitive.

>> RED
RED =

0.00000 1.00000 1.00000
1.00000 0.50000 1.00000

```
   0.50000   0.75000   0.00000
   1.00000   0.00000   0.75000
>> GREEN
GREEN =

   0.00000   0.00000   0.00000
   1.00000   0.75000   0.50000
   1.00000   1.00000   0.50000
   0.75000   0.50000   0.75000

>> BLUE
BLUE =

   0.00000   0.50000   0.00000
   0.00000   0.00000   1.00000
   1.00000   1.00000   0.50000
   0.50000   0.50000   1.00000
```

Concatenar as três matrizes para criar uma única matriz tridimensional (RGB) usando a função cat (usar a ajuda para aprender sobre a função: help cat).
Exemplo:
RGB = cat (3, RED, GREEN, BLUE);

```
>> RGB = cat(3, RED, GREEN, BLUE);
```

Visualize o conteúdo da matriz RGB. Os dois primeiros índices dos elementos da matriz definem, respectivamente, os números de linha e coluna. O terceiro índice refere-se a uma das três matrizes originais. Se considerarmos agora matriz RGB como uma imagem, os dois primeiros índices definem as coordenadas de um pixel, e o terceiro, um dos componentes RGB do pixel.
Exemplo (experimente os comandos abaixo):
RGB(:,:,1)
RGB(:,:,2)
RGB(:,:,3)

```
>> RGB(:,:,1)
ans =

   0.00000   1.00000   1.00000
   1.00000   0.50000   1.00000
   0.50000   0.75000   0.00000
   1.00000   0.00000   0.75000
```

>> RGB(:,:,2)
ans =

0.00000	**0.00000**	**0.00000**
1.00000	**0.75000**	**0.50000**
1.00000	**1.00000**	**0.50000**
0.75000	**0.50000**	**0.75000**

>> RGB(:,:,3)
ans =

0.00000	**0.50000**	**0.00000**
0.00000	**0.00000**	**1.00000**
1.00000	**1.00000**	**0.50000**
0.50000	**0.50000**	**1.00000**

Visualize a matriz resultante com a função:
imshow (RGB);
>> imshow(RGB)

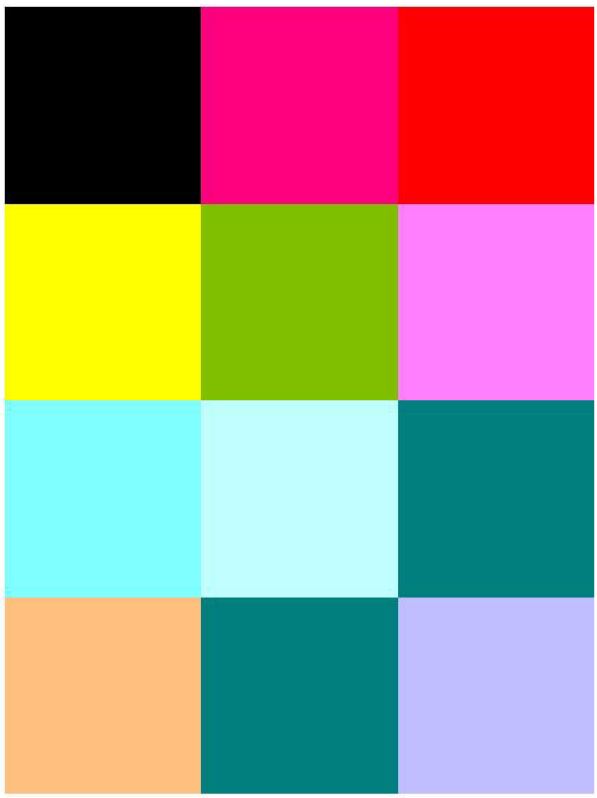


Imagem indexada

Crie uma matriz com o nome MAP, com três linhas e três colunas e valores entre 0 e 1, para representar um mapa de cores. Crie uma matriz bidimensional com o nome X, com valores inteiros entre 1 e 3. Visualize a imagem indexada usando:
imshow (X, MAP);

```
>> MAP = [0 0 1; 0 1 0; 1 0 0]
MAP =

   0   0   1
   0   1   0
   1   0   0

>> X = [1 2 3; 3 1 2; 2 3 1]
X =

   1   2   3
   3   1   2
   2   3   1

>> imshow (X,MAP)
```

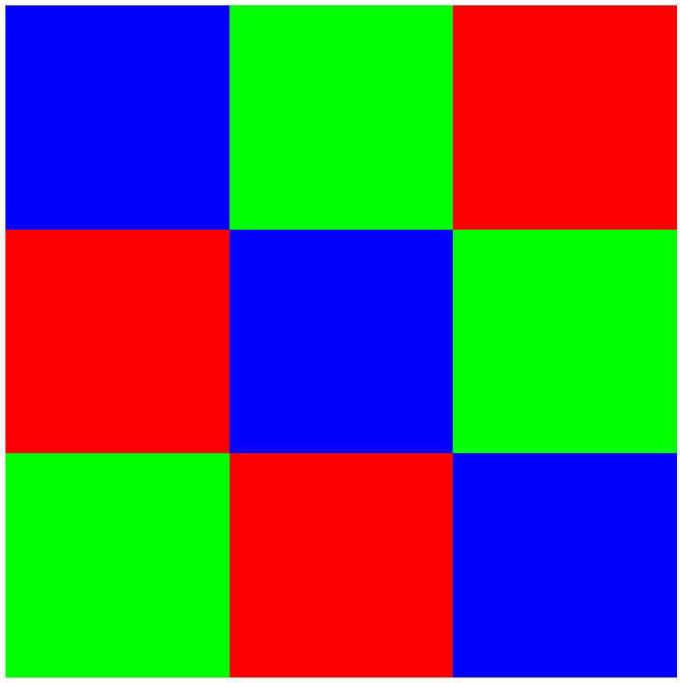


Imagem de Níveis de Cinza

Crie uma matriz X com valores entre 0 e 1.

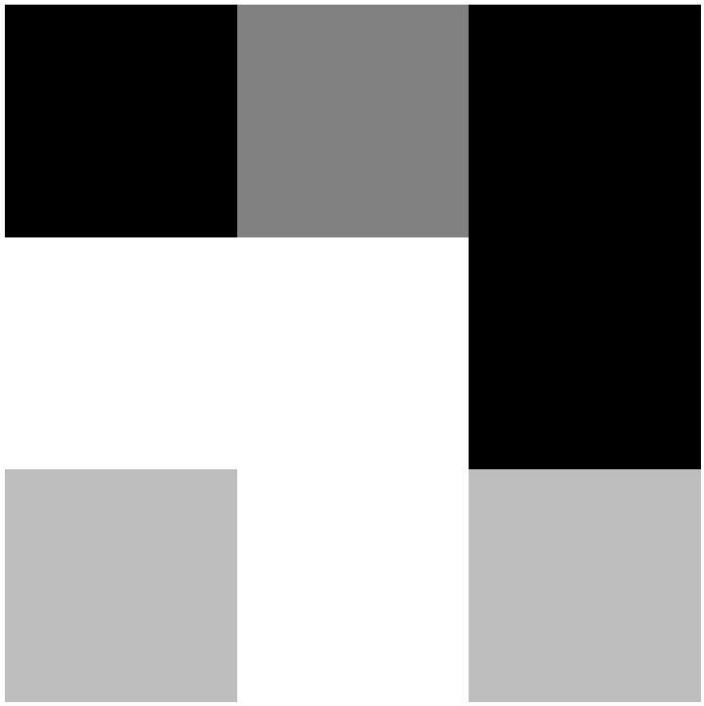
```
>> X = [0 0.5 0; 1 1 0; 0.75 1 0.75]
X =

   0.00000   0.50000   0.00000
   1.00000   1.00000   0.00000
   0.75000   1.00000   0.75000
```

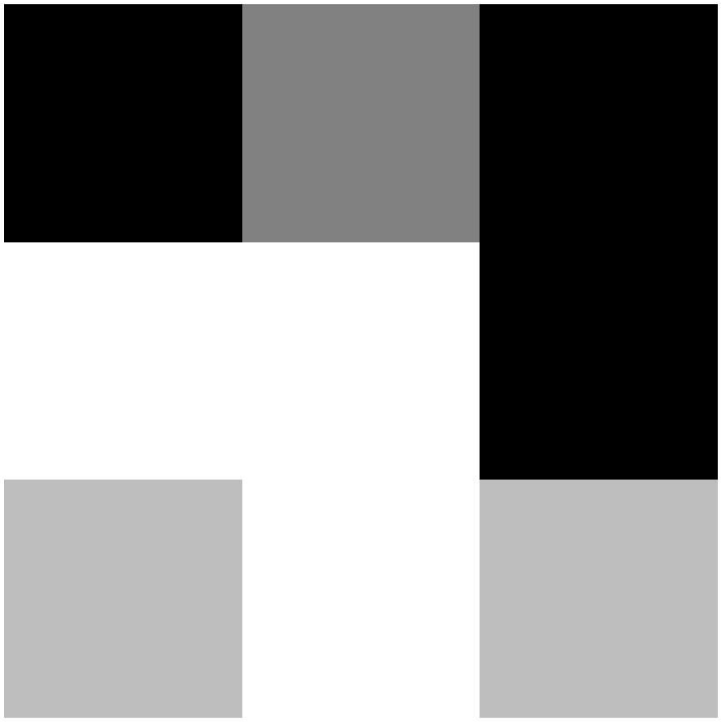
Visualize a matriz X como uma imagem de níveis de cinza (intensidade). Experimente os comandos abaixo:

```
imshow (X);
imshow (X, [0 1]);
imshow (X, [0 2]);
imshow (X, [0 16]);
```

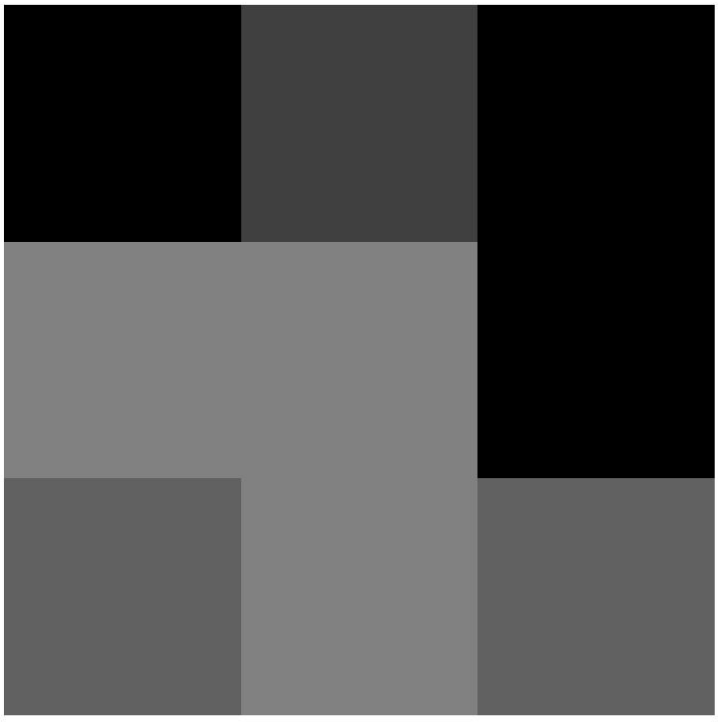
```
>> imshow(X)
```



>> imshow(X, [0 1])



>> imshow(X, [0 2])



>> imshow(X, [0 16])

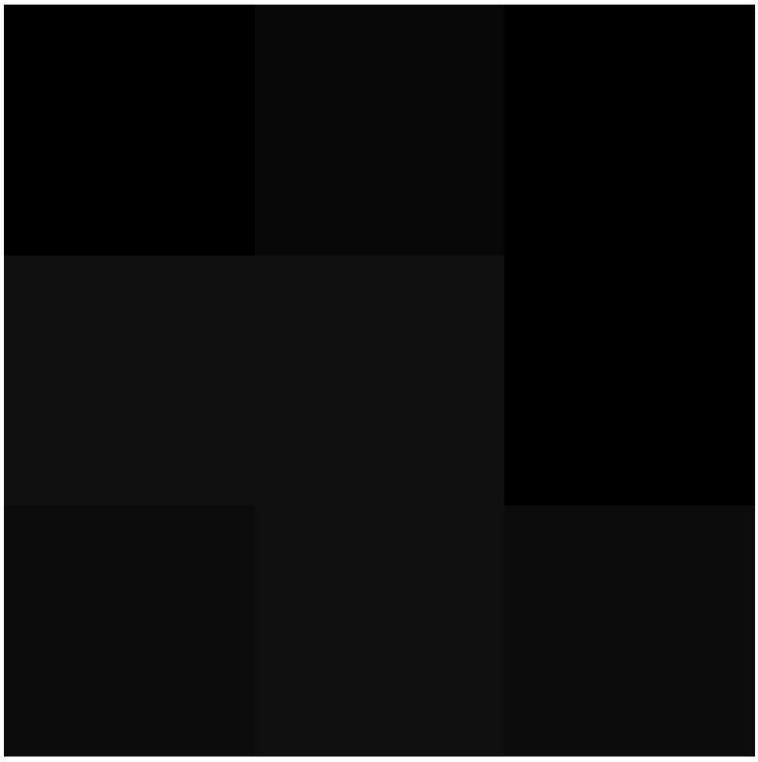


Imagem Binária

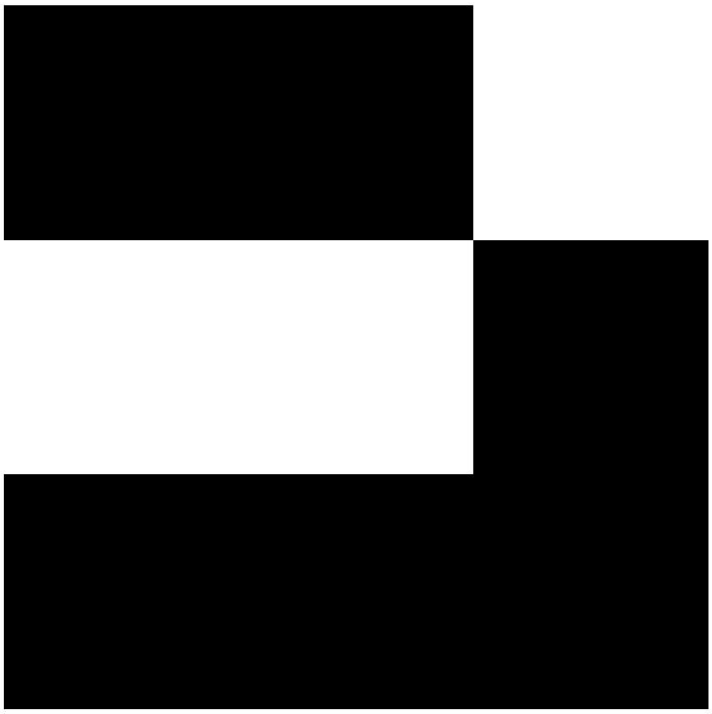
Crie uma matriz X usando exclusivamente valores 0 ou 1
(imagem binária).

>> X = [0 0 1; 1 1 0; 0 0 0]
X =

0	**0**	**1**
1	**1**	**0**
0	**0**	**0**

Visualize a matriz X usando o comando:
imshow (X);

>> imshow(X)



3. Leitura e gravação de imagens:

Leitura de Imagens

Estudar a função imread usando a ajuda do Octave/Matlab.

>> help imread
'imread' is a function from the file /usr/share/octave/4.0.3/m/image/imread.m

-- Function File: [IMG, MAP, ALPHA] = imread (FILENAME)
-- Function File: [...] = imread (URL)
-- Function File: [...] = imread (..., EXT)
-- Function File: [...] = imread (..., IDX)
-- Function File: [...] = imread (..., PARAM1, VAL1, ...)
Read images from various file formats.

Read an image as a matrix from the file FILENAME. If there is no file FILENAME, and EXT was specified, it will look for a file with the extension EXT. Finally, it will attempt to download and read an image from URL.

The size and class of the output depends on the format of the image. A color image is returned as an MxNx3 matrix. Gray-level and black-and-white images are of size MxN. Multipage images will have an additional 4th dimension.

The bit depth of the image determines the class of the output: "uint8", "uint16" or "single" for gray and color, and "logical" for black and white. Note that indexed images always return the indexes for a colormap, independent if MAP is a requested output. To obtain the actual RGB image, use 'ind2rgb'. When more than one indexed image is being read, MAP is obtained from the first. In some rare cases this may be incorrect and 'imfinfo' can be used to obtain the colormap of each image.

See the Octave manual for more information in representing images.

Some file formats, such as TIFF and GIF, are able to store multiple images in a single file. IDX can be a scalar or vector specifying the index of the images to read. By default, Octave will only read the first page.

Depending on the file format, it is possible to configure the reading of images with PARAM, VAL pairs. The following options are supported:

'"Frames" or "Index"'
This is an alternative method to specify IDX. When specifying it in this way, its value can also be the string "all".

'"Info"'
This option exists for MATLAB compatibility and has no effect. For maximum performance while reading multiple images from a single file, use the Index option.

'"PixelRegion"'
Controls the image region that is read. Takes as value a cell array with two arrays of 3 elements '{ROWS COLS}'. The elements in the array are the start, increment and end pixel to be read. If the increment value is omitted, defaults to 1. For example, the following are all equivalent:

```
imread (filename, "PixelRegion", {[200 600] [300 700]});
imread (filename, "PixelRegion", {[200 1 600] [300 1 700]});
imread (filename)(200:600, 300:700);
```

See also: imwrite, imfinfo, imformats.

Leia as imagens arara_full.png e arara_full_256.bmp usando a função imread.

Primeiras 16 colunas de cada:
>> imread("arara_full.png")
ans =

ans(:,:,1) =

Columns 1 through 16:

237	220	222	221	222	219	221	216	217	213	210	210	210	209	205	201
215	197	200	196	197	195	197	193	196	193	191	191	192	191	187	184
220	200	202	197	198	197	200	197	195	194	193	193	193	191	189	188
219	200	202	199	199	198	201	200	200	200	198	197	195	194	193	193
220	203	205	204	204	203	206	205	199	199	197	195	193	192	192	194
217	200	205	203	206	204	206	204	205	202	200	198	196	196	196	197
224	205	210	208	211	209	211	209	207	206	202	200	199	197	196	196
225	207	211	208	210	210	211	209	210	207	206	202	200	197	194	192
233	208	211	209	208	207	210	209	213	212	210	207	203	199	197	196
235	211	214	213	211	208	213	209	213	212	211	207	203	200	199	200
237	212	216	215	214	211	215	211	212	211	211	209	205	202	201	202
239	215	217	218	218	215	218	217	212	213	212	209	205	202	201	202
243	218	218	219	220	217	221	219	215	215	213	210	206	202	201	201
247	221	221	222	223	222	224	220	218	216	214	211	208	205	203	201
247	222	225	226	227	226	227	223	220	217	213	212	212	210	206	203
245	221	226	229	231	230	231	226	221	217	213	213	214	213	209	207
247	223	222	231	232	231	227	229	226	216	210	214	217	213	209	210
247	225	222	229	229	228	227	228	220	215	212	215	216	213	211	213
246	226	224	227	226	228	227	225	217	218	216	216	215	213	214	214
246	229	226	226	225	229	226	222	218	220	220	218	216	214	214	212
246	230	227	227	224	228	224	221	220	222	220	219	220	218	214	209
247	230	227	226	222	225	221	219	220	220	218	220	221	218	213	209
249	229	225	224	220	221	219	219	220	218	219	222	221	215	210	209
251	229	223	222	217	218	217	220	220	217	219	222	220	212	207	209
251	225	222	220	219	215	218	222	220	219	220	220	215	209	207	208
252	226	224	222	221	213	215	220	218	218	218	217	213	207	206	208
253	228	225	223	219	210	211	215	214	214	213	211	208	205	205	207
253	228	225	221	217	209	210	213	209	209	207	206	205	204	206	209
255	229	224	221	217	208	210	215	208	208	206	204	205	206	207	210
255	228	223	220	217	209	211	214	209	209	207	205	206	208	209	210
254	226	220	218	216	209	208	210	206	207	206	205	206	209	210	211
251	223	217	216	216	207	206	205	203	205	205	205	207	211	213	213
248	219	218	217	215	207	208	204	205	205	206	207	208	209	210	213

```
>> imread("arara_full_256.bmp")
ans =

 Columns 1 through 16:

191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 126 190 190 190 190 190 190 190 190 190 190 190 190 190 182
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 191 191 191 191 191 191 190 190 190 190 190 190 190 190
191 190 191 191 191 191 191 191 191 190 190 190 190 190 190 190
191 191 191 191 191 190 191 191 191 191 191 190 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 190 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 191 191 191 191 191 191 191 191 191 190 190 190 190 190 191
191 191 191 191 191 191 191 191 191 191 190 190 190 190 190 191
191 191 191 191 191 191 191 191 191 191 190 190 190 191 191 191
191 191 191 191 191 191 191 191 190 190 190 190 190 191 191 191
191 191 191 191 191 190 190 190 190 190 190 190 190 191 191 191
191 191 191 191 191 190 191 190 190 190 190 190 191 191 191 191
191 191 191 191 191 191 191 190 190 190 190 191 191 191 191 191
191 191 191 191 191 191 191 182 190 190 190 190 190 191 191 191
```

Tente descobrir, com a ajuda da função whos se as imagens são RGB ou indexadas (para saber mais sobre a função use help whos).

>> arara_full = imread("arara_full.png");
>> arara_full_256 = imread("arara_full_256.bmp");
>> whos arara_full arara_full_256
Variables in the current scope:

Attr	Name	Size	Bytes	Class
	arara_full	296x460x3	408480	uint8
	arara_full_256	296x460	136160	uint8

Total is 544640 elements using 544640 bytes

Se analizarmos a parte “Size” da tabela, podemos perceber que arara_full tem o tamanho da imagem em pixels (296x460) multiplicado por 3, um para cada corhttps://www.youtube.com/watch?v=9KABcmczPdg RGB, logo essa seria a imagem RGB, enquanto a arara_full_256 a imagem indexada.

Visualize as imagens usando a função imshow.

>> imshow("arara_full.png")

>> imshow("arara_full_256.bmp")

Escrita de Imagens

Gravar em disco cópias das imagens lidas no exercício anterior com as extensões trocadas (.bmp ↔.png) usando a função imwrite.

>> imwrite(arara_full, "arara_full.bmp");
>> imwrite(arara_full_256, "arara_full_256.png");

Ler as imagens gravadas (imread) e visualize-as (imshow).

Primeiras 16 colunas:

>> imread("arara_full.bmp")
ans =

ans(:,:,1) =

Columns 1 through 16:

```
237 220 222 221 222 219 221 216 217 213 210 210 210 209 205 201
215 197 200 196 197 195 197 193 196 193 191 191 192 191 187 184
220 200 202 197 198 197 200 197 195 194 193 193 193 191 189 188
219 200 202 199 199 198 201 200 200 200 198 197 195 194 193 193
220 203 205 204 204 203 206 205 199 199 197 195 193 192 192 194
217 200 205 203 206 204 206 204 205 202 200 198 196 196 196 197
224 205 210 208 211 209 211 209 207 206 202 200 199 197 196 196
225 207 211 208 210 210 211 209 210 207 206 202 200 197 194 192
233 208 211 209 208 207 210 209 213 212 210 207 203 199 197 196
235 211 214 213 211 208 213 209 213 212 211 207 203 200 199 200
237 212 216 215 214 211 215 211 212 211 211 209 205 202 201 202
239 215 217 218 218 215 218 217 212 213 212 209 205 202 201 202
243 218 218 219 220 217 221 219 215 215 213 210 206 202 201 201
247 221 221 222 223 222 224 220 218 216 214 211 208 205 203 201
247 222 225 226 227 226 227 223 220 217 213 212 212 210 206 203
245 221 226 229 231 230 231 226 221 217 213 213 214 213 209 207
247 223 222 231 232 231 227 229 226 216 210 214 217 213 209 210
247 225 222 229 229 228 227 228 220 215 212 215 216 213 211 213
246 226 224 227 226 228 227 225 217 218 216 216 215 213 214 214
246 229 226 226 225 229 226 222 218 220 220 218 216 214 214 212
246 230 227 227 224 228 224 221 220 222 220 219 220 218 214 209
247 230 227 226 222 225 221 219 220 220 218 220 221 218 213 209
249 229 225 224 220 221 219 219 220 218 219 222 221 215 210 209
251 229 223 222 217 218 217 220 220 217 219 222 220 212 207 209
251 225 222 220 219 215 218 222 220 219 220 220 215 209 207 208
252 226 224 222 221 213 215 220 218 218 218 217 213 207 206 208
253 228 225 223 219 210 211 215 214 214 213 211 208 205 205 207
253 228 225 221 217 209 210 213 209 209 207 206 205 204 206 209
255 229 224 221 217 208 210 215 208 208 206 204 205 206 207 210
255 228 223 220 217 209 211 214 209 209 207 205 206 208 209 210
254 226 220 218 216 209 208 210 206 207 206 205 206 209 210 211
251 223 217 216 216 207 206 205 203 205 205 205 207 211 213 213
248 219 218 217 215 207 208 204 205 205 206 207 208 209 210 213
```

>> imread("arara_full_256.png")
ans =

Columns 1 through 16:

```
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 126 190 190 190 190 190 190 190 190 190 190 190 190 190 182
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
```

191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 190 190 190 190 190 190 190 190 190 190 190 190 190 190
191 190 191 191 191 191 191 191 190 190 190 190 190 190 190 190
191 190 191 191 191 191 191 191 191 190 190 190 190 190 190 190
191 191 191 191 191 190 191 191 191 191 191 190 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 190 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 190 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 190 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 190
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 191 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 191 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 190 190 191
191 191 191 191 191 191 191 191 191 191 191 191 191 190 190 190
191 191 191 191 191 191 191 191 191 191 190 190 190 190 190 191
191 191 191 191 191 191 191 191 191 191 190 190 190 190 190 191
191 191 191 191 191 191 191 191 191 191 190 190 190 191 191 191
191 191 191 191 191 191 191 191 190 190 190 190 190 191 191 191
191 191 191 191 191 190 190 190 190 190 190 190 190 191 191 191
191 191 191 191 191 190 191 190 190 190 190 190 191 191 191 191
191 191 191 191 191 191 191 190 190 190 190 191 191 191 191 191
191 191 191 191 191 191 191 182 190 190 190 190 190 191 191 191

>> imshow("arara_full.bmp")

>> imshow("arara_full_256.png")

4.Conversão de Imagens:

Aprenda a usar as funções ind2gray, gray2ind, rgb2ind, ind2rgb, rgb2gray e gray2rgb. Observe os tipos de dados (double ou uint8) dos elementos das matrizes resultantes.

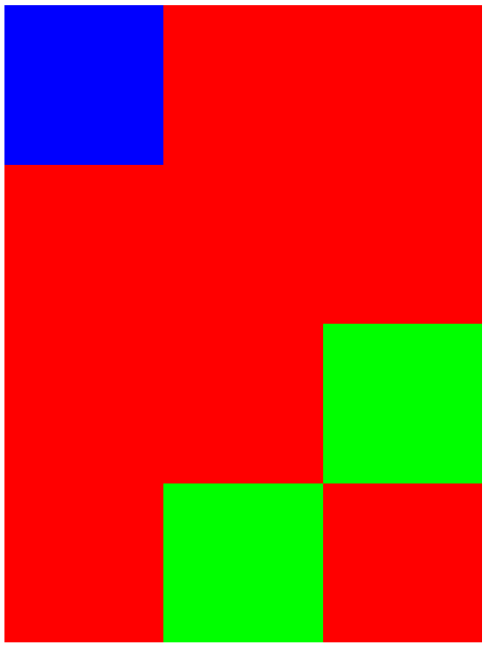
>> whos gray_ind ind_rgb rgb_ind ind_gray rgb_gray
Variables in the current scope:

Attr	Name	Size	Bytes	Class
	gray_ind	3x3	9	uint8
	ind_rgb	3x3x3	216	double
	rgb_ind	4x3	12	uint8
	ind_gray	3x3	72	double
	rgb_gray	4x3	96	double

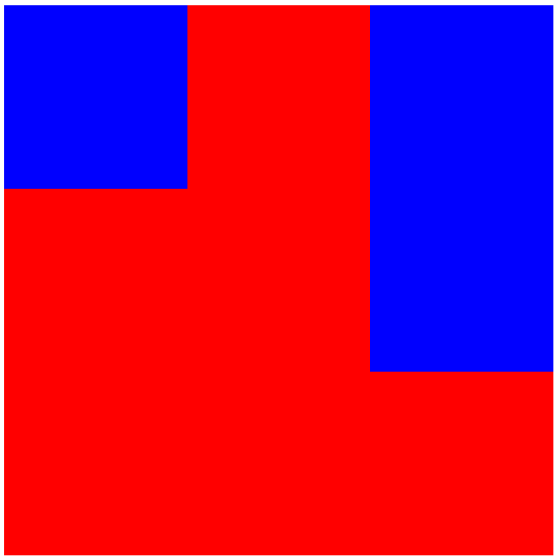
Total is 69 elements using 405 bytes

Transformar as imagens criadas nos exercícios anteriores em diferentes tipos de imagem e visualizar as imagens resultantes.

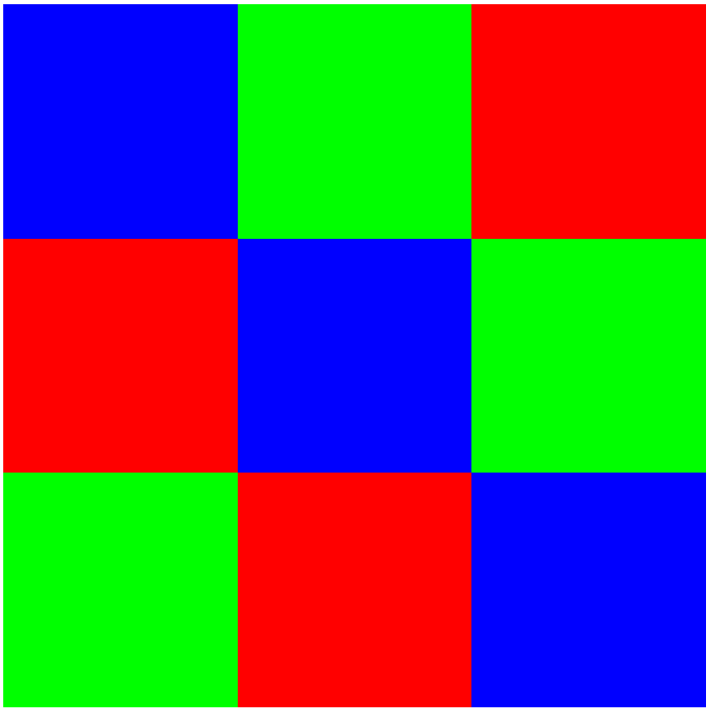
>> imshow(rgb_ind, MAP)



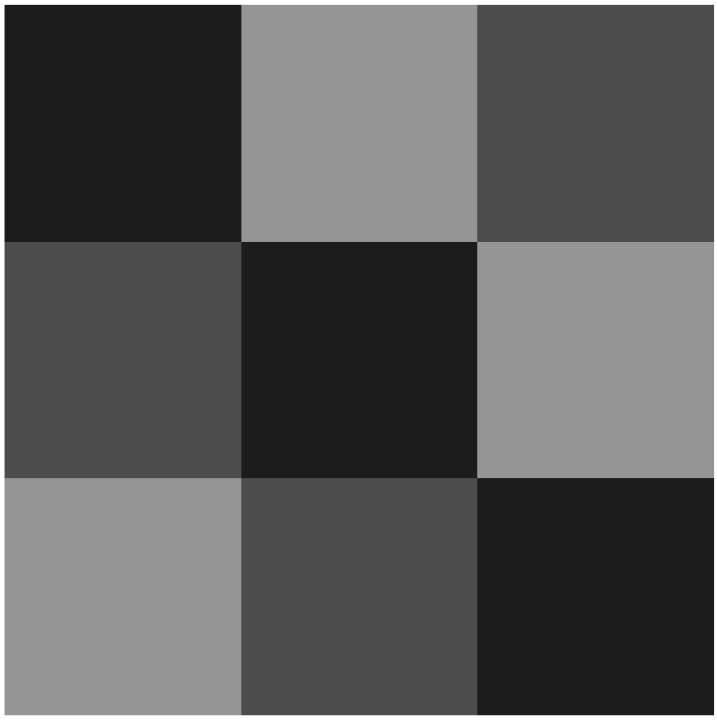
>> imshow(gray_ind, MAP)



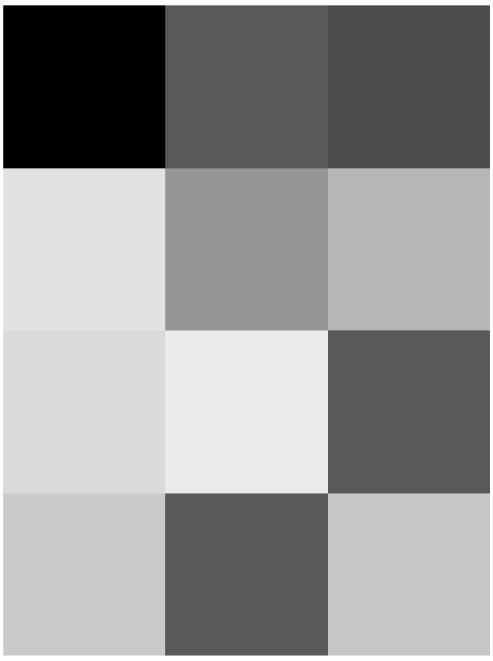
>> imshow(ind_rgb)



>> imshow(ind_gray)



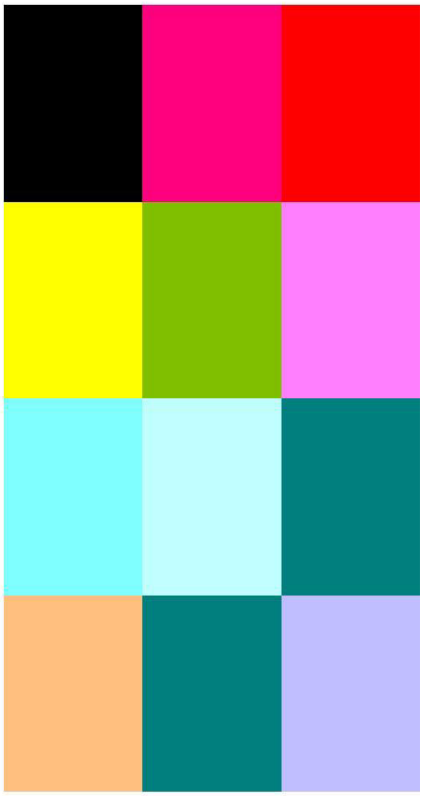
>> imshow(rgb_gray)

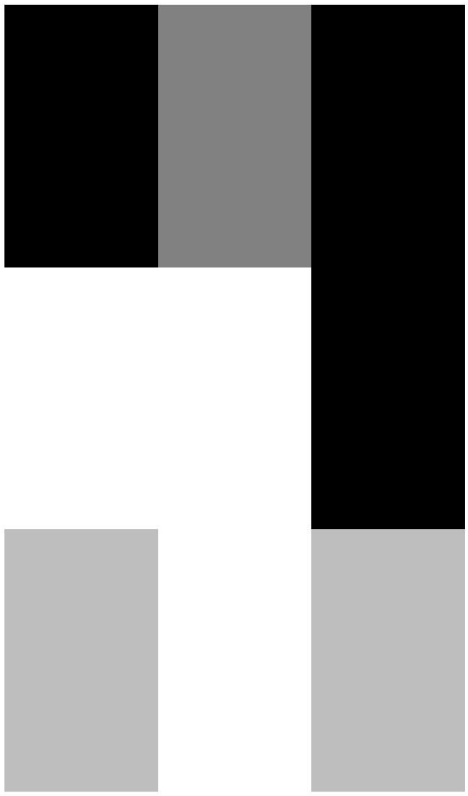


5. Visualização:

Aprenda a usar a função figure. Mostrar duas imagens diferentes em diferentes janelas.

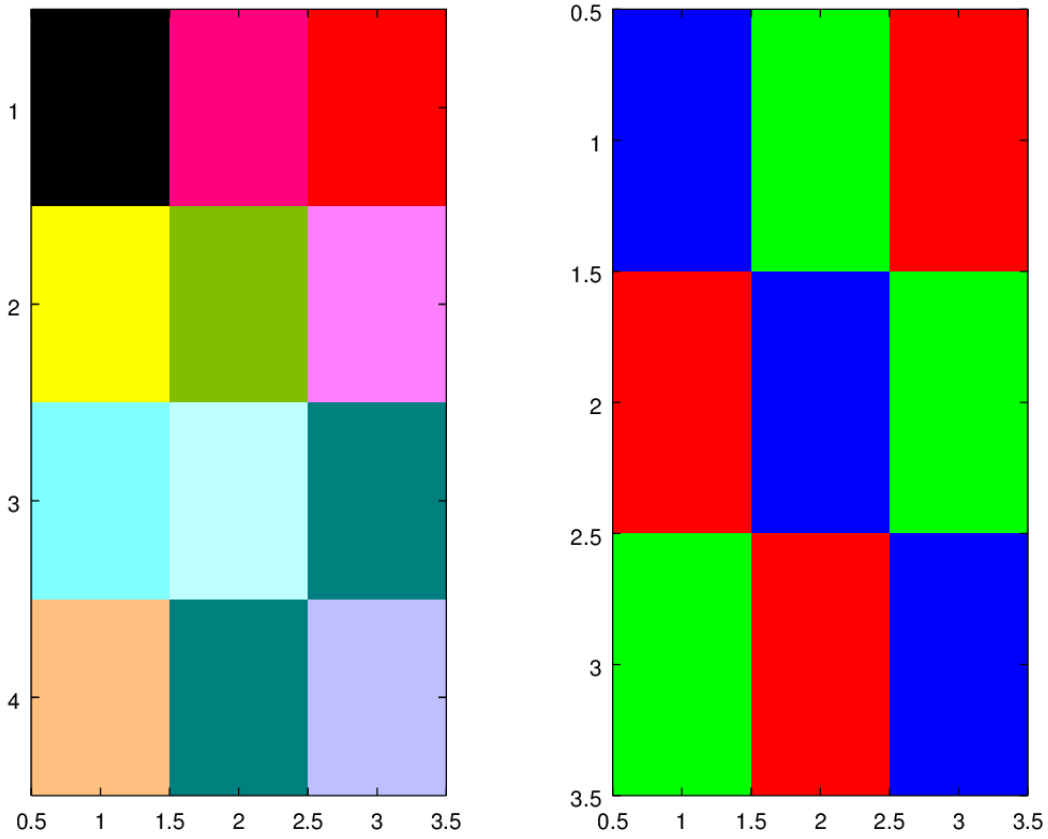
>> figure(1)
>> subimage(RGB)
>> figure(2)
>> subimage(X)





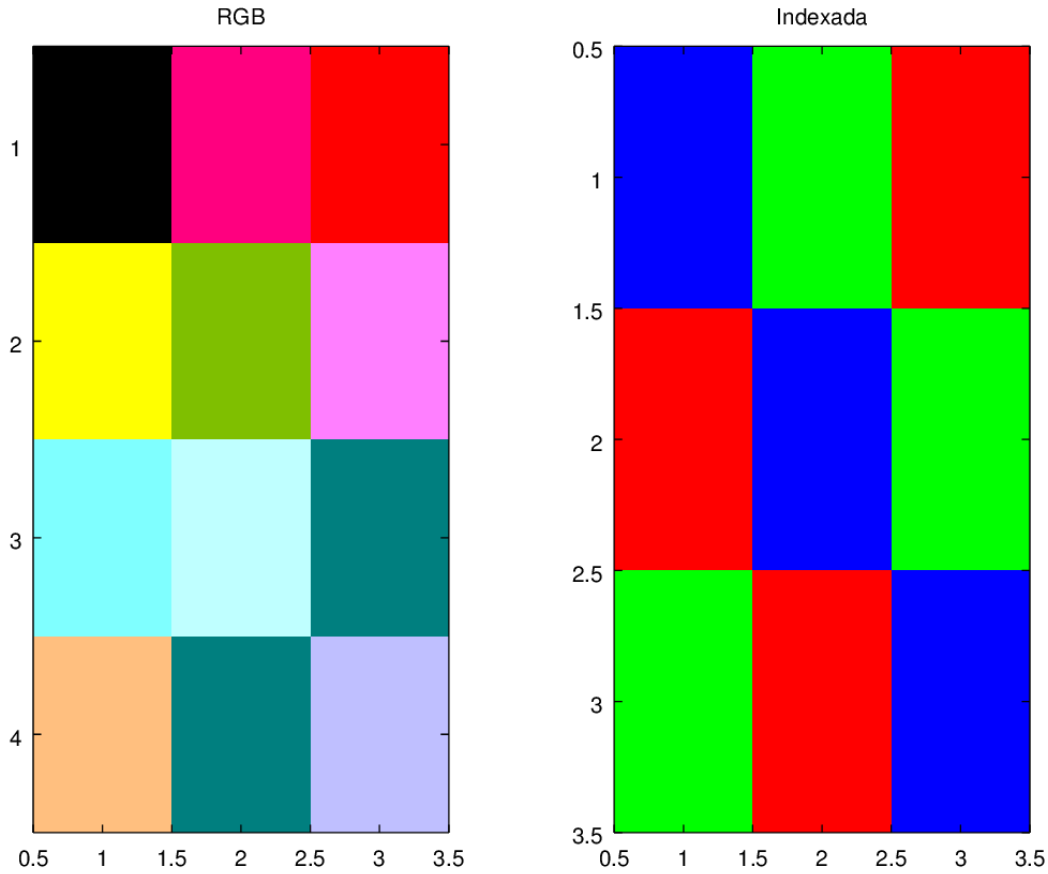
Aprenda a usar as funções subplot e subimage. Mostrar as duas imagens de exercícios anteriores em uma única janela usando
estas funções.

>> subplot(1,2,1), subimage(RGB)
>> subplot(1,2,2), subimage(X, MAP)



Aprenda a usar a função title. Coloque títulos nas janelas dos exercícios anteriores.

>> subplot(1,2,1), title("RGB")
>> subplot(1,2,2), title("Indexada")



6. Matrizes:

Criar a matriz A usando o seguinte comando:
A = [16 3 2 13 19; 5 10 11 8 3; 6 7 9 12 8; 4 15 14 1 13; 1 2 3 4 5]

>> A = [16 3 2 13 19; 5 10 11 8 3; 6 7 9 12 8; 4 15 14 1 13; 1 2 3 4 5]
A =

16	**3**	**2**	**13**	**19**
5	**10**	**11**	**8**	**3**
6	**7**	**9**	**12**	**8**
4	**15**	**14**	**1**	**13**
1	**2**	**3**	**4**	**5**

Calcular a soma das quatro células dos cantos da matriz, referenciando os respectivos índices.

```
>> A(1,1) + A(1,5) + A(5,1) + A(5,5)
ans =  41
```

Verifique a saída dos seguintes comandos:
A

```
>> A
A =

   16    3    2   13   19
    5   10   11    8    3
    6    7    9   12    8
    4   15   14    1   13
    1    2    3    4    5
```

A(1,3) + A(3,1)

```
>> A(1,3) + A(3,1)
ans =  8
```

A(1: 3,3)

```
>> A(1: 3,3)
ans =

    2
   11
    9
```

A(1:4,2:4)

```
>> A(1:4,2:4)
ans =

    3    2   13
   10   11    8
    7    9   12
   15   14    1
```

A(:, 3)

```
>> A(:, 3)
ans =

    2
   11
    9
   14
    3
```

A(1:3, :)

```
>> A(1:3, :)
ans =

   16    3    2   13   19
    5   10   11    8    3
    6    7    9   12    8
```

Criar uma matriz 5x3 B, e aplicar o comando A(:, [3 5 2]) = B(:, 1:3)

```
>> B = [1 7 9; 2 23 1; 8 8 9; 5 13 3; 1 6 0]
B =

    1    7    9
    2   23    1
    8    8    9
    5   13    3
    1    6    0
>> A(:, [3 5 2]) = B(:, 1:3)
A =

   16    9    1   13    7
    5    1    2    8   23
    6    9    8   12    8
    4    3    5    1   13
    1    0    1    4    6
```

A(:)

```
>> A(:)
ans =

   16
    5
    6
    4
    1
    9
    1
    9
    3
    0
    1
    2
    8
    5
    1
   13
    8
   12
    1
    4
    7
   23
    8
   13
    6
```

A’

```
>> A'
ans =

   16    5    6    4    1
    9    1    9    3    0
    1    2    8    5    1
   13    8   12    1    4
    7   23    8   13    6
```

A(:, [1 2 2 3 3 3 4 4 4 4])

>> A(:, [1 2 2 3 3 3 4 4 4 4])
ans =

16	**9**	**9**	**1**	**1**	**1**	**13**	**13**	**13**	**13**
5	**1**	**1**	**2**	**2**	**2**	**8**	**8**	**8**	**8**
6	**9**	**9**	**8**	**8**	**8**	**12**	**12**	**12**	**12**
4	**3**	**3**	**5**	**5**	**5**	**1**	**1**	**1**	**1**
1	**0**	**0**	**1**	**1**	**1**	**4**	**4**	**4**	**4**

A([1 2 2 3 3 3 4 4 4 4], :)

>> A([1 2 2 3 3 3 4 4 4 4], :)
ans =

16	**9**	**1**	**13**	**7**
5	**1**	**2**	**8**	**23**
5	**1**	**2**	**8**	**23**
6	**9**	**8**	**12**	**8**
6	**9**	**8**	**12**	**8**
6	**9**	**8**	**12**	**8**
4	**3**	**5**	**1**	**13**
4	**3**	**5**	**1**	**13**
4	**3**	**5**	**1**	**13**
4	**3**	**5**	**1**	**13**

A(A>10) = 0

>> A(A>10) = 0
A =

0	**9**	**1**	**0**	**7**
5	**1**	**2**	**8**	**0**
6	**9**	**8**	**0**	**8**
4	**3**	**5**	**1**	**0**
1	**0**	**1**	**4**	**6**

Estude os comandos zeros, ones, eye, size.

>> zeros(2,5)
ans =

0 0 0 0 0
0 0 0 0 0

zeros: Gera uma matriz composta apenas por zeros.

>> ones(2,5)
ans =

1 1 1 1 1
1 1 1 1 1

ones: Gera uma matriz composta apenas por ums.

>> eye(5,5)
ans =

Diagonal Matrix

1 0 0 0 0
0 1 0 0 0
0 0 1 0 0
0 0 0 1 0
0 0 0 0 1

eye: Gera uma matriz diagonal.

>> size(A)
ans =

5 5

size: Retorna o número de linhas e colunas da matriz dada como parâmetro.

Use help arith para aprender sobre os operadores: "+", "-", "*", "/", "./", "^", "*." "^."

>> help +

-- Operator: +
Addition operator.

See also: plus.

Operador usado para realizar somas.

>> help -

-- Operator: -
Subtraction or unary negation operator.

See also: minus.

Operador usado para realizar subtrações.

>> help *

-- Operator: *
Multiplication operator.

See also: .*, times.

Operador usado para realizar multiplicações.

>> help /

-- Operator: /
Right division operator.

See also: ./, , rdivide, mrdivide.

Operador usado para realizar divisões.

>> help ./
'./' is the file /home/pedro/Downloads/

-- Operator: ./
Element by element right division operator.

See also: /, ., rdivide, mrdivide.

Operador usado para realizar divisões elemento por elemento.

>> help ^

-- Operator: ^
Power operator. This may return complex results for real inputs.
Use 'realsqrt', 'cbrt', 'nthroot', or 'realroot' to obtain real
results when possible.

See also: power, **, .^, ., realpow, realsqrt, cbrt, nthroot.**

Operador usado para realizar potênciação.

>> help .*

-- Operator: .*
Element by element multiplication operator.

See also: *, times.

Operador usado para realizar multiplicações elemento por elemento.

>> help .^

-- Operator: .^
Element by element power operator. If several complex results are possible, returns the one with smallest non-negative argument

(angle). Use 'realpow', 'realsqrt', 'cbrt', or 'nthroot' if a real
result is preferred.

See also: ., ^, **, power, realpow, realsqrt, cbrt, nthroot.**

Operador usado para realizar potenciação elemento por elemento.

Experimente com os operadores: "*",". *", "/", "./"

```
>> M1 = [1 2 3; 4 0 2; 8 9 1]
M1 =

   1   2   3
   4   0   2
   8   9   1

>> M2 = [2 2 6; 7 1 4; 10 5 2]
M2 =

    2    2    6
    7    1    4
   10    5    2

>> M1 * M2
ans =

   46   19   20
   28   18   28
   89   30   86

>> M1 .* M2
ans =

    2    4   18
   28    0    8
   80   45    2

>> M1 / M2
ans =

   0.656627  -0.337349   0.204819
  -0.132530   0.746988  -0.096386
   0.692771  -1.722892   1.867470
```

```
>> M1 ./ M2
```

ans =

0.50000	**1.00000**	**0.50000**
0.57143	**0.00000**	**0.50000**
0.80000	**1.80000**	**0.50000**