```
# somando dois números

a = 4
b = 5

soma = a + b

print("o resultado da soma é: " + str((soma)))
```

```
o resultado da soma é: 9
```

```
# criando um vetor

vetor = []

vetor.append(4)

print(vetor)

vetor.append(-58)

print(vetor)
```

```
[4]
[4, -58]
```

```
# percorrendo o vetor

for i in vetor:
  print(i)
```

```
4
-58
```

```
# mostrando na tela 0 até 4 (5 posições)

for i in range(5):
  print(str(i))
```

```
0
1
2
3
4
```

```
# criando um vetor e colocando nele números de 1 a 5

vetor1 = []

for i in range(5):
```

```
  vetor1.append(i)

print(vetor1)
```

```
[0, 1, 2, 3, 4]
```

```
# removendo o elemento 2 do vetor

vetor1.remove(2)

print(vetor1)
```

```
[0, 1, 3, 4]
```

```
# criando uma cópia do vetor1

vetor2 = vetor1.copy()

print(vetor2)
```

```
[0, 1, 3, 4]
```

```
# criando uma matriz
import numpy as np  #biblioteca que suporta o processamento de grandes, multi-dimensionais arranjos e matrizes, juntamente com uma grande coleção de funções matemáticas de alto nível para operar sobre estas matrizes.

matriz = np.array([[5, 8, 4], [6, 10, 64], [22, 74, 61]])

print(matriz)
```

```
[[ 5  8  4]
 [ 6 10 64]
 [22 74 61]]
```

```
# acessando linha 0 coluna 0 da matriz

matriz[0,0]
```

```
5
```

```
# acessando os elementos da segunda coluna da matriz
matriz[:,1]
```

```
array([ 8, 10, 74])
```

```
# acessando os elementos da terceira linha da matriz
matriz[2,:]
```

```
array([22, 74, 61])
```

```
# modificando a entrada linha 1 coluna 2 para o valor: 80

matriz[0][1] = 80

print(matriz)
```

```
[[ 5 80  4]
 [ 6 10 64]
 [22 74 61]]
```

```
# criando uma matriz 5 x 5 toda com zeros

a = np.zeros((5,5))

print(a)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

```
# criando uma matriz 4 x 4 toda com um

a1 = np.ones((4,4))

print(a1)
```

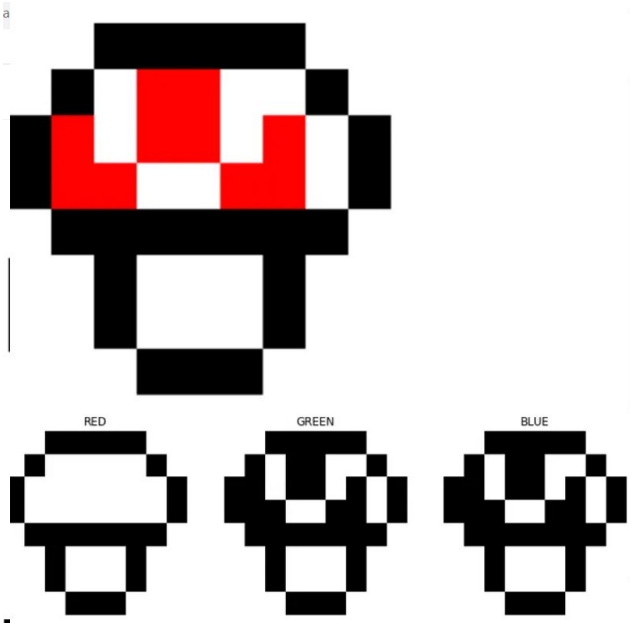
```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

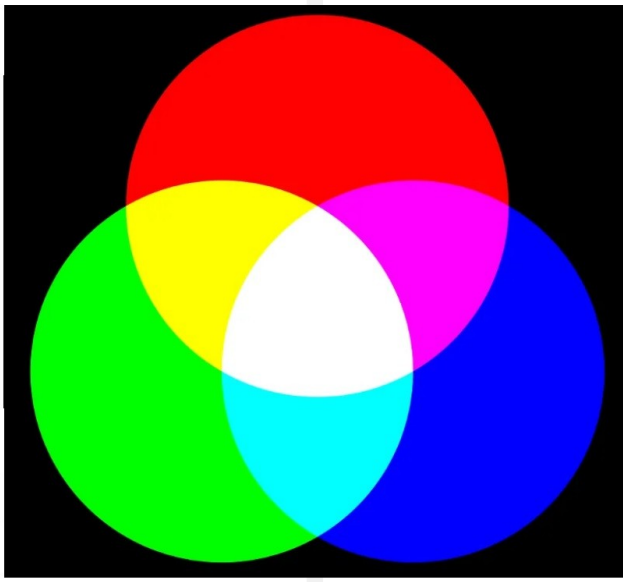
```
# modificando a segunda coluna da matriz a1 para 0

a1[:,1] = 0

print(a1)
```

```
[[1. 0. 1. 1.]
 [1. 0. 1. 1.]
 [1. 0. 1. 1.]
 [1. 0. 1. 1.]]
```

Até agora vimos matrizes que possuem linhas e colunas. Queremos também declarar as camadas! Para entender melhor essa loucura, pense num cubo mágico:

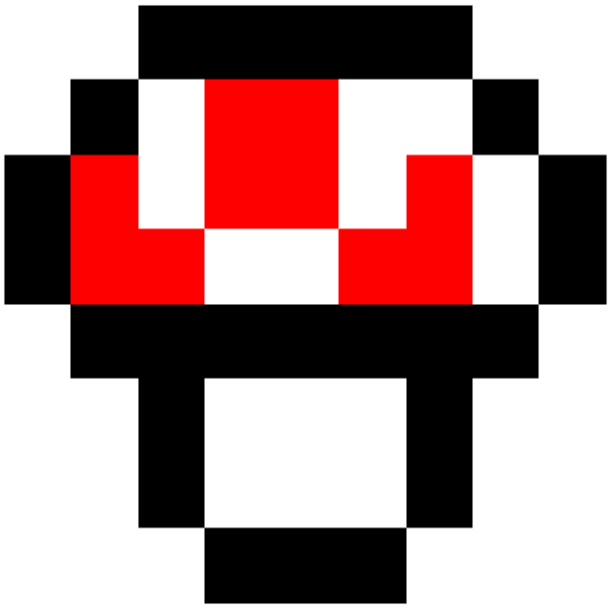
Criaremos uma matriz 8x9x3. As primeira, segunda e terceira camadas da nossa matriz serão compostas pelas variações de tons das camadas vermelha, azul e verde, respectivamente, da imagem do cogumelo do Mário. Primeiro criaremos as camadas separadamente.



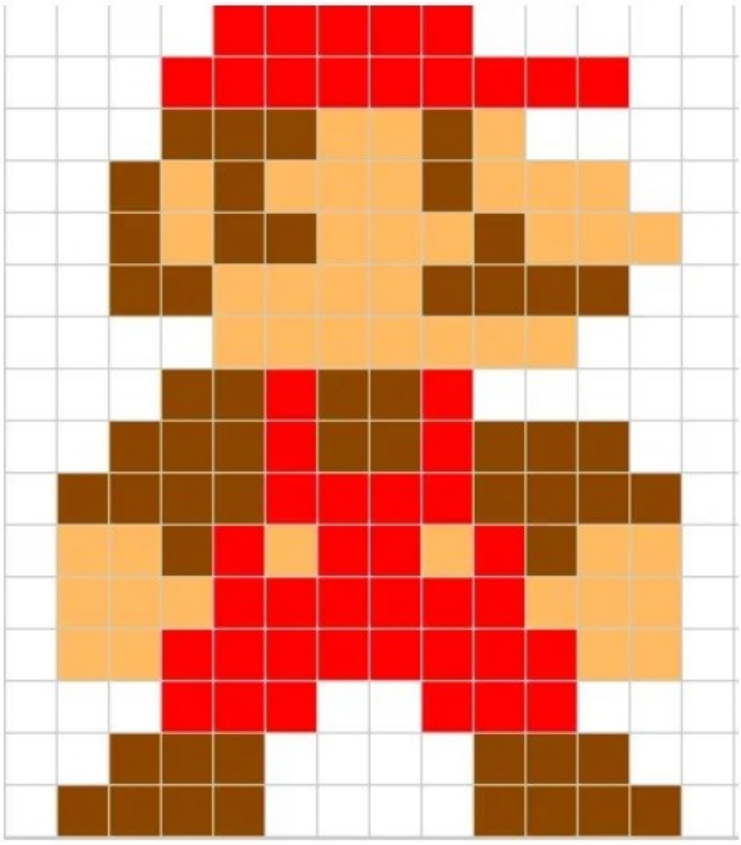


```
# Cada camada é uma matriz! Para declarar uma matriz de 1’s com 8
linhas, 9 colunas e 3 camadas

R = np.array([
    [1, 1, 0, 0, 0, 0, 0, 1, 1],
    [1, 0, 1, 1, 1, 1, 1, 0, 1],
    [0, 1, 1, 1, 1, 1, 1, 1, 0],
    [0, 1, 1, 1, 1, 1, 1, 1, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0, 0, 1, 1, 1]
], dtype=np.float32)



G = np.array([
    [1, 1, 0, 0, 0, 0, 0, 1, 1],
    [1, 0, 1, 0, 0, 1, 1, 0, 1],
    [0, 0, 1, 0, 0, 1, 0, 1, 0],
    [0, 0, 0, 1, 1, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0, 0, 1, 1, 1]
], dtype=np.float32)



B = np.array([
    [1, 1, 0, 0, 0, 0, 0, 1, 1],
    [1, 0, 1, 0, 0, 1, 1, 0, 1],
    [0, 0, 1, 0, 0, 1, 0, 1, 0],
    [0, 0, 0, 1, 1, 0, 0, 1, 0],
    [1, 0, 0, 0, 0, 0, 0, 0, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 0, 1, 1, 1, 0, 1, 1],
    [1, 1, 1, 0, 0, 0, 1, 1, 1]
], dtype=np.float32)



# cria matriz 8x9x3 nula
M = np.zeros((8,9,3), dtype=np.float32)
# atribui R à 1ª camada de M
M[:,:,0] = R
# atribui G à 2ª camada de M
M[:,:,1] = G
# atribui B à 3ª camada de M
```

```
M[:,:,2] = B

#Para mostrar o cogumelo no display:

import matplotlib.pyplot as plt
plt.figure(figsize=(4,4))
im = plt.imshow(M, aspect='auto')
plt.axis("off")
plt.show()
```



Desafio (14 colunas x 16 linhas)



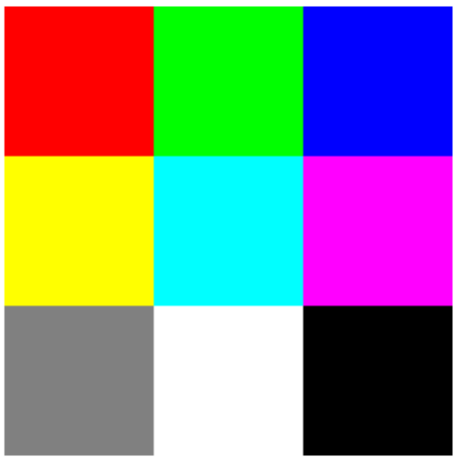
Ateliê Drisol Artesanatos

Tabela de Mistura de Cores

Drisol

www.drisolartes.blogspot.com

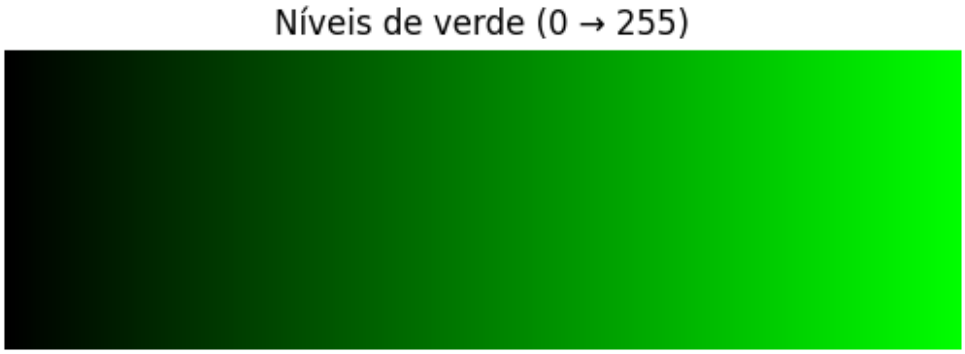
```python
import numpy as np
import matplotlib.pyplot as plt

# Criar uma matriz 3x3x3 (3x3 pixels, 3 canais RGB)
M = np.zeros((3,3,3), dtype=np.uint8)

# Definir algumas cores básicas usando níveis 0–255
M[0,0] = [255, 0, 0]     # vermelho
M[0,1] = [0, 255, 0]     # verde
M[0,2] = [0, 0, 255]     # azul

M[1,0] = [255, 255, 0]   # amarelo (R+G)
M[1,1] = [0, 255, 255]   # ciano (G+B)
M[1,2] = [255, 0, 255]   # magenta (R+B)

M[2,0] = [128, 128, 128] # cinza médio
M[2,1] = [255, 255, 255] # branco
M[2,2] = [0, 0, 0]       # preto

# Mostrar a matriz como imagem
plt.figure(figsize=(3,3))
plt.imshow(M)
plt.axis("off")
plt.show()
```



```
0
```

```python
import numpy as np
import matplotlib.pyplot as plt

# Criar uma imagem 1x256 pixels (uma linha), com 3 canais (RGB)
M = np.zeros((80, 256, 3), dtype=np.uint8)  # altura 50 px só para
visualizar melhor
```

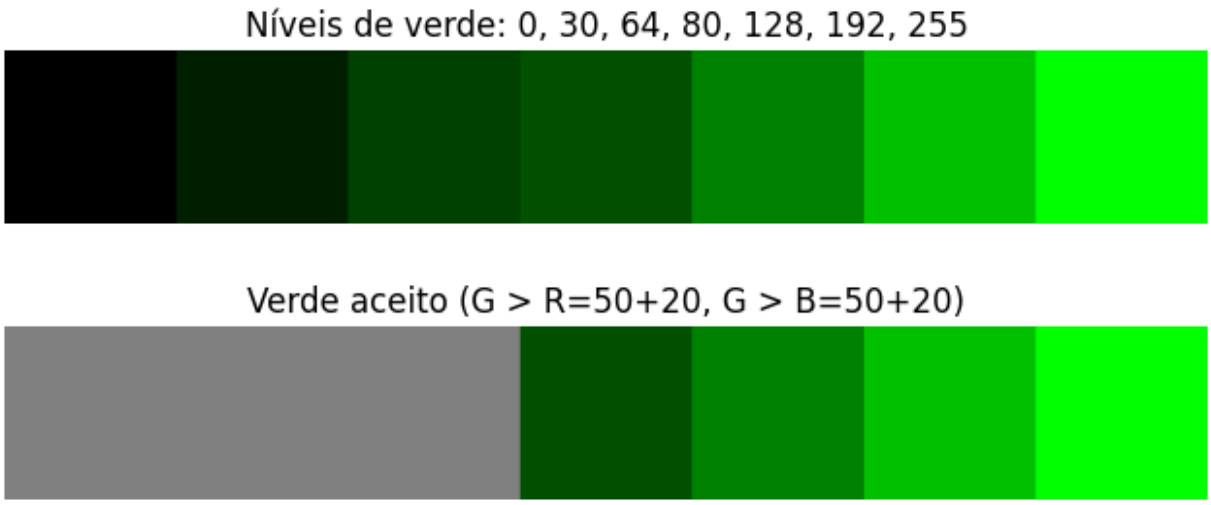
```
# Preencher apenas o canal G (verde) variando de 0 a 255
for i in range(256):
    M[:, i, 1] = i  # canal G = intensidade


# Mostrar a imagem
plt.figure(figsize=(8,2))
plt.imshow(M)
plt.axis("off")
plt.title("Níveis de verde (0 → 255)")
plt.show()

print(M[:,:,2])
```




```
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

```
import numpy as np
import matplotlib.pyplot as plt

# Definir os níveis de verde que queremos mostrar
niveles = [0, 30, 64, 80, 128, 192, 255]

# Criar uma imagem 100x(5*100) pixels, cada bloco com 100px de largura
M = np.zeros((100, 100*len(niveles), 3), dtype=np.uint8)

# Preencher cada bloco com o respectivo nível de verde
for i, g in enumerate(niveles):
    M[:, i*100:(i+1)*100, 1] = g  # canal G recebe intensidade

# Mostrar a imagem
plt.figure(figsize=(8,2))
```

```
plt.imshow(M)
plt.axis("off")
plt.title("Níveis de verde: 0, 30, 64, 80, 128, 192, 255")
plt.show()


# ========================
# 2) Contexto de DOM_MARGIN
# ========================
# Suponha que R = 50 e B = 50
R_ref, B_ref = 50, 50
DOM_MARGIN = 20

# Criar uma nova imagem com os mesmos blocos, mas agora:
# Verde só é válido se G > R_ref + DOM_MARGIN e G > B_ref + DOM_MARGIN
M2 = np.zeros((100, 100*len(niveles), 3), dtype=np.uint8)

for i, g in enumerate(niveles):
    # aplica a regra de dominância
    if (g > R_ref + DOM_MARGIN) and (g > B_ref + DOM_MARGIN):
        # verde aceito (colocamos o valor de G)
        M2[:, i*100:(i+1)*100, 1] = g
    else:
        # não passa pelo critério → deixamos em cinza (pra ver a
exclusão)
        M2[:, i*100:(i+1)*100] = [128,128,128]

plt.figure(figsize=(8,2))
plt.imshow(M2)
plt.axis("off")
plt.title(f"Verde aceito (G > R={R_ref}+{DOM_MARGIN}, G > B={B_ref}+
{DOM_MARGIN})")
plt.show()
```



```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

# ============ 1) Carregar imagens ============
fg = Image.open('/content/kr_gino.jpg').convert('RGB')   # imagem com
fundo verde
bg = Image.open('/content/Header-
Desktop_Floresta_Europeia.jpg').convert('RGB')   # nova cena de fundo

# Redimensionar o background para o mesmo tamanho do foreground
bg = bg.resize(fg.size, resample=Image.BILINEAR)

# Para NumPy (H, W, 3) e uint8
fg = np.array(fg, dtype=np.uint8)
bg = np.array(bg, dtype=np.uint8)

# ============ 2) Parâmetros do chroma key (RGB) ============
# Regras simples de "verde dominante"
G_MIN = 80       # mínimo absoluto de G para considerar "verde"
DOM_MARGIN = 20  # quão acima G deve estar de R e B (em valores
absolutos)
K = 0          # fator de dominância (opção por razão): G > K*R e G >
K*B

R = fg[..., 0].astype(np.int16)
G = fg[..., 1].astype(np.int16)
B = fg[..., 2].astype(np.int16)

# Máscara por diferenças absolutas (robusta e intuitiva):
mask_abs = (G > G_MIN) & (G > R + DOM_MARGIN) & (G > B + DOM_MARGIN)

# (Opcional) Máscara por razão (útil quando luz varia muito):
mask_ratio = (G > G_MIN) & (G > (K * R)) & (G > (K * B))

# Use uma das duas ou a interseção/união dependendo do seu cenário.
# Aqui, vou usar a interseção para ser um pouco mais seletivo:
mask = mask_abs & mask_ratio

# ============ 3) Composição ============
out = fg.copy()
out[mask] = bg[mask]

# ============ 4) Visualização ============
fig, axs = plt.subplots(1, 3, figsize=(12, 4))
axs[0].imshow(fg); axs[0].set_title('Foreground (fundo verde)');
axs[0].axis('off')
axs[1].imshow(mask, cmap='gray'); axs[1].set_title('Máscara de
verde'); axs[1].axis('off')
axs[2].imshow(out); axs[2].set_title('Resultado (chroma key)');
axs[2].axis('off')
```

```
plt.tight_layout()
plt.show()
```


```
# TAREFA

import cv2
import numpy as np
import matplotlib.pyplot as plt
import os

# ====== Caminhos das imagens ======
FG_PATH = "/content/image2.jpg"   # cena com os telões verdes
BG_PATH = "/content/Header-Desktop_Floresta_Europeia.jpg"  # imagem a
exibir nos telões

# ====== Parâmetros do chroma key (RGB) ======
G_MIN      = 80     # mínimo absoluto de G para ser verde
DOM_MARGIN = 50     # G deve superar R e B por pelo menos DOM_MARGIN
K_RATIO    = 0      # se <=0, ignora a regra de razão
```