Primeira Avaliação Parcial: Avaliação Individual

Neste módulo trabalhamos com os recursos da **NumPy** para processar dados de tipo único de forma eficiente. Um exemplo deste tipo de dados são as imagens que podemos ser armazenadas como *ndarrays* **NumPy**. O código a seguir gera uma imagem colorida, de 1000 por 1000 *pixels*, armazenada no formato **RGB**. Isto significa que, de cada *pixel* armazenamos a intensidade da cor dos canais **R** (*red* ou vermelho), **G** (green ou verde) e **B** (blue o azul). A intensidade da cor é dada por um inteiro de 8 *bits* sem sinal.

In [ ]:
```
# importado a biblioteca numpy
import numpy as np
print(np.__version__)
```

In [ ]:
```
# Definindo a altura e a largura da imagem que represnearão:
height = 1000  # altura ou o número de linhas da imagem
width = 1000   # largura ou o número de colunas da imagem
```

In [ ]:
```
# gerando uma imagem aleatória de 1000 x 1000 pixels
random_image = np.random.randint(0, 256, size=(height, width, 3), d
print("Veja o forma da matriz criada: {}".format(random_image.shape
print("Veja como fica a cor de um pixel: {} ...".format(random_imag
print("Ou a componente vermelha dos 3 primeiro pixels da segunda li
print("Ou a componente verde dos 3 últimos pixels da segunda coluna
```

Exercício 1

Crie uma função para a qual passamos como parâmetros a altura e a largura e retorna uma imagem, gerada de forma aleatória, em formato **RGB**, com estas dimensões. A imagem deve ser representada utilizando um *ndarray* como o do exemplo anterior. Se a altura ou a largura não forme passadas como a imagem deve ser gerada com 720 linhas e 1280 colunas.

In [ ]:
```
# Implementar aqui
def geradorImagens(parametros):
    pass
```

In [ ]:
```
#teste
print(geradorImagens().shape) # deve retornar (1280, 720, 3)
print(geradorImagens(10).shape) # deve retornar (10, 720, 3)
print(geradorImagens(10,10).shape) # deve retornar (10, 10, 3)
print(geradorImagens(5,5)) # deve retornar uma matriz 5x5x3
```

Para simplificar o tratamento durante o processamento das imagens, frequentemente elas são convertidas e imagens em tons de cinza. Estas imagens contem, para cada pixel, uma única intensidade de cor representada como um inteiro de 8 *bits* sem sinal. Para converter uma imagem **RGB** em uma imagem em tons de cinza podemos utilizar dois métodos diferentes.

1. Calculamos a intensidade da cor de cada *pixel* como a média das intensidades dos canais *_RGB*
2. Os valores **RGB** são convertidos para tons de cinza usando a fórmula **NTSC**: $0.299 \times \text{Vermelho} + 0.587 \times \text{Verde} + 0.114 \times \text{Azul}$. Esta fórmula representa de perto a percepção relativa da pessoa média sobre o brilho da luz vermelha, verde e azul.

Exercício 2

Crie uma função que recebe uma imagem **RGB** e converte a mesma em tons de cinza. A função deve retornar então a nova imagem. Esta função.

- Deve verificar se o argumento de entrada é de fato uma imagem **RGB** com base no tipo e no atributo `shape` do *ndarray*. Casso o argumento esteja errado deve-se lançar uma exceção.
- A função recebe como parâmetro de entrada a forma de conversão a ser utilizada: media dos canais ou **NTSC**. Por padrão o algoritmo de utilizado deve ser o **NTSC**.
- A função deve utilizas as *ufunc* e operações definidas na *_NumPy* de forma a minimizar o uso de laços no processamento dos *ndarrays*.

In [ ]:
```
#implementar aqui
def rgb2gray(parametros):
    pass
```

Exercício 3 (Opcional, valendo ponto extra)

Refaça a implementação da função anterior utilizando estruturas de repetição e implemente um teste de demonstre, usando uma imagem **RGB** gerada de forma aleatória de 10.000 linhas por 10.000 colunas, o ganho de desempenho quando utilizamos as *ufunc* e os operadores de **NumPy**.

Respostas

Faça suas implementações neste notebook e envie mesmo via **Moodle** até o final do prazo.