

# ICE-E Módulo 6: Imagens e Gráficos em Python, Método de Monte-Carlo

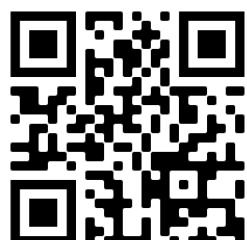
Nuno C. Marques

DI – NOVA FCT

Ensino a distância/Ensino remoto de emergência

*Apoio multimedia para uso exclusivo em ICE-E 2021.*

*É proibida gravar aulas online.*



<http://bit.ly/ICE-E-2021>

# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

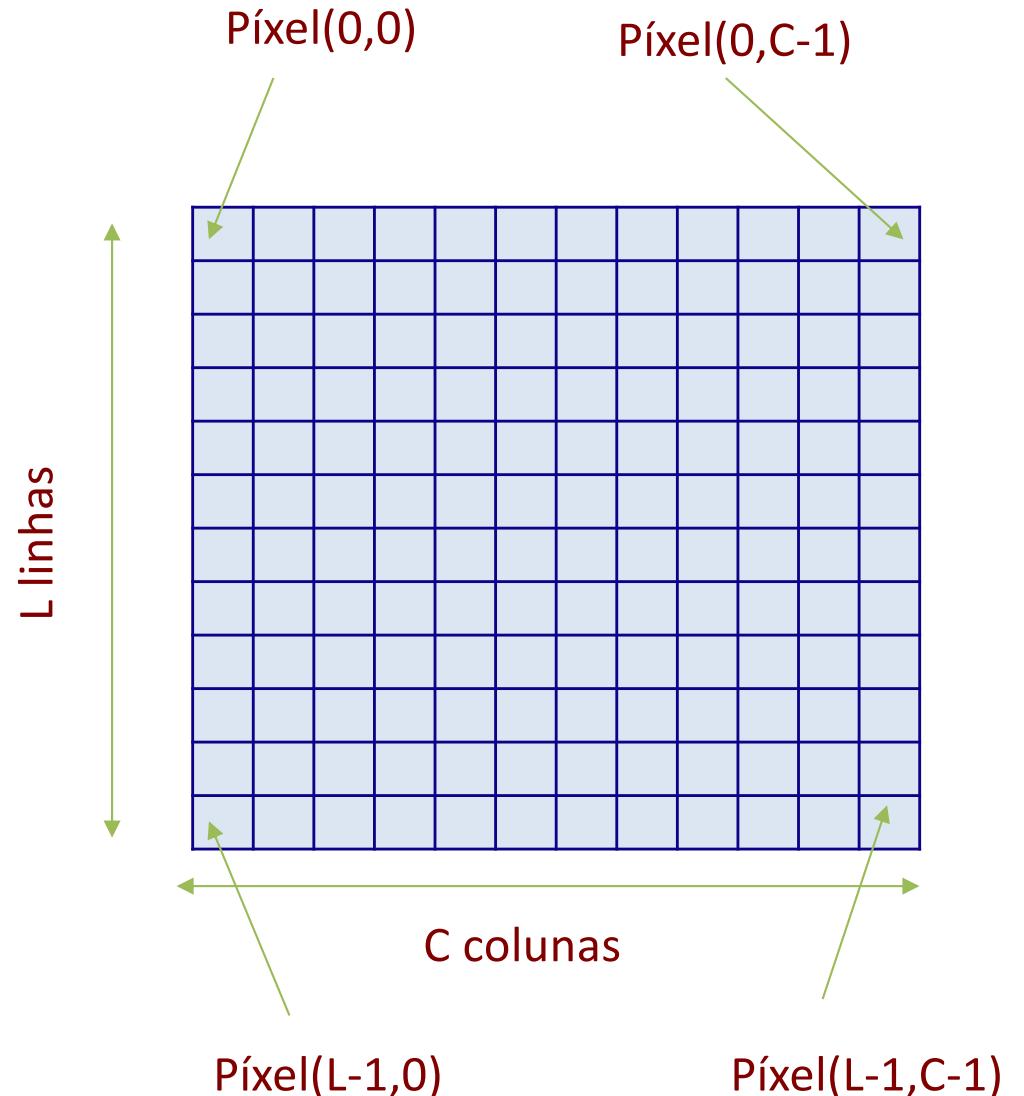
# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

# Constituição do ecrã

- Ecrã constituído por uma matriz de píxeis (*picture elements*).
- Organizado em **L** linhas (por exemplo, 768).
- Cada linha tem **C** colunas (por exemplo, 1024).



# Imagen é composta por píxeis

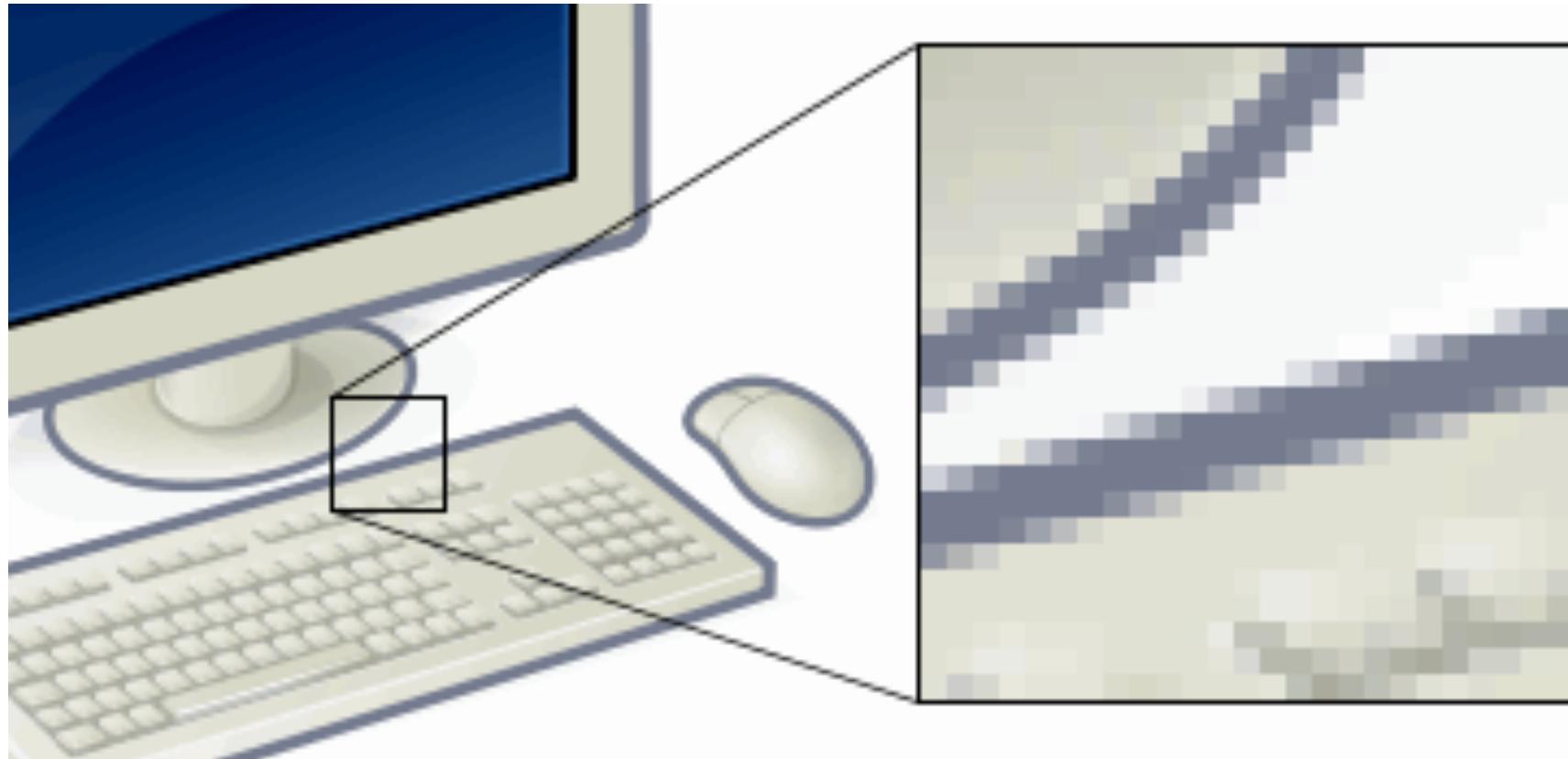
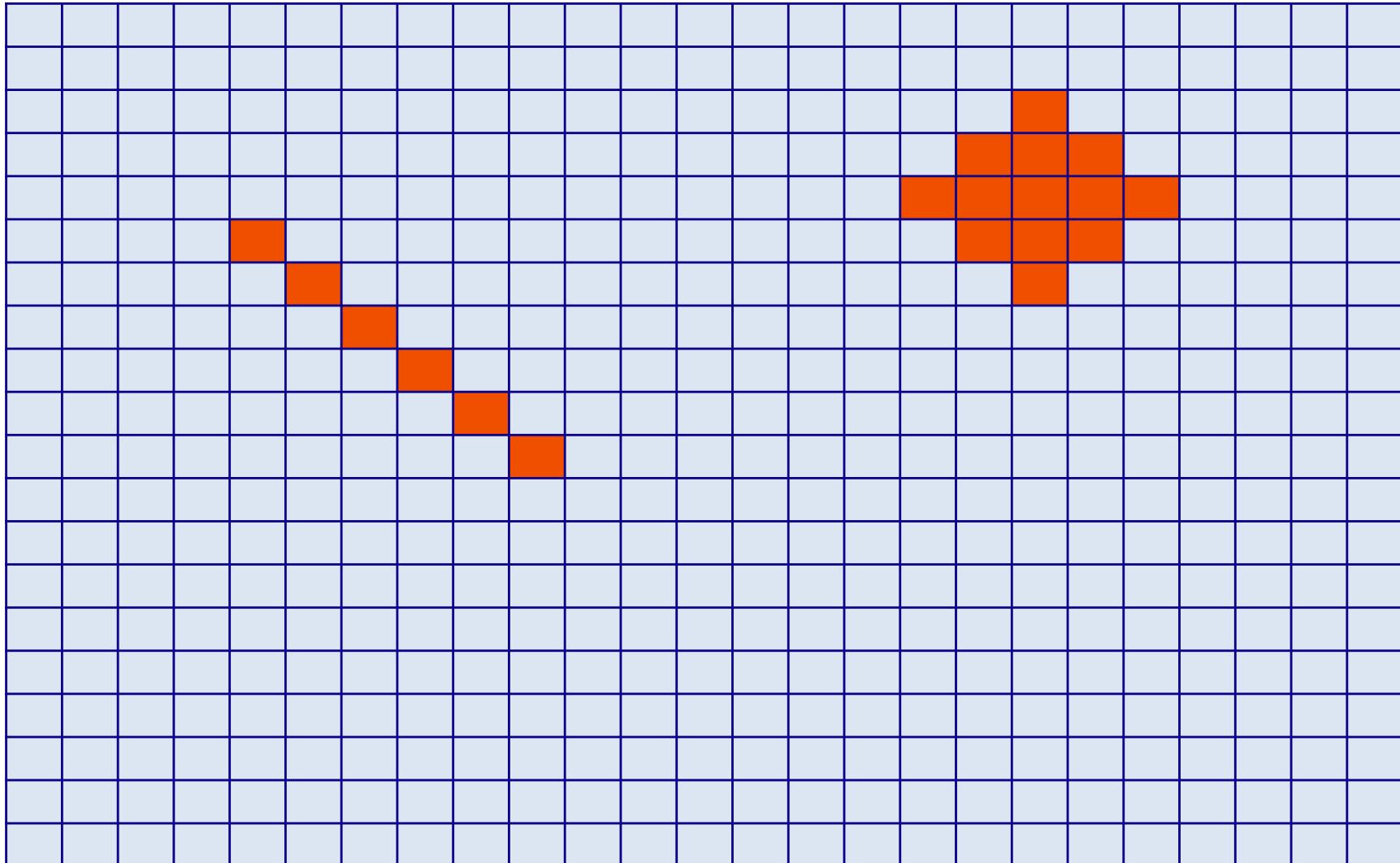


Imagen ampliada para mostrar os píxeis

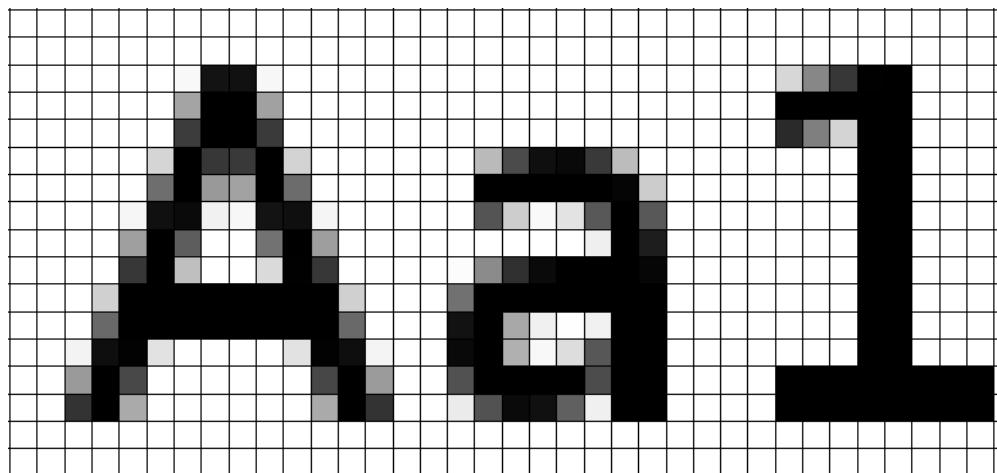
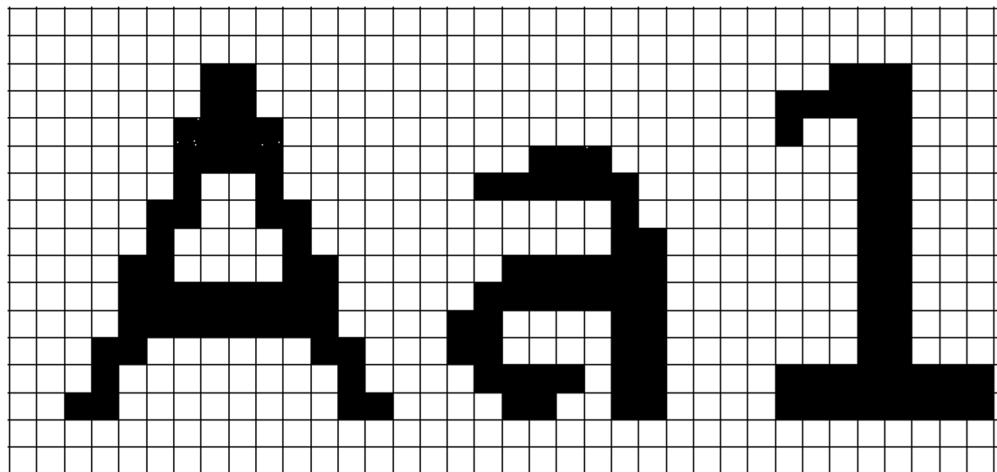
Image from: wikipedia, author: ed g2s

# Píxeis formam figuras geométricas

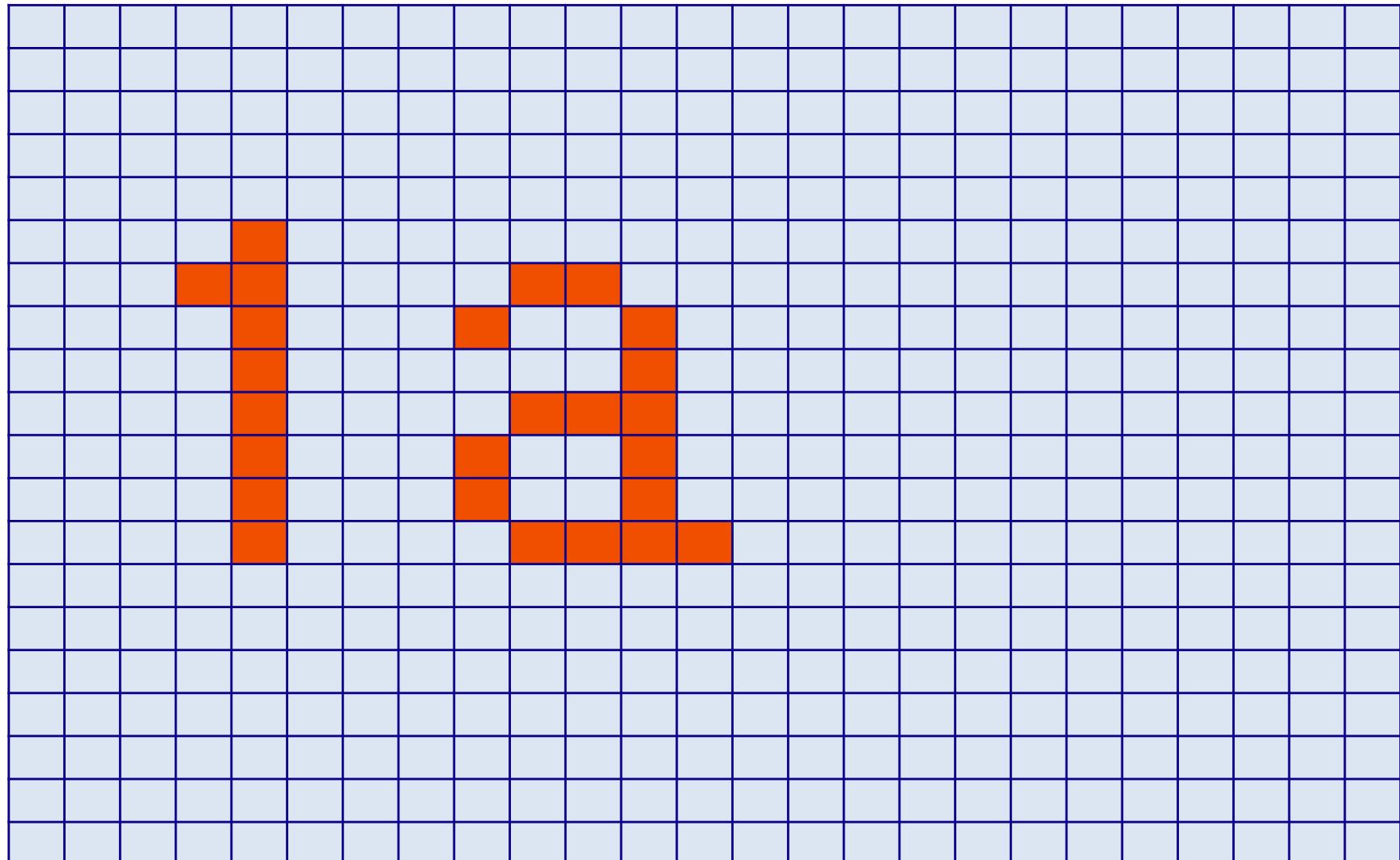


# Píxeis formam letras e números [2]

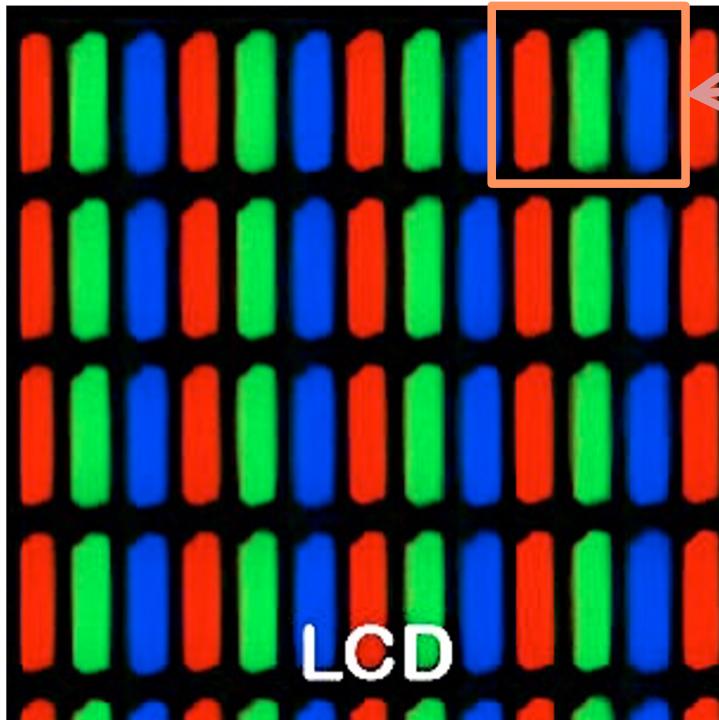
- 
- Uso de níveis intermédios para melhorar a imagem



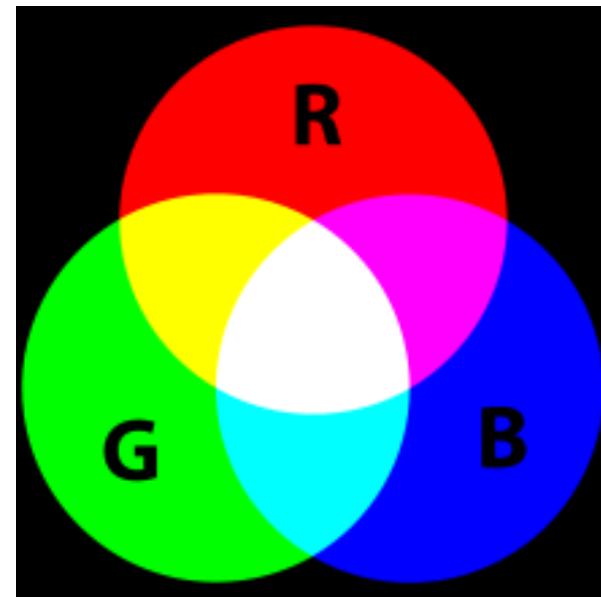
# Píxeis formam letras e números [1]



# Representação da cor [1]



- Cor: cada pixel formado por 3 componentes primários.
- RGB:
  - *Red* – vermelho;
  - *Green* – verde;
  - *Blue* – azul.

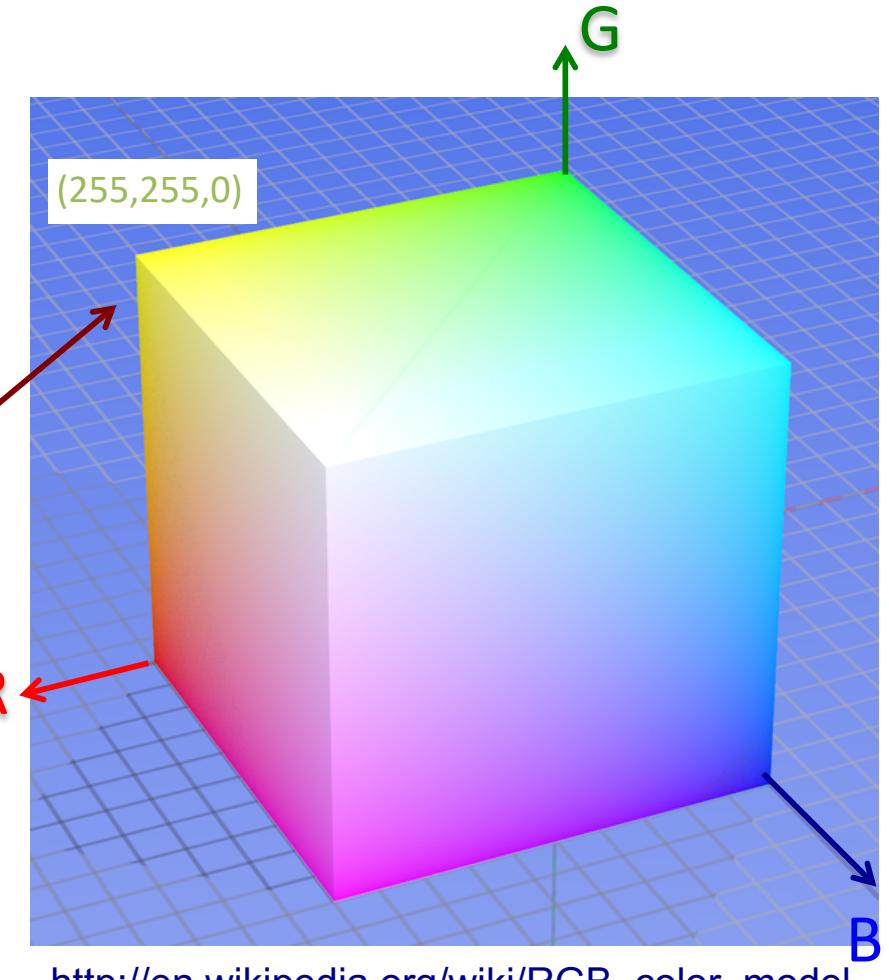
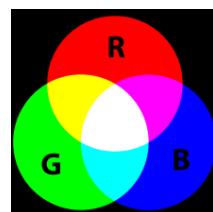


# Representação da cor [2]

- Cor de cada píxel codificada, usualmente, em 3 bytes (24 bits); 1 byte (0-255) por componente.
  - (red, green, blue)
- Os três números representam a contribuição de cada uma das cores primárias.

## Exemplo:

- O terno (255, 255, 0) resulta num brilhante amarelo.



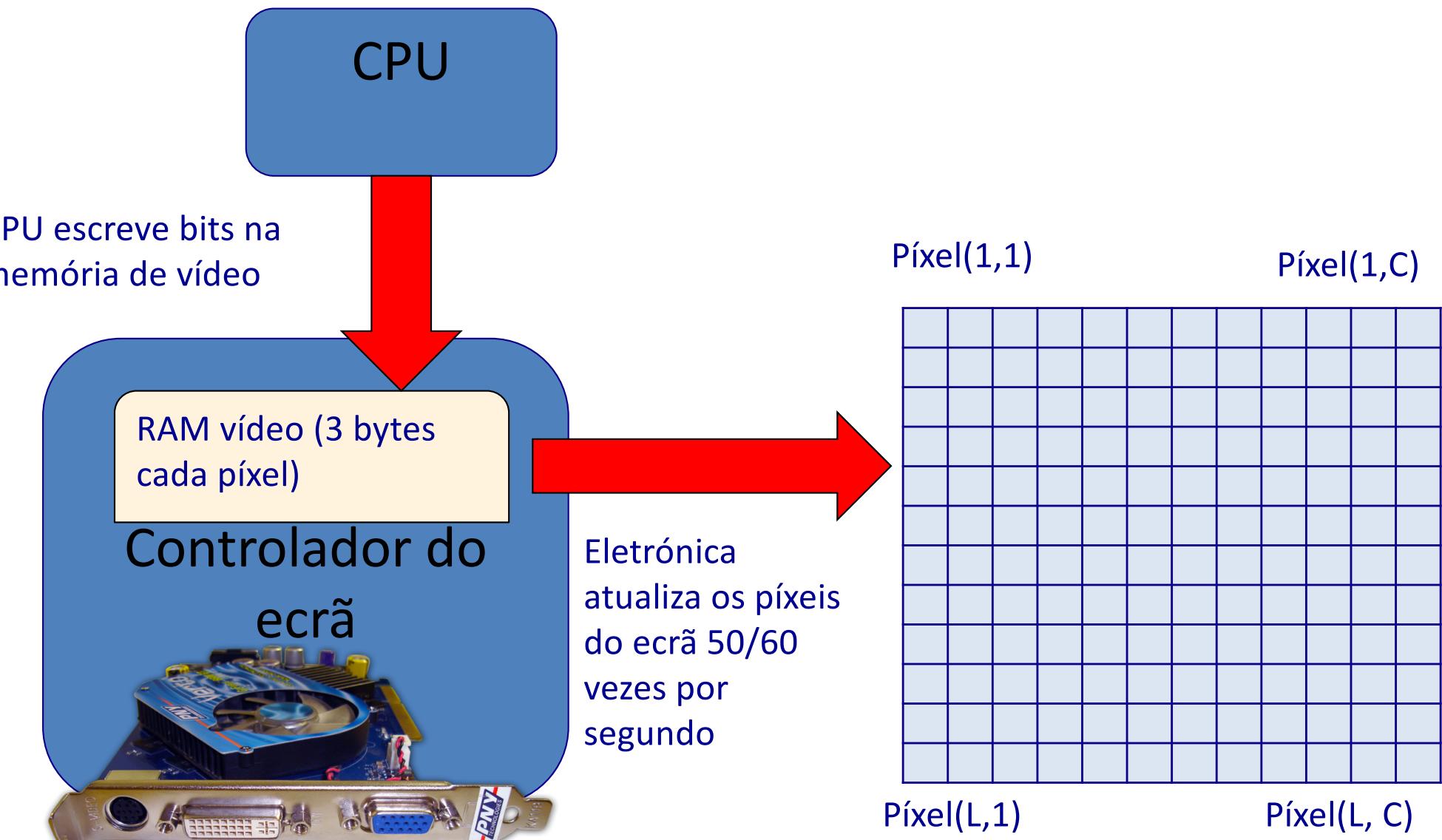
Para testar cores: [http://www.rapidtables.com/web/color/RGB\\_Color.htm](http://www.rapidtables.com/web/color/RGB_Color.htm)

# Representação da cor – Exemplos

Cor representada por 3 valores de 8 bits (True Color)

RGB Value			Color
Red	Green	Blue	
0	0	0	black
255	255	255	white
255	255	0	yellow
255	130	255	pink
146	81	0	brown
157	95	82	purple
140	0	0	maroon

# Manipulação do ecrã (simplificado)



# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

- A biblioteca *matplotlib* é uma das bibliotecas existentes em Python que nos permite criar facilmente diferentes tipos de gráficos.
  - A documentação online desta biblioteca está disponível em <https://matplotlib.org/>
- Para poder aceder a uma biblioteca, deve-se importá-la antes, com a instrução **import**.
- Além da forma simples de import, já vista anteriormente, é possível indicar um nome mais simples a usar quando se acedem às funções da biblioteca

```
In : import math as m  
In : m.pi  
Out: 3.141592653589793
```

- No caso da biblioteca *matplotlib*, vamos usar o seguintes import que acede ao módulo que contém as funções de criação de gráficos como **plt**.

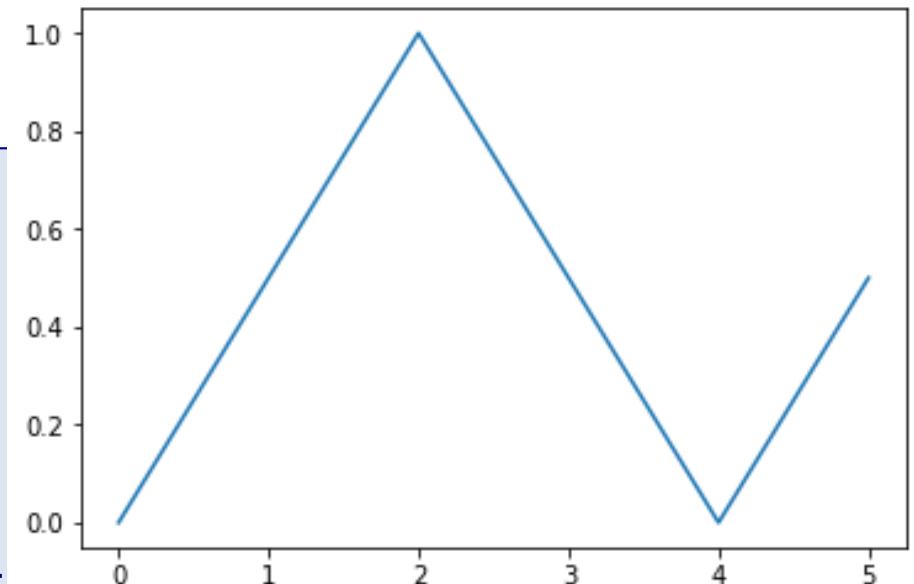
```
In : import matplotlib as plt
```

- O Spyder (versão 4) apresenta por omissão os gráficos num separador próprio. Este modo não permite controlar alguma da funcionalidade dos gráficos.
- Há uma opção que possibilita a apresentação dos gráficos numa janela independente.
- Para apagar o que já está na janela, usa-se o comando `plt.clf()`.

# Gráficos de Funções [1]

- Para se desenhar o gráfico de uma função dar os seguintes passos:
  1. Preencher um vetor **x** com os valores das abcissas.
  2. Preencher um vetor **y** com os valores das ordenadas.
  3. Usar a função **plt.plot(x, y, fmt)** para traçar o gráfico, onde **fmt** especifica o formato a usar (estilo, marcador e cor)
    - **fmt** é opcional

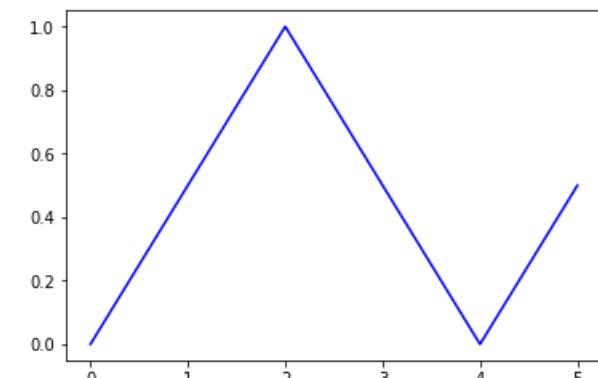
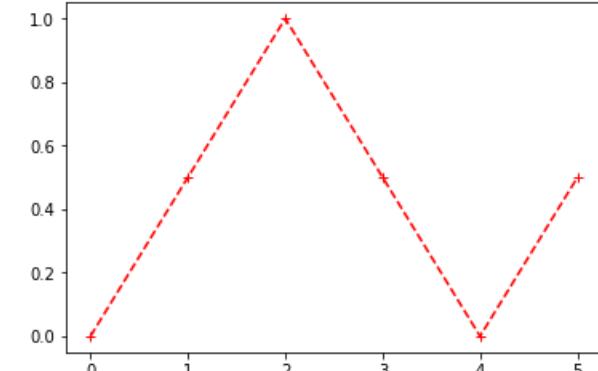
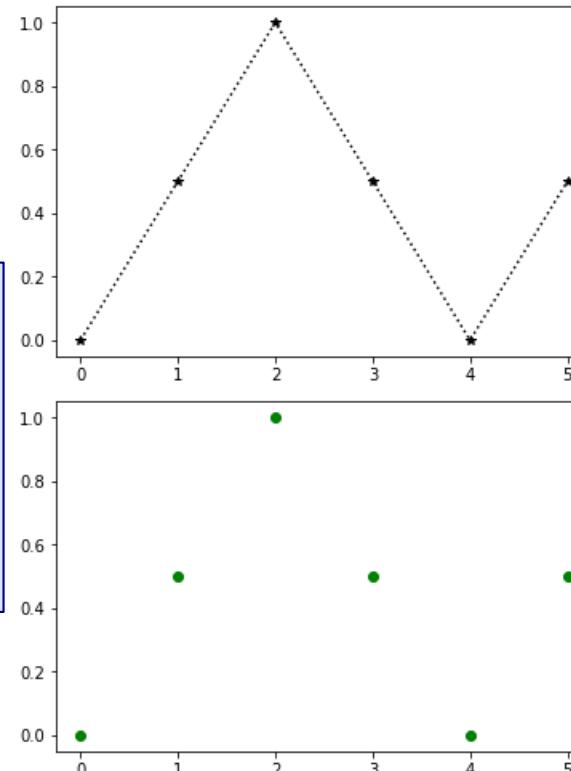
```
In : import matplotlib.pyplot as plt  
In : X = [0, 1, 2, 3, 4, 5]  
In : Y = [0, 0.5, 1, 0.5, 0, 0.5]  
In : plt.plot(X, Y)  
Out: [<matplotlib.lines.Line2D at 0x1
```



# Gráficos de Funções [2]

- O formato é uma string com 3 partes (opcionais): cor, marcador e estilo.
  - Algumas cores aceitas: 'b'-blue, 'g'-green, 'r'-red, 'y'-yellow, 'k'-black
  - Marcadores: '.'-ponto, 'o'-círculo, '+'-mais, 'x'-vezes, '\*'-estrela
  - Estilos: '-' -a cheio, '--' -a tracejado, ':' -a ponteado, '-.' -a traço ponto
    - ver **help(plt.plot)**.

```
In : plt.plot(X, Y, 'k*:')  
In : plt.plot(X, Y, 'r+--')  
In : plt.plot(X, Y, 'go')  
In : plt.plot(X, Y, 'b-')
```



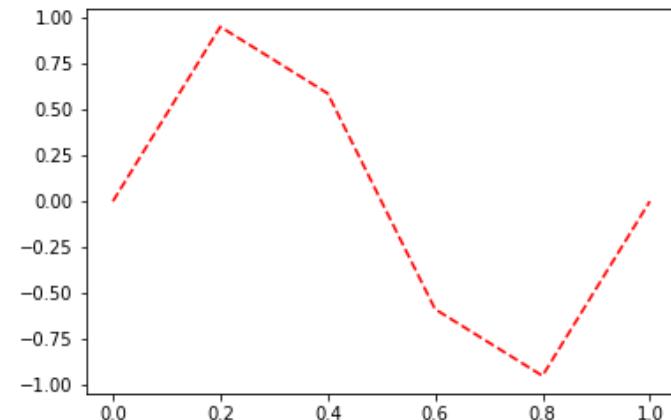
- Para desenhar um gráfico de uma função devem definir-se dois vetores, X e Y, com as abcissas e ordenadas da função na “zona de interesse”.
- Para o vetor X é conveniente usar um número de pontos suficiente para que a imagem apareça bem definida.
- Naturalmente um vetor com n valores, de 0 a n-1, pode ser definido através de um range.
- A criação do vetor de abcissas da zona de interesse com esse número de pontos pode ser feita por compreensão.

```
In : P = list(range(0,6)) # lista de inteiros 0 a 5
In : P
Out: [0, 1, 2, 3, 4, 5]
In : X = [0.2*v for v in P]
In : X
Out: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]
```

# Gráficos de Funções [4]

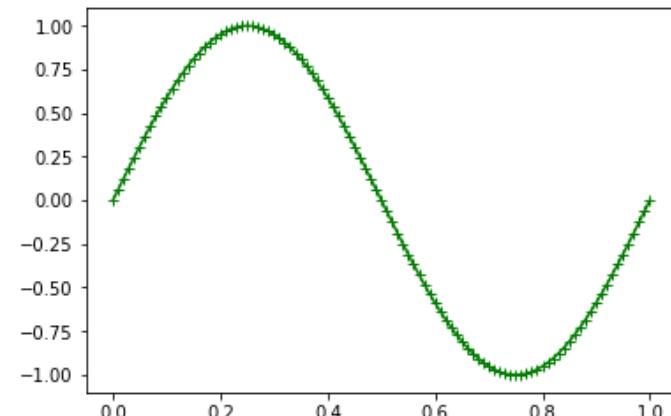
- Uma vez definido o vetor X, o vetor  $Y = f(X)$  pode igualmente ser definido por compreensão.

```
In : X  
Out: [0.0, 0.2, 0.4, 0.6, 0.8, 1.0]  
  
In : import math as m  
  
In : Y = [m.sin(x*2*m.pi) for x in X]  
  
In : plt.plot(X,Y, 'r--')
```



- Pode ser usada uma melhor definição se forem usados mais pontos.

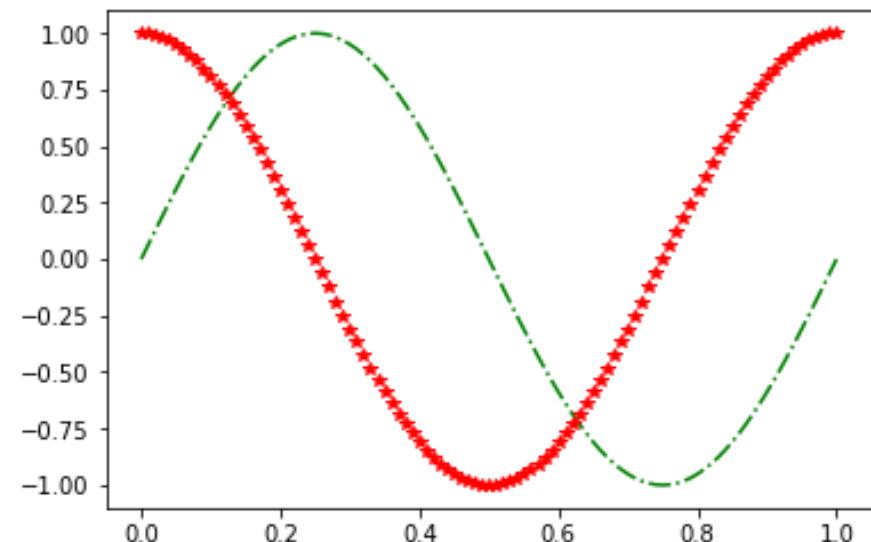
```
In : P = list(range(0,101))  
In : X = [a/100.0 for a in P]  
  
In : Y = [m.sin(x*2*m.pi) for x in X]  
  
In : plt.plot(X,Y, 'g+-')
```



# Gráficos Multi-Funções [1]

- Podem ser feitos gráficos com mais de uma função, se as instruções de plot forem feitas “sem interrupção”.

```
In : P = list(range(0,101))
In : X = [a/100.0 for a in P]
In : S = [m.sin(x*2*m.pi) for x in X]
In : C = [m.cos(x*2*m.pi) for x in X]
In : plt.plot(X,S,'g-.')
....: plt.plot(X,C,'r*')
....: plt.show(). # opcional
```

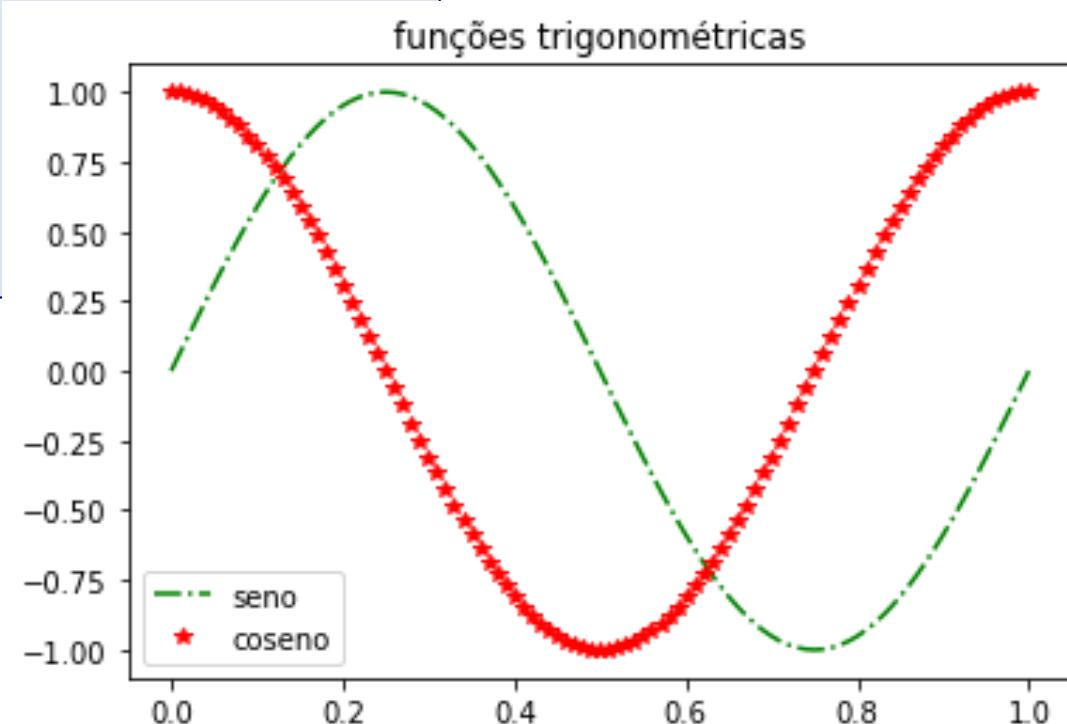


- O comando plt.show() é opcional no Ipython.

# Gráficos Multi-Funções [2]

- Outros comandos de plot permitem definir o título, e legenda do gráfico

```
In : plt.title['funções trigonométricas']
....: plt.plot(X,S,'g-.')
....: plt.plot(X,C,'r*')
....: plt.legend['seno', 'coseno']
....: plt.show()
```



- Para guardar a figura num ficheiro .png, pode usar-se o método

```
plt.savefig(FicheiroPNG)
```

# Sumário

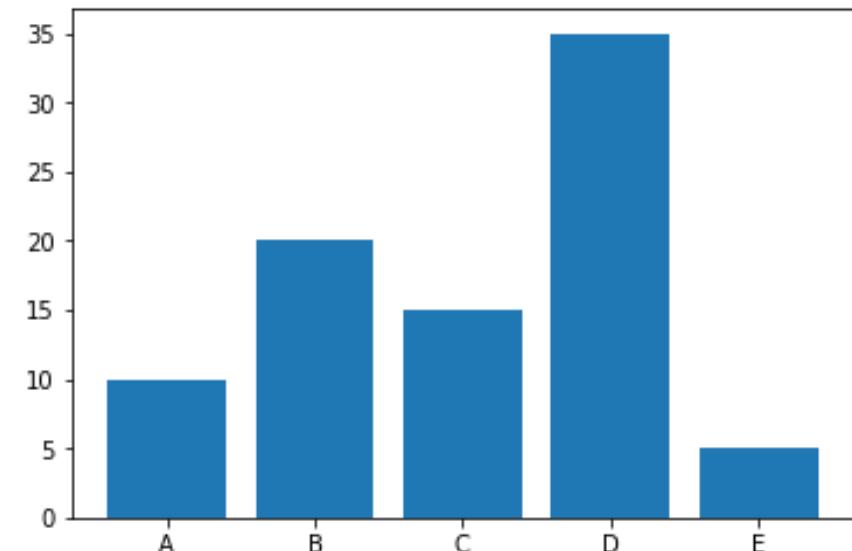
---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

# Gráficos de barras [1]

- Para se desenhar um gráfico de barras usar o seguinte procedimento:
  1. Preencher um vetor **V** com os valores de cada barra..
  2. Preencher um vetor **X** com a “legenda” de cada barra.
  3. Usar a função **plt.bar(X, V)** para traçar o gráfico

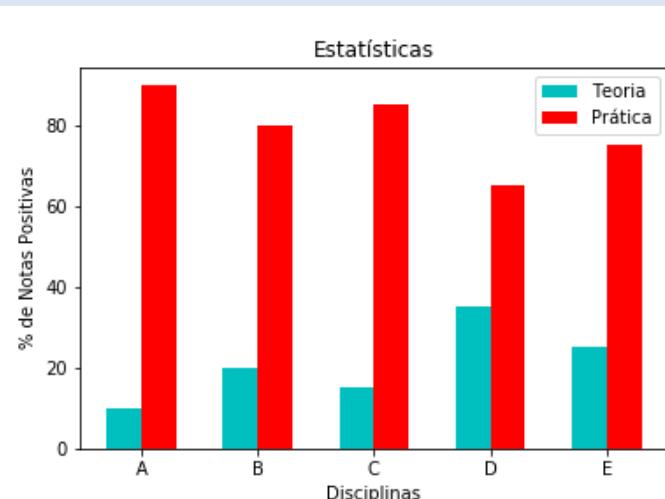
```
In : X = ["A", "B", "C", "D", "E"]  
In : v = [10, 20, 15, 35, 5]  
In : plt.bar(X, v)  
....: plt.show()
```



# Gráficos de barras [2]

- Os gráficos podem ter várias dimensões e ser formatados (tamanho, posição e cor das barras, e ainda título, legendas nomes dos eixos).

```
In : V1 = [10, 20, 15, 35, 5]    # alturas das barras 1
In : P = range(0,len(V1))        # gama de valores a considerar
In : X1 = [0.85 + v for v in P] # posição das barras 1 no eixo X
In : V2 = [90, 80, 85, 65, 95]
In : X2 = [1.15 + v for v in P]
In : plt.bar(X1,V1, width = 0.3, color = 'c') # 0.3 = 0.15 esquerda + 0.15 direita
....: plt.bar(X2,V2, width = 0.3, color = 'r')
....: plt.xticks( P , ['A','B','C','D','E']) # nomes das barras
....: plt.title('Estatísticas')
....: plt.xlabel('Disciplinas')
....: plt.legend(['Teoria','Prática'])
....: plt.ylabel('% de Notas Positivas')
....: plt.show()
```



# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

# Gráficos de imagens [1]

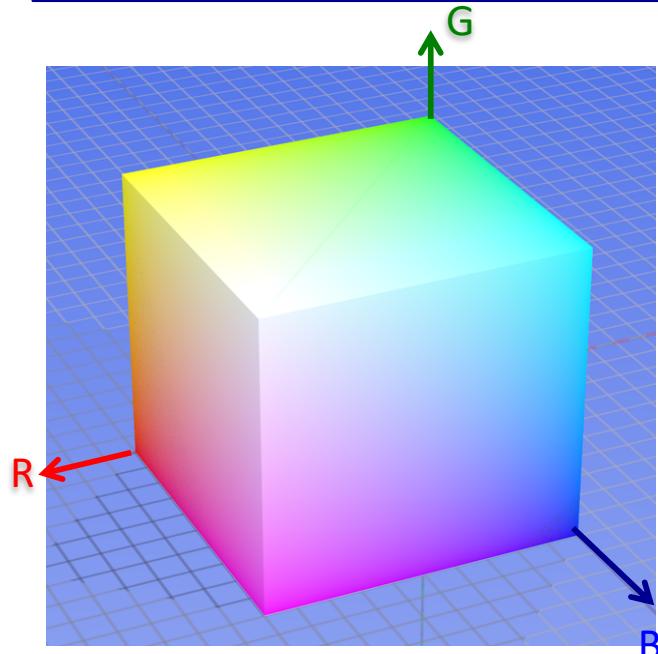
Para se desenhar um gráfico imagem usar o seguinte procedimento:

1. Definir uma grelha (rectangular) correspondente à imagem.
  - A grelha é especificada por uma matriz **M**
2. A cada célula é atribuída uma cor definida pelas suas componentes [R,G,B]
  - Por exemplo, a cor amarela corresponde a [1,1,0] (cores vermelha e verde saturadas , nenhuma componente de azul).
3. Criar um objeto **cm**, com a função **ListedColorMap**, da biblioteca **matplotlib.colors** que irá corresponder a um dicionário de n cores.
  - O objeto é especificado com uma matriz de n linhas (uma para cada cor), com 3 colunas correspondentes à contribuição das cores primárias [R,G,B].
4. Preencher a matriz **M** com um inteiro **c**, de 0 a n-1 para cada elemento
  - Cada elemento da grelha ficará com a cor indexada pelo valor **c**.
5. Desenhar a imagem usando a função **plt.imshow(M,cmap=cm)**

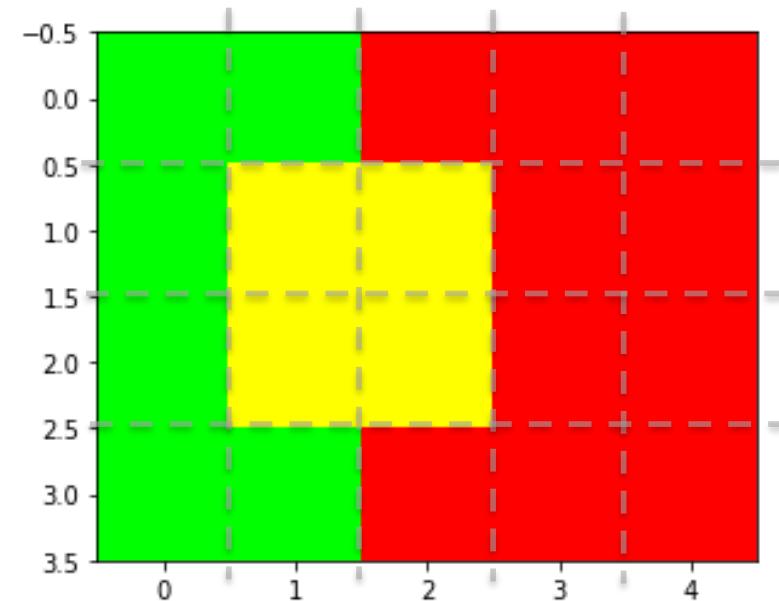
# Gráficos de imagens [2]

Exemplo:

```
In : from matplotlib.colors import ListedColormap  
In : cores = [[0,1,0],[1,0,0],[1,1,0]] # [green, red, yellow]  
In : imagem = [[0,0,1,1,1],[0,2,2,1,1],[0,2,2,1,1],[0,0,1,1,1]]  
In : my_cmap = ListedColormap(cores)  
In : plt.imshow(imagem,cmap = my_cmap)
```



0	0	1	1	1
0	2	2	1	1
0	2	2	1	1
0	0	1	1	1



# Sumário

---

- Representação de imagens num ecrã.
  - Píxeis. Sistema RGB.
- Gráficos em Python.
  - Biblioteca matplotlib.
  - Gráficos de pontos e de linhas.
  - Gráficos de barras.
  - Gráficos de imagens.
- Cálculo de áreas pelo método de Monte Carlo.
  - Números pseudo-aleatórios.

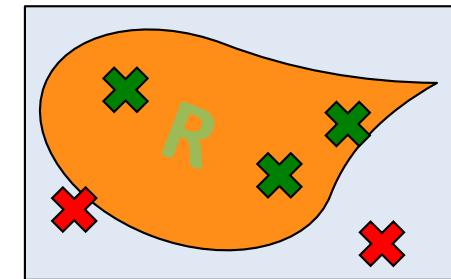
- 
- Métodos de Monte Carlo:
    - Métodos usados para conceber algoritmos que recorrem a números (pseudo-)aleatórios.
  - Números pseudo-aleatórios:
    - Uma sequência de números pseudo-aleatórios tem uma aparência aleatória (aparentemente, sem padrão), mas é previsível e reproduzível.
  - Algumas aplicações:
    - Simulações de fenômenos físicos, químicos, biológicos, bioquímicos, etc., jogos, cálculo de áreas.

Método para calcular a área  $A_R$  da região  $R$ :

1. Definir uma região  $S$  tal que:
  - a)  $S$  contém  $R$ ;
  - b) a área de  $S$  ( $A_S$ ) é conhecida (e fácil de calcular).
2. Gerar  $n$  pontos pseudo-aleatórios de  $S$  e contar o número  $k$  de pontos que pertencem a  $R$ .
3. O valor estimado para  $A_R$  é proporcional ao número de pontos que pertencem a  $R$ , pelo que se pode estimar:

$$A_R = (k / n) \times A_S.$$

- Nota: Os resultados são melhores quando:
  - $S$  é uma região contendo  $R$ , tão “estritamente” quanto possível; e
  - $n$  é grande.

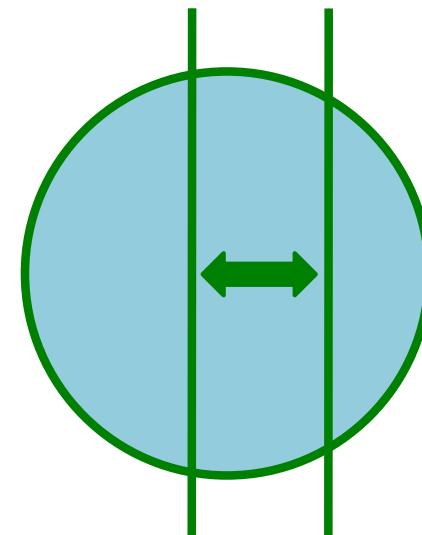


- A biblioteca `random` disponibiliza funções para gerar números pseudo-aleatórios
- A função `random` retorna números **reais** pseudo-aleatórios uniformemente distribuídos no intervalo  $]0, 1[$ .
- Para obter um número real entre:
  - $]0, 1[$  `random.random()`
  - $]0, b[$  (com  $0 < b$ ) `b * random.random()`
  - $]a, b[$  (com  $a < b$ ) `a + (b - a) * random.random()`
- A função `random.randint(a,b)` retorna números **inteiros** pseudo-aleatórios uniformemente distribuídos entre  $a$  e  $b$  (inclusive).

# Problema da fatia de um círculo

Problema:

- Faça um programa que, dado o raio de um círculo e duas abscissas, **estima** a área da respetiva fatia vertical do círculo e **representa** graficamente a estimativa.
- **Sugestão:** use o método de Monte Carlo para efetuar a estimativa.

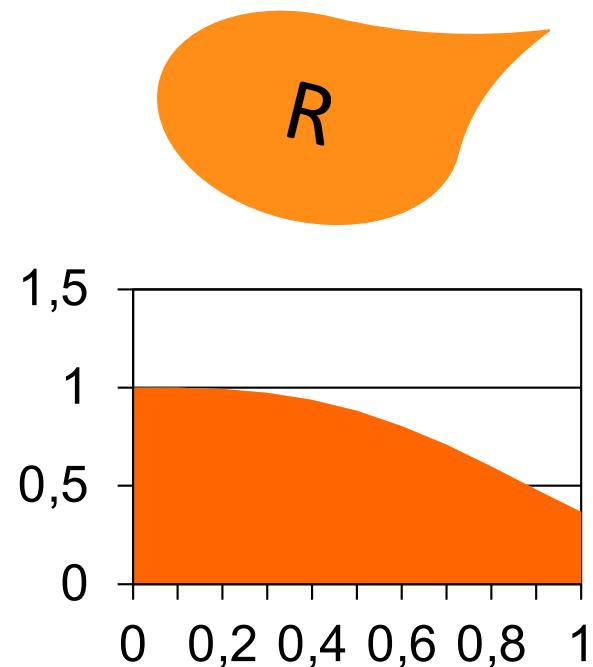


Algumas aplicações do cálculo de áreas:

- Estimar o valor de um número irracional (p.e.  $\pi$ ).
- Estimar a área de uma região definida por várias funções ou por funções com muitas variáveis.
- Estimar o valor de um integral.

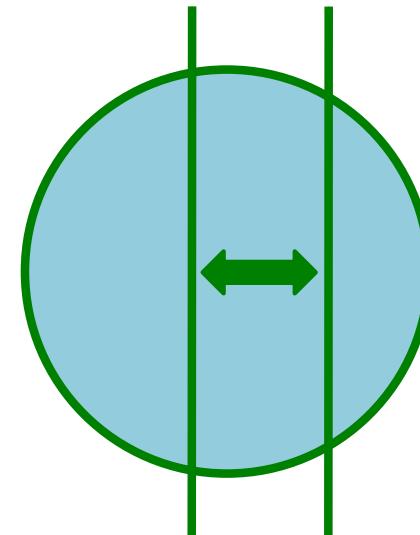
Exemplo:

$$\int_0^1 e^{-x^3} dx$$



## 1. Compreender totalmente o problema.

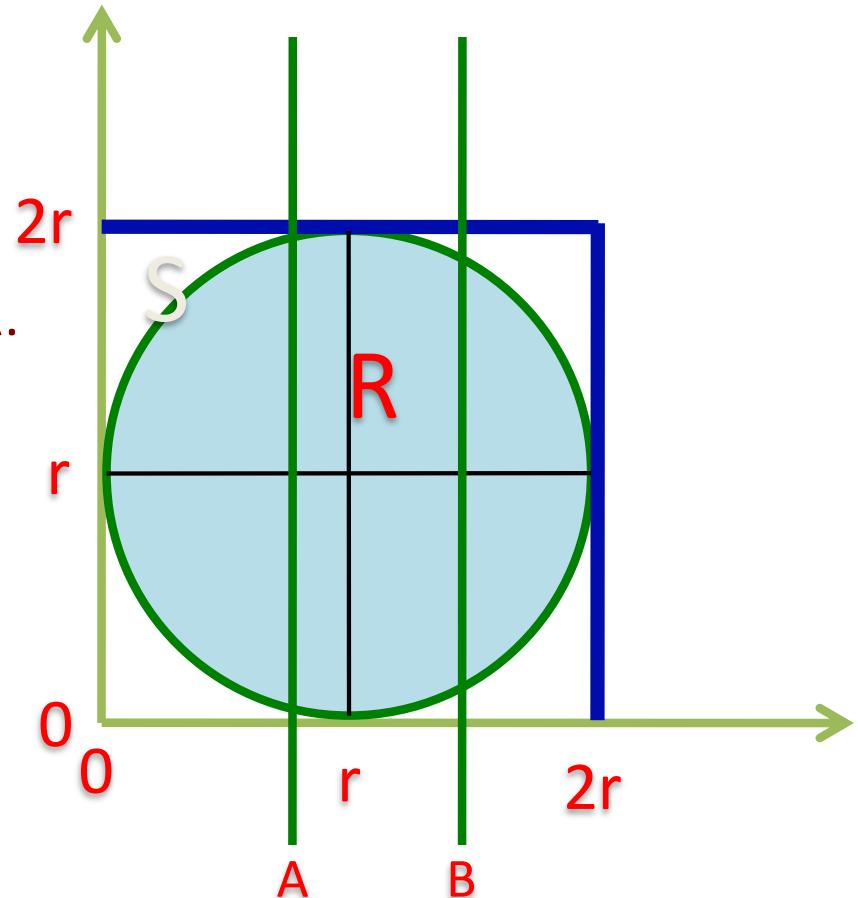
- Pretende-se estimar a área da fatia vertical do círculo usando o método de Monte Carlo.
- Pretende-se uma representação gráfica da estimativa.



## Algoritmo:

- A região  $S$  é  $[0, 2r] \times [0, 2r]$ .
- A área de  $S$  é  $(2r)^2$ .
- Geram-se  $n$  pontos uniformemente distribuídos em  $S$ .
- Contam-se os  $k$  pontos pertencendo a  $R$ .
- O valor estimado para a área de  $R$  é  $(k / n) \times (2r)^2$ .
- Como se contam os pontos?
  - Geram-se  $n$  pontos  $x, y$  tais que  
 $x \geq 0$  e  $x \leq 2r$   
 $y \geq 0$  e  $y \leq 2r$
  - Conta-se os pontos em que

$$x \geq 0 \text{ e } x \leq 2r, \text{ e} \\ (x-r)^2 + (y-r)^2 \leq r^2$$



---

## 2. Caracterizar o problema.

- **Problema:** Estimativa da área de uma fatia vertical de um círculo com representação gráfica.

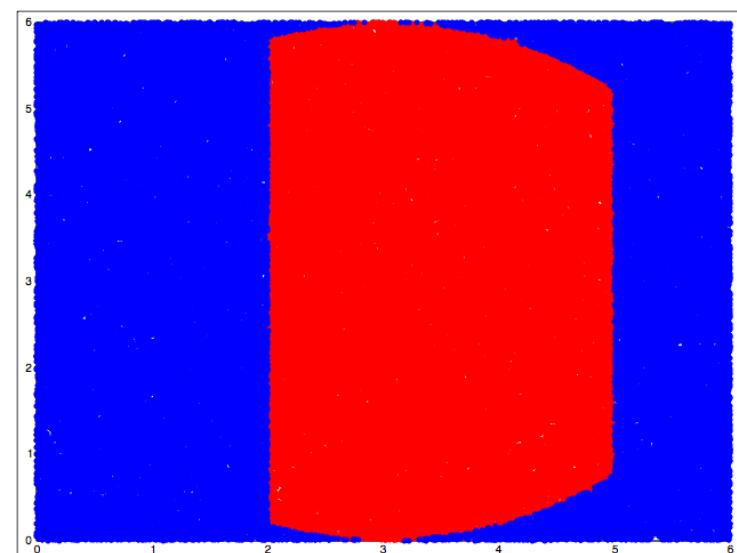
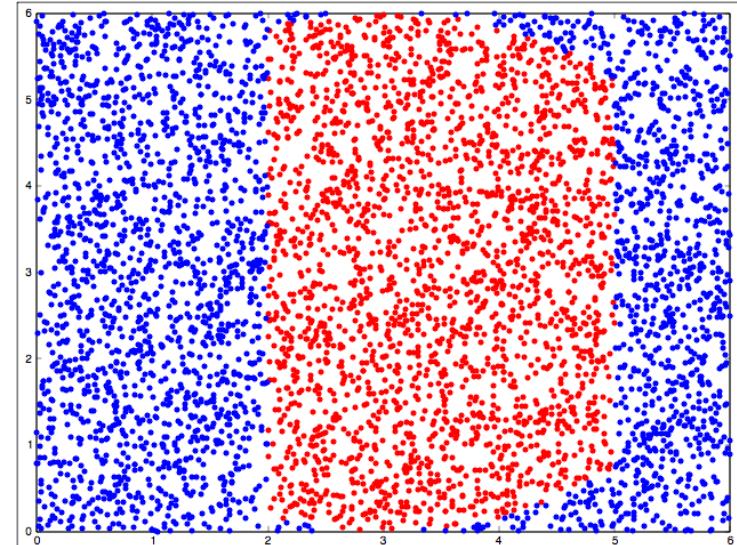
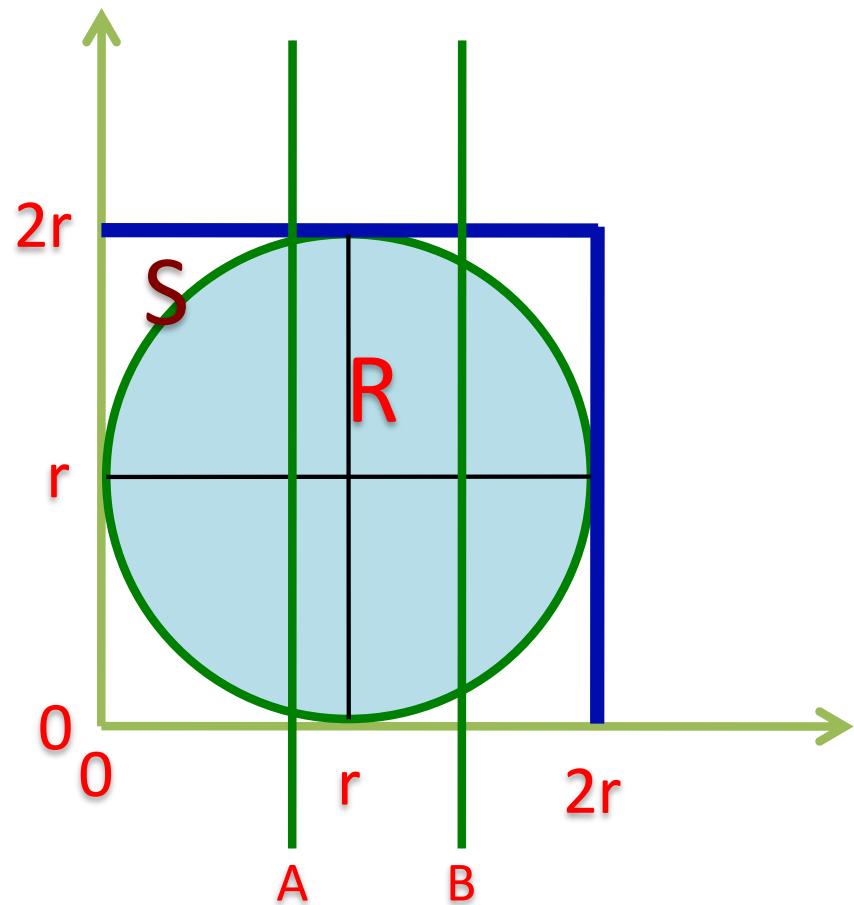
- **Entrada:** (real) raio, (real) xEsq, (real) xDir, (inteiro) número de pontos a gerar.

- **Saída:** (real) estimativa da área da fatia do círculo.  
Adicionalmente é gerado um gráfico.

## 3. Generalizar o problema (sempre que for possível).

- Não vamos generalizar este problema.

# Representação gráfica



- 
4. Desenhar o algoritmo para resolver o problema.
    - a) Conceber o algoritmo, decompondo o problema em sub-problemas.
      - Como estimar a área e representar a estimativa?
        - Aplicar o método de Monte Carlo,
          - ★ computando uma estimativa da área; e
          - ★ devolvendo os pontos gerados que ficam dentro e fora da fatia.
        - Desenhar o gráfico com os pontos gerados dentro e fora da fatia.

---

b) Identificar, caracterizar e generalizar cada sub-problema.

- **Problema:** Estimativa da área de uma fatia vertical de um círculo.
- **Entrada:** (real) raio, (real) xEsq, (real) xDir, (inteiro) número de pontos a gerar.
- **Saída:** (real) estimativa da área da fatia do círculo, (**matriz**) pontos gerados dentro da fatia, (**matriz**) pontos gerados fora da fatia.

**Nota:** A função retorna um tuplo, constituído por um real, e duas matrizes de reais.

---

b) Identificar, caracterizar e generalizar cada sub-problema.

- **Problema:** Desenho de dois conjuntos de pontos.
- **Entrada:** (matriz) pontos do conjunto 1, (matriz) pontos do conjunto 2.
- **Saída:** nenhuma. É gerado um gráfico.

**Nota:** Temos aqui uma situação em que a função não retorna nenhum valor. Neste caso a assinatura da função não contem nenhuma instrução de retorno.

- 
- c) Conceber o algoritmo, assumindo que os sub-problemas estão resolvidos.
  - Estimativa da área com representação gráfica (raio, xEsq, xDir, numPts):
    - Sub-problema 1:
      - `[area, ptsDentro, ptsFora] ← areaFatiaCirc(raio, xEsq, xDir, numPts)`
    - Sub-problema 2:
      - `desenha2ConjPts ( ptsDentro, ptsFora )`

- 
- 5. Para cada sub-problema, desenhar o algoritmo para o resolver.

## Sub-problema 2:

desenha2ConjPts( conj1, conj2 ):

desenha os pontos na matriz conj1

na mesma figura, desenha os pontos na matriz conj2,

---

5. Para cada sub-problema, desenhar o algoritmo para o resolver.

## Sub-problema 1:

areaFatiaCirc( raio, xEsq, xDir, numPts ):

    numPtsDentro  $\leftarrow$  0 ;    ptsDentro  $\leftarrow$  [] ;    ptsFora  $\leftarrow$  [];

    Para i de 0 a numPts - 1

        Gerar ponto aleatório (x,y) em  $[0, 2*raio] \times [0, 2*raio]$

        Se  $(x,y)$  entre as retas e dentro do círculo então

            numPtsDentro  $\leftarrow$  numPtsDentro + 1

            acrescentar  $(x,y)$  à matriz ptsDentro

        senão

            acrescentar  $(x,y)$  à matriz ptsFora

    retornar:

        numPtsDentro / numPts \*  $(2*raio)^2$ , ptsDentro, ptsFora

- Em Python, as funções podem retornar um conjunto diverso de valores:
  - Uma função pode não retornar qualquer valor, retornando `None`.
    - Quando não existe `return` ou se tem `return None`.
  - Uma função pode retornar um objeto.
    - Quando se faz `return expressão`, em que a expressão é um objeto.
  - Uma função pode retornar vários objetos.
    - Quando se faz `return expressão_1, ..., expressão_n`, em que `expressão_1, ..., expressão_n` são objetos.

- O 2º sub-problema (mais simples) usa os comandos e funções gráficas vistas anteriormente.

```
def desenha2ConjPts(conj1, conj2)
    """Desenha dois conjuntos de pontos com cores diferentes.
    As matrizes conj1 e conj2 tem:
        na primeira coluna, as abcissas dos pontos;
        na segunda coluna, as ordenadas dos pontos.
    """
    plt.clf()      # apaga graficos das janelas abertas
    plt.plot(conj1[0], conj1[1], ".r")
    plt.plot(conj2[0], conj2[1], ".b")
```

- 
6. Para cada sub-problema (começando pelos mais simples), implementar o respectivo algoritmo e testar o “sub-programa”.
- **Problema:** Estimativa da área de uma fatia vertical de um círculo.
  - **Entrada:** (real) raio, (real) xEsq, (real) xDir, (inteiro) número de pontos a gerar.
  - **Saída:** (real) estimativa da área da fatia do círculo, (matriz) pontos gerados dentro da fatia, (matriz) pontos gerados fora da fatia.

# Função areaFatiaCirc [1]

- Para o 1º sub-problema (mais complexo ) começamos pela sua documentação.

```
def areaFatiaCirc(raio, xEsq, xDir, numPts)
    """ Estima a área de uma fatia vertical de um círculo (com
    o raio dado) usando o método de Monte Carlo.
    O centro do círculo é (raio, raio). A fatia é definida pelos
    pontos do círculo cujas abscissas estão entre xEsq e xDir (com
    xEsq < xDir).
    O inteiro numPts é o número de pontos gerados.
    As matrizes ptsDentro e ptsFora, com duas linhas, x,y têm os
    pontos gerados que ficam dentro e fora da fatia.
    """

```

# Função areaFatiaCirc [2]

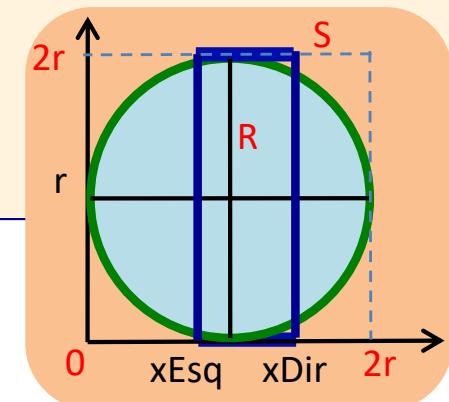
- E agora o código
  - (deixando para o próximo slide o ciclo FOR)

```
def areaFatiaCirc(raio, xEsq, xDir, numPts):  
    """ ... """  
    numPtsDentro = 0  
    ptsDentro = [[],[]]  
    ptsFora = [[],[]]  
    diametro = (2 * raio)  
    areaFora = diametro ** 2  
    for i in range(0, numPts) :  
        ...  
        area = len(numPtsDentro) / numPts * areaFora;  
    return area, ptsDentro, ptsFora
```

# Função areaFatiaCirc [3]

- E agora o ciclo FOR

```
for i in range(0, numPts) : # conta os pontos e gera matrizes
    x = diametro * random.random()
    y = diametro * random.random()
    if x >= xEsq and x <= xDir \ # Continua na linha seguinte
        (x - raio)** 2 +(y - raio) ** 2 <= raio ** 2 :
            ptsDentro[0].append(x) # conta
            ptsDentro[1].append(y)
    else :
        ptsFora[0].append(x) # não conta
        ptsFora[1].append(y)
```



---

7. Implementar o algoritmo que resolve o problema e testar o programa pedido.

- **Problema:** Estimativa da área de uma fatia vertical de um círculo com representação gráfica.
- **Entrada:** (real) raio, (real) xEsq, (real) xDir, (inteiro) número de pontos a gerar.
- **Saída:** (real) estimativa da área da fatia do círculo. É gerado um gráfico.

- E agora a função principal , que chama as duas funções anteriores.

```
def estimaEDesenhaFatiaCirc(raio, xEsq, xDir, numPts):
    """ Estima a area de uma fatia vertical de um círculo
        (com o raio dado), usando o metodo de Monte Carlo, e
        apresenta a estimativa graficamente.

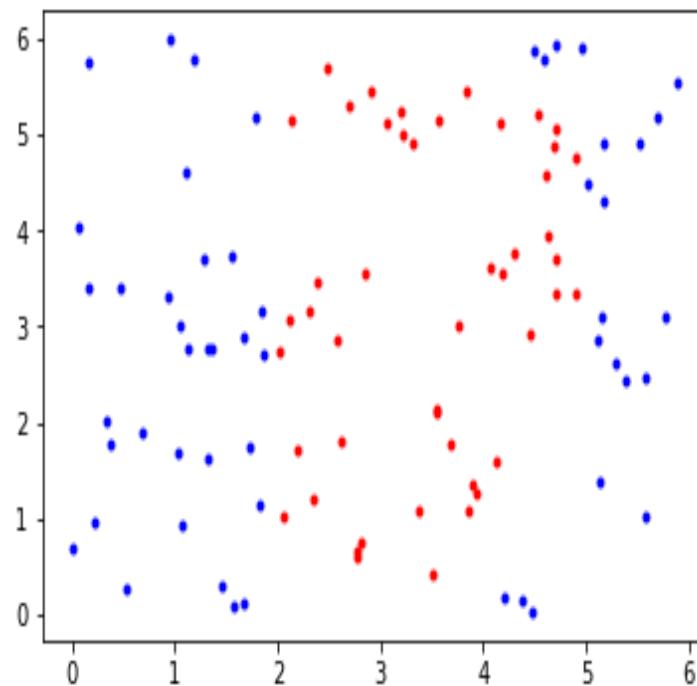
        O centro do circulo é (raio, raio). A fatia é definida pelos
        pontos do circulo cujas abcissas estao entre xEsq e xDir
        (com xEsq < xDir).

        O inteiro numPts e' o numero de pontos gerados.

    """
    area, ptsDentro, ptsFora = areaFatiaCirc(raio, xEsq,
                                              xDir, numPts)
    desenha2ConjPts(ptsDentro, ptsFora)
    return area
```

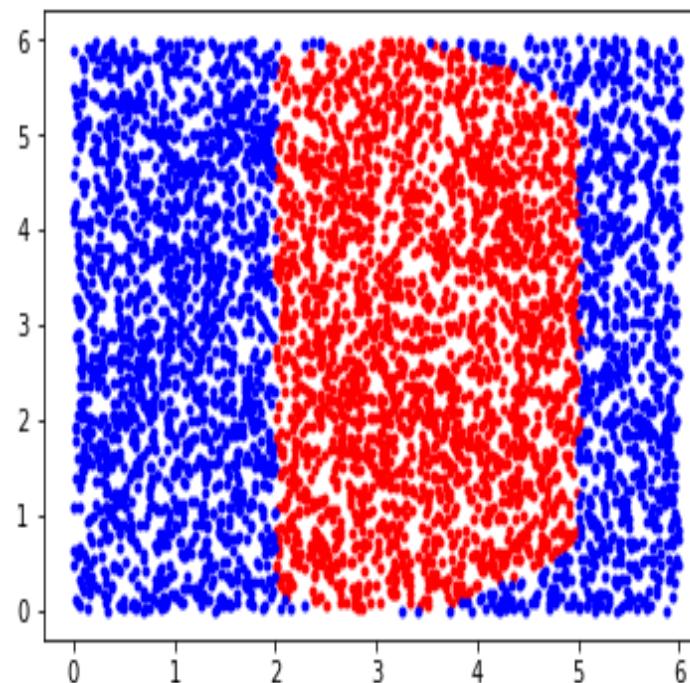
# Alguns Testes [1]

```
In : raio = 3  
In : xesq = 2  
In : xdir = 5  
In : npontos = 100  
In : estimaEDesenhaFatiaCirc(raio, xesq,xdir,npontos)  
Out: 16.92
```



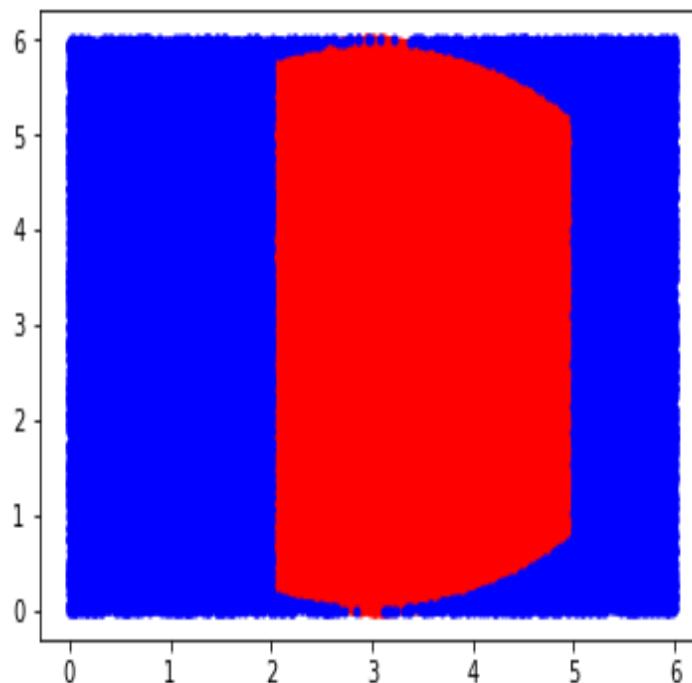
## Alguns Testes [2]

```
In : raio = 3  
In : xesq = 2  
In : xdir = 5  
In : npontos = 5000  
In : estimaEDesenhaFatiaCirc(raio, xesq,xdir,npontos)  
Out: 16.7688
```



# Alguns Testes [3]

```
In : raio = 3  
In : xesq = 2  
In : xdir = 5  
In : npontos = 100000  
In : estimaEDesenhaFatiaCirc(raio, xesq,xdir,npontos)  
Out: 16.92684
```



# Figura com (3, 0, 6, 100000)

```
In : raio = 3
In : xesq = 0
In : xdir = 6
In : npontos = 100000
In : estimaEDesenhaFatiaCirc(raio, xesq,xdir,npontos)
Out: 28.28196
In : math.pi * 3 ** 2
Out: 28.274333882
```

