Python para Aplicações Científicas

Ministrante:
Weslley da Cunha Santos

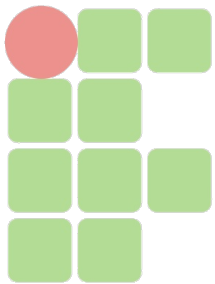
IFPB – Campus João Pessoa

2017

Por que usar o Python?

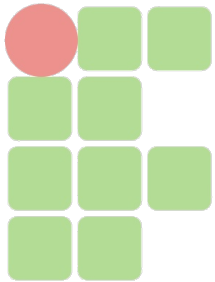
As Necessidades do Cientista

- Obtenção de dados como os oriundos de simulações ou experimentos;
- Manipular e processar dados;
- Visualizar resultados que sejam simples e rápidos de interpretar, mas também figuras e gráficos que facilitem a visualização do objeto em estudo como para a publicação em relatórios e publicações.



Soluções com Python

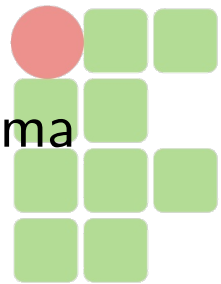
- Vantagens:
  - Imenso conjunto de bibliotecas e funções já inclusas para métodos numéricos, plotar gráficos e ferramentas para processamento de dados;
  - Fácil e rápido de aprender;
  - Linguagem simples, de desenvolvimento e execução rápidos;
  - Código livre e aberto.
- Desvantagens:
  - Não possui algumas ferramentas disponíveis em outros softwares como o Matlab;
  - Possui um desempenho inferior a linguagens compiladas como C e C++ para aplicações que exijam muito processamento em um curto espaço de tempo.



Ambiente de trabalho

Por que usar o Anaconda?

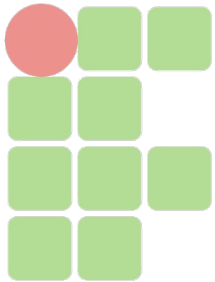
- O Python (https://www.python.org) por padrão não vem com as ferramentas de aplicação científica, já que o mesmo é uma linguagem de programação de propósito geral.
- O projeto Scipy (https://www.scipy.org) desenvolve várias ferramentas como o próprio Scipy, o Numpy, o Matplotlib, o Ipython entre outros pacotes que incluem ferramentas científicas no Python.
- Já o projeto Anaconda (https://www.continuum.io/downloads) empacota todas estas ferramentas em um único pacote fácil de instalar e utilizar, bem como o ambiente de desenvolvimento Spyder (https://github.com/spyder-ide/spyder) que fornece uma interface similar ao Matlab.



Ambiente de Trabalho

Usando o repl.it

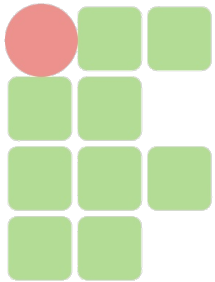
- Acessar o site https://repl.it/
- Selecionar a linguagem de programação (Python3)



Ambiente de Trabalho

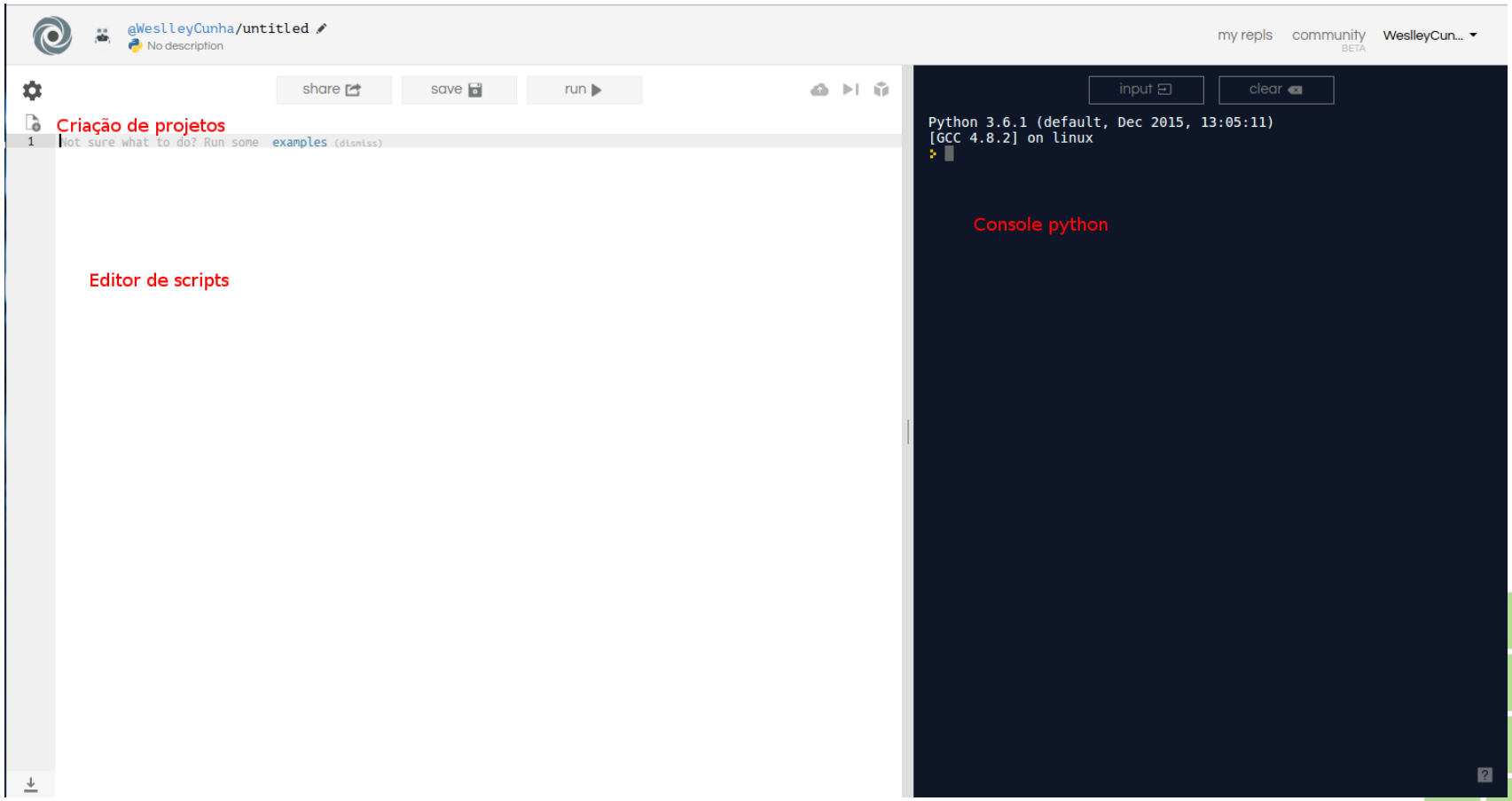
Anaconda Navigator

- Caso queira salvar seu código, pode-se cadastrar ou acessar o site com a sua conta da Google.



Ambiente de Trabalho

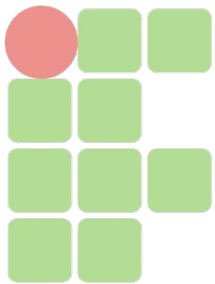
- A IDE utilizada será a do próprio site.



Ambiente de Trabalho

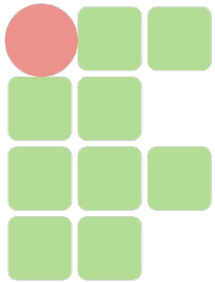
repl.it

- O repl.it pode ser dividido em duas partes principais:
  - **Console :** O IPython ou Python Interativo (https://ipython.org/) é um interpretador de Python.
  - **Editor de scripts:** Onde podem ser escritos os programas para serem salvos e executados sempre que necessários.



Operações Aritméticas com Escalares

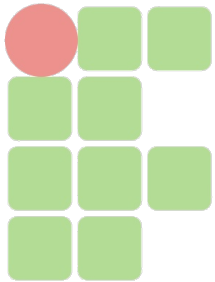
Operação	Símbolo	Exemplo	Resultado
Adição	$+$	$3 + 2$	5
Subtração	$-$	$5 - 2$	3
Multiplicação	$*$	6 * 4	24
Divisão	$/$	$5/4$	1.25
Divisão Inteira	$//$	$5//4$	1
Exponenciação	$**$	2**3	8



Precedência de Operadores

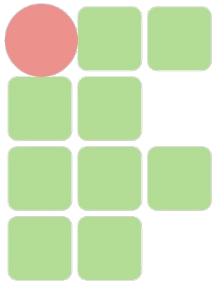
- Em expressões com muitas operações, uma ordem de execução deve ser seguida:

  1. Parênteses mais interno;
  2. Exponenciação;
  3. Multiplicação e divisão;
  4. Soma e subtração.



Operações Lógicas com Escalares

Operação	Símbolo	Exemplo	Resultado
Maior	$>$	$3 > 2$	True
Menor	$<$	$5 < 2$	False
Maior ou igual	$>=$	$6 >= 4$	True
Menor ou igual	$<=$	$5 <= 4$	False
Igual	$==$	$5 == 4$	False
Diferente	$!=$	2!=3	True



Precedência de Operadores

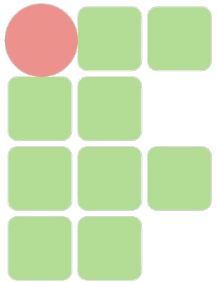
Vamos exercitar!

Realize as seguintes operações matemáticas:

$$2 + 3^2 \tag{1}$$

$$(2 + 45) - 2^4 + \frac{32}{2} \tag{2}$$

$$4,5 \times 3,7 - (1,2)^3 + 4^{(1/2)} \tag{3}$$



Precedência de Operadores

Vamos exercitar!

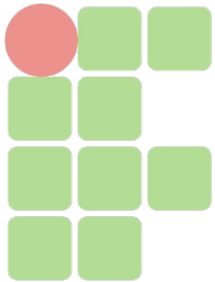
Realize as seguintes operações matemáticas:

$$2 + 3^2 \tag{1}$$

$$(2 + 45) - 2^4 + \frac{32}{2} \tag{2}$$

$$4,5 \times 3,7 - (1,2)^3 + 4^{(1/2)} \tag{3}$$

R.: (1) 11; (2) 47.0; (3) 16.922000000000004



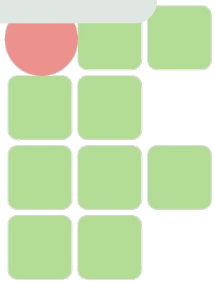
Funções Matemáticas do Python

- O Python possui uma biblioteca, por meio da qual é possível acessar as funções matemáticas. Sendo assim, é necessário inclui-la.

```
1 >> import math
```

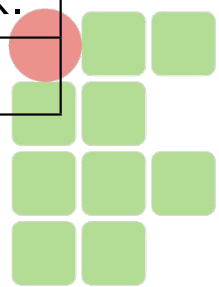
Observação!

Estas funções não podem ser usadas com números complexos. Para aplicar as mesmas funções à números complexos, deve-se utilizar as mesmas funções, porém, da biblioteca cmath.



Funções Matemáticas do Python

Função	Descrição
math.sqrt(x)	Raiz quadrada de x.
math.fabs(x)	Módulo (ou valor absoluto) de x.
math.log(x)	Logaritmo neperiano de x.
math.factorial(x)	Fatorial de x.
math.sin(x)	Seno de x.
math.cos(x)	Cosseno de x.
math.tan(x)	Tangente de x.
math.ceil(x)	Arredonda para o inteiro maior do que x.
math.floor(x)	Arredonda para o inteiro menor do que x.
math.exp(x)	Retorna e^x.



Funções Matemáticas do Python

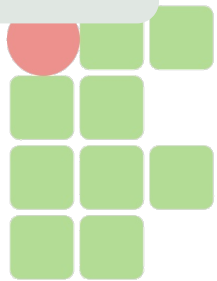
Vamos exercitar!

Resolva cada uma das expressões abaixo:

$$\frac{25.7 \times 64 - 7^3}{45 + 5^2} \tag{4}$$

$$(2+7)^3 + \frac{273^2}{log(2)^3} \tag{5}$$

$$\cos\left(\frac{5\pi}{2}\right) + \sin\left(\frac{7\pi}{8}\right)^2 \tag{6}$$



Funções Matemáticas do Python

Vamos exercitar!

Resolva cada uma das expressões abaixo:

$$\frac{25.7 \times 64 - 7^3}{45 + 5^2} \tag{4}$$

$$(2+7)^3 + \frac{273^2}{log(2)^3} \tag{5}$$

$$\cos\left(\frac{5\pi}{2}\right) + \sin\left(\frac{7\pi}{8}\right)^2 \tag{6}$$

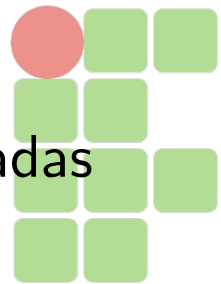
R.: (4) 18.597142857142856; (5) 2732825.1358154626; (6) 0.956053023802351 2

Variáveis

- As variáveis podem ser utilizadas em expressões matemáticas somente após serem declaradas;
- Uma variável é declarada no momento em que recebe uma atribuição. Por exemplo:

```
1 >> a = 10
2 >> a
3 10
4 >> A = 15
5 >> A
6 15
```

- O Python é case sensitive;
- Não se pode utilizar variáveis com o nome de palavras reservadas da linguagem.



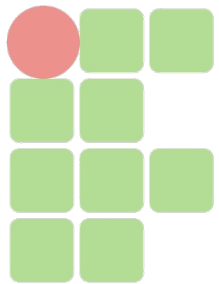
Variáveis

Vamos exercitar!

Calcule as raízes da equação de segundo grau apresentada na Eq. 7:

$$x^2 - 2x - 15 = 0 \tag{7}$$

Para isso, considere as seguintes variáveis: $a = 1$, $b = -2$ e $c = -15$. Em seguida, realize os cálculos necessários para obtenção das raízes. De preferência, utilize outras variáveis para armazenar os resultados obtidos (**delta**, **x1** e **x2**, por exemplo).



Variáveis

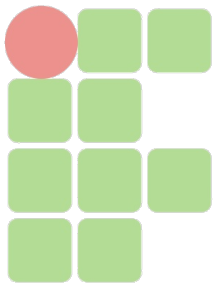
Vamos exercitar!

Calcule as raízes da equação de segundo grau apresentada na Eq. 7:

$$x^2 - 2x - 15 = 0 \tag{7}$$

Para isso, considere as seguintes variáveis: $a = 1$, $b = -2$ e $c = -15$. Em seguida, realize os cálculos necessários para obtenção das raízes. De preferência, utilize outras variáveis para armazenar os resultados obtidos (**delta**, **x1** e **x2**, por exemplo).

R.: $X_1 = 5$; $X_2 = -3$.



Entrada de Dados

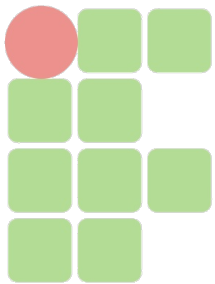
- A função **input** é utilizada para a entrada de dados padrão.
- A entrada, por padrão, é lida e armazenada como uma string (<class 'str'>).
- É possível especificar o tipo do dado que será lido. Veja os exemplos:

```
1 >>> a = input()
2 10
3 >>> a
4 '10'
5 >>> a = int(input())
6 10
7 >>> a
8 10
```

Saída de Dados

- A função **print** é utilizada para a saída de dados padrão. Veja os exemplos:

```
1 >>> a = 10
2 >>> print(a)
3 10
4 >>> print('a = %d' % a)
5 'a = 10'
```



Controle de Fluxo

Estruturas de Decisão

- São utilizadas para controlar a ordem, na qual o código será executado.
- As estruturas de decisão são: if/elif/else.
- Exemplo:

```
>>> a = 10
>>> if a == 1:
...     print(1)
... elif a == 2:
...     print(2)
... else:
...     print('A lot')
A lot
```

Controle de Fluxo

Estruturas de Repetição

- São utilizadas para executar algumas linhas de código repetidas vezes.
- As estruturas de decisão são: **for/range** e **while/break/continue**.
- Exemplo do **for/range**:

```
>>> for i in range(4):
...     print(i)
0
1
2
3
```

Controle de Fluxo

Estruturas de Repetição

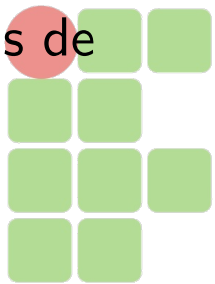
- São utilizadas para executar algumas linhas de código repetidas vezes.
- As estruturas de decisão são: **for/range** e **while/break/continue**.
- Exemplo do **while/break/continue**:

```
1 >>> a = 0
2 >>> while a <= 3
3 ...     print(a)
4 ...     a = a + 1;
5 0
6 1
7 2
8 3
```

Numpy

Criando e Manipulando Dados Numéricos

- O que o Numpy disponibiliza?
  - Pacotes adicionais no Python para arrays multidimensionais;
  - Mais próximo ao *hardware* (eficiência);
  - Projetado para computação científica (comodidade);
  - Também conhecido como *computação orientada a arrays*.
- Um array contém:
  - Valores de um experimento/simulação em intervalo de tempo discreto;
  - Sinal gravado por um dispositivo de medição (sinal sonoro, por exemplo);
  - *Pixels* de uma imagem (nível de cinza ou cor);
  - Dados 3D medidos em relação às coordenadas X, Y e Z (imagens de Ressonância Magnética, por exemplo).



Numpy

Exemplo de Implementação

```
1 >> import numpy as np
2 >> a = np.array([0, 1, 2, 3])
3 >> a
4 array([0, 1, 2, 3])
```

Observação!

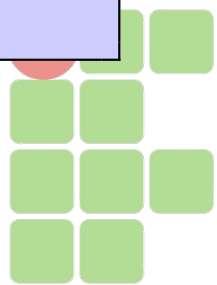
Por convenção, importa-se a biblioteca Numpy da forma como foi definida na linha 1 do código acima.

Numpy

Criando Arrays

- Construção manual de arrays:
  - 1D:

```
1 >>> a = np.array([0, 1, 2, 3])
2 >>> a
3 array([0, 1, 2, 3])
4 >>> a.ndim
5 1
6 >>> a.shape
7 (4,)
8 >>> len(a)
9 4
```



Numpy

Criando Arrays

- Construção manual de arrays:
  - 2D, 3D, ...:

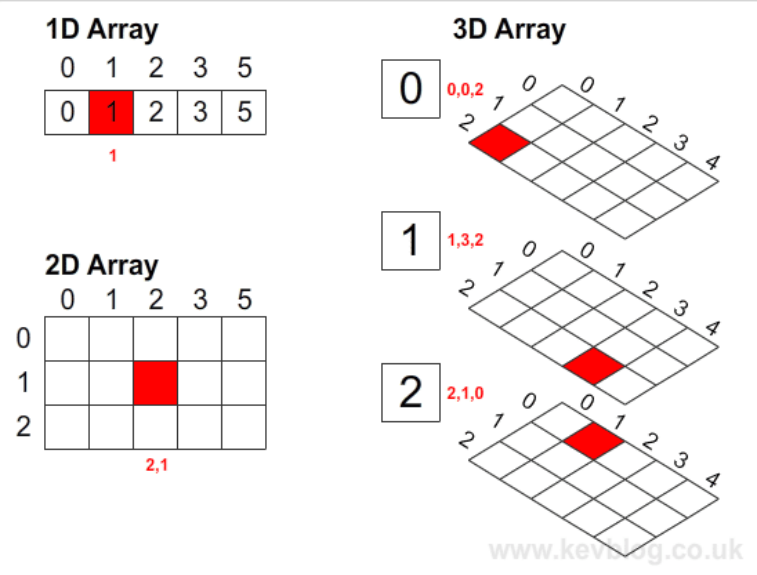
```
1  >>> b = np.array([[0, 1, 2], [3, 4, 5]])
2  >>> b
3  array([[0, 1, 2],
4         [3, 4, 5]])
5  >>> b.ndim
6  2
7  >>> b.shape
8  (2, 3)
9  >>> len(b) # retorna o tamanho da primeira dimensao
10 2
```

Numpy

Criando Arrays

Vamos exercitar!

Crie arrays 1D, 2D e 3D tendo como base a figura abaixo. Em seguida, analise cada um dos seus atributos.

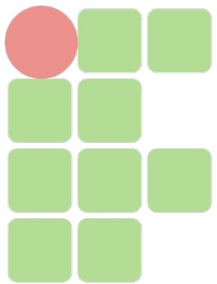


Numpy

Criando Arrays

- Funções para criar arrays:
  - Espaçados uniformemente:

```
1 >>> a = np.arange(10) # 0, ..., n - 1
2 >>> a
3 array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
4 >>> b = np.arange(1, 9, 2) # inicio, fim (exclui esse
      valor), passo
5 >>> b
6 array([1, 3, 5, 7])
```

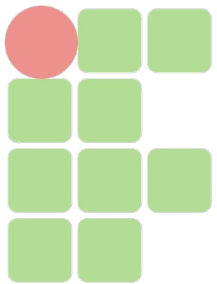


Numpy

Criando Arrays

- Funções para criar arrays:
  - Pelo número de pontos:

```
1 >>> c = np.linspace(0, 1, 6) # inicio, fim, num. de
      pts
2 >>> c
3 array([0. , 0.2, 0.4, 0.6, 0.8, 1. ])
4 >>> d = np.linspace(0, 1, 5, endpoint=False)
5 >>> d
6 array([0. , 0.2, 0.4, 0.6, 0.8])
```



Numpy

Criando Arrays

- Funções para criar arrays:
  - Arrays comuns:

```
>>> a = np.ones((3, 3))
>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> b = np.zeros((2, 2))
>>> b
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> c = np.random.rand(4)
>>> c
array([ 0.25298236,  0.43479153,  0.77938292,
    0.19768507])
```

Numpy

Criando Arrays

- Funções para criar arrays:
  - Arrays comuns:

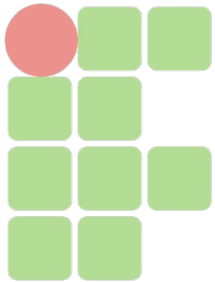
```
>>> d = np.eye(3)
>>> d
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> e = np.diag(np.array([1, 2, 3, 4]))
>>> e
array([[1, 0, 0, 0],
       [0, 2, 0, 0],
       [0, 0, 3, 0],
       [0, 0, 0, 4]])
```

Numpy

Criando Arrays

Vamos exercitar!

1. Crie um matriz identidade com dimensão 5×5.
2. Crie uma matriz 3×3 com todos os elementos iguais a 1.
3. Crie uma matriz 2×3 com elementos de valores aleatórios entre 0 e 20.
4. Crie o vetor: [9, 8, 7, 6, 5, 4, 3, 2, 1].



Numpy

Indexação e Cortes de Arrays

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

Observação!

O índice dos arrays começa em 0, como nas outras sequências do Python, C/C++, dentre outros. Deve-se atentar para isso, uma vez que no Matlab os índices iniciam em 1.

Numpy

Indexação e Cortes de Arrays

- Assim como no Python, é possível inverter o array da seguinte forma:

```
>>> a[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

- Acesso em arrays multidimensionais:

```
>>> a = np.diag(np.arange(3))
>>> a
array([[0, 0, 0],
       [0, 1, 0],
       [0, 0, 2]])
>>> a[1, 1]
1
```

Numpy

Indexação e Cortes de Arrays

- Modificação em arrays multidimensionais:

```
>>> a
array([[ 0,  0,  0],
       [ 0,  1,  0],
       [ 0, 10,  2]])
>>> a[2, 1] = 10
>>> a[1]
array([0, 1, 0])
```

- Em um array 2D, a primeira dimensão corresponde à linha, ao passo que a segunda dimensão corresponde à coluna.
- Para arrays multidimensionais, o comando a[0] é interpretado tomando todos os elementos que não foram especificados.

Numpy

Indexação e Cortes de Arrays

- Assim como nas sequências do Python, é possível tomar "fatias" de um array. Por exemplo:

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[2:9:3] # [inicio:fim:passo]
array([2, 5, 8])
>>> a[:4]
array([0, 1, 2, 3])
```

Observação!

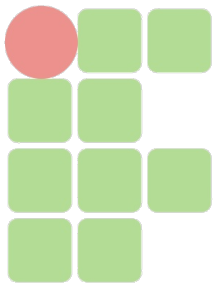
Note que o último índice não é incluído!

Numpy

Indexação e Cortes de Arrays

- Outros exemplos...

```
>>> a[1:3]
array([1, 2])
>>> a[::2]
array([0, 2, 4, 6, 8])
>>> a[3:]
array([3, 4, 5, 6, 7, 8, 9])
```



Numpy

Indexação e Cortes de Arrays

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```

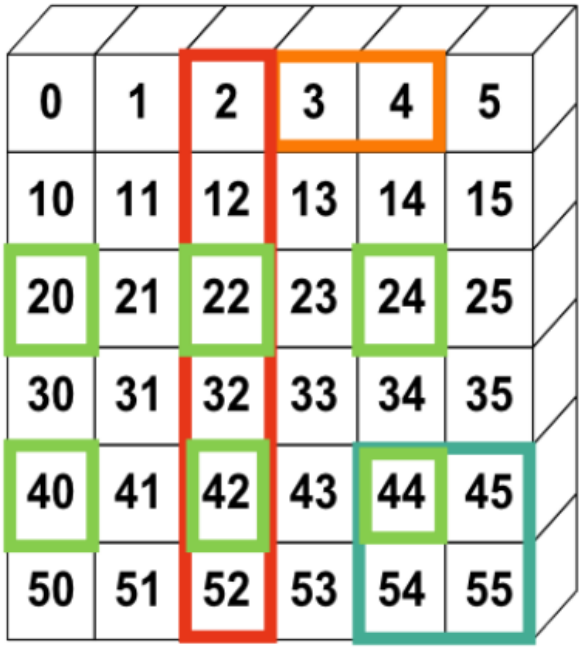


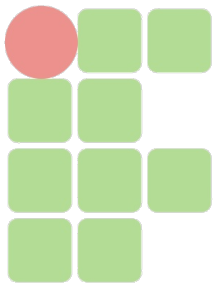
Figura: Cortes em um array bidimensional.

Numpy

Operações com Arrays

- Operações Básicas:
  - Com escalares:

```
1 >>> a = np.array([1, 2, 3, 4])
2 >>> a + 1
3 array([2, 3, 4, 5])
4 >>> 2**a
5 array([ 2, 4, 8, 16])
```

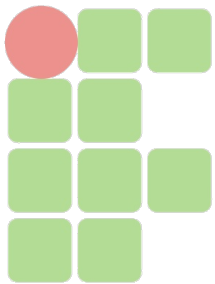


Numpy

Operações com Arrays

- Operações Básicas:
  - Entre arrays:

```
>>> b = np.ones(4) + 1
>>> a - b
array([-1., 0., 1., 2.])
>>> a * b
array([ 2., 4., 6., 8.])
```



Numpy

Operações com Arrays

- Entre arrays:

```
>>> c = np.ones((3, 3))
>>> c * c
array([[ 1., 1., 1.],
       [ 1., 1., 1.],
       [ 1., 1., 1.]])
>>> c.dot(c)
array([[ 3., 3., 3.],
       [ 3., 3., 3.],
       [ 3., 3., 3.]])
```

Observação!

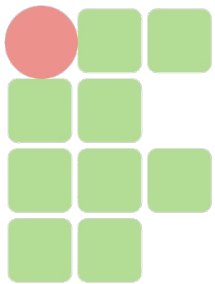
A multiplicação de arrays 2D não corresponde a uma multiplicação de matrizes, mas a uma multiplicação elemento a elemento.

Numpy

Operações com Arrays

Vamos exercitar!

1. Crie dois arrays 1D de forma automática: o primeiro com números pares entre 0 e 8; o segundo com números ímpares entre 1 e 9. Experimente realizar as operações aritméticas sobre eles.
2. Gere o seguinte array:
   [2**0, 2**1, 2**2, 2**3, 2**4]

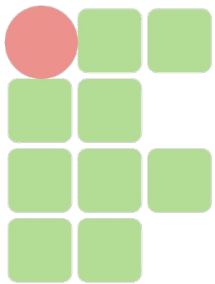


Numpy

Operações com Arrays

- Outras Operações:
  - Comparações – Parte 1:

```
1 >>> a = np.array([1, 2, 3, 4])
2 >>> b = np.array([4, 2, 2, 4])
3 >>> a == b
4 array([False, True, False, True], dtype=bool)
5 >>> a > b
6 array([False, False, True, False], dtype=bool)
```

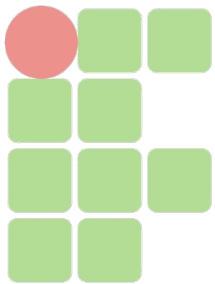


Numpy

Operações com Arrays

- Outras Operações:
  - Comparações – Parte 2:

```
1 >>> a = np.array([1, 2, 3, 4])
2 >>> b = np.array([4, 2, 2, 4])
3 >>> c = np.array([1, 2, 3, 4])
4 >>> np.array_equal(a, b)
5 False
6 >>> np.array_equal(a, c)
7 True
```

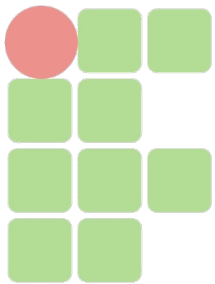


Numpy

Operações com Arrays

- Outras Operações:
  - Operações Lógicas:

```
>>> a = np.array([1, 1, 0, 0], dtype=bool)
>>> b = np.array([1, 0, 1, 0], dtype=bool)
>>> np.logical_or(a, b)
array([ True, True, True, False], dtype=bool)
>>> np.logical_and(a, b)
array([ True, False, False, False], dtype=bool)
```



Numpy

Outras Operações Importantes

- Considere um array **x**.

Operação	Retorno
np.abs(x)	Array com o valor absoluto dos elementos de x.
np.sin(x)	Array com o seno dos elementos de x.
np.cos(x)	Array com o cosseno dos elementos de x.
np.log(x)	Array com o log. neperiano dos elementos de x.
np.exp(x)	Array formado por e^x.
np.invert(x)	A inversa da matriz x.
np.sum(x)	O somatório de todos os elementos de x.

Numpy

Métodos de um Array

- Considere um array **x**.

Atributo	Descrição
x.diagonal()	Retorna um vetor com a diagonal de x (2D).
x.dot(c)	Retorna a multiplicação das matrizes x e c.
x.max()	Retorna o elemento de x com maior valor.
x.min()	Retorna o elemento de x com menor valor.
x.sum()	Retorna a soma de todos os elementos de x.
x.mean()	Retorna a média dos valores de x.
x.sort()	Ordena os elementos de x e atribui ao próprio x.

Numpy

Vamos exercitar!

$$\begin{bmatrix} 2 & 45 & 22 & 12 \\ 44 & 34 & 68 & 3 \\ 5 & 7 & 78 & 2 \\ 12 & 34 & 5 & 9 \end{bmatrix} \tag{8}$$

Com base na matriz da Eq. 8:

1. Obtenha um vetor contendo o seno dos elementos da segunda linha;
2. Obtenha sua inversa, sua transposta e um vetor com os elementos de sua diagonal principal;
3. Obtenha a soma e a média dos seus elementos.

Algumas funções para Álgebra Linear

Scipy - Linalg

- O módulo **scipy.linalg** permite as operações padrão para álgebra linear.
- Duas funções que são importantes são a **scipy.linalg.det()** e a **scipy.linalg.inv()** que calculam o determinante e a matriz inversa.

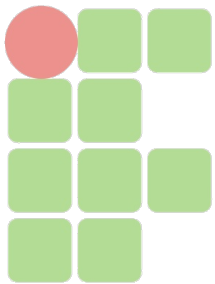
```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[3, 2],[6, 4]])
>>> linalg.det(arr)
0.0
>>> arr = np.array([[1, 0],[0, 1]])
>>> linalg.det(arr)
1.0

```

Algumas funções para Álgebra Linear

Scipy - Linalg

```
>>> import numpy as np
>>> from scipy import linalg
>>> arr = np.array([[1, 2],[3, 4]])
>>> iarr = linalg.inv(arr)
>>> iarr
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```



Scripts

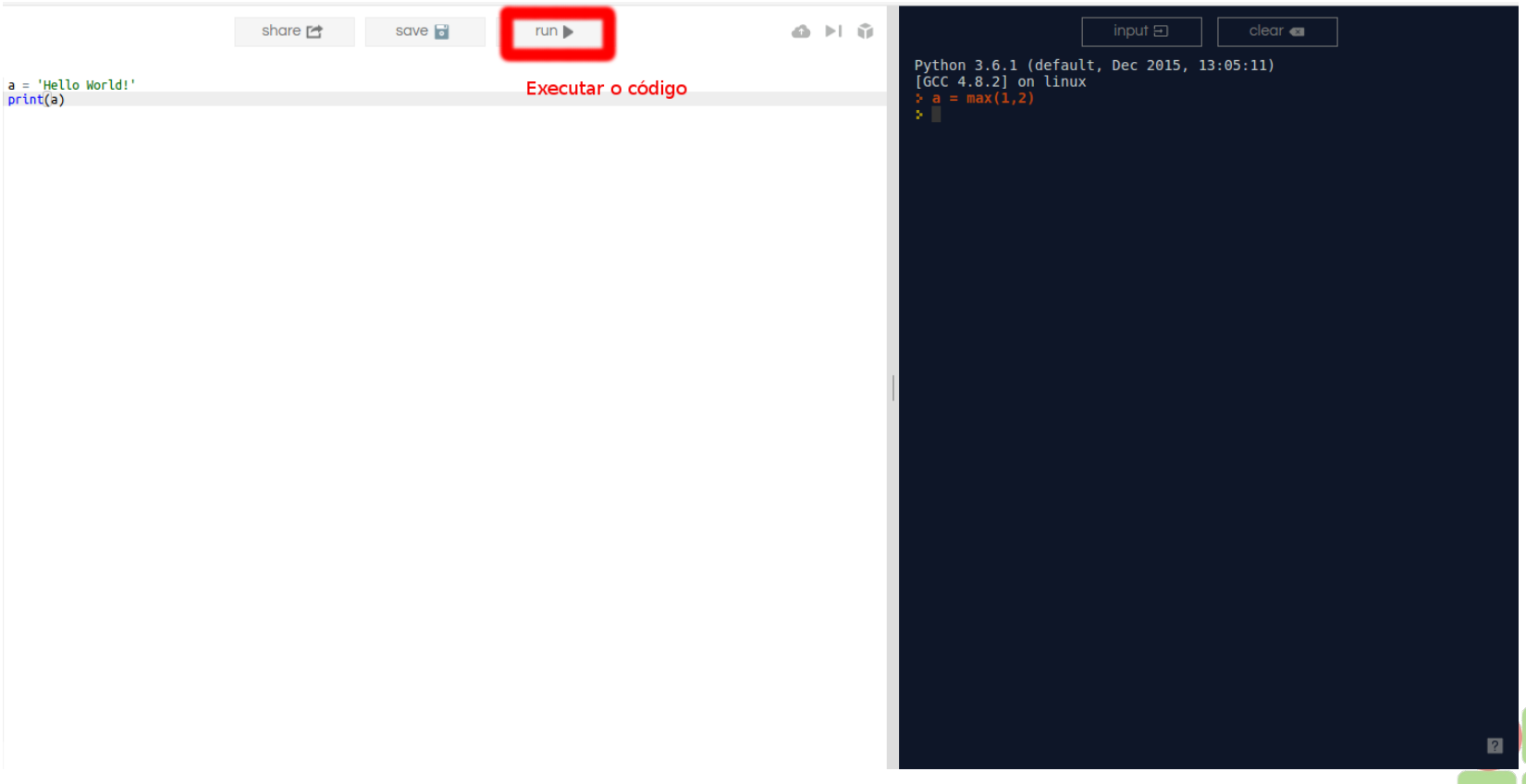
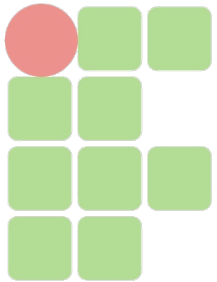


Figura: Execução de um script no repl.it

Scripts

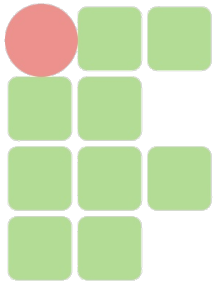
- Os scripts com extensão .py ajudam a:
  - Fornecer uma visão geral do que vai ser executado
  - Permitir que erros possam ser corrigidos mais rapidamente
  - Alterar o código de maneira mais rápida.



Scripts

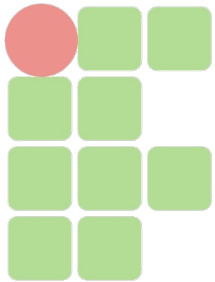
Vamos exercitar!

Crie um script para ler os valores dos coeficientes **a**, **b**, e **c** de uma equação de segundo grau. Em seguida, calcule as raízes dessa equação e exiba-as.



Matplotlib

- Matplotlib é provavelmente o pacote Python mais utilizado para gráficos 2D.
- Ele fornece uma maneira muito rápida de visualizar dados usando Python, e figuras com qualidade de publicação e em diversos formatos.
- O Pyplot fornece uma interface procedural para a biblioteca de plotagem orientada a objeto, matplotlib.
- Ele foi desenvolvido após o Matlab. Portanto, a maioria dos comandos de plotagem do pyplot são análogos aos do Matlab e possuem argumentos semelhantes.



Matplotlib

- Para importar o pyplot basta incluir o seguinte comando:

```
1 import matplotlib.pyplot as plt
```

- Ou:

```
1 from matplotlib import pyplot as plt
```

Observação!

Assim como no Numpy, o Pyplot, por convenção, é importado dessa forma (como **plt**).

Matplotlib

Primeiro Exemplo

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
5  C, S = np.cos(X), np.sin(X)
6
7  plt.plot(X, C)
8  plt.plot(X, S)
9  plt.title("Seno e Cosseno");
10 plt.ylabel("Amplitude");
11 plt.xlabel("Angulo");
12 plt.grid("on");
13
14 plt.show()
15 plt.savefig("plot.png")
```

Matplotlib

Primeiro Exemplo

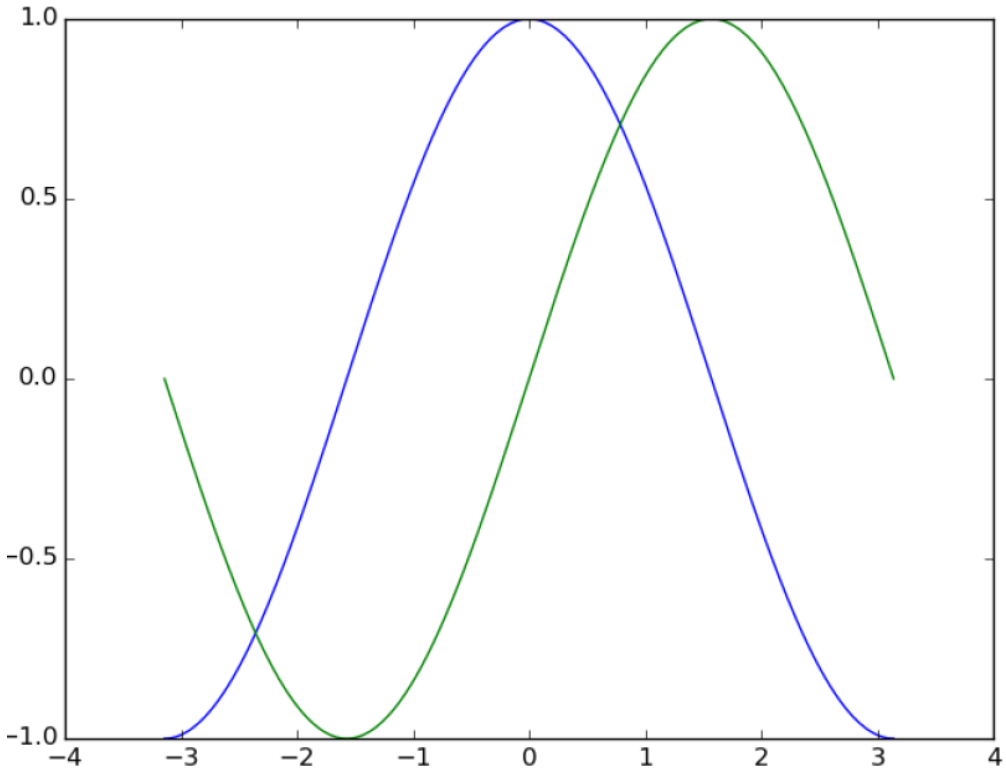
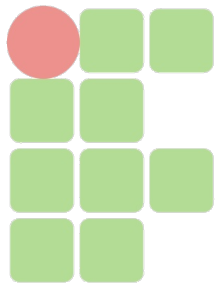


Figura: Saída após a execução do código anterior.

Matplotlib

Comando Plot

- No comando plot alguns especificadores e propriedades podem ser utilizados.
- Por padrão a linha é preenchida, mas ela pode ser tracejada (–), pontilhada (:), traço-ponto (-.), dentre outros.
- Junto com o padrão da linha, a cor pode ser alterada: vermelha (r), verde (g), amarela (y), preto (b), ciano (c) e branco (w).
- Ver documentação: <https://goo.gl/25DF0l>.



Matplotlib

Comando Plot

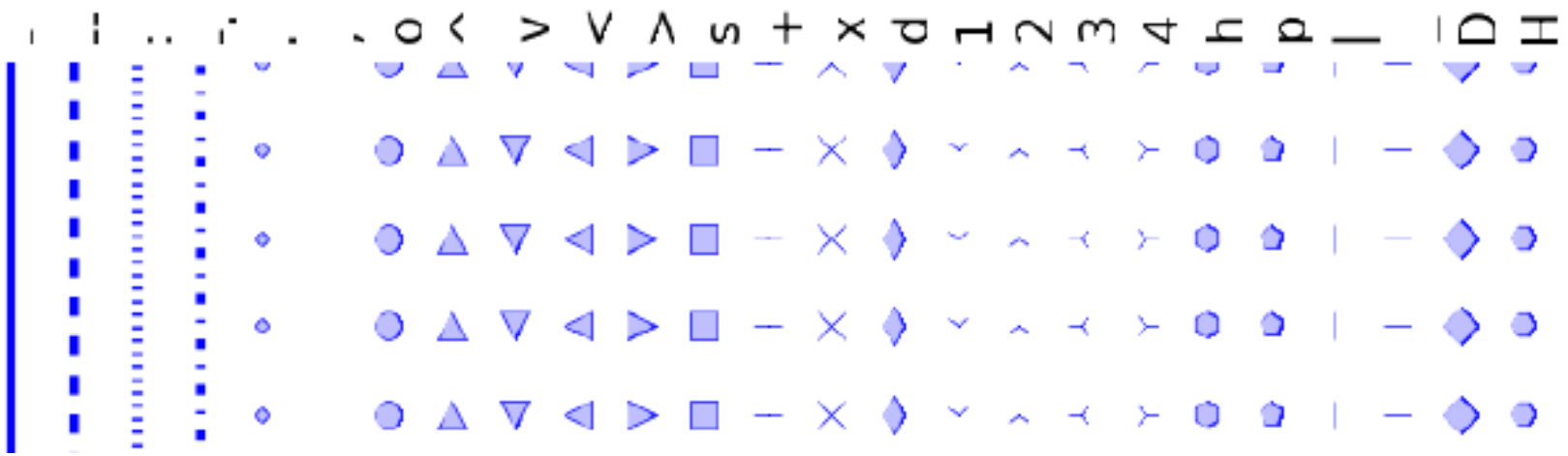


Figura: Estilos da linha.

Matplotlib

Comando Plot

- É possível também modificar a espessura da linha. Veja o exemplo:

```
1 plt.plot(X, C, color="blue", linewidth=2.5, linestyle
      ="-")
2 plt.plot(X, S, color="red", linewidth=2.5, linestyle=
      "-")
```

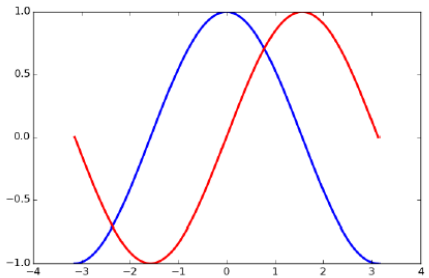


Figura: Espessura, cor e estilo da linha.

Matplotlib

Comando Plot

- Existe uma outra forma de escrever o código visto anteriormente.

```
1 plt.plot(X, C, "b-", linewidth=2.5)
2 plt.plot(X, S, "r-", linewidth=2.5)
```

- Além de ser mais compacta, essa é a forma utilizada no *software* Matlab.

Observação!

Os especificadores de cor e largura da linha do gráfico são inseridos como strings.

Matplotlib

Estabelecendo Limites aos Eixos

- É possível estabelecer limites para os eixos do gráfico. Veja o exemplo:

```
1 plt.xlim(X.min() * 1.1, X.max() * 1.1)
2 plt.ylim(C.min() * 1.1, C.max() * 1.1)
```

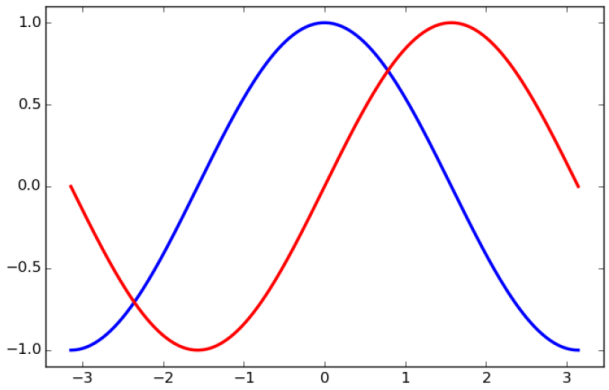


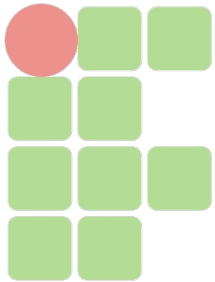
Figura: Limitando os eixos do gráfico.

Matplotlib

Plotando gráficos

Vamos exercitar!

Crie um script para ler os valores dos coeficientes **a**, **b**, e **c** de uma função de segundo grau. Em seguida, plote essa função no intervalo de -10 a 10 (eixo x) com 100 pontos nesse intervalo. Varie a cor, a espessura e o estilo da linha. Dê nome aos eixos e atribua um título. Além disso, inclua o grid no gráfico.

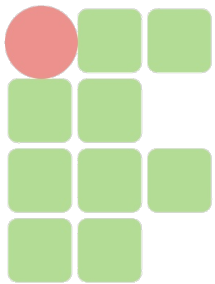


Matplotlib

Imagens

- Usando o Matplotlib também é possível carregar e manipular imagens.

```
>>> img = plt.imread('data/elephant.png')
>>> img.shape, img.dtype
((200, 300, 3), dtype('float32'))
>>> plt.imshow(img)
<matplotlib.image.AxesImage object at ...>
>>> plt.savefig('plot.png')
```

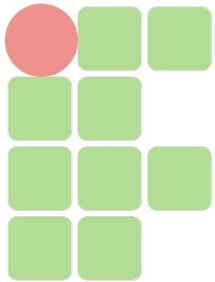


Matplotlib

Imagens

Vamos exercitar!

Crie um script para carregar uma imagem. Em seguida, apresente os seus atributos e exiba-a.



Agradecemos pela presença!

- Contato:
  - Weslley Cunha: weslley@ieee.org

Até a próxima!!!