Pró-reitora de Pós-Graduação, Pesquisa e Inovação

Especialização em Ciências de Dados e Analytics

Programação para Ciência de Dados

Parte 2 - Matplotlib

Agenda

- Biblioteca Matplotlib
  - Visão geral e instalação
  - Uso com funções
  - Uso com OO
  - Plots e outros tipos de gráficos

Matplotlib

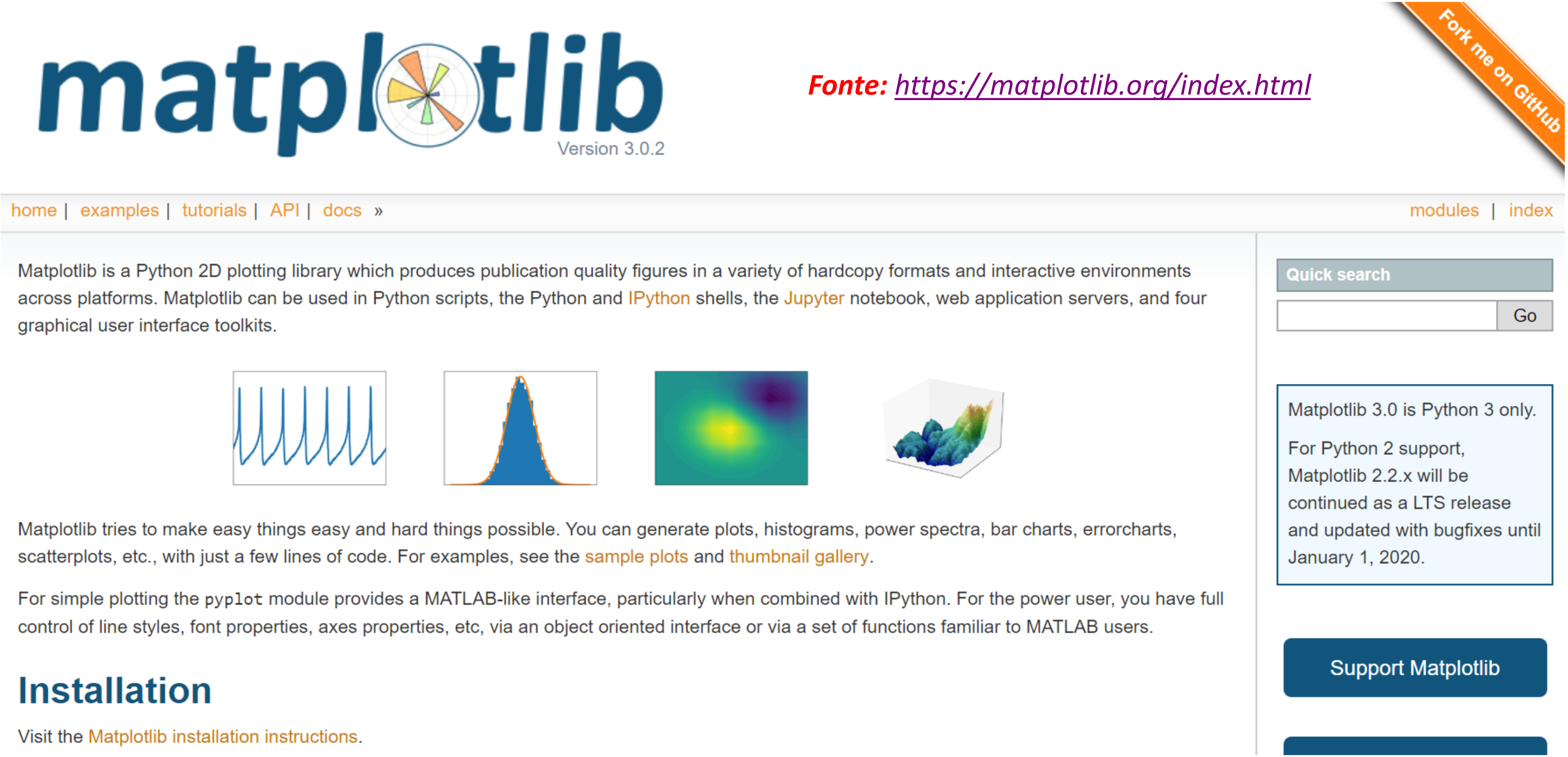
- Uma das Bibliotecas para visualização de dados mais popular em python

- Oferece diversos recursos para visualização de dados, com funcionalidades similares a correspondente em Matlab

Instalação

- Com o Anaconda já instalado, o processo de instalação do Matplotlib é muito simples:
  - conda install matplotlib

    OU
  - pip install matplotlib



Fonte: *https://matplotlib.org/index.html*

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

Matplotlib tries to make easy things easy and hard things possible. You can generate plots, histograms, power spectra, bar charts, errorcharts, scatterplots, etc., with just a few lines of code. For examples, see the sample plots and thumbnail gallery.

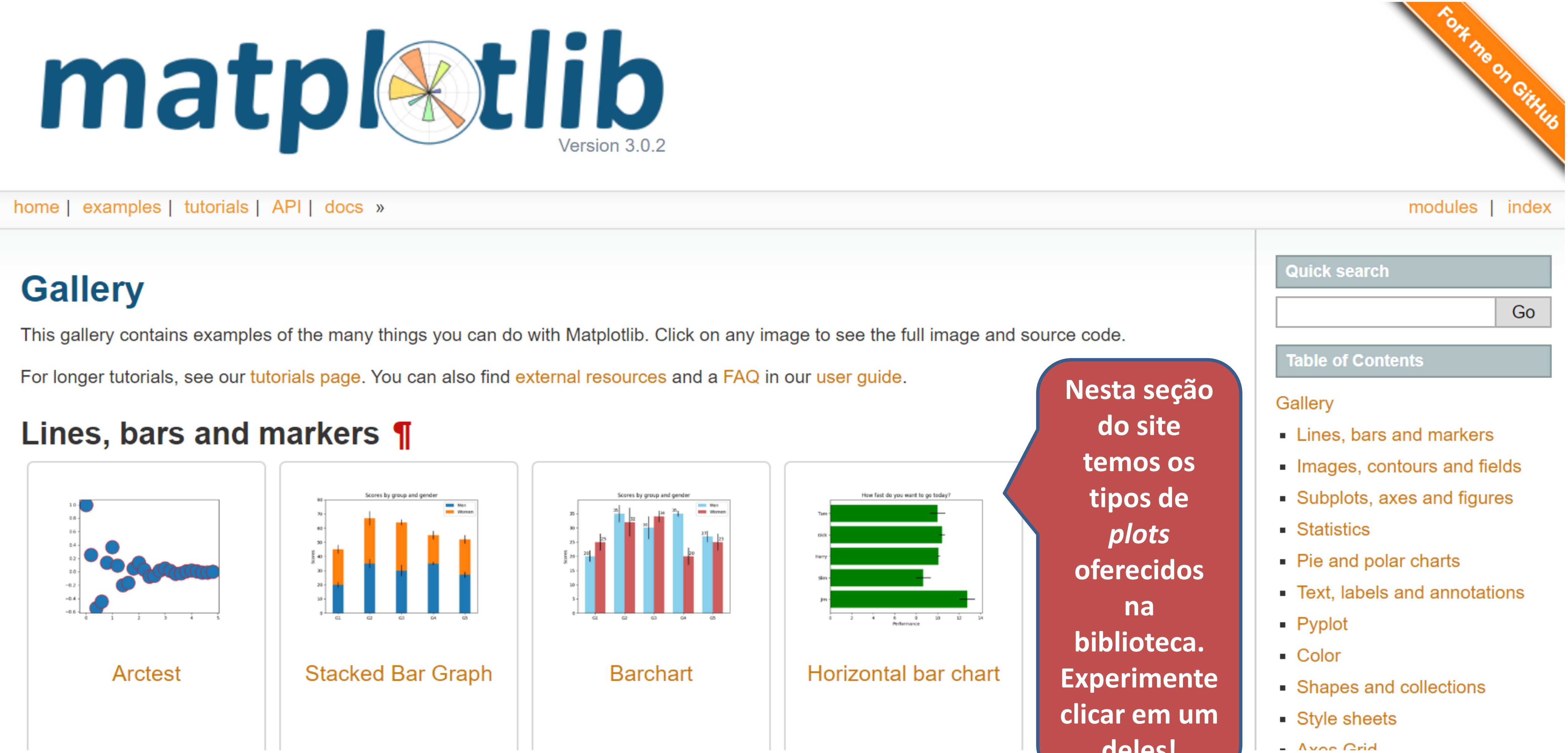
For simple plotting the `pyplot` module provides a MATLAB-like interface, particularly when combined with IPython. For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users.

Installation

Visit the Matplotlib installation instructions.

Matplotlib 3.0 is Python 3 only.

For Python 2 support, Matplotlib 2.2.x will be continued as a LTS release and updated with bugfixes until January 1, 2020.


Version 3.0.2
Fork me on GitHub
Scores by group and gender
Scores by group and gender
How fast do you want to go today?
Performance

home | examples | tutorials | API | docs » modules | index

Quick search

Go

Table of Contents

Gallery

This gallery contains examples of the many things you can do with Matplotlib. Click on any image to see the full image and source code.

For longer tutorials, see our tutorials page. You can also find external resources and a FAQ in our user guide.

Lines, bars and markers ¶

Arctest

Stacked Bar Graph

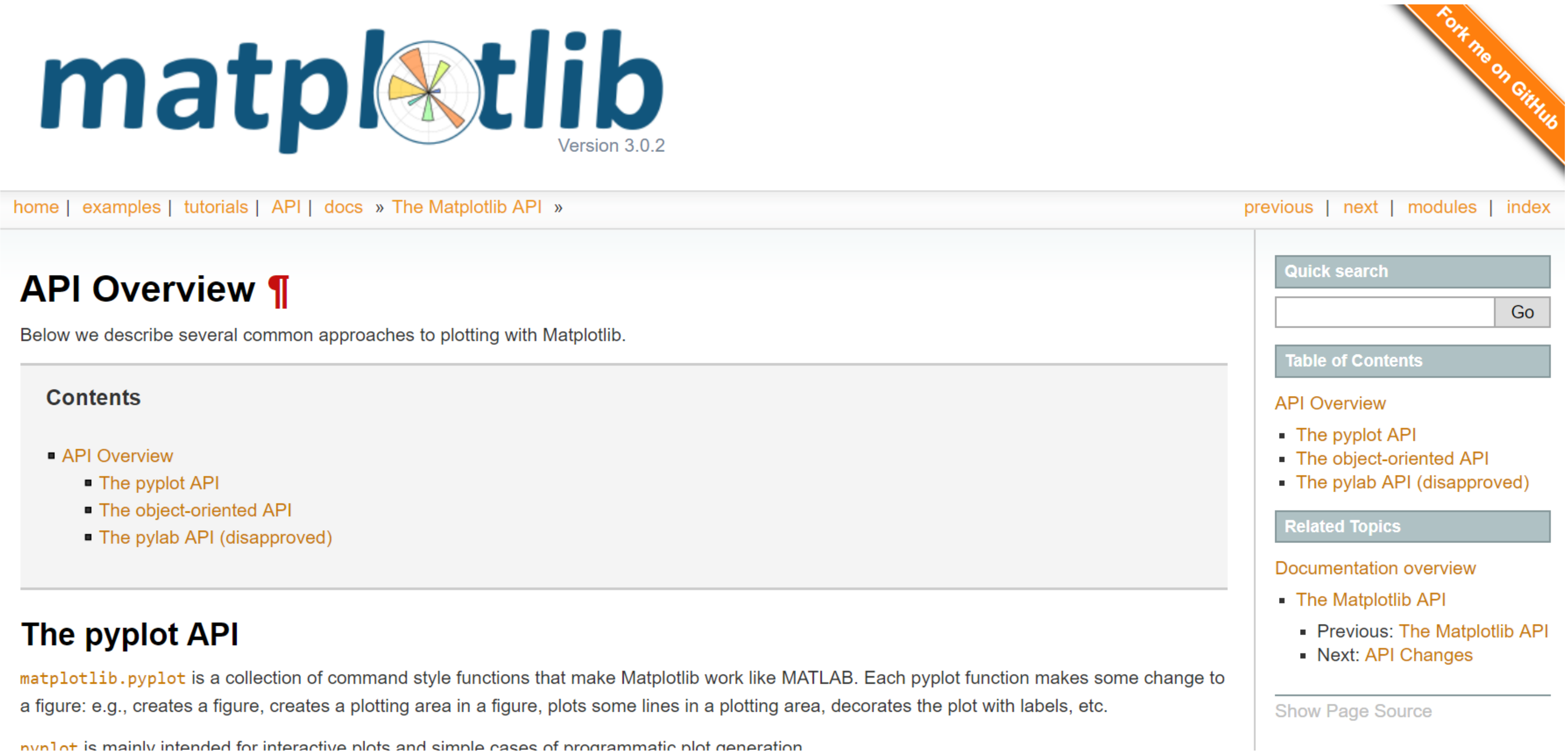
Barchart

Horizontal bar chart

Gallery

- Lines, bars and markers
- Images, contours and fields
- Subplots, axes and figures
- Statistics
- Pie and polar charts
- Text, labels and annotations
- Pyplot
- Color
- Shapes and collections
- Style sheets
- Axes Grid

Nesta seção do site temos os tipos de *plots* oferecidos na biblioteca. Experimente clicar em um deles!



matplotlib
Version 3.0.2

home | examples | tutorials | API | docs » The Matplotlib API »
previous | next | modules | index

API Overview ¶

Below we describe several common approaches to plotting with Matplotlib.

Contents

The pyplot API

`matplotlib.pyplot` is a collection of command style functions that make Matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

`pyplot` is mainly intended for interactive plots and simple cases of programmatic plot generation

Table of Contents

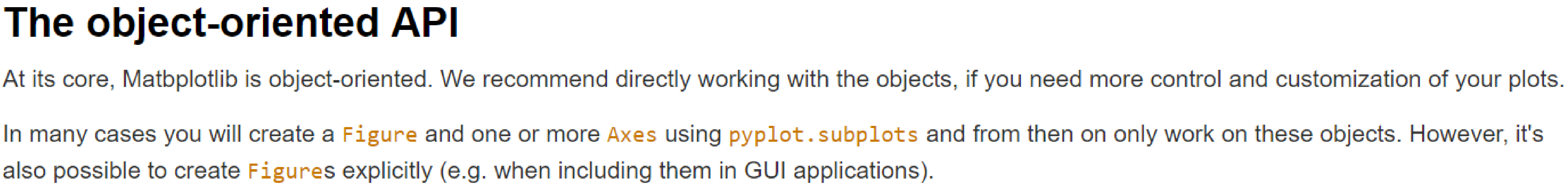
Related Topics

Documentation overview
- The Matplotlib API
  - Previous: The Matplotlib API
  - Next: API Changes

Show Page Source

Matplotlib :: Visão Geral

- matplotlib.pyplot
  - Classe mais recomendada
  - A classe *pylab* está sendo depreciada
- Funcional X OO
  - https://matplotlib.org/api/_as_gen/matplotlib.pyplot.html#module-matplotlib.pyplot



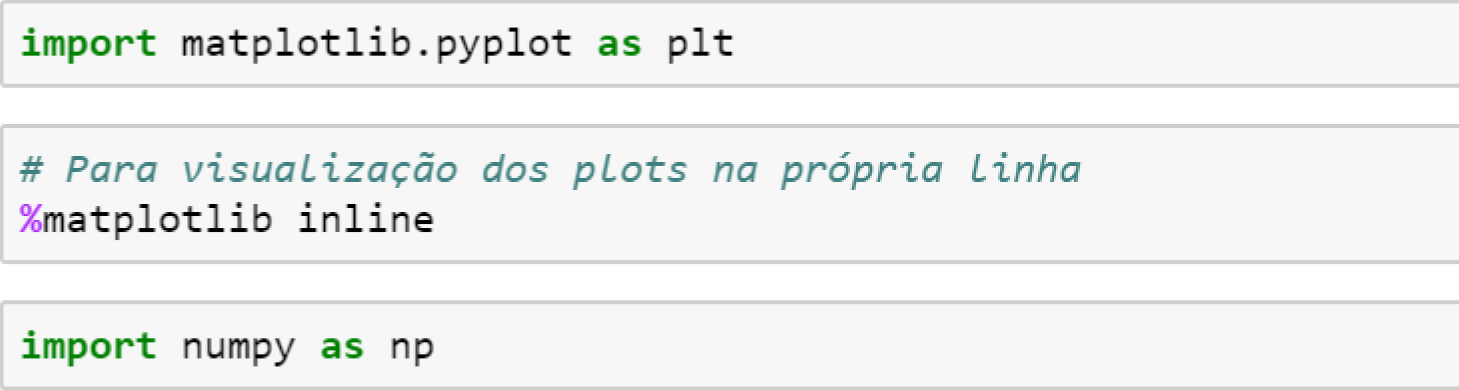
The object-oriented API

At its core, Matbplotlib is object-oriented. We recommend directly working with the objects, if you need more control and customization of your plots.

In many cases you will create a `Figure` and one or more `Axes` using `pyplot.subplots` and from then on only work on these objects. However, it's also possible to create `Figure`s explicitly (e.g. when including them in GUI applications).

Matplotlib :: Funcional

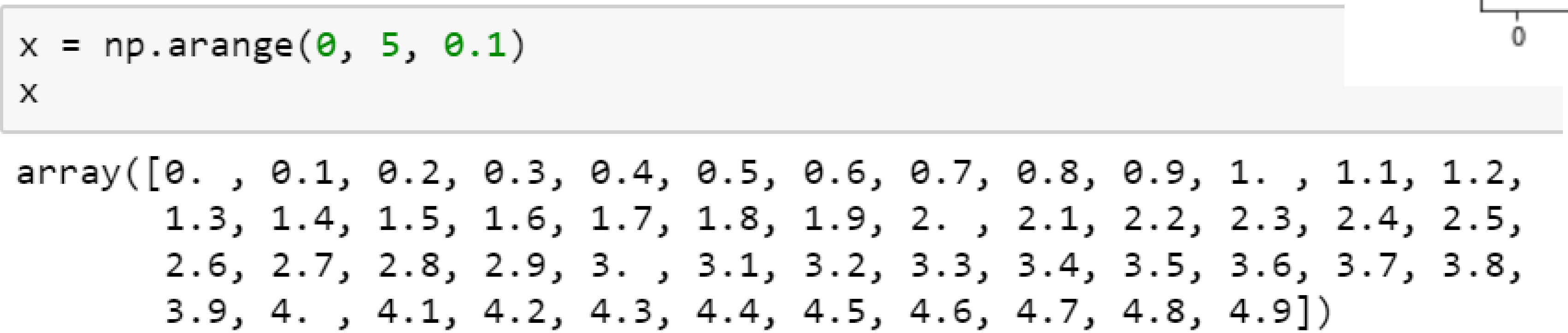
- matplotlib.pyplot



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

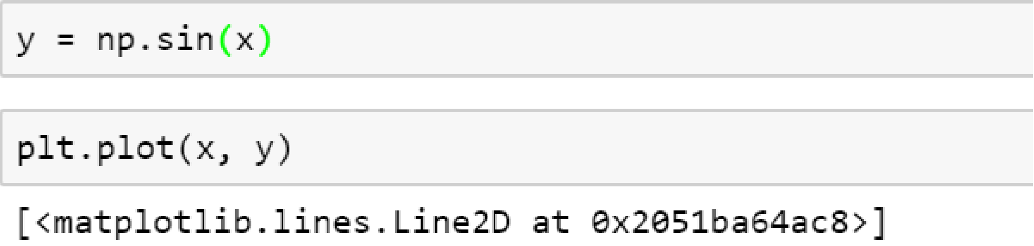
```
# Para visualização dos plots na própria linha
%matplotlib inline
```

```
import numpy as np
```



```
x = np.arange(0, 5, 0.1)
x
```

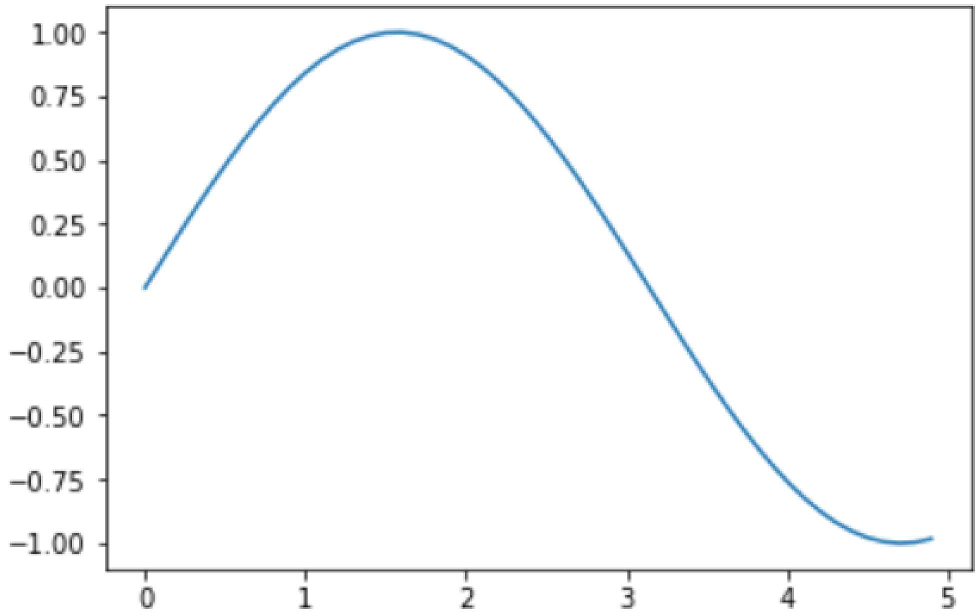
```
array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. , 1.1, 1.2,
       1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5,
       2.6, 2.7, 2.8, 2.9, 3. , 3.1, 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8,
       3.9, 4. , 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, 4.8, 4.9])
```



```
y = np.sin(x)
```

```
plt.plot(x, y)
```

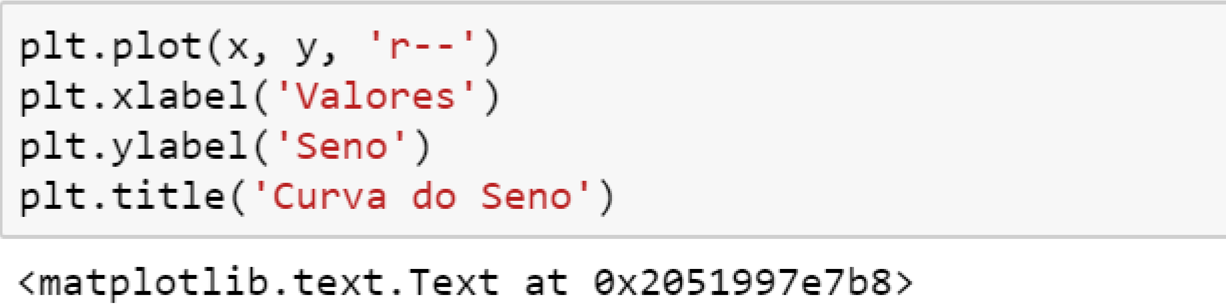
```
[<matplotlib.lines.Line2D at 0x2051ba64ac8>]
```



O gráfico foi exibido de imediato por conta do ***inline***

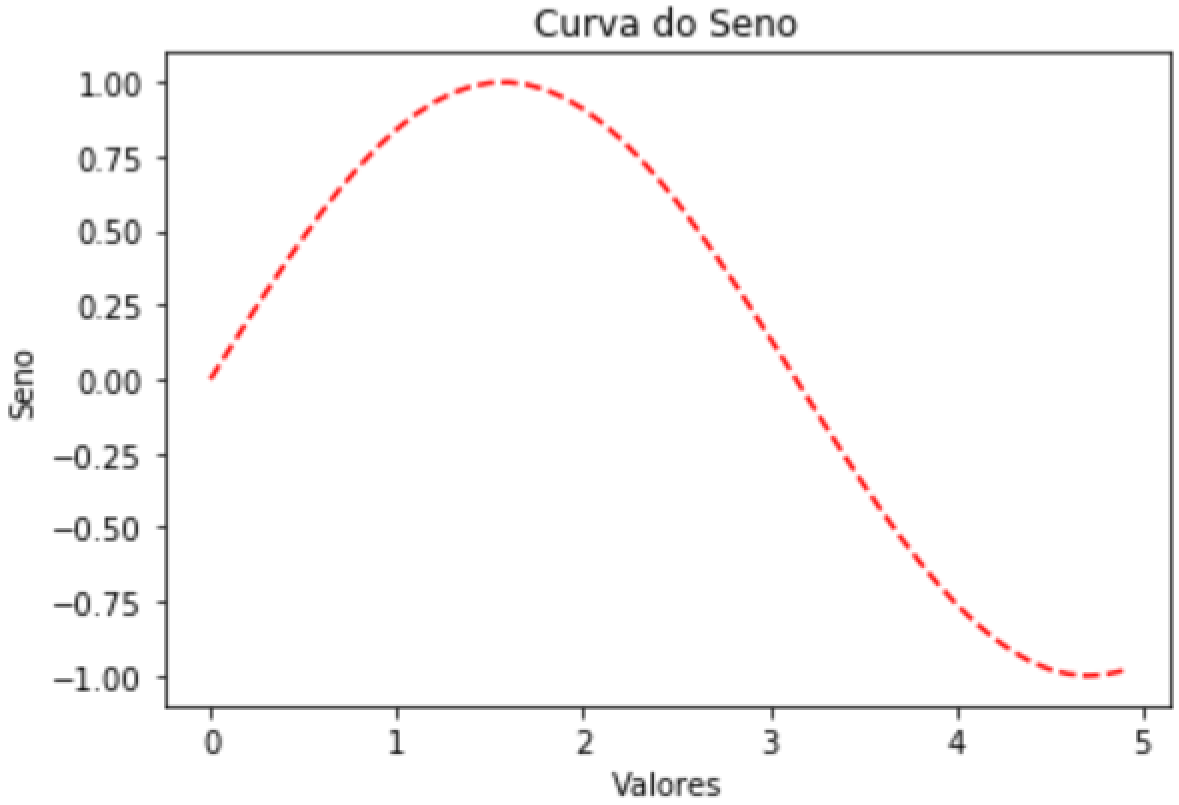
Matplotlib::Funcional

- Acessando atributos do *plot*



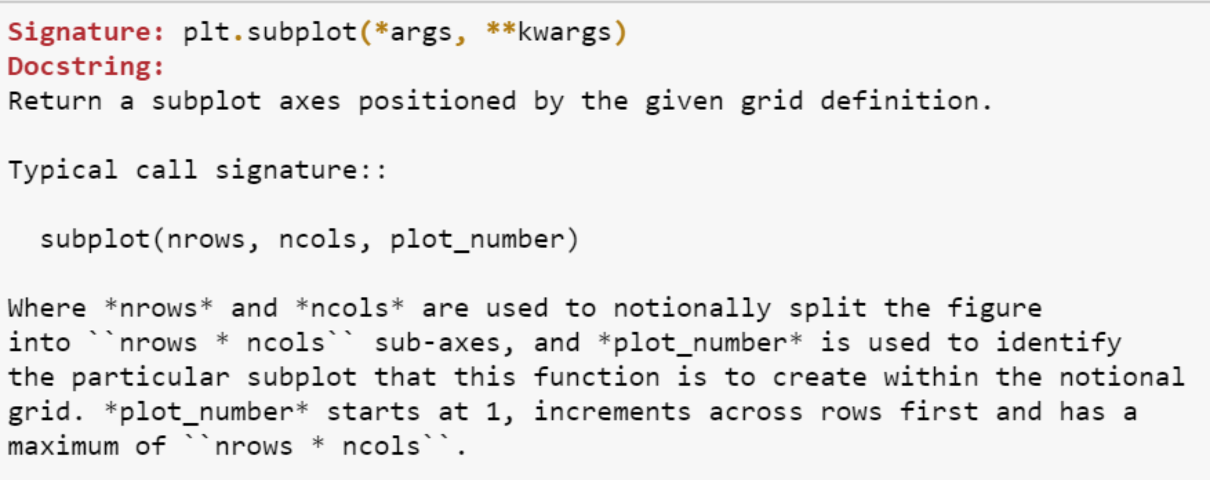
```
plt.plot(x, y, 'r--')
plt.xlabel('Valores')
plt.ylabel('Seno')
plt.title('Curva do Seno')
```

```
<matplotlib.text.Text at 0x2051997e7b8>
```

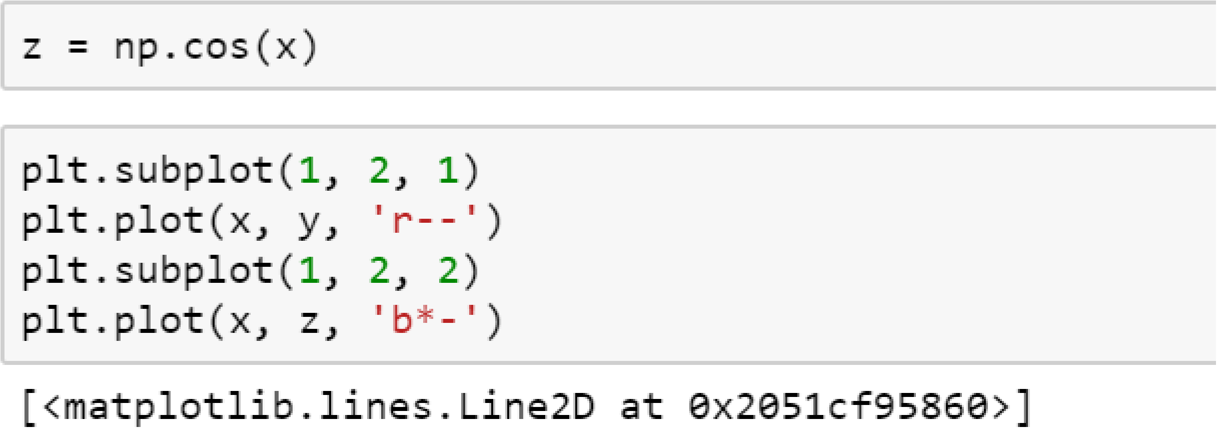


Matplotlib::Funcional

- Criando múltiplos plots



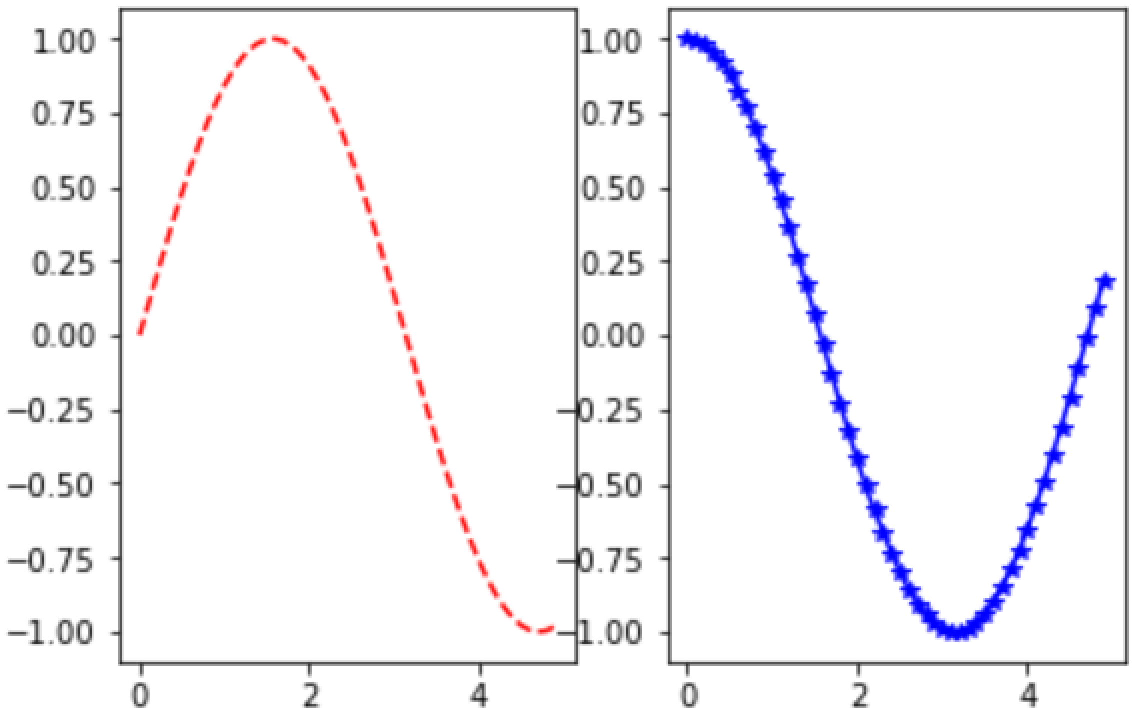
```
Signature: plt.subplot(*args, **kwargs)
Docstring:
Return a subplot axes positioned by the given grid definition.

Typical call signature::

  subplot(nrows, ncols, plot_number)

Where *nrows* and *ncols* are used to notionally split the figure
into ``nrows * ncols`` sub-axes, and *plot_number* is used to identify
the particular subplot that this function is to create within the notional
grid. *plot_number* starts at 1, increments across rows first and has a
maximum of ``nrows * ncols``.
```



```
z = np.cos(x)
```

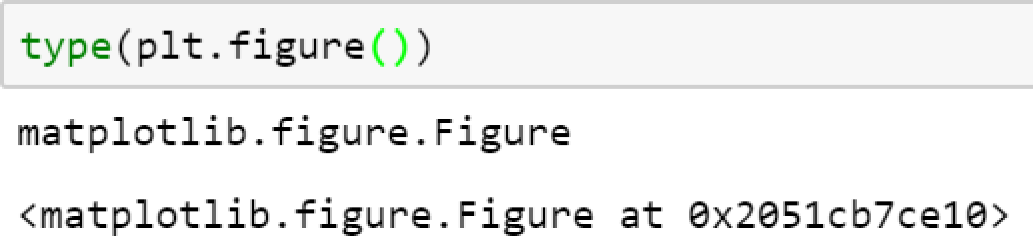
```
plt.subplot(1, 2, 1)
plt.plot(x, y, 'r--')
plt.subplot(1, 2, 2)
plt.plot(x, z, 'b*-')
```

```
[<matplotlib.lines.Line2D at 0x2051cf95860>]
```



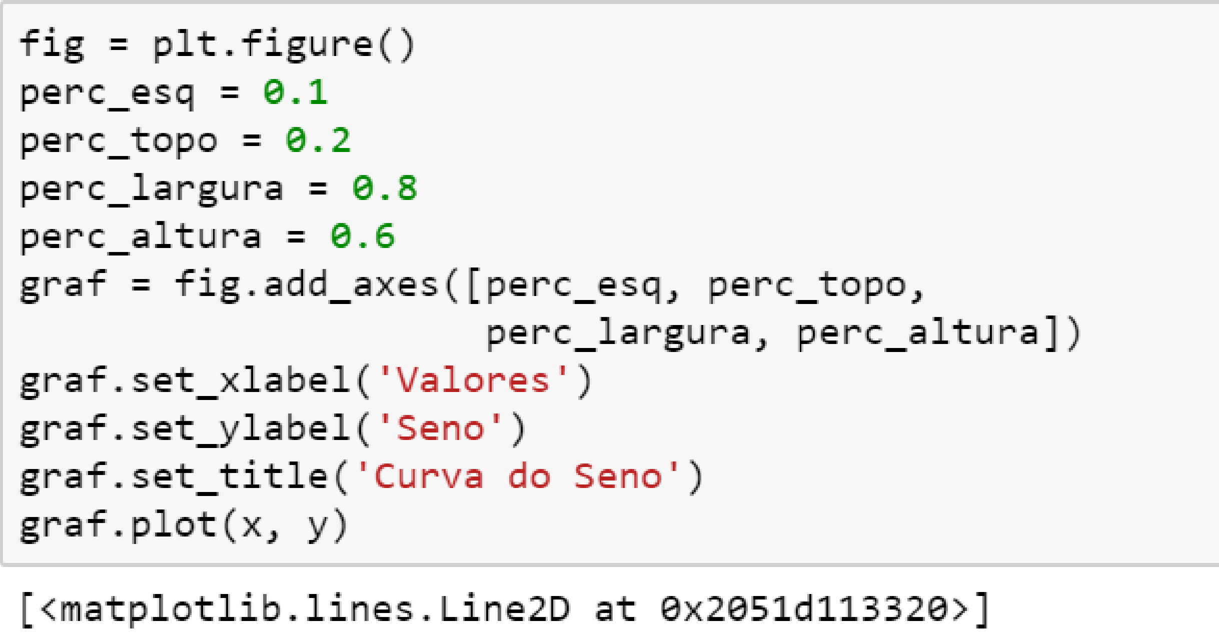
Matplotlib::OO

- Trabalhando com objetos
  - *A classe principal é figure*



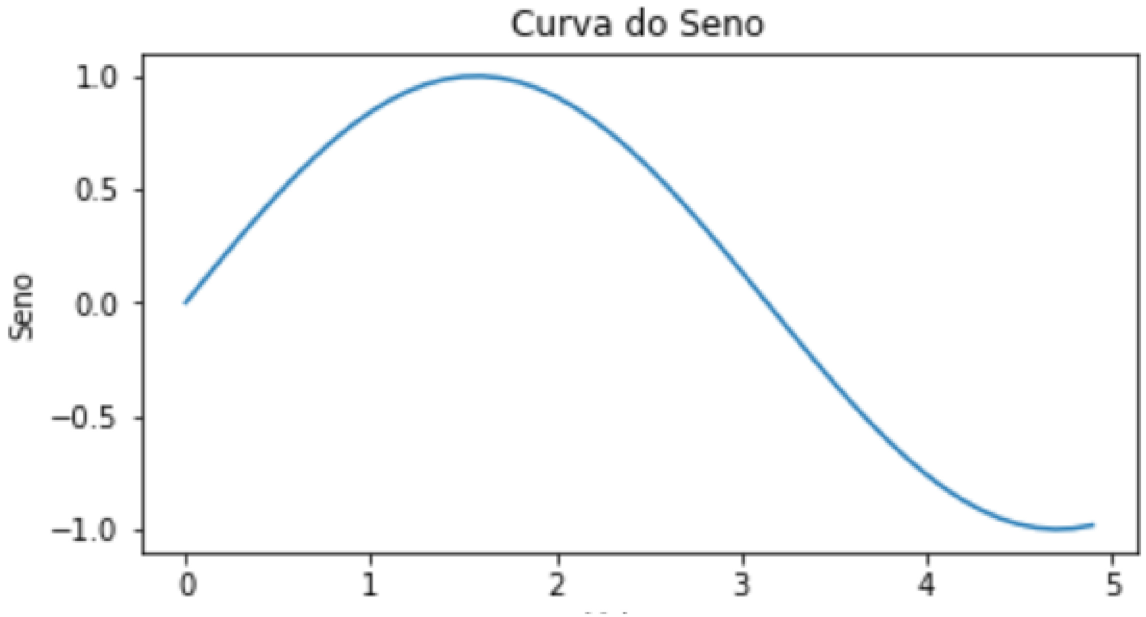
```
type(plt.figure())
```

```
matplotlib.figure.Figure

<matplotlib.figure.Figure at 0x2051cb7ce10>
```



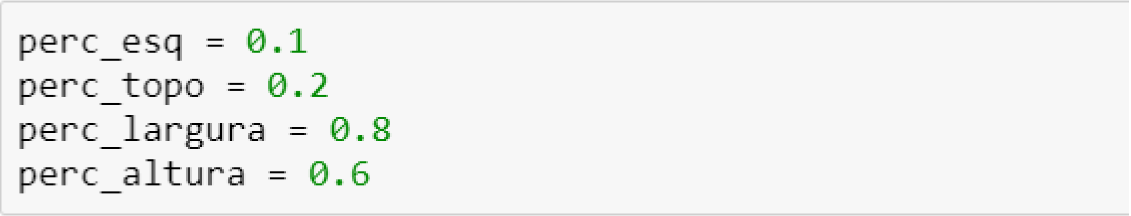
```
fig = plt.figure()
perc_esq = 0.1
perc_topo = 0.2
perc_largura = 0.8
perc_altura = 0.6
graf = fig.add_axes([perc_esq, perc_topo,
                     perc_largura, perc_altura])
graf.set_xlabel('Valores')
graf.set_ylabel('Seno')
graf.set_title('Curva do Seno')
graf.plot(x, y)
```

```
[<matplotlib.lines.Line2D at 0x2051d113320>]
```

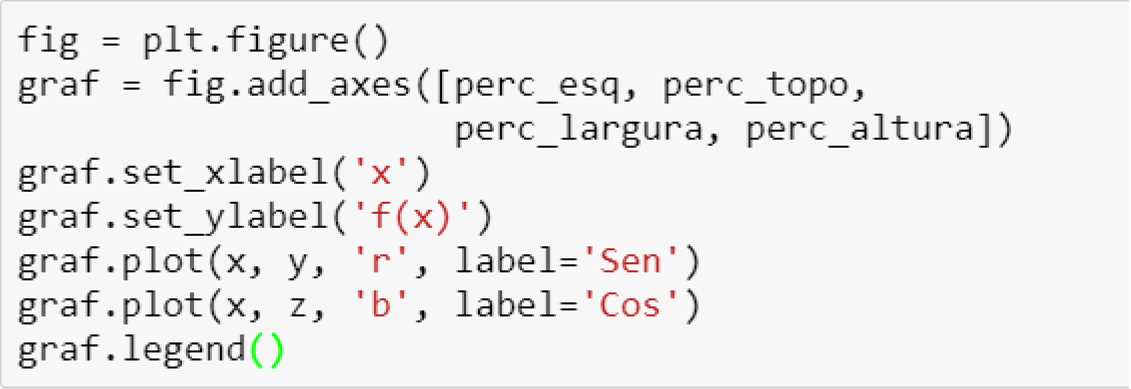


Matplotlib::OO

- Adicionando múltiplas curvas
  - *Basta plotar as novas curvas*

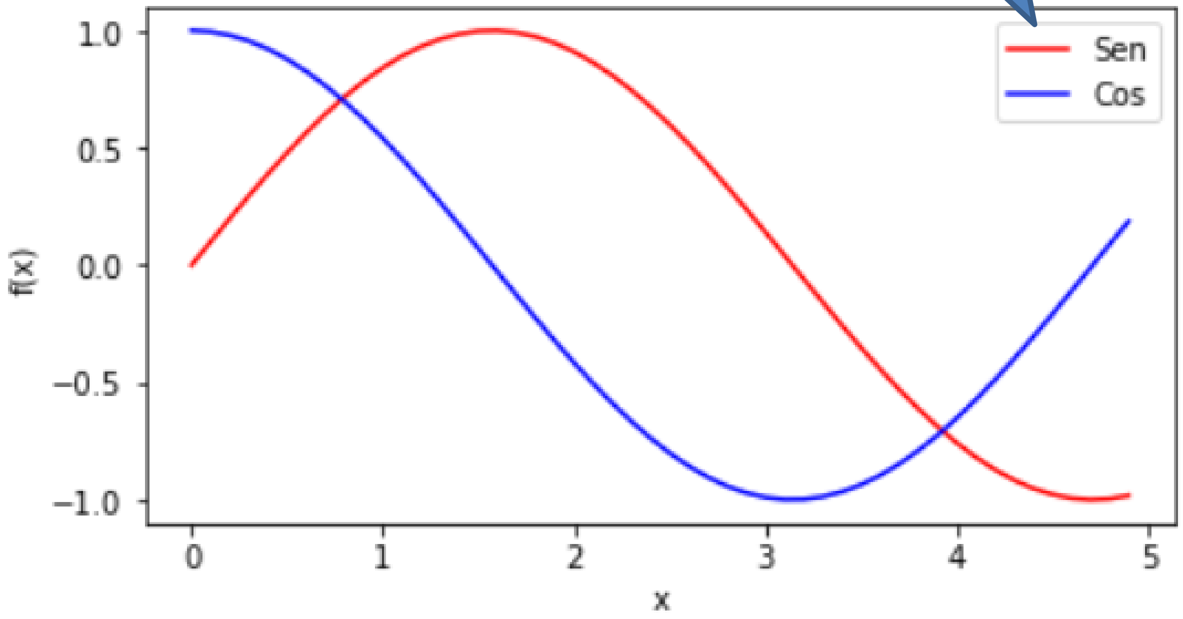


```
perc_esq = 0.1
perc_topo = 0.2
perc_largura = 0.8
perc_altura = 0.6
```



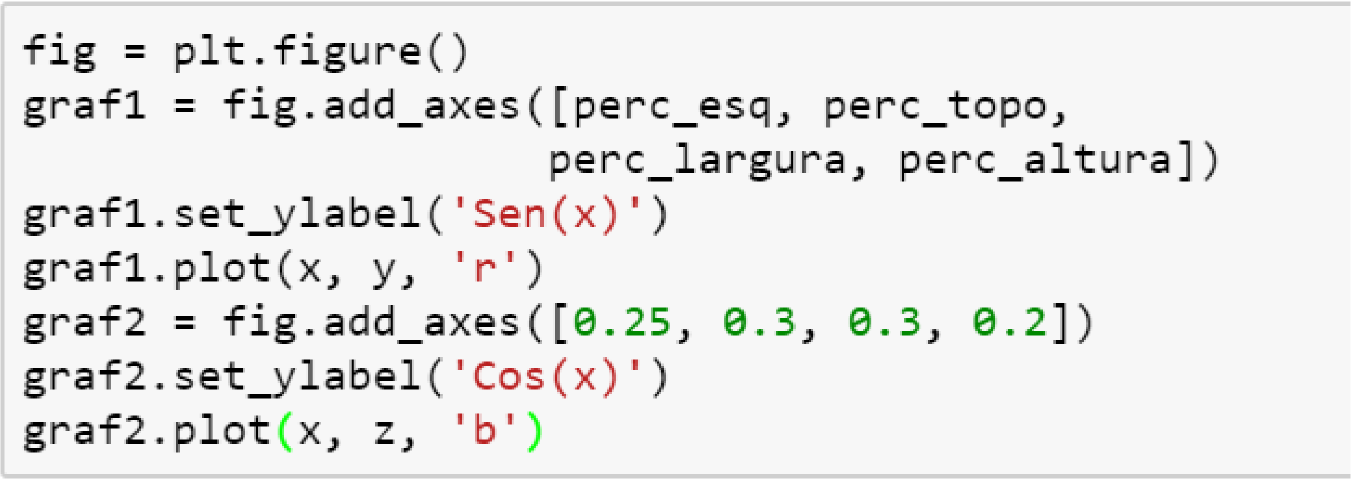
```
fig = plt.figure()
graf = fig.add_axes([perc_esq, perc_topo,
                     perc_largura, perc_altura])
graf.set_xlabel('x')
graf.set_ylabel('f(x)')
graf.plot(x, y, 'r', label='Sen')
graf.plot(x, z, 'b', label='Cos')
graf.legend()
```

Observe que adicionamos legendas às curvas!



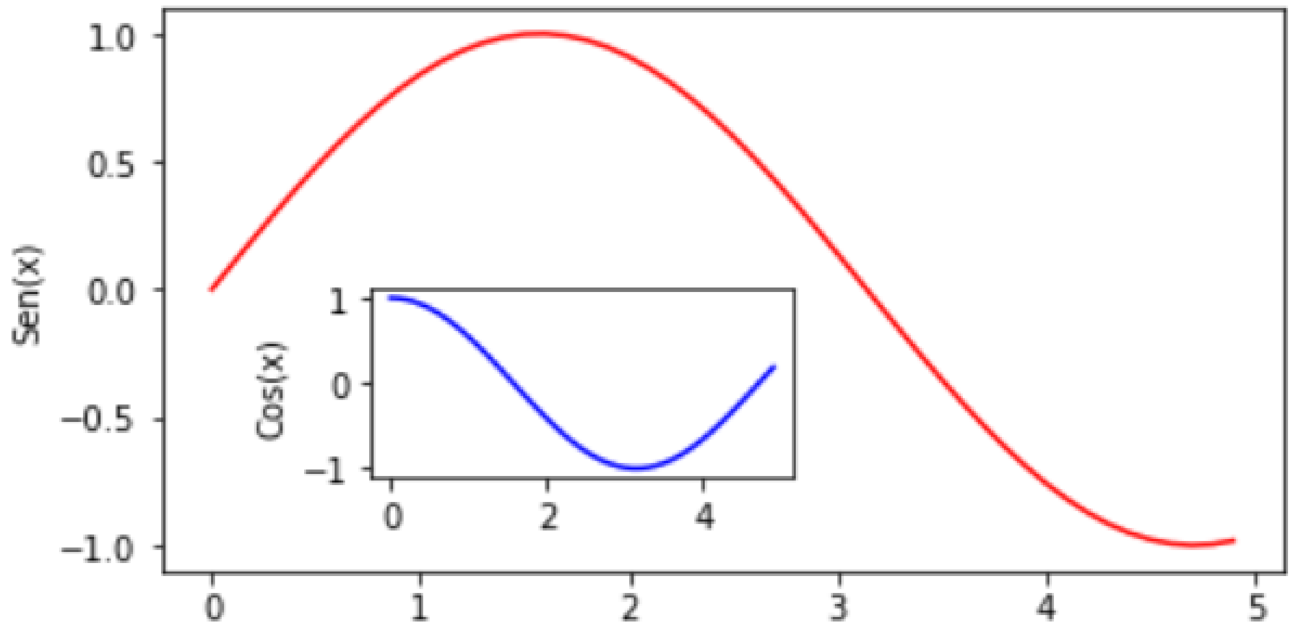
Matplotlib::OO

- Adicionando múltiplos gráficos
  - *Basta inserir o novo axes na figure*



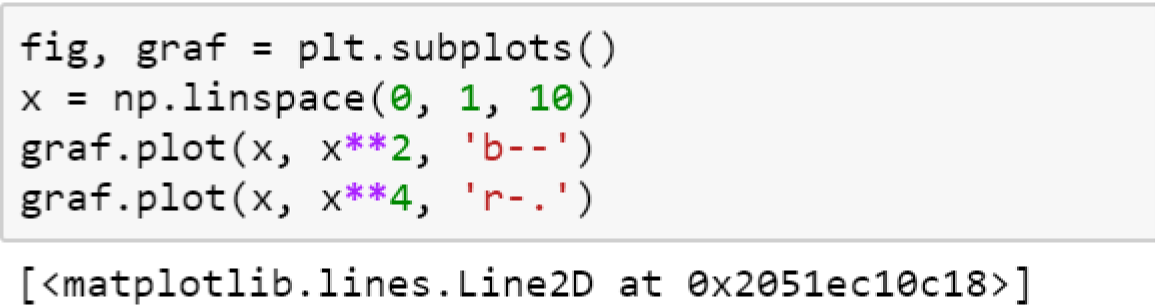
```
fig = plt.figure()
graf1 = fig.add_axes([perc_esq, perc_topo,
                      perc_largura, perc_altura])
graf1.set_ylabel('Sen(x)')
graf1.plot(x, y, 'r')
graf2 = fig.add_axes([0.25, 0.3, 0.3, 0.2])
graf2.set_ylabel('Cos(x)')
graf2.plot(x, z, 'b')
```

```
[<matplotlib.lines.Line2D at 0x2051e95fa20>]
```



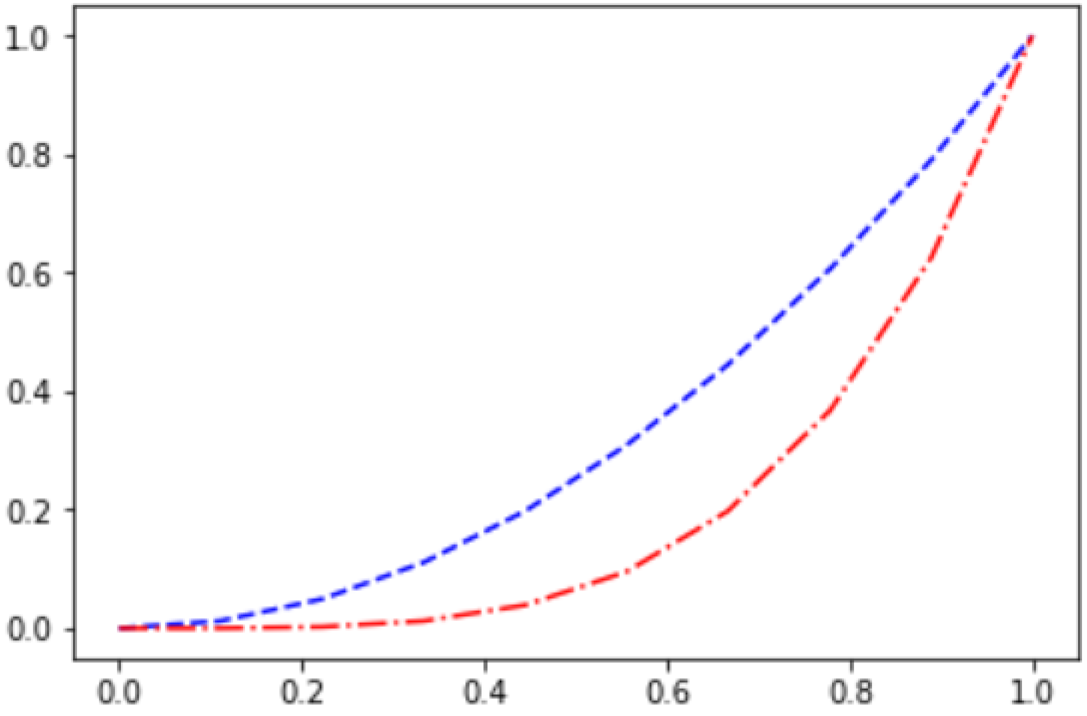
Matplotlib::OO

- Função *subplots*



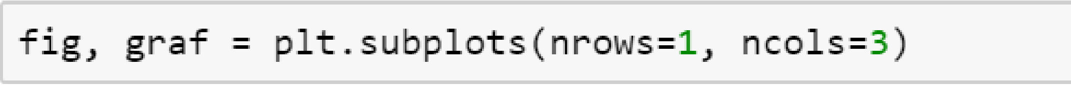
```
fig, graf = plt.subplots()
x = np.linspace(0, 1, 10)
graf.plot(x, x**2, 'b--')
graf.plot(x, x**4, 'r-.')
```

```
[<matplotlib.lines.Line2D at 0x2051ec10c18>]
```

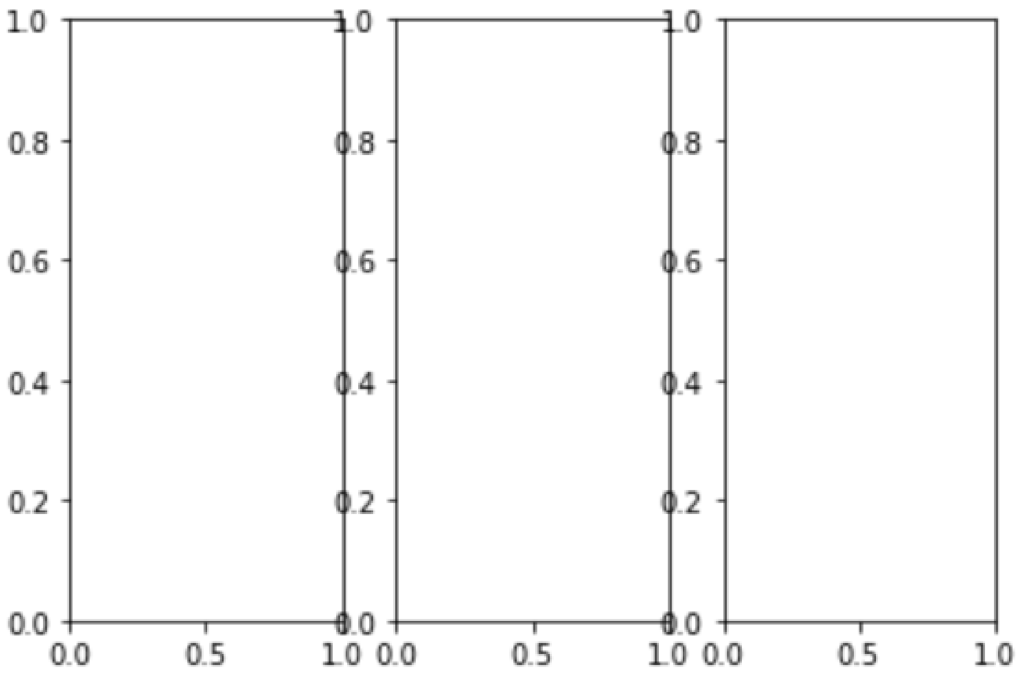


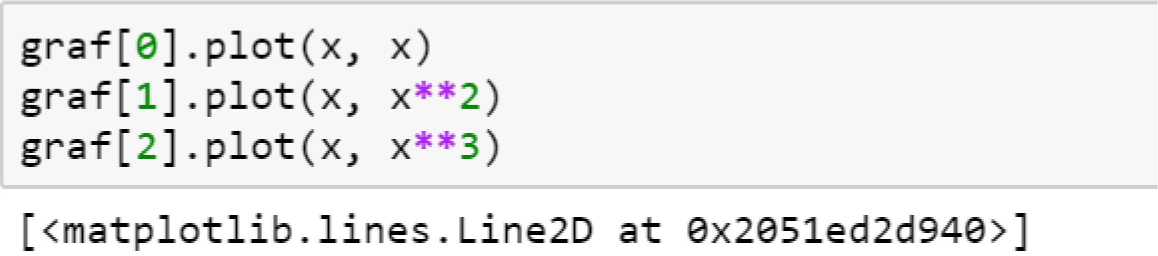
Matplotlib::OO

- Criando e acessando múltiplos *plots* com a função *subplots*



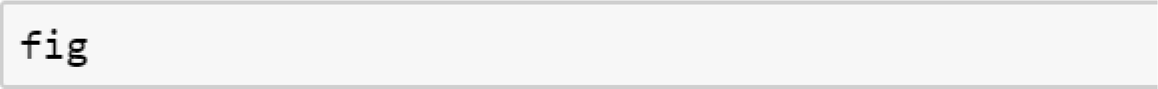
```
fig, graf = plt.subplots(nrows=1, ncols=3)
```



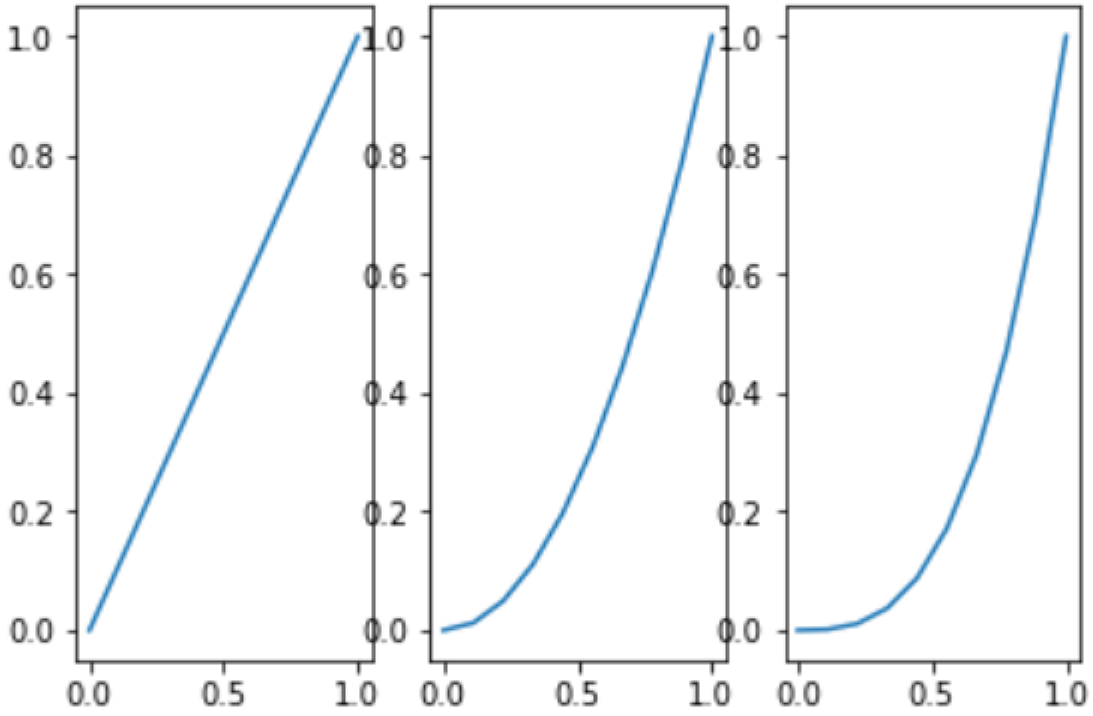


```
graf[0].plot(x, x)
graf[1].plot(x, x**2)
graf[2].plot(x, x**3)
```

```
[<matplotlib.lines.Line2D at 0x2051ed2d940>]
```

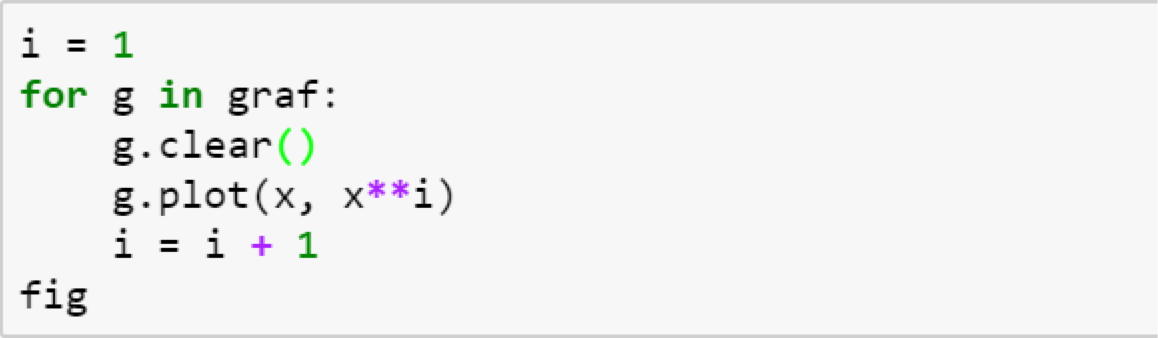


```
fig
```

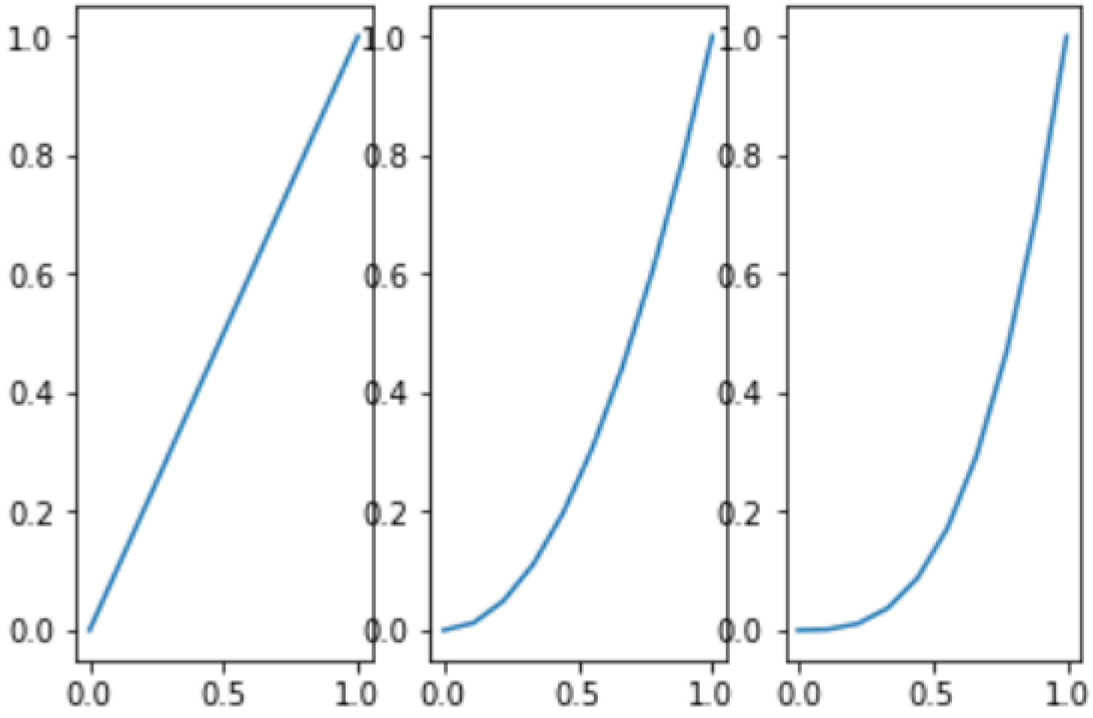


Matplotlib::OO

- Criando e acessando múltiplos *plots* com a função *subplots*

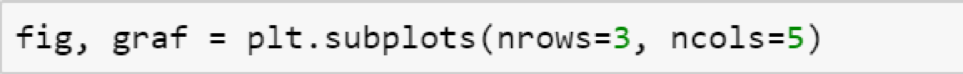


```
i = 1
for g in graf:
    g.clear()
    g.plot(x, x**i)
    i = i + 1
fig
```

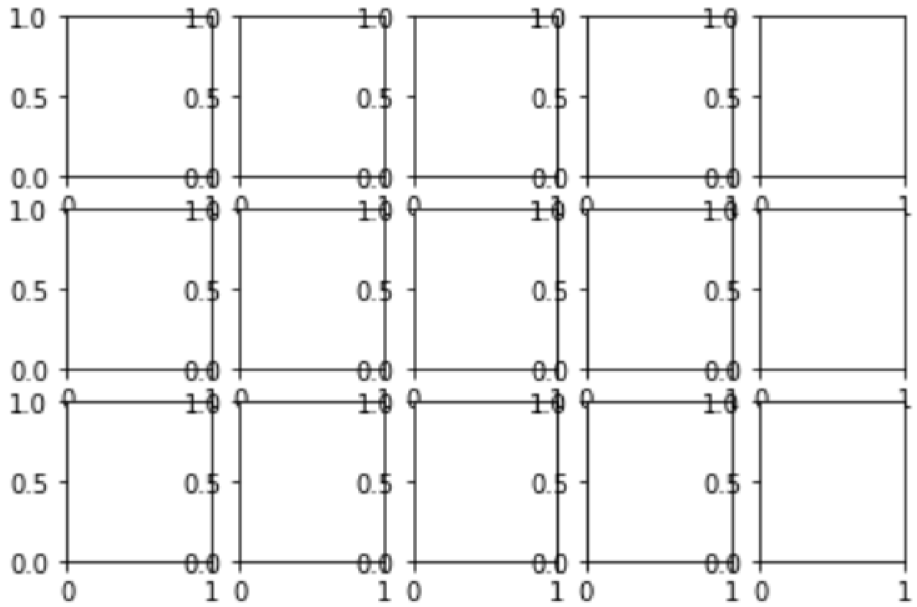


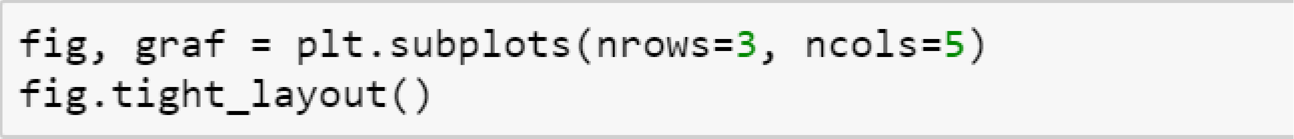
Matplotlib::OO

- Ajustando o layout na visualização dos *plots*

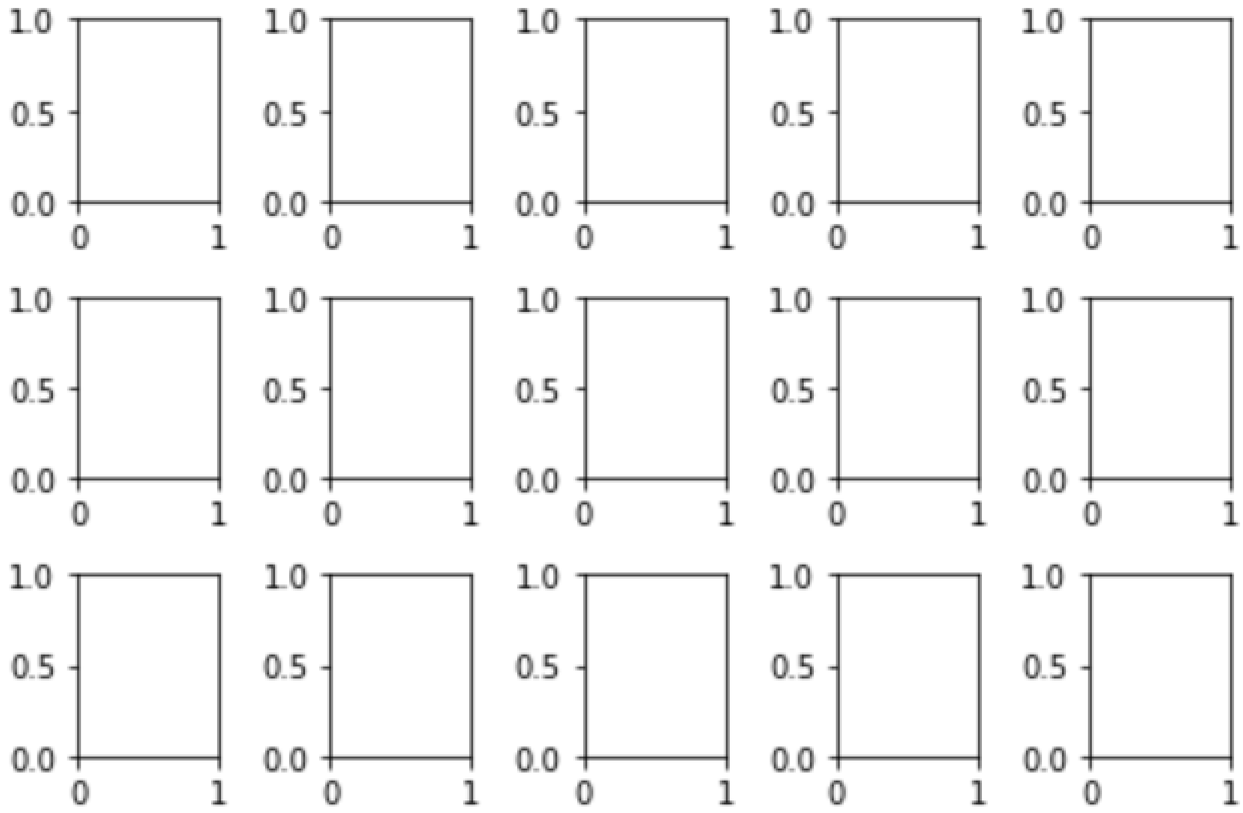


```
fig, graf = plt.subplots(nrows=3, ncols=5)
```



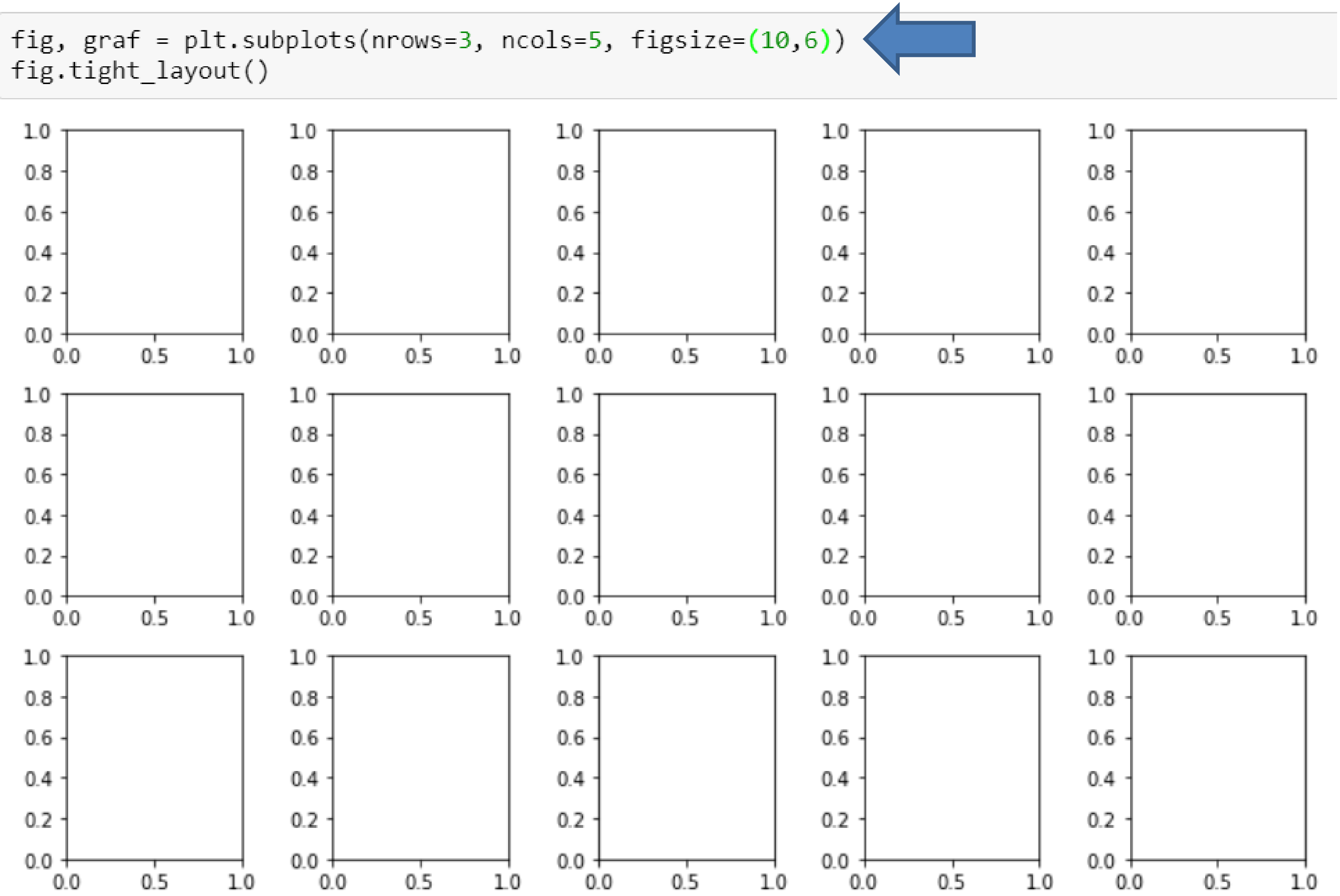


```
fig, graf = plt.subplots(nrows=3, ncols=5)
fig.tight_layout()
```



Matplotlib

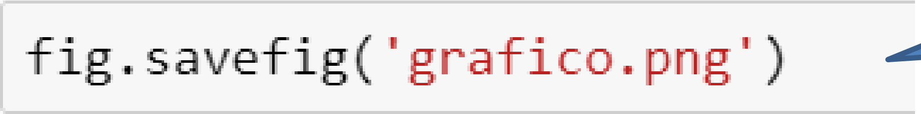
- Ajustando o layout na visualização dos *plots*



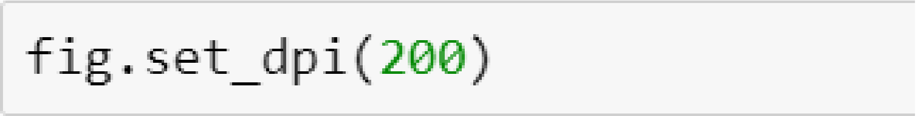
```
fig, graf = plt.subplots(nrows=3, ncols=5, figsize=(10,6))
fig.tight_layout()
```

Matplotlib

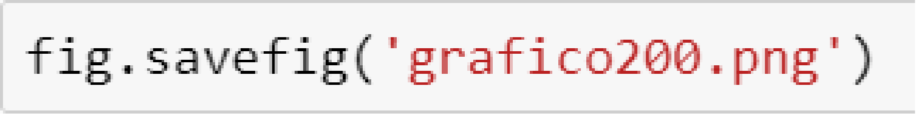
- Exportando a *figure* para arquivo
  - *savefig*



```python
fig.savefig('grafico.png')
```



```python
fig.set_dpi(200)
```

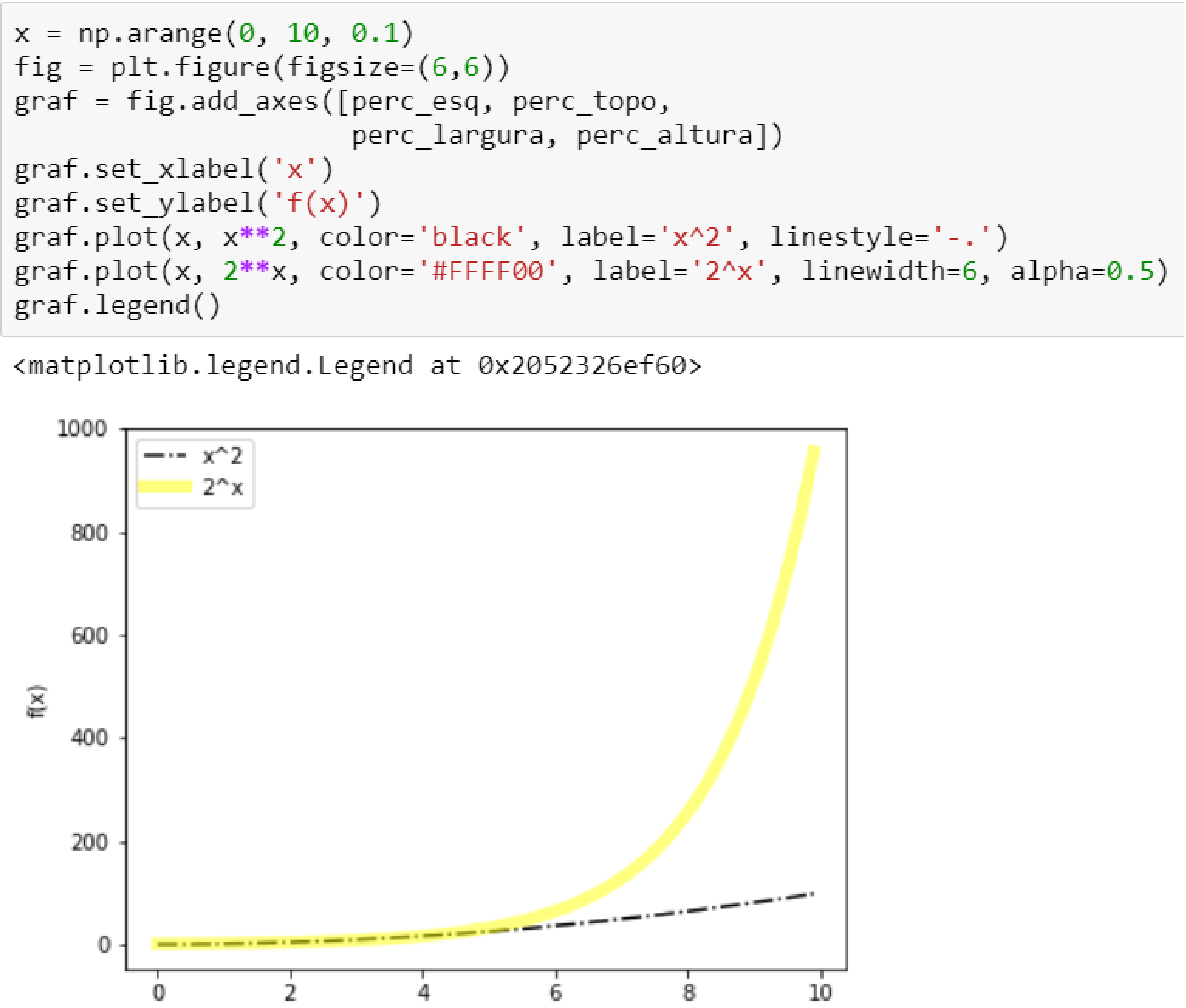


```python
fig.savefig('grafico200.png')
```

O padrão é exportar em 72 DPI. Para salvar em outros formatos de arquivo, basta mudar a extensão (jpg, tif, bmp, png).

Matplotlib

- Personalizando as curvas na visualização dos *plots*
  - *color*
  - *linewidth (lw)*
  - *linestyle (ls)*
  - *alpha*
  - *marker*
  - *markersize*
  - ...



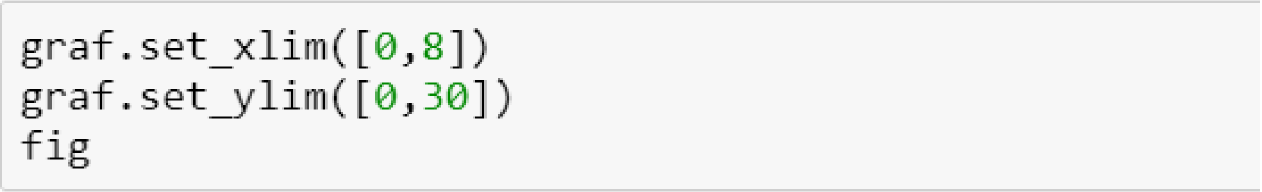
```
x = np.arange(0, 10, 0.1)
fig = plt.figure(figsize=(6,6))
graf = fig.add_axes([perc_esq, perc_topo,
                     perc_largura, perc_altura])
graf.set_xlabel('x')
graf.set_ylabel('f(x)')
graf.plot(x, x**2, color='black', label='x^2', linestyle='-.')
graf.plot(x, 2**x, color='#FFFF00', label='2^x', linewidth=6, alpha=0.5)
graf.legend()
```

```
<matplotlib.legend.Legend at 0x2052326ef60>
```

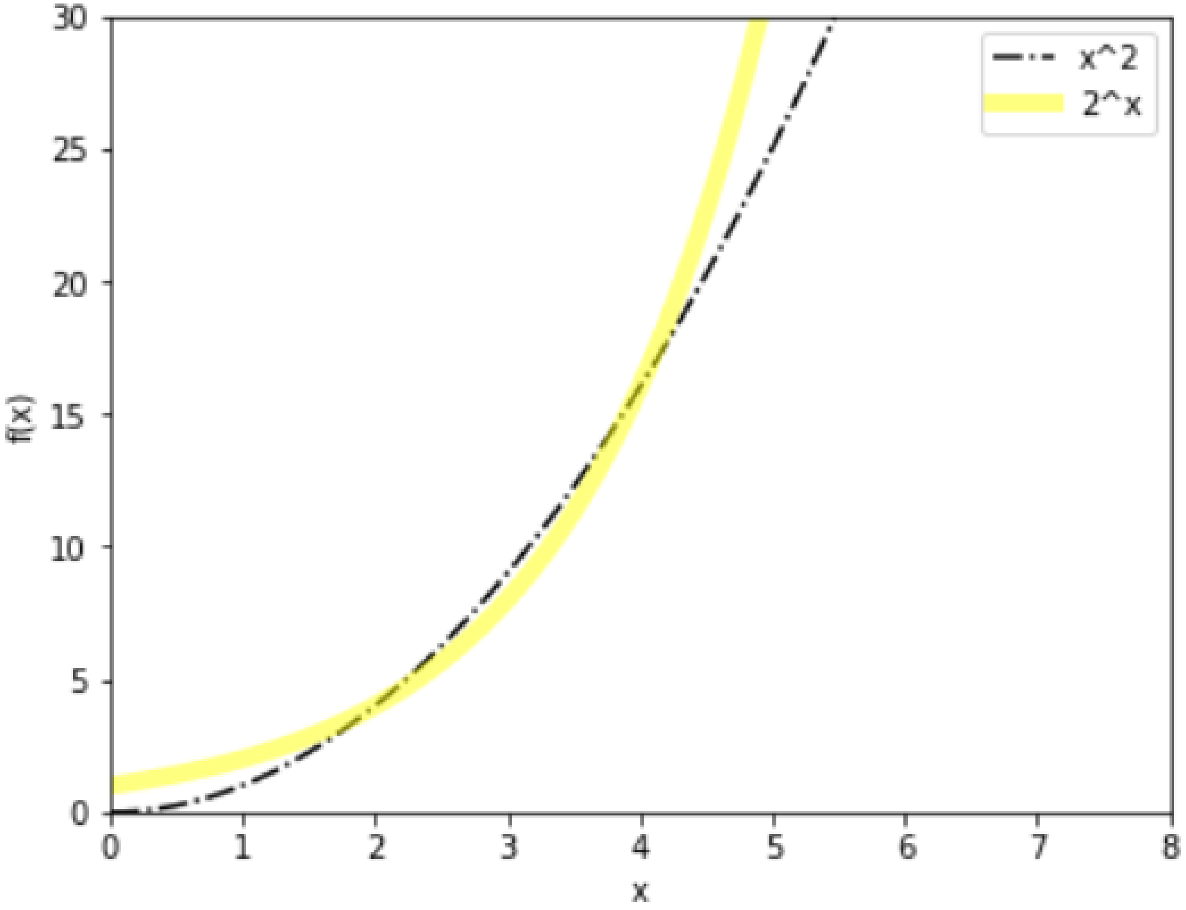
Consulte:
https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html#matplotlib.pyplot.plot

Matplotlib

- Definindo o limite das curvas na visualização dos *plots*
  - *set_xlim*
  - *set_ylim*

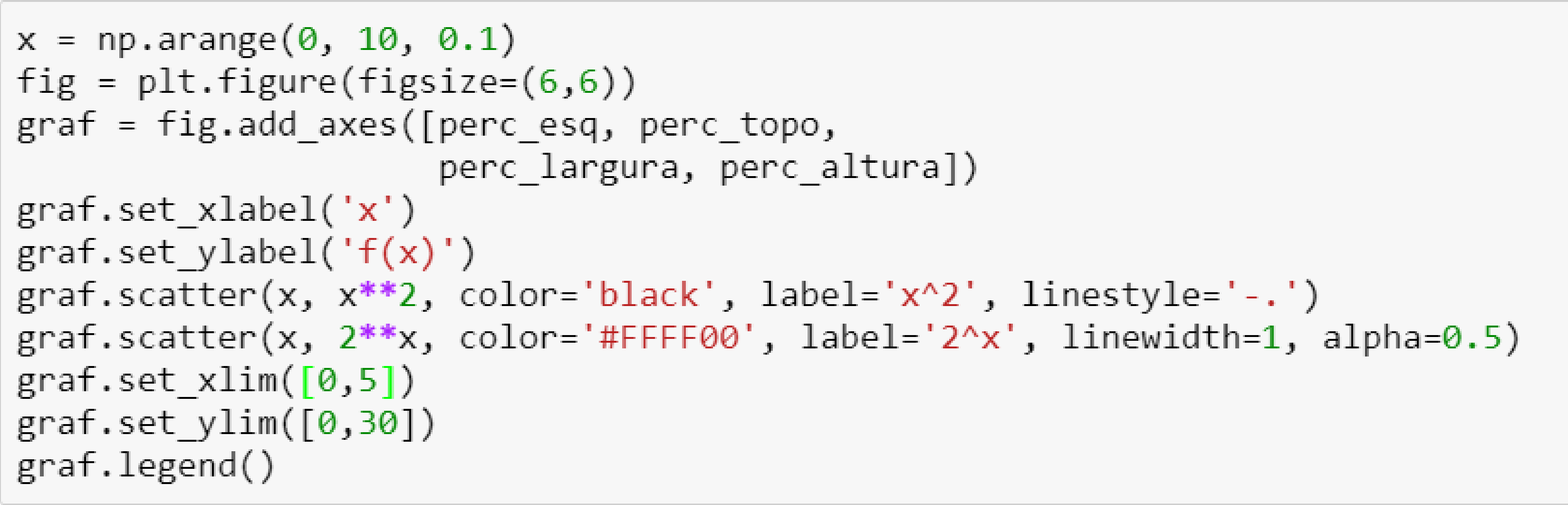


```
graf.set_xlim([0,8])
graf.set_ylim([0,30])
fig
```



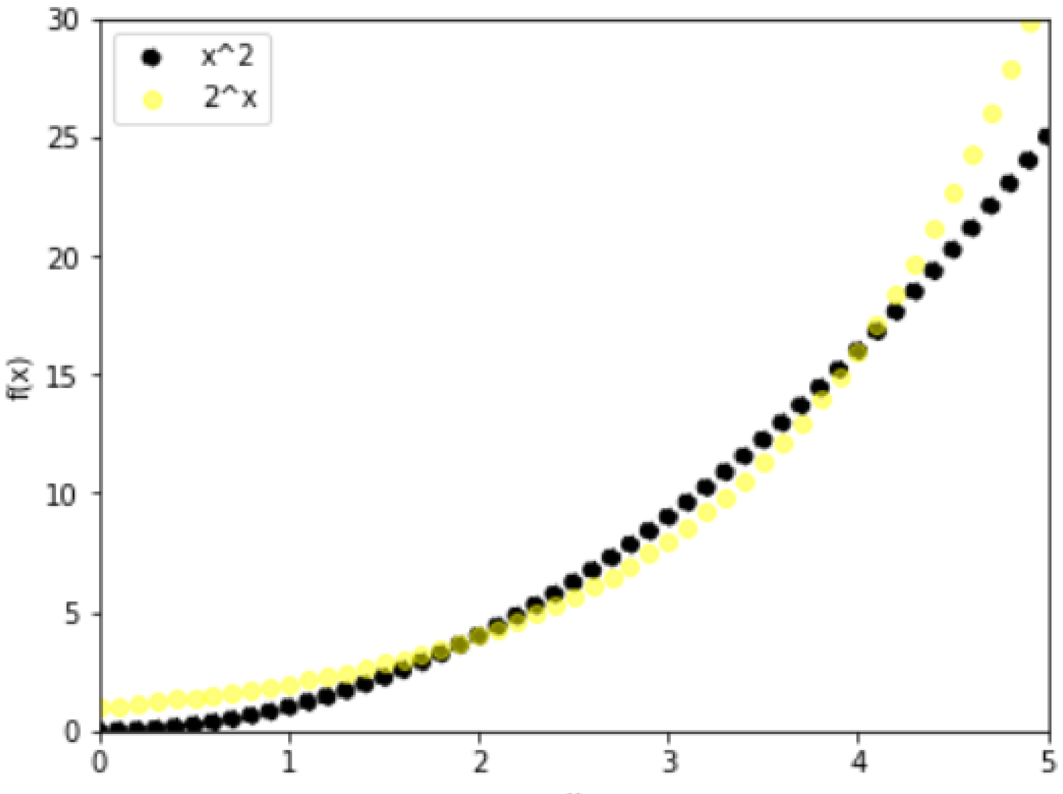
Matplotlib

- Outros tipos de gráficos:
  - *scatter*



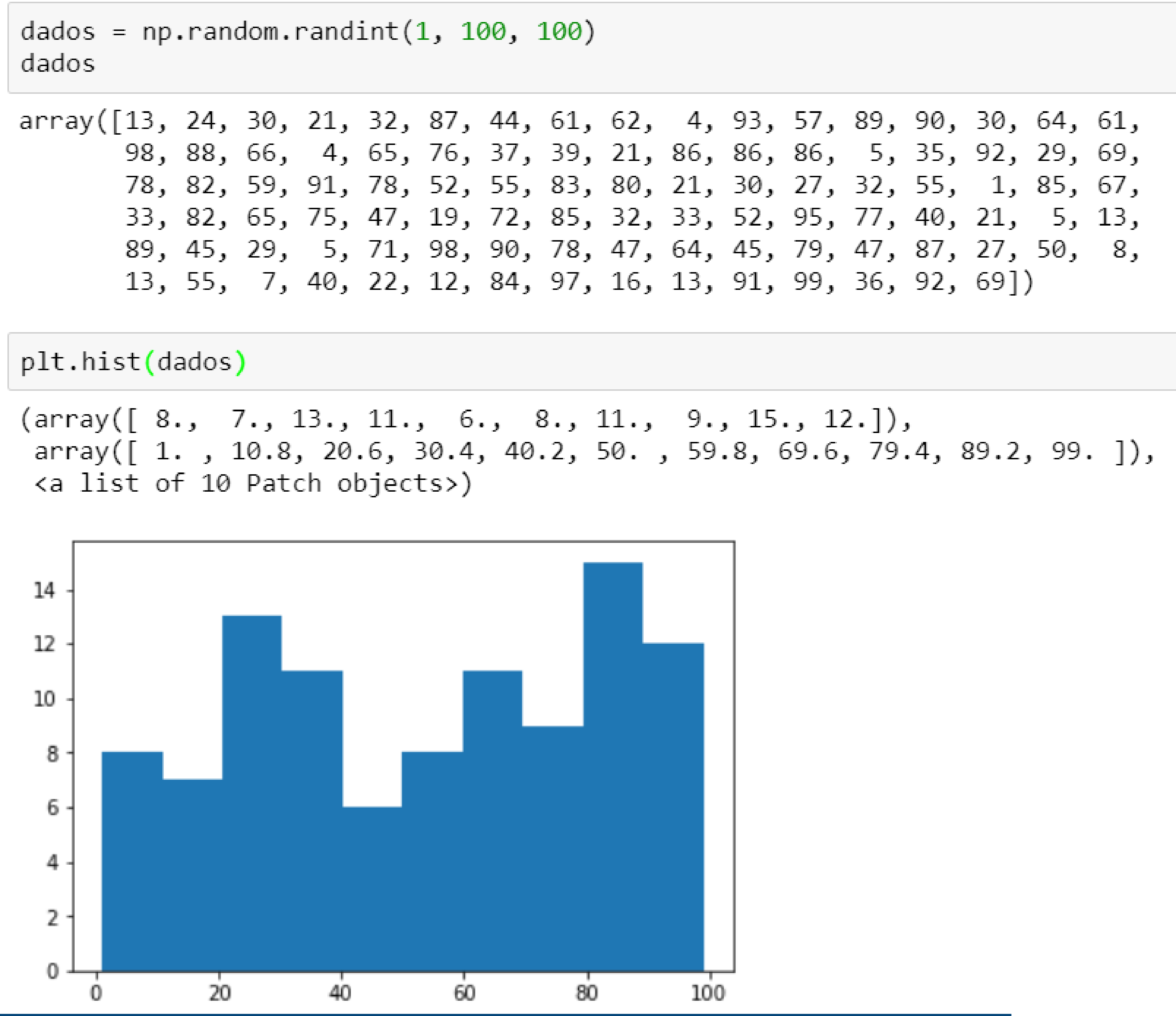
```
x = np.arange(0, 10, 0.1)
fig = plt.figure(figsize=(6,6))
graf = fig.add_axes([perc_esq, perc_topo,
                     perc_largura, perc_altura])
graf.set_xlabel('x')
graf.set_ylabel('f(x)')
graf.scatter(x, x**2, color='black', label='x^2', linestyle='-.')
graf.scatter(x, 2**x, color='#FFFF00', label='2^x', linewidth=1, alpha=0.5)
graf.set_xlim([0,5])
graf.set_ylim([0,30])
graf.legend()
```

```
<matplotlib.legend.Legend at 0x20523531e10>
```



Matplotlib

- Outros tipos de gráficos:
  - *hist*



```
dados = np.random.randint(1, 100, 100)
dados
```

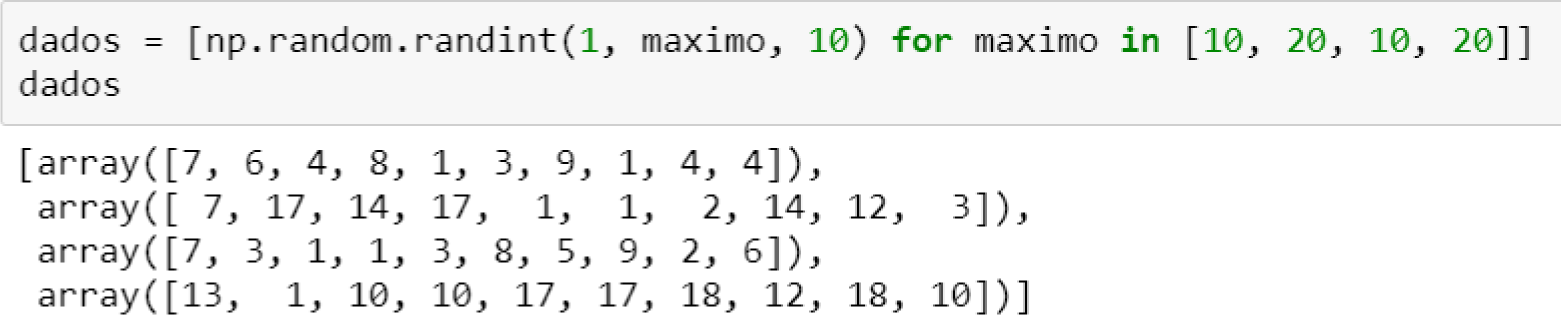
```
array([13, 24, 30, 21, 32, 87, 44, 61, 62,  4, 93, 57, 89, 90, 30, 64, 61,
       98, 88, 66,  4, 65, 76, 37, 39, 21, 86, 86, 86,  5, 35, 92, 29, 69,
       78, 82, 59, 91, 78, 52, 55, 83, 80, 21, 30, 27, 32, 55,  1, 85, 67,
       33, 82, 65, 75, 47, 19, 72, 85, 32, 33, 52, 95, 77, 40, 21,  5, 13,
       89, 45, 29,  5, 71, 98, 90, 78, 47, 64, 45, 79, 47, 87, 27, 50,  8,
       13, 55,  7, 40, 22, 12, 84, 97, 16, 13, 91, 99, 36, 92, 69])
```

```
plt.hist(dados)
```

```
(array([ 8.,  7., 13., 11.,  6.,  8., 11.,  9., 15., 12.]),
 array([ 1. , 10.8, 20.6, 30.4, 40.2, 50. , 59.8, 69.6, 79.4, 89.2, 99. ]),
 <a list of 10 Patch objects>)
```

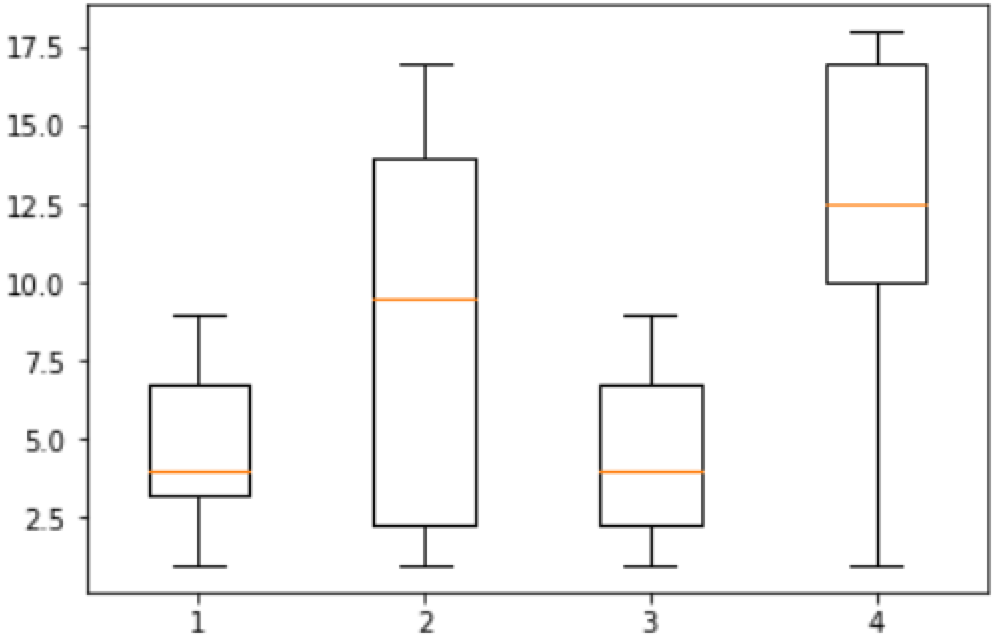
Matplotlib

- Outros tipos de gráficos:
  - *boxplot*



```
dados = [np.random.randint(1, maximo, 10) for maximo in [10, 20, 10, 20]]
dados
```

```
[array([7, 6, 4, 8, 1, 3, 9, 1, 4, 4]),
 array([ 7, 17, 14, 17,  1,  1,  2, 14, 12,  3]),
 array([7, 3, 1, 1, 3, 8, 5, 9, 2, 6]),
 array([13,  1, 10, 10, 17, 17, 18, 12, 18, 10])]
```



Referências

- https://matplotlib.org/gallery/index.html
- http://www.labri.fr/perso/nrougier/teaching/matplotlib/
- http://scipy-lectures.org/matplotlib/matplotlib.html