Anaconda

SURP 2022 Python Bootcamp

Ohio State Astronomy

Slides by: James W. Johnson

Anaconda

- These really big – no seriously like *really* big – snakes from South America
- Like 30 feet long dude
- Don't hurt people as much as the movies would suggest, but still a few meanies out there

Whoops Wrong Anaconda

A package manager for python

- Terminal: *conda*
- Fulfills a similar purpose as *pip*

Comes with NumPy, SciPy, Matplotlib, Pandas, etc.

Even if you didn’t know about anaconda, chances are you’ve used these packages in the past.

NumPy

Perhaps the most widely used python distribution

Contains

- An N-dimensional array object
- A highly optimized mathematical library (including linear algebra routines)
- Fast Fourier Transforms (FFTs)

Some common functions: linspace, logspace, arange, sin/cos/tan/etc., where, genfromtxt, savetxt, argsort, isnan/isinf/isreal, zeros, histogram, linalg.X, and the list definitely goes on.

SciPy

Widely used throughout STEM fields

Contains

- Numerical integration
- Interpolation
- Optimization
- Linear Algebra
- Statistics

Some popular functions: interpolate.X, optimize.X, integrate.X

Matplotlib

One of the most widely used plotting packages

- Seaborn is also popular

If you’re using matplotlib as an astronomer, chances are you’re using pyplot.

Some popular functions: pyplot.plot, pyplot.scatter, pyplot.errorbar, pyplot.imshow

Bottom Line

You have a wealth of tools at your disposal through Anaconda.

Questions about how to use a specific function and what happens under the hood can usually be answered by its documentation.

With the number of tools available, there are many ways to write the same program.

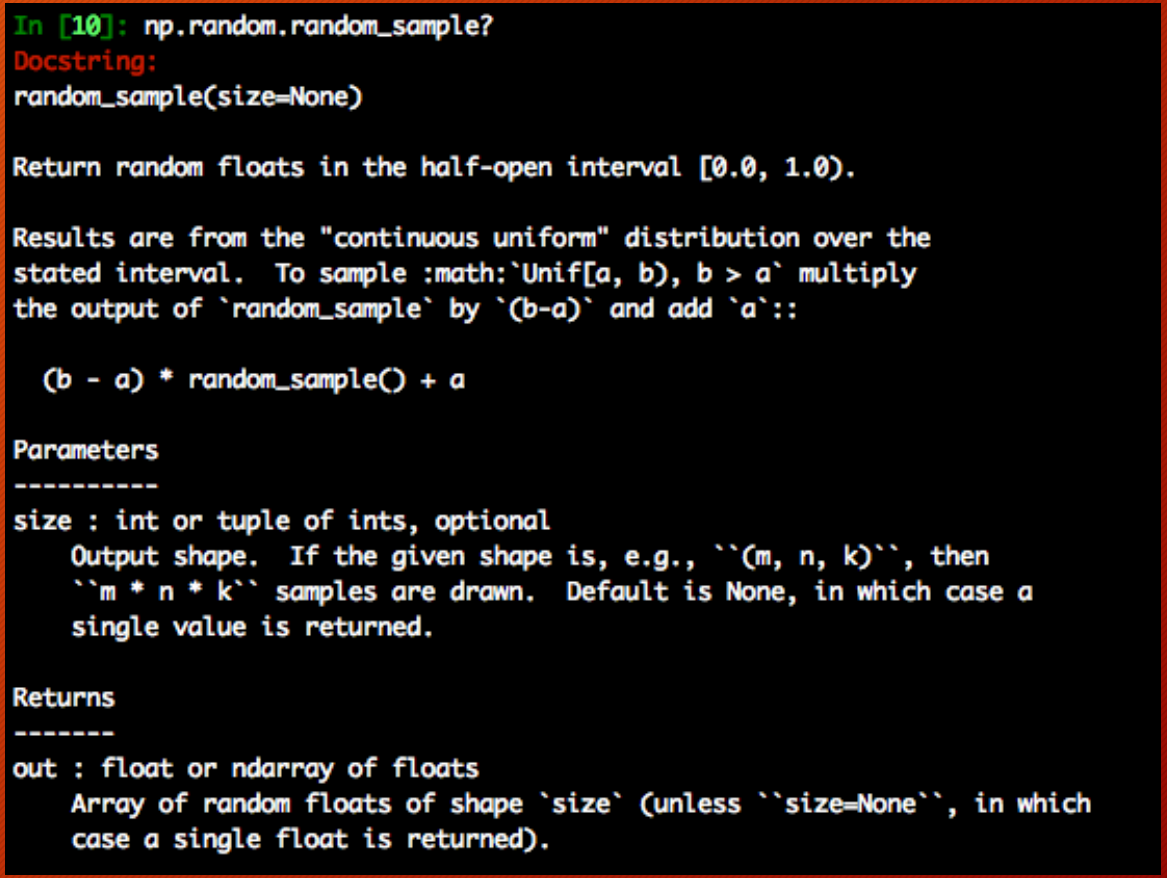
Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample

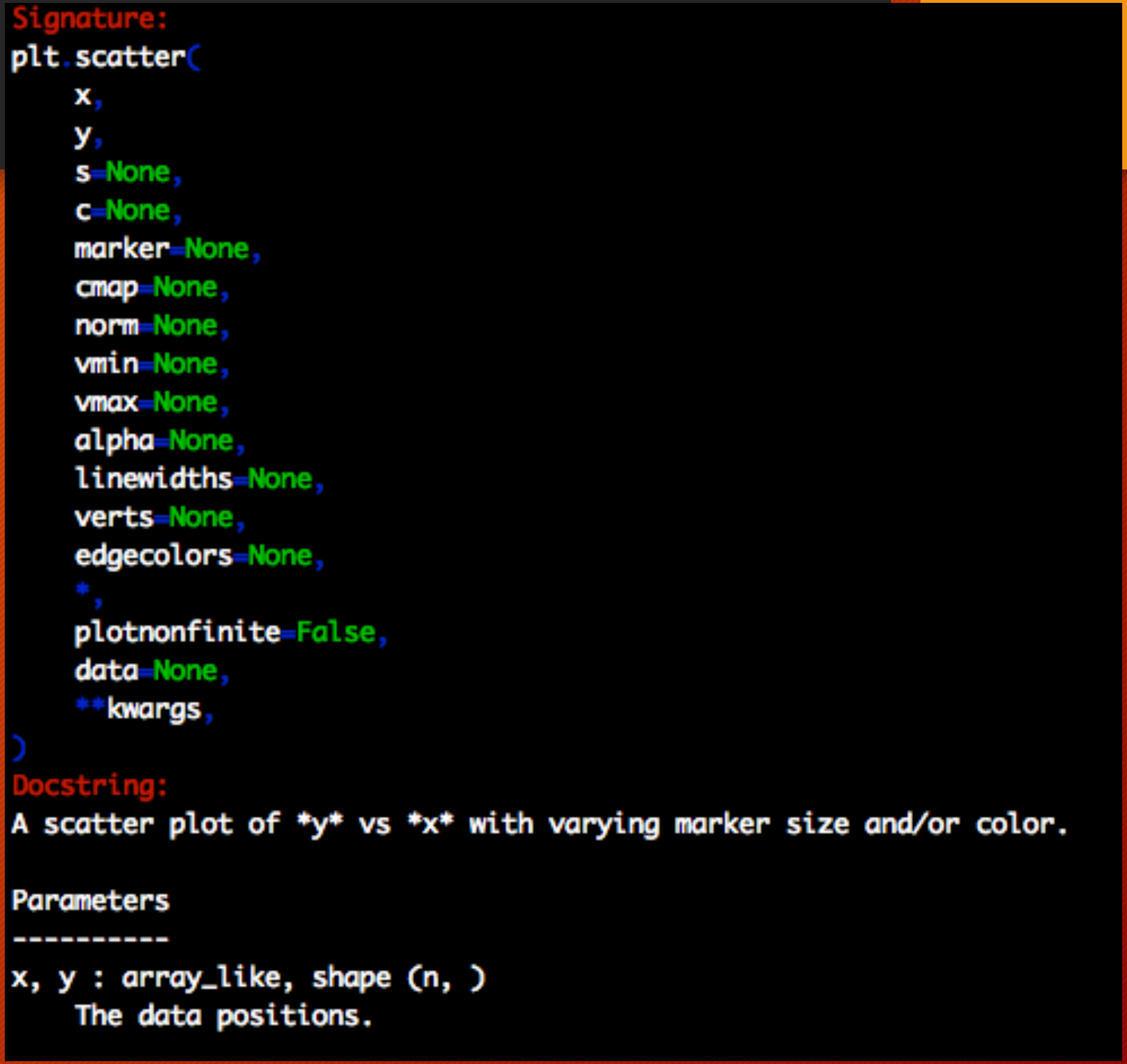


```
In [10]: np.random.random_sample?
Docstring:
random_sample(size=None)

Return random floats in the half-open interval [0.0, 1.0).

Results are from the "continuous uniform" distribution over the
stated interval.  To sample :math:`Unif[a, b), b > a` multiply
the output of `random_sample` by `(b-a)` and add `a`::

  (b - a) * random_sample() + a

Parameters
----------
size : int or tuple of ints, optional
    Output shape.  If the given shape is, e.g., ``(m, n, k)``, then
    ``m * n * k`` samples are drawn.  Default is None, in which case a
    single value is returned.

Returns
-------
out : float or ndarray of floats
    Array of random floats of shape `size` (unless ``size=None``, in which
    case a single float is returned).
```

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample
- matplotlib.pyplot.scatter

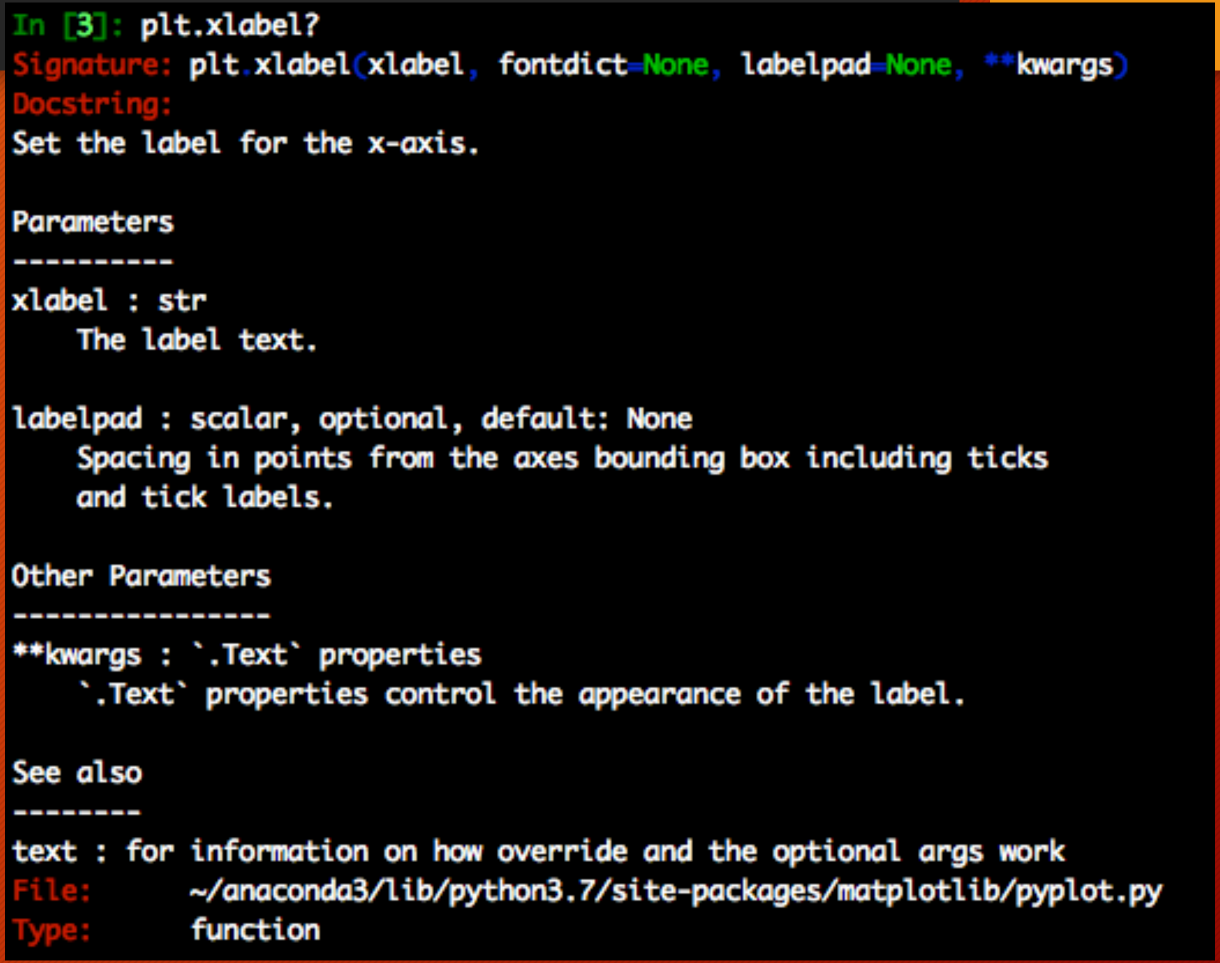


```
Signature:
plt.scatter(
    x,
    y,
    s=None,
    c=None,
    marker=None,
    cmap=None,
    norm=None,
    vmin=None,
    vmax=None,
    alpha=None,
    linewidths=None,
    verts=None,
    edgecolors=None,
    *,
    plotnonfinite=False,
    data=None,
    **kwargs,
)
Docstring:
A scatter plot of *y* vs *x* with varying marker size and/or color.

Parameters
----------
x, y : array_like, shape (n, )
    The data positions.
```

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample
- matplotlib.pyplot.scatter
- matplotlib.pyplot.xlabel

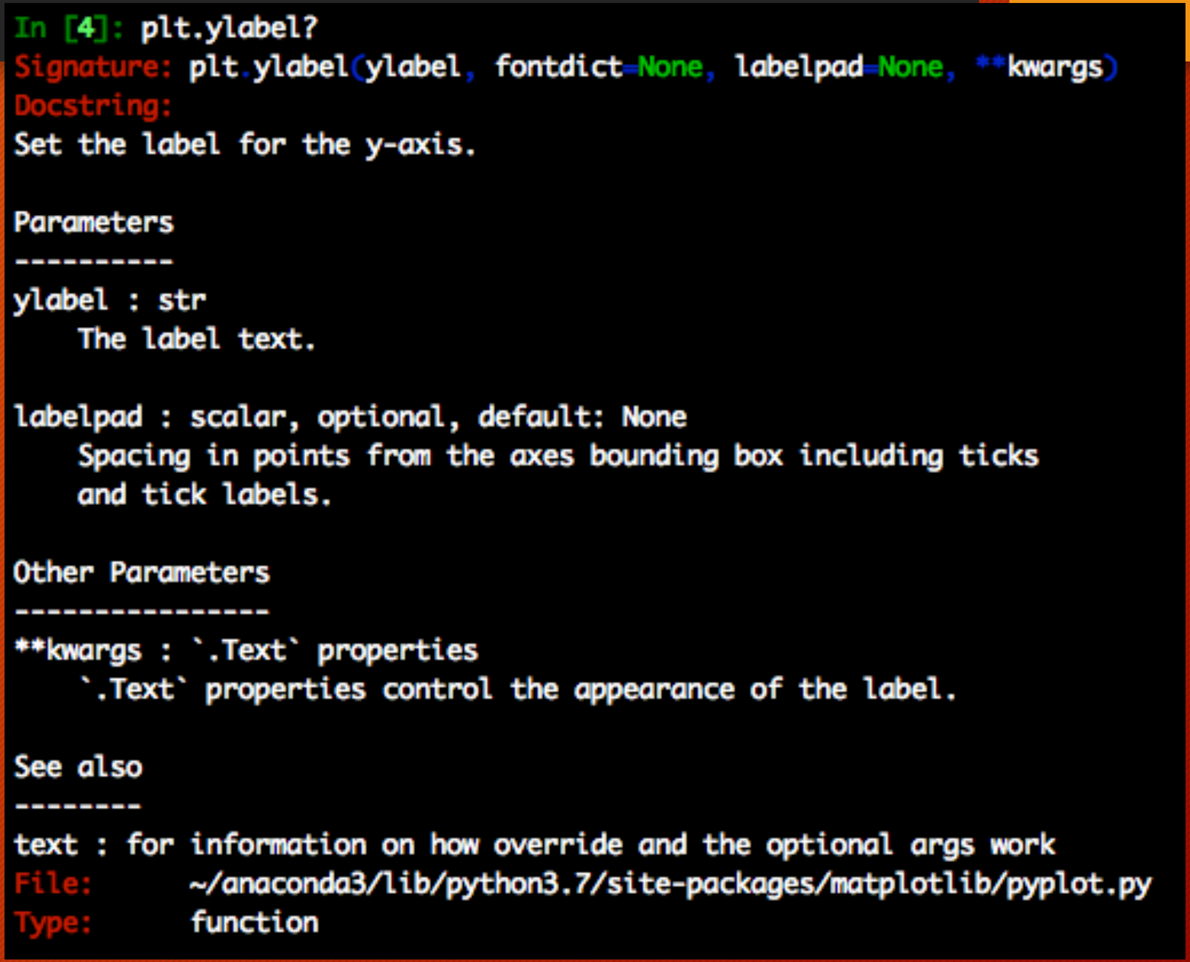


```
In [3]: plt.xlabel?
Signature: plt.xlabel(xlabel, fontdict=None, labelpad=None, **kwargs)
Docstring:
Set the label for the x-axis.

Parameters
----------
xlabel : str
    The label text.

labelpad : scalar, optional, default: None
    Spacing in points from the axes bounding box including ticks
    and tick labels.

Other Parameters
----------------
**kwargs : `.Text` properties
    `.Text` properties control the appearance of the label.

See also
--------
text : for information on how override and the optional args work
File:      ~/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py
Type:      function
```

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample
- matplotlib.pyplot.scatter
- matplotlib.pyplot.xlabel
- matplotlib.pyplot.ylabel



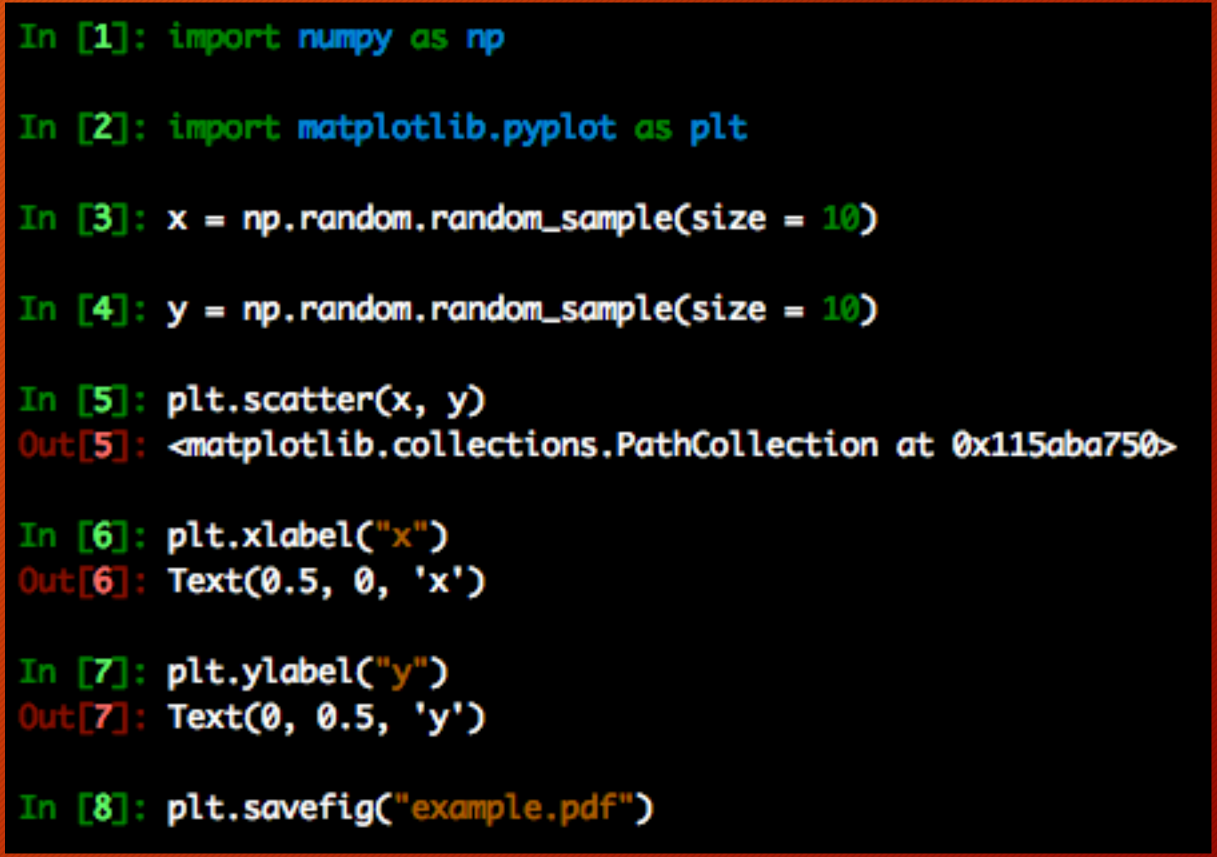
```
In [4]: plt.ylabel?
Signature: plt.ylabel(ylabel, fontdict=None, labelpad=None, **kwargs)
Docstring:
Set the label for the y-axis.

Parameters
----------
ylabel : str
    The label text.

labelpad : scalar, optional, default: None
    Spacing in points from the axes bounding box including ticks
    and tick labels.

Other Parameters
----------------
**kwargs : `.Text` properties
    `.Text` properties control the appearance of the label.

See also
--------
text : for information on how override and the optional args work
File:      ~/anaconda3/lib/python3.7/site-packages/matplotlib/pyplot.py
Type:      function
```

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample
- matplotlib.pyplot.scatter
- matplotlib.pyplot.xlabel
- matplotlib.pyplot.ylabel

The solution:



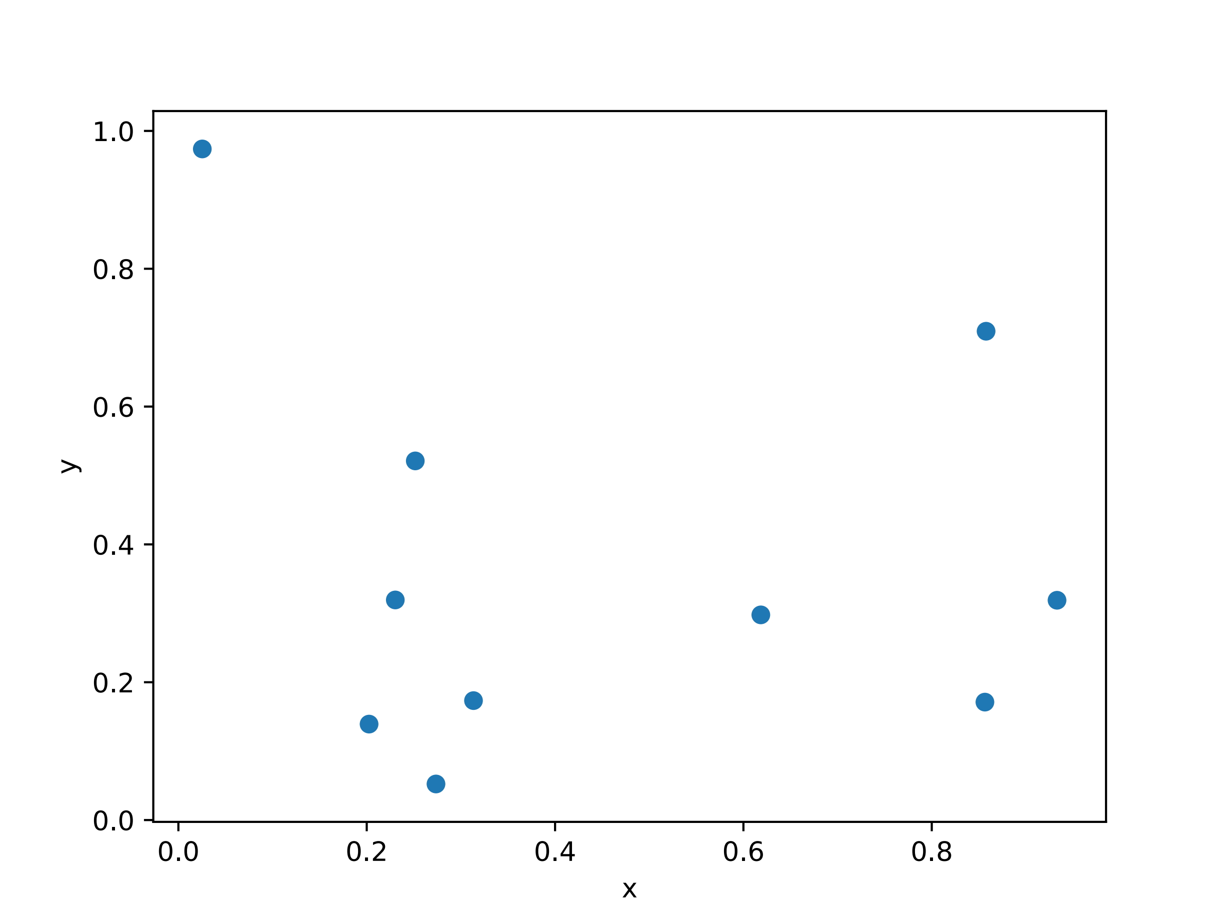
```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: x = np.random.random_sample(size = 10)

In [4]: y = np.random.random_sample(size = 10)

In [5]: plt.scatter(x, y)
Out[5]: <matplotlib.collections.PathCollection at 0x115aba750>

In [6]: plt.xlabel("x")
Out[6]: Text(0.5, 0, 'x')

In [7]: plt.ylabel("y")
Out[7]: Text(0, 0.5, 'y')

In [8]: plt.savefig("example.pdf")
```

Example

Problem: Randomly generate 10 (x, y) points, scatter plot them using matplotlib, and save the figure.

- numpy.random.random_sample
- matplotlib.pyplot.scatter
- matplotlib.pyplot.xlabel
- matplotlib.pyplot.ylabel

The output:



Plotting: Some Useful Matplotlib Functions

pyplot.scatter
pyplot.plot
pyplot.fill_between
pyplot.errorbar
pyplot.legend
pyplot.subplots
pyplot.xlabel
pyplot.ylabel

The plotting functions in pyplot have similar signatures when working with a subplot (e.g. ax.scatter, ax.plot, ax.errorbar).
Subplots are more flexible than calling pyplot directly, and can be made with:

```
fig = plt.figure(…)
ax = fig.add_subplot(…)
ax.set_xlabel(“x”)
ax.set_ylabel(“y”)
```