Basic Plotting

In this lesson, we will discuss the basic use of plot command and how to save figures.

WE'LL COVER THE FOLLOWING

Python's 2-D and 3-D plotting library Matplotlib produces high-quality figures in a variety of formats and interactive environments. Using this library, you can generate simple plots, bar charts, scatter plots, contour plots, error charts, etc. Matplotlib makes these things a lot easier because it is easy to get started with. It has support for LaTeX formatted labels and texts.

What makes Matplotlib highly suitable for generating figures for scientific publications, is that all aspects of the figure can be controlled programmatically. This is important for reproducibility and is convenient when one needs to regenerate the figure with updated data or change its appearance.

The submodule `matplotlib.pyplot` provides an object-oriented interface for the plotting library.

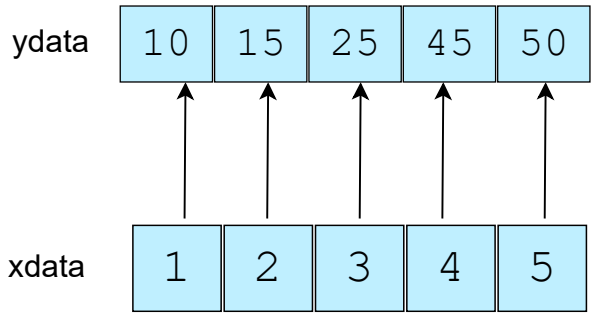
> 💡 In this course, we will import the `matplotlib.pyplot` module under the standard alias `plt`.

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

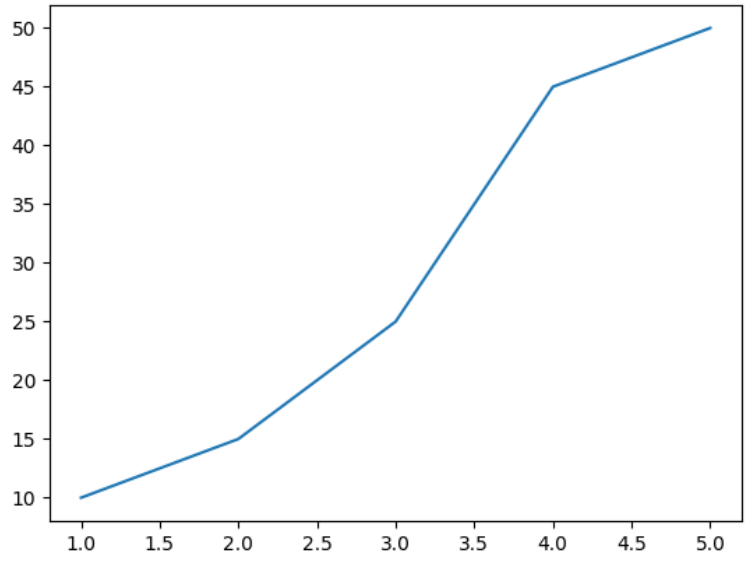
A simple example

The `plot` function takes two input arrays in the order `plot(xdata, ydata)` and

plots each element of `xdata` against its corresponding `ydata`.



1 of 2



2 of 2

Length of `xdata` and `ydata` should be equal.

Let’s start with a basic plotting example:

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2, 100)
y = x ** 2
fig = plt.figure(figsize=(10, 5), dpi=100)
ax1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])      #add_axes([bottom x, bottom y, width, height])
ax1.plot(x, y, 'b')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Example figure');
```

Note that one hundred `y` values are computed in the simple line `y = x ** 2`. Python treats arrays in the same fashion as regular variables when you perform mathematical operations. The math is simply applied to every value in the array and it runs much faster than if you were to do every calculation separately.

In line 4, we have generated the data for the x-axis and in line 5, we have generated data for the y-axis.

> 💡 **Plotting features are objects.**

All of the plotting commands we have used so far are functions that are part of the `matplotlib` package. Not surprisingly, `matplotlib` has an object-oriented design. Plots may be created by making use of the object-oriented structure. This requires a bit of additional typing, but in the end, we gain a lot of additional flexibility.

Using the OO syntax, we first create a `figure` object and specify the size using the `figsize` keyword argument. `figsize` is a tuple of the width and height of the figure in inches. We then also added `dpi` (dots-per-inch). In the example above, we create a 10x5 unit figure and since the resolution (dpi) is 100, this translates to 1000 x 500 pixels.

Then we add an axis to the figure with the `add_axes` command (note that it is `axes` with an `e`) by specifying the *relative* location of the axis in the figure. The location of the left, bottom, width, and height are specified in relative coordinates (both the horizontal and vertical directions run from 0 to 1). To plot, we use the `plot` method of the axis. We will discuss this command in detail shortly.

For scientific purposes, it is absolutely necessary to label the axes of the graph and provide it with the title. This can be done with the very basic commands on lines 9 - 11.

Arguments of `plot`

`plot` takes a large number of keyword arguments. A keyword argument is an optional argument that may be added to a function.

1. `plot` can be used with one argument as `plot(y)`, which plots `y` values along the vertical axis and enumerates the horizontal axis starting at 0.
2. `plot(x, y)` plots `y` vs `x`, and
3. `plot(x, y, formatstring)` plots `y` vs `x` using colors and markers defined in formatstring, which can:
    - define the `color`, for example `b` for blue, `r` for red, and `g` for green.
    - define the `linetype` `-` for line, `--` for dashed, `:` for dots.
    - define `marker`, for example `o` for circles and `s` for squares. You can even combine them: `r--` gives a red dashed line, while `go` gives green circular markers.

```
plot(x, y, color='g', marker='o', linestyle='-', linewidth=4, markersize=15)
```

Keyword arguments can be given in any order, but all keyword arguments should come after regular arguments. `plot(linewidth=6, [1, 2, 3], [2, 4, 3])` gives an error.

The full list of options can be found using the `help("matplotlib.pyplot.plot")` command in the code playground below.

```
import matplotlib.pyplot as plt

help("matplotlib.pyplot.plot")
```

Configure the arguments in line 8 to produce different variations of the graph.

```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2 *np.pi , 100)
y = np.sin(x)
fig = plt.figure(figsize = (10, 5))
ax1 = fig.add_axes([0.1, 0.1, 0.8, 0.8])
ax1.plot(x, y, color='g', marker='o', linestyle='-', linewidth=4, markersize=15)
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('y = sin(x)');
```

The next lesson contains an important note on saving figures in this course.