# 1 Bokeh Tutorial `bokeh.plotting` interface

- This section of the tutorial covers the `bokeh.plotting` interface.
- This interface is a "mid-level" interface, and the main idea can be described by the statement:
- Starting from simple default figures (with default tools, grids and axes), add markers and other shapes whose visual attributes are tied to data.
- We will see that it is possible to customize and change all of the defaults, but having them means that it is possible to get up and running very quickly.

## 1.1 Imports and Setup

When using the `bokeh.plotting` interface, there are a few common imports:

- Use the figure function to create new plot objects to work with.
- Call the functions `output_file`, `output_notebook`, and `output_server` (possibly in combination) to tell Bokeh how to display or save output.
- Execute show and save to display or save plots and layouts.

```
In [1]:
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
```

In this case, we are in the Jupyter notebook, so call `output_notebook()`. We only need to call this once, and all subsequent calls to `show()` will display inline in the notebook.

## 1.2 Some Basic Scatter Plots

In this section you will see how to use Bokeh's various marker types to create simple scatter plots.

```
In [3]:
# create a new plot with default tools, using figure
p = figure(plot_width=400, plot_height=400)

# add a circle renderer with a size, color, and alpha
p.circle([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=15, line_color="na

show(p) # show the results
```

All Bokeh markers accept size (measured in screen space units) as a property. Circles also have radius (measured in "data" space units).

- **EXERCISE:** Try changing the example above to set a 'radius' value instead of 'size'

To scatter square markers instead of circles, you can use the square method on figures.

```
# create a new plot using figure
p = figure(plot_width=400, plot_height=400)

# add a square renderer with a size, color, alpha, and sizes
p.square([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], size=[10, 15, 20, 25, 3

show(p) # show the results
```

Note that in the example above, we are also specifying different sizes for each individual marker. In general, all of a glyph's properties can be "vectorized" in this fashion. Also note that we have passed

color as a shorthand to set both the line and fill colors easily at the same time. This is a convenience specific to `bokeh.plotting`.

There are many marker types available in Bokeh, you can see details and example plots for all of them in the reference guide by clicking on entries in the list below:

```
asterisk()
circle()
circle_cross()
circle_x()
cross()
diamond()
diamond_cross()
inverted_triangle()
square()
square_cross()
square_x()
triangle()
x()
```

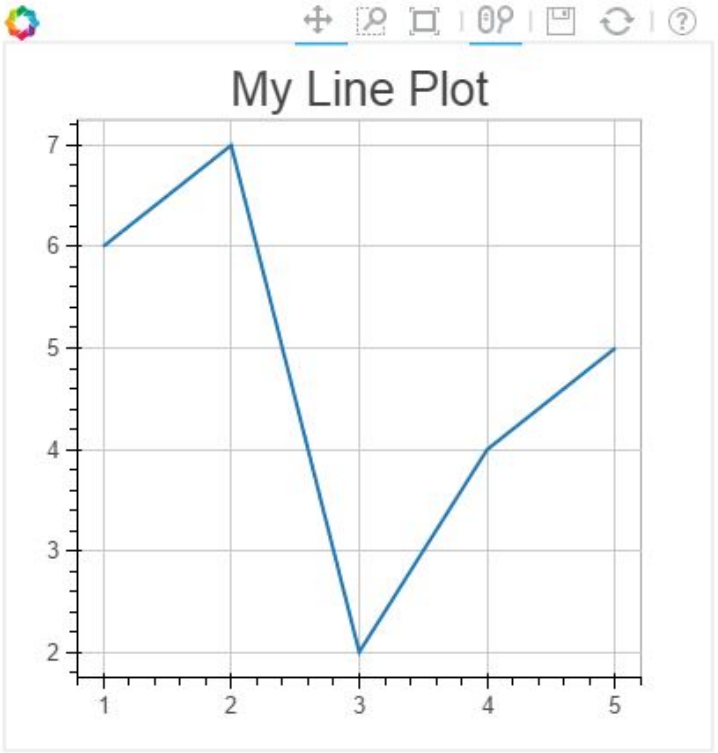
```
In [6]:
# EXERCISE: Plot some different markers in this cell
```

### 1.3 Basic Line Plots

```
# create a new plot (with a title) using figure
p = figure(plot_width=400, plot_height=400,
     title="My Line Plot")

# add a line renderer
p.line([1, 2, 3, 4, 5], [6, 7, 2, 4, 5], line_width=2)

show(p) # show the results
```




The example below shows how to use the `image_rgba` method to display raw RGBA data.

Note: This example makes use of the NumPy library

```
In [8]:
from __future__ import division
import numpy as np

# set up some data
N = 20
img = np.empty((N,N), dtype=np.uint32)
view = img.view(dtype=np.uint8).reshape((N, N, 4))
for i in range(N):
    for j in range(N):
        view[i, j, 0] = int(i/N*255) # red
        view[i, j, 1] = 158          # green
        view[i, j, 2] = int(j/N*255) # blue
        view[i, j, 3] = 255          # alpha

# create a new plot (with a fixed range) using figure
p = figure(x_range=[0,10], y_range=[0,10])

# add an RGBA image renderer
p.image_rgba(image=[img], x=[0], y=[0],
     dw=[10], dh=[10])

show(p) # show the results
```

# 2 Other Kinds of Glyphs

Bokeh supports many other kinds of glyphs. You can click on the User Guide links below to see how to create plots with these glyphs using the `bokeh.plotting` interface.

- Rectangles and Ovals
- Segments and Rays
- Wedges and Arcs
- Specialized Curves

## 2.1 Plots with Multiple Glyphs

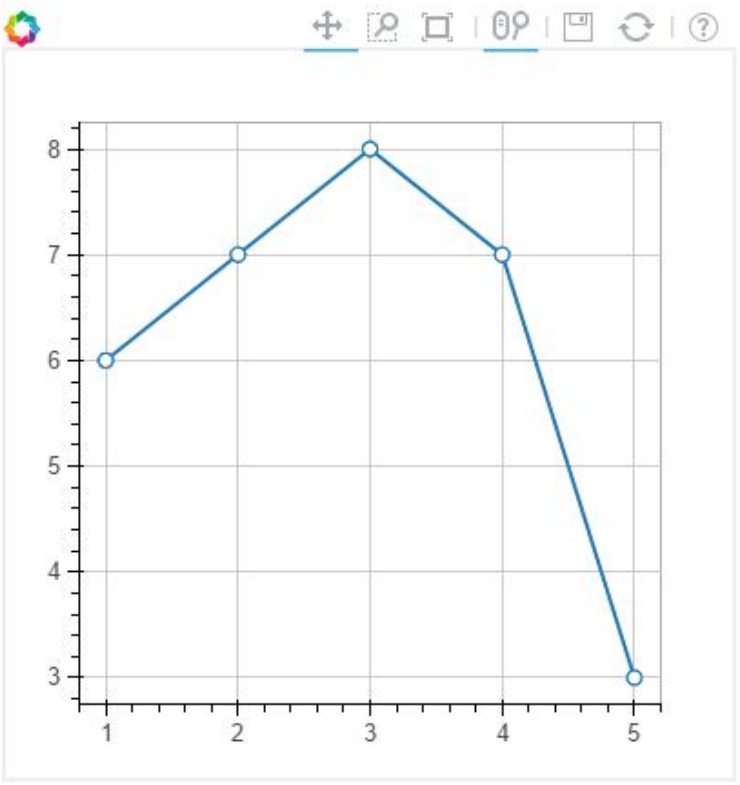
It is possible to combine more than one glyph on a single figure. You just need to call multiple glyph methods on one figure object:

```
In [10]:
# set up some data
x = [1, 2, 3, 4, 5]
y = [6, 7, 8, 7, 3]

# create a new plot with figure
p = figure(plot_width=400, plot_height=400)

# add both a line and circles on the same plot
p.line(x, y, line_width=2)
p.circle(x, y, fill_color="white", size=8)

show(p) # show the results
```


8
7
6
5
4
3
1
2
3
4
5