Bokeh Tutorial

(http://bokeh.pydata.org/)

06. Linking and Interactions

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

(https://bokeh.pydata.org)

BokehJS 0.12.14 successfully loaded.

Now that we know from the previous chapter how multiple plots can be placed together in a layout, we can start to look at how different plots can be linked togeher, or how plots can be linked to widgets.

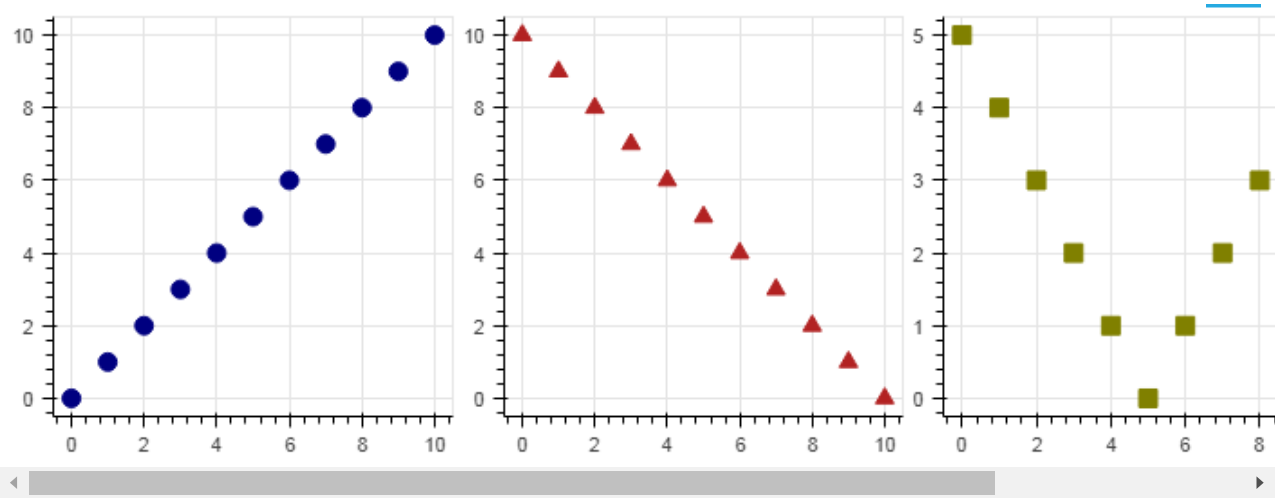
Linked Interactions

It is possible to link various interactions between different Bokeh plots. For instance, the ranges of two (or more) plots can be linked, so that when one of the plots is panned (or zoomed, or otherwise has its range changed) the other plots will update in unison. It is also possible to link selections between two plots, so that when items are selected on one plot, the corresponding items on the second plot also become selected.

Linked panning

Linked panning (when multiple plots have ranges that stay in sync) is simple to spell with Bokeh. You simply share the appropriate range objects between two (or more) plots. The example below shows how to accomplish this by linking the ranges of three plots in various ways:

In [2]:
```
from bokeh.layouts import gridplot

x = list(range(11))
y0, y1, y2 = x, [10-i for i in x], [abs(i-5) for i in x]

plot_options = dict(width=250, plot_height=250, tools='pan,wheel_zoom')

# create a new plot
s1 = figure(**plot_options)
s1.circle(x, y0, size=10, color="navy")

# create a new plot and share both ranges
s2 = figure(x_range=s1.x_range, y_range=s1.y_range, **plot_options)
s2.triangle(x, y1, size=10, color="firebrick")

# create a new plot and share only one range
s3 = figure(x_range=s1.x_range, **plot_options)
s3.square(x, y2, size=10, color="olive")

p = gridplot([[s1, s2, s3]])

# show the results
show(p)
```



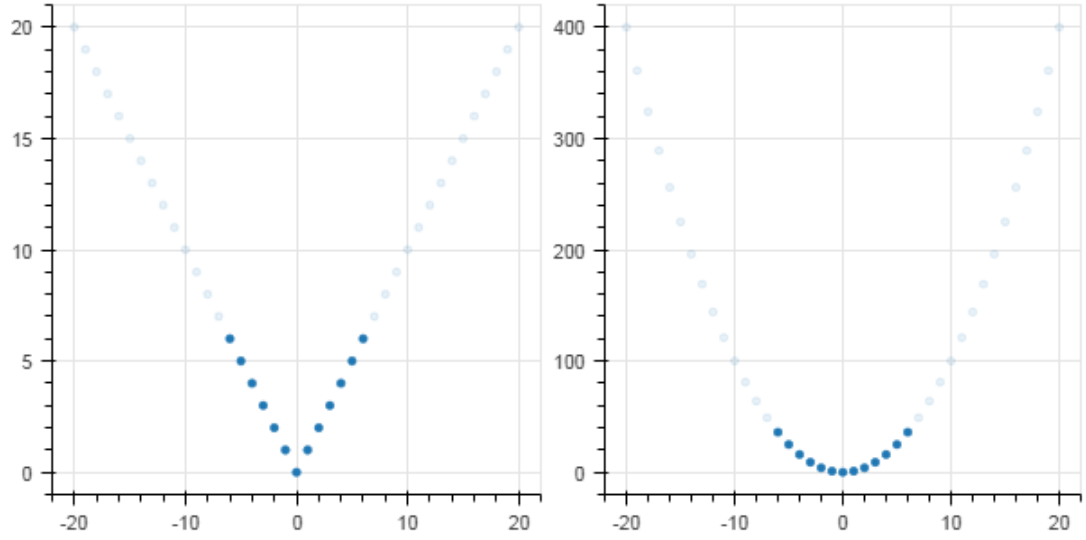
In [3]:
```
# EXERCISE: create two plots in a gridplot, and link their ranges
```

Linked brushing

Linking selections is accomplished in a similar way, by sharing data sources between plots. Note that normally with `bokeh.plotting` and `bokeh.charts` creating a default data source for simple plots is handled automatically. However to share a data source, we must create them by hand and pass them explicitly. This is illustrated in the example below:

In [3]:
```
from bokeh.models import ColumnDataSource

x = list(range(-20, 21))
y0, y1 = [abs(xx) for xx in x], [xx**2 for xx in x]

# create a column data source for the plots to share
source = ColumnDataSource(data=dict(x=x, y0=y0, y1=y1))

TOOLS = "box_select,lasso_select,help"

# create a new plot and add a renderer
left = figure(tools=TOOLS, width=300, height=300)
left.circle('x', 'y0', source=source)

# create another new plot and add a renderer
right = figure(tools=TOOLS, width=300, height=300)
right.circle('x', 'y1', source=source)

p = gridplot([[left, right]])

show(p)
```

(https://bokeh.pydata



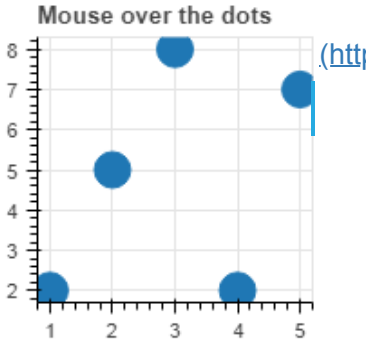
In [5]:
```
# EXERCISE: create two plots in a gridplot, and link their data sources
```

Hover Tools

Bokeh has a Hover Tool that allows additional information to be displayed in a popup whenever the user hovers over a specific glyph. Basic hover tool configuration amounts to providing a list of `(name, format)` tuples. The full details can be found in the User's Guide here (http://bokeh.pydata.org/en/latest/docs/user_guide/tools.html#hovertool).

The example below shows some basic usage of the Hover tool with a circle glyph, using hover information defined in utils.py:

In [9]:
```
from bokeh.models import HoverTool

source = ColumnDataSource(
        data=dict(
            x=[1, 2, 3, 4, 5],
            y=[2, 5, 8, 2, 7],
            desc=['A', 'b', 'C', 'd', 'E'],
        )
    )

hover = HoverTool(
        tooltips=[
            ("index", "$index"),
            ("(x,y)", "($x, $y)"),
            ("desc", "@desc"),
        ]
    )

p = figure(plot_width=200, plot_height=200, tools=[hover], title="Mouse over the dots")

p.circle('x', 'y', size=20, source=source)

show(p)
```



Widgets

Bokeh supports direct integration with a small basic widget set. Thse can be used in conjunction with a Bokeh Server, or with `CustomJS` models to add more interactive capability to your documents. You can see a complete list, with example code in the Adding Widgets (http://bokeh.pydata.org/en/latest/docs/user_guide/interaction.html#adding-widgets) section of the User's Guide.

To use the widgets, include them in a layout like you would a plot object:

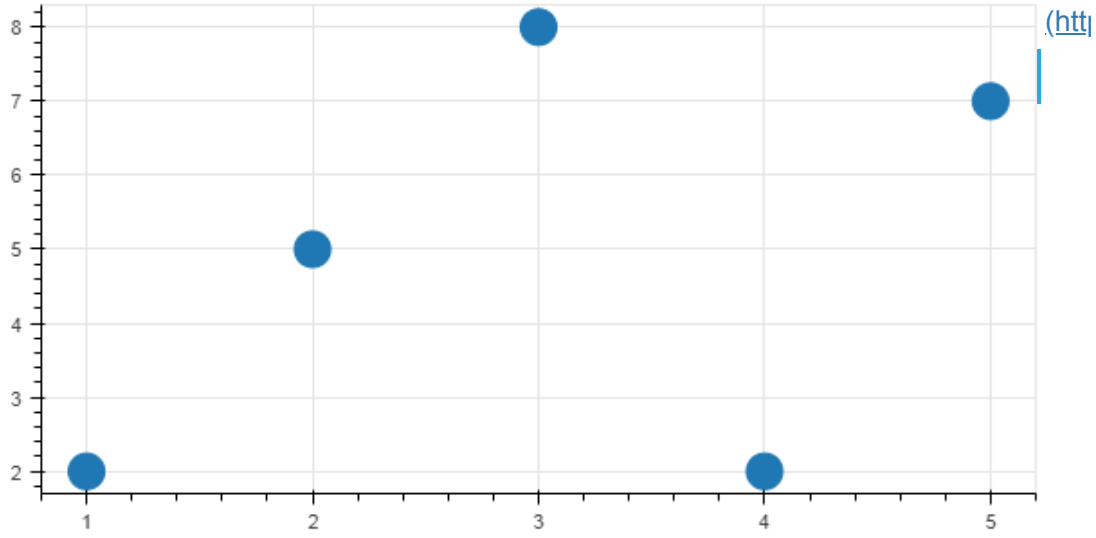
In [5]:
```
from bokeh.layouts import widgetbox
from bokeh.models.widgets import Slider


slider = Slider(start=0, end=10, value=1, step=.1, title="foo")

show(widgetbox(slider))
```

foo: **1**

In [8]:
```
# EXERCISE: create and show a Select widget
```

CustomJS Callbacks

In [6]:
```
from bokeh.models import TapTool, CustomJS, ColumnDataSource

callback = CustomJS(code="alert('hello world')")
tap = TapTool(callback=callback)

p = figure(plot_width=600, plot_height=300, tools=[tap])

p.circle(x=[1, 2, 3, 4, 5], y=[2, 5, 8, 2, 7], size=20)

show(p)
```



(http

Lots of places to add callbacks

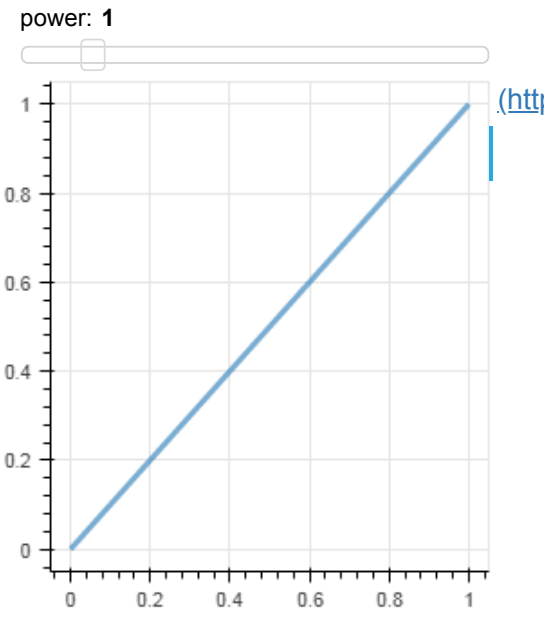
- Widgets - Button, Toggle, Dropdown, TextInput, AutocompleteInput, Select, Multiselect, Slider, (DateRangeSlider), DatePicker,
- Tools - TapTool, BoxSelectTool, HoverTool,
- Selection - ColumnDataSource, AjaxDataSource, BlazeDataSource, ServerDataSource
- Ranges - Range1d, DataRange1d, FactorRange

Callbacks for widgets

Widgets that have values associated can have small JavaScript actions attached to them. These actions (also referred to as "callbacks") are executed whenever the widget's value is changed. In order to make it easier to refer to specific Bokeh models (e.g., a data source, or a glyhph) from JavaScript, the `CustomJS` obejct also accepts a dictionary of "args" that map names to Python Bokeh models. The corresponding JavaScript models are made available automaticaly to the `CustomJS` code.

And example below shows an action attached to a slider that updates a data source whenever the slider is moved:

In [10]:
```
from bokeh.layouts import column
from bokeh.models import CustomJS, ColumnDataSource, Slider

x = [x*0.005 for x in range(0, 201)]

source = ColumnDataSource(data=dict(x=x, y=x))

plot = figure(plot_width=300, plot_height=300)
plot.line('x', 'y', source=source, line_width=3, line_alpha=0.6)

slider = Slider(start=0.1, end=6, value=1, step=.1, title="power")

update_curve = CustomJS(args=dict(source=source, slider=slider), code="""
    var data = source.get('data');
    var f = slider.value;
    x = data['x']
    y = data['y']
    for (i = 0; i < x.length; i++) {
        y[i] = Math.pow(x[i], f)
    }
    source.change.emit();
""")
slider.js_on_change('value', update_curve)


show(column(slider, plot))
```

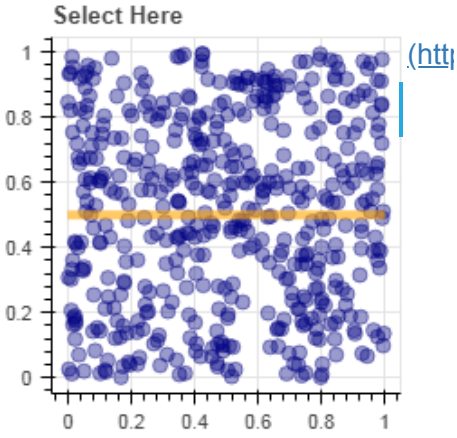


Calbacks for selections

It's also possible to make JavaScript actions that execute whenever a user selection (e.g., box, point, lasso) changes. This is done by attaching the same kind of CustomJS object to whatever data source the selection is made on.

The example below is a bit more sophisticated, and demonstrates updating one glyph's data source in response to another glyph's selection:

In [11]:
```
from random import random

x = [random() for x in range(500)]
y = [random() for y in range(500)]
color = ["navy"] * len(x)

s = ColumnDataSource(data=dict(x=x, y=y, color=color))
p = figure(plot_width=250, plot_height=250, tools="lasso_select", title="Select Here")
p.circle('x', 'y', color='color', size=8, source=s, alpha=0.4)

s2 = ColumnDataSource(data=dict(xm=[0,1],ym=[0.5, 0.5]))
p.line(x='xm', y='ym', color="orange", line_width=5, alpha=0.6, source=s2)

s.callback = CustomJS(args=dict(s2=s2), code="""
    var inds = cb_obj.get('selected')['1d'].indices;
    var d = cb_obj.get('data');
    var ym = 0

    if (inds.length == 0) { return; }

    for (i = 0; i < d['color'].length; i++) {
        d['color'][i] = "navy"
    }
    for (i = 0; i < inds.length; i++) {
        d['color'][inds[i]] = "firebrick"
        ym += d['y'][inds[i]]
    }

    ym /= inds.length
    s2.get('data')['ym'] = [ym, ym]

    cb_obj.trigger('change');
    s2.trigger('change');
""")

show(p)
```



More

For more interactions, see the User Guide - http://bokeh.pydata.org/en/latest/docs/user_guide/interaction.html (http://bokeh.pydata.org/en/latest/docs/user_guide/interaction.html)