Bokeh Tutorial

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

11. Running Bokeh Applications

The architecture of Bokeh is such that high-level “model objects” (representing things like plots, ranges, axes, glyphs, etc.) are created in Python, and then converted to a JSON format that is consumed by the client library, BokehJS. Using the Bokeh Server, it is possible to keep the “model objects” in python and in the browser in sync with one another, creating powerful capabilities:

- respond to UI and tool events generated in a browser with computations or queries using the full power of python
- automatically push updates the UI (i.e. widgets or plots), in a browser
- use periodic, timeout, and asychronous callbacks drive streaming updates

This capability to synchronize between python and the browser is the main purpose of the Bokeh Server.

NOTE: Exercises below require work outside the notebook

Hello Bokeh

To try out the example below, copy the code into a file `hello.py` and then execute:

```
bokeh serve --show hello.py
```

```
# hello.py

from bokeh.io import curdoc
from bokeh.layouts import column
from bokeh.models.widgets import TextInput, Button, Paragraph

# create some widgets
button = Button(label="Say HI")
input = TextInput(value="Bokeh")
output = Paragraph()

# add a callback to a widget
def update():
    output.text = "Hello, " + input.value
button.on_click(update)

# create a layout for everything
layout = column(button, input, output)

# add the layout to curdoc
curdoc().add_root(layout)
```

Let's try an exercise to modify this example to add another widget.

In [1]:
```
# EXERCISE: add a Select widget to this example that offers several different greetings
```

Linking Plots and Widgets

To try out the example below, copy the code into a file `app.py` and then execute:

```
bokeh serve --show app.py
```

```
# app.py

from numpy.random import random

from bokeh.io import curdoc
from bokeh.layouts import column, row
from bokeh.plotting import ColumnDataSource, Figure
from bokeh.models.widgets import Select, TextInput

def get_data(N):
    return dict(x=random(size=N), y=random(size=N), r=random(size=N) * 0.03)

source = ColumnDataSource(data=get_data(200))

p = Figure(tools="", toolbar_location=None)
r = p.circle(x='x', y='y', radius='r', source=source,
             color="navy", alpha=0.6, line_color="white")

COLORS = ["black", "firebrick", "navy", "olive", "goldenrod"]
select = Select(title="Color", value="navy", options=COLORS)
input = TextInput(title="Number of points", value="200")

def update_color(attrname, old, new):
    r.glyph.fill_color = select.value
select.on_change('value', update_color)

def update_points(attrname, old, new):
    N = int(input.value)
    source.data = get_data(N)
input.on_change('value', update_points)

layout = column(row(select, input, width=400), row(p))

curdoc().add_root(layout)
```

In [2]:
```
# EXERCISE: add more widgets to change more aspects of this plot
```

Streaming Data

It is possible to efficiently stream new data to column data sources by using the `stream` method. This method accepts two argmuments:

- `new_data` — a dictionary with the same structure as the column data source
- `rollover` — a maximum column length on the client (earlier data is dropped) *[optional]*

If no `rollover` is specified, data is never dropped on the client and columns grow without bound.

It is often useful to use periodic callbacks in conjuction with streaming data The `add_periodic_callback` method of `curdoc()` accepts a callback function, and a time interval (in ms) to repeatedly execute the callback.

To try out the example below, copy the code into a file `stream.py` and then execute:

```
bokeh serve --show stream.py
```

```
# stream.py
from math import cos, sin

from bokeh.io import curdoc
from bokeh.models import ColumnDataSource
from bokeh.plotting import figure

p = figure(x_range=(-1.1, 1.1), y_range=(-1.1, 1.1))
p.circle(x=0, y=0, radius=1, fill_color=None, line_width=2)

# this is the data source we will stream to
source = ColumnDataSource(data=dict(x=[1], y=[0]))
p.circle(x='x', y='y', size=12, fill_color='white', source=source)

def update():
    x, y = source.data['x'][-1], source.data['y'][-1]

    # construct the new values for all columns, and pass to stream
    new_data = dict(x=[x*cos(0.1) - y*sin(0.1)], y=[x*sin(0.1) + y*cos(0.1)])
    source.stream(new_data, rollover=8)

curdoc().add_periodic_callback(update, 150)
curdoc().add_root(p)
```

In [3]:
```
### EXERCISE: starting with the above example, create your own streaming plot
```

In [ ]:
```
```