Day 7, More on widgets and dashboarding

- we'll get into some more of the nitty gritty details of different things we can use with widgets now

In [1]:
```
# lets import our usual stuff
import pandas as pd
import numpy as np
%matplotlib inline
```

In [2]:
```
#import widgets
import ipywidgets
```

So, this is how we've used widgets before for the most part:

In [4]:
```
@ipywidgets.interact(name = ['Linda', 'Tina', 'Louise'])
def print_name(name):
    print(name) # just a simple print out
```

But now, lets go into ipywidgets in a bit more detail and look at specific functions.

For example, we can create a little display that increments integer numbers:

In [5]:
```
itext = ipywidgets.IntText()
```

To be able to display our widget we need to use the "display" function.

Note: depending on the version of jupyter you are using, you may or maynot have to use "display" to display things. We'll do it both ways, starting with display:

In [6]:
```
from IPython.display import display
```

In [7]:
```
# lets show the thing!
display(itext)
# you can see there are little numbers at the
#  end that we can toggle up and down
```

Note that if I make another display of itext, this one and the previous one have values that are tied together:

In [8]:
```
display(itext)
```

Also, the value of itext is then stored - so we could in theory generate a toggle and then do stuff with its value:

In [9]:
```
itext.value
```

Out[9]: 3

Note if I go up and change the toggle value I have to re-run this cell to print out the newly stored value.

I can also set the value "by hand":

In [10]:
```
itext.value = 10
```

Once I run this cell, now the toggle values are updated above.

We can also do fun things like create progress bars:

In [11]:
```
ip = ipywidgets.IntProgress()
```

In [13]:
```
display(ip)
```

So, its probably hard to see, but there is indeed a bar there. I can show this better by setting the progress to 90% by hand:

In [14]:
```
ip.value = 90
```

We can also check out more of the interer counters, like this nifty slider:

In [15]:
```
irange = ipywidgets.IntRangeSlider(min = -10, max = 10, step = 1)
```

In [16]:
```
display(irange)
```

Again, the value of the slider is stored if we want to use it:

In [17]:
```
irange.value
```

Out[17]: (-5, 8)

We can use ipywidgets to make clickers:

In [18]:
```
button1 = ipywidgets.Button(description = "I am a Clicker")
```

In [19]:
```
display(button1)
```

```
I have clicked. Click.
I have clicked. Click.
```

We note that nothing happens when we press this button.

Now, lets make a function that can act when we press this button:

In [20]:
```
def say_click(event):
    print("I have clicked. Click.")
```

Now we can tell our button that we've created to say click when we click.

In [21]:
```
button1.on_click(say_click)
```

We'll note that this is "back reactive" - so now if we go up to our displayed button and press click it says it has clicked were before it did nothing.

Now that we've played with some toy examples, let's see how we can combine them to make interactive things and how we can layer widgets to create more complex interactives.

In [22]:
```
# lets start by making a progress bar again:
ip = ipywidgets.IntProgress()
# now, lets add in a button that will add 10
button_plus = ipywidgets.Button(description = "+10")
# and one that will subtract 10
button_minus = ipywidgets.Button(description = "-10")
```

Now we'll use one of the layout features of widgets to put these in a horizontal order with `HBox`:

In [23]:
```
ipywidgets.HBox([button_minus, ip, button_plus])
```

We note if we click these, nothing happens this is because we haven't associated actions to our clicks as before.

So, no matter what we press, we don't see anything happening:

In [24]:
```
ip.value
```

Out[24]: 0

Let's now associate a change in the value of our progress bar when we click the down button:

In [25]:
```
def click_down(event):
    ip.value -= 10
```

And lets tie this change in value to the click with the "on_click" function of our down button:

In [26]:
```
button_minus.on_click(click_down)
```

Same type of thing, but for our up button:

In [27]:
```
def click_up(event):
    ip.value += 10
button_plus.on_click(click_up)
```

Let's try this again:

In [28]:
```
ipywidgets.HBox([button_minus, ip, button_plus])
```

Note also again that these associations with these up and down click functions are in a sense "back reactive" - so now our previous instance of this clicker and progress bar updates as well!

Let's try one more example - an integer slider:

In [29]:
```
islider = ipywidgets.IntSlider(min = 0,
                               max = 10,
                               step = 1,
                               orientation = 'vertical')
```

Let's give this slider a base color that is sort of purple-y, using a hex code:

In [30]:
```
islider.style.handle_color = "#750075"
```

In [31]:
```
islider
```

Note here (and above with our boxes) I'm not using "display" note this slider slides up and down, nothing too exciting.

Let's create a new widget object called a color picker:

In [32]:
```
cp = ipywidgets.ColorPicker()
cp
```

When we show this we can click on the little box and it popus up a color picker we can mess around with.

Now, lets link the slider and the color picker:

In [33]:
```
ipywidgets.link( (cp, 'value'), (islider.style, 'handle_color'))
```

Out[33]:
```
<traitlets.traitlets.link at 0x120c36a58>
```

In [34]:
```
ipywidgets.VBox([cp, islider])
```

Note how previous instances of these objects are tied together also note how the .link function sort of intuitievly knows how to link these two interactive widgets together.

A more practicle example: Plotting analytical orbits

Let's now change over from toy examples to something that we can use. We'll make use of "interact" in ipywidgets to do this.

In [35]:
```
import matplotlib.pyplot as plt
```

In [40]:
```
theta = np.arange(0, 2*np.pi, 0.001)

a = 5

def plot_ellipse(ecc):
    #r = b/np.sqrt(1-ecc*(np.cos(theta))**2) ## double check formula, wr
ong!
    r = a*(1-ecc**2)/(1-ecc*np.cos(theta))
    x = r*np.cos(theta)
    y = r*np.sin(theta)
    plt.plot(x, y)
    plt.xlim(-10,10)
    plt.ylim(-10,10)
    plt.show()

# note here I'm doing this differently then using it as
#  a "decorator" function => this is just a different way to call things
ipywidgets.interact(plot_ellipse, ecc = (0.0,0.99, 0.01))
```

Out[40]:
```
<function __main__.plot_ellipse(ecc)>
```

Exercise

Include the radius as another slider

Bonus: Use a slider to select radius & eccentricity and plot both analytical and numerical solutions (you can use whatever solver you'd like).

Bonus bonus: re-do for multi-body

In [ ]: