SECTION 21: GUIs - 45 minutes, 7 parts
3/7 Interact Functionality with GUIs

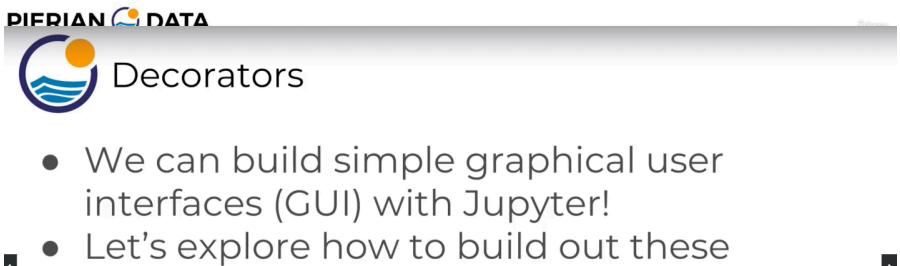
- **building GUIs with Jupyter**
  - -> graphical user interfaces
  - -> there are multiple libraries for this
  - -> these have licensing issues
  - -> styling widgets
  - -> interactive elements
  - -> keeping the lecture notebooks, for reference code
  - -> interact functionality

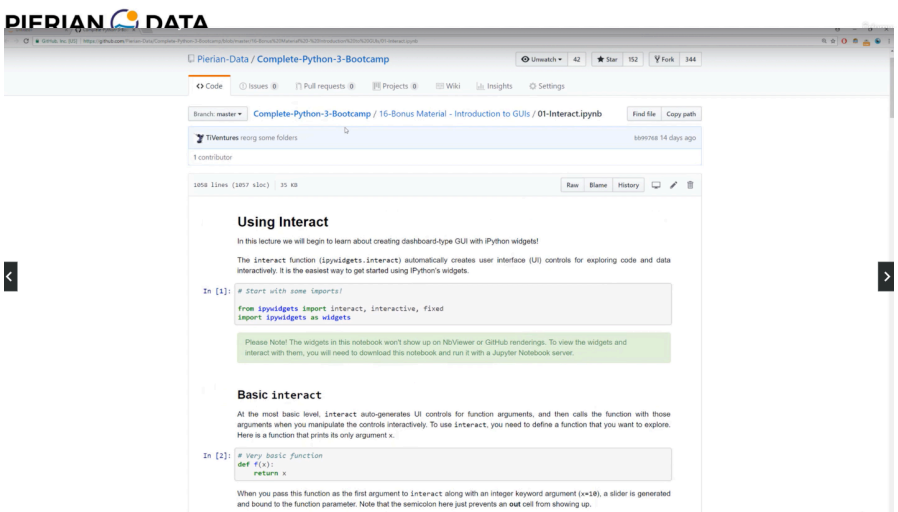
- **in an ipynb**
  - -> we can't see the sliders for this
  - -> we can't execute the code for this
  - -> this is under bonus material
  - **-> the interact function**
    - -> he imports ipywidgets
    - -> he then imports it as ipywidgets
    - -> auto-generating the user interface control for a user function argument
    - -> then calling the function with the arguments which we can manipulate interactively
    - -> he defines a function
    - -> and then uses the name of that function as the argument of another, with a second argument

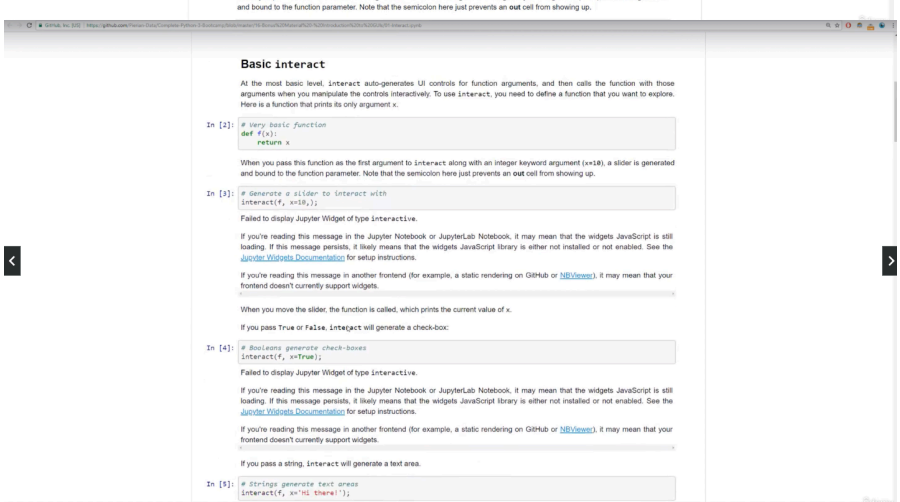
```
In [80]: def func(x):
             return x

In [81]: interact(func,x=10)

         x                    4

         4

Out[81]: <function __main__.func>
```

    - -> this returns a slider with different inputs to the function and their outputs
    - -> passing booleans into this returns a checkbox instead of a slider
    - -> strings return an auto-textbox

  - **-> this can also be done with a decorator**
    - -> decorators are @interact() in this case
    - -> the second element can be run







```
In [88]: @interact(x=True,y=1.0)
         def g(x,y):
             return (x,y)
```

x ☑

y 0.80

(True, 0.8)

- **-> widget abbreviations <- the .interact() method**
  - -> he gives more arguments to the widget
  - -> this allows us to add different ranges to the slider
  - -> there is an integer and flat slider
  - -> he is creating a minimum, maximum, step value and value
  - -> there are ways to abbreviate all of this functionality

```
In [79]: from ipywidgets import interact,interactive,fixed
         import ipywidgets as widgets

In [84]: def func(x):
             return x

In [85]: interact(func,x=10)
```

x 10

10

```
Out[85]: <function __main__.func>

In [ ]:
```

```
In [79]: from ipywidgets import interact,interactive,fixed
         import ipywidgets as widgets

In [84]: def func(x):
             return x

In [87]: interact(func,x='Hello')
```

x type whatever

'type whatever'

```
Out[87]: <function __main__.func>

In [ ]:
```

```
In [97]: interact(func,x=widgets.IntSlider(min=-100,max=100,step=1,value=0))
```

x 100

100

```
Out[97]: <function __main__.func>

In [99]: interact(func,x=(-10,10,1))
```

x 0

```
import ipywidgets as widgets

In [90]: def func(x):
             return x

In [91]: interact(func,x=fixed('Hello'))
```

'Hello'

```
Out[91]: <function __main__.func>

In [89]: @interact(x=True,y=fixed(1.0))
         def g(x,y):
             return (x,y)
```

x ☑

(True, 1.0)

```
In [ ]:
```

- -> min, max, step
- -> there is another function called the interact function
- -> the arguments of this are function and the second is in

```
In [100]: interact(func,x=(-10.0,10.0,.1))
```

x 3.30

3.3

in this case in the abbreviated format

```
In [101]: @interact(x=(0.0,20.0,0.5))
          def h(x=5.0):
              return x
```

x 10.00

9.0

x ☑

(True, 1.0)

```
In [97]: interact(func,x=widgets.IntSlider(min=-100,max=100,step=1,value=0))
```

x 100

100

```
Out[97]: <function __main__.func>

In [100]: interact(func,x=(-10.0,10.0,.1))
```

x 3.30

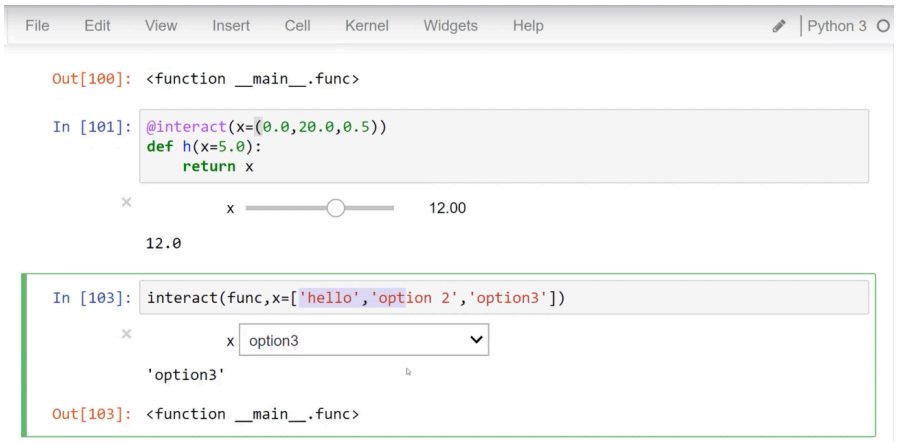
3.3

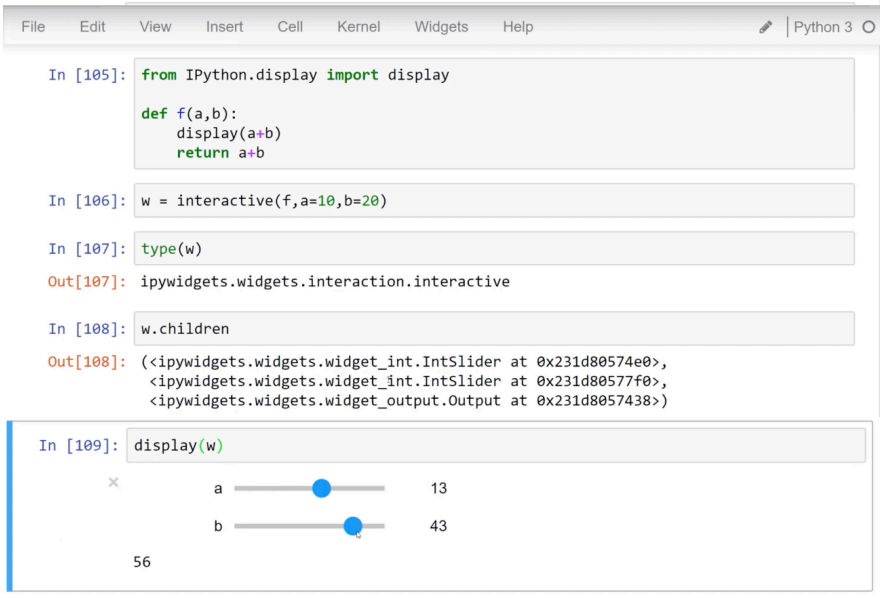
```
Out[100]: <function __main__.func>

In [ ]:
```

- -> this can also be done with the @interact decorator
- -> this is widget abbreviation <- start, stop, step
- -> he defines another function in this example

- -> the floating point slider
- -> increasing in increments of .1's

- -> we can also have drop down menus for this, with text to be entered
  - -> adding a list for this creates a drop down menu
- -> dictionaries with this
  - -> he passes a dictionary into this method
  - -> the output of this is equal to the value





- **-> the .interactive() method**
  - **-> about this method**
    - -> to reuse the widgets we already have, or access the data that is bound to the user interface controls
    - -> the value of this isn't returned automatically
    - -> he imports modules
    - -> then defines a function
    - -> he uses display() in the definition for this
    - -> then runs interactive() on this
    - -> the type of this is an interactive object

  - **-> the children of this attribute**
    - -> this builds out a larger UI using the interactive functionality
    - -> using display on this method prints two sliders
    - -> this is to allow the user to see the inputs / outputs
    - -> there is a port for the output of the function which can be removed by entering interactive(..); <- a semi-colon at the end of this