# Using Interact

In this lecture we will begin to learn about creating dashboard-type GUI with iPython widgets!

The `interact` function ( `ipywidgets.interact` ) automatically creates user interface (UI) controls for exploring code and data interactively. It is the easiest way to get started using IPython's widgets.

In [1]:

```
# Start with some imports!

from ipywidgets import interact, interactive, fixed
import ipywidgets as widgets
```

> Please Note! The widgets in this notebook won't show up on NbViewer or GitHub renderings. To view the widgets and interact with them, you will need to download this notebook and run it with a Jupyter Notebook server.

## Basic `interact`

At the most basic level, `interact` auto-generates UI controls for function arguments, and then calls the function with those arguments when you manipulate the controls interactively. To use `interact` , you need to define a function that you want to explore. Here is a function that prints its only argument `x` .

In [2]:

```
# Very basic function
def f(x):
    return x
```

When you pass this function as the first argument to `interact` along with an integer keyword argument ( `x=10` ), a slider is generated and bound to the function parameter. Note that the semicolon here just prevents an **out** cell from showing up.

In [3]:

```
# Generate a slider to interact with
interact(f, x=10,);
```

When you move the slider, the function is called, which prints the current value of `x` .

If you pass `True` or `False` , `interact` will generate a check-box:

In [4]:

```
# Booleans generate check-boxes
interact(f, x=True);
```

If you pass a string, `interact` will generate a text area.

In [5]:

```
# Strings generate text areas
interact(f, x='Hi there!');
```

`interact` can also be used as a decorator. This allows you to define a function and interact with it in a single shot. As this example shows, `interact` also works with functions that have multiple arguments.

In [6]:

```
# Using a decorator!
@interact(x=True, y=1.0)
def g(x, y):
    return (x, y)
```

## Fixing arguments using `fixed`

There are times when you may want to explore a function using `interact`, but fix one or more of its arguments to specific values. This can be accomplished by wrapping values with the `fixed` function.

In [7]:

```
# Again, a simple function
def h(p, q):
    return (p, q)
```

When we call `interact`, we pass `fixed(20)` for q to hold it fixed at a value of `20`.

In [8]:

```
interact(h, p=5, q=fixed(20));
```

Notice that a slider is only produced for `p` as the value of `q` is fixed.

## Widget abbreviations

When you pass an integer-valued keyword argument of `10` (`x=10`) to `interact`, it generates an integer-valued slider control with a range of `[-10,+3\times10]`. In this case, `10` is an *abbreviation* for an actual slider widget:

```
IntSlider(min=-10,max=30,step=1,value=10)
```

In fact, we can get the same result if we pass this `IntSlider` as the keyword argument for `x`:

In [9]:

```
# Can call the IntSlider to get more specific
interact(f, x=widgets.IntSlider(min=-10,max=30,step=1,value=10));
```

This examples clarifies how `interact` processes its keyword arguments:

1. If the keyword argument is a `Widget` instance with a `value` attribute, that widget is used. Any widget with a `value` attribute can be used, even custom ones.
2. Otherwise, the value is treated as a *widget abbreviation* that is converted to a widget before it is used.

The following table gives an overview of different widget abbreviations:

| Keyword argument | Widget |
|---|---|
| `True` or `False` | Checkbox |
| `'Hi there'` | Text |
| `value` or `(min,max)` or `(min,max,step)` if integers are passed | IntSlider |
| `value` or `(min,max)` or `(min,max,step)` if floats are passed | FloatSlider |
| `['orange','apple']` or `{'one':1,'two':2}` | Dropdown |

Note that a dropdown is used if a list or a dict is given (signifying discrete choices), and a slider is used if a tuple is given (signifying a range).

You have seen how the checkbox and text area widgets work above. Here, more details about the different abbreviations for sliders and drop-downs are given.

If a 2-tuple of integers is passed `(min,max)`, an integer-valued slider is produced with those minimum and maximum values (inclusively). In this case, the default step size of `1` is used.

In [10]:

```
# Min,Max slider with Tuples
interact(f, x=(0,4));
```

If a 3-tuple of integers is passed `(min,max,step)`, the step size can also be set.

In [11]:

```
# (min, max, step)
interact(f, x=(0,8,2));
```

A float-valued slider is produced if the elements of the tuples are floats. Here the minimum is `0.0`, the maximum is `10.0` and step size is `0.1` (the default).

In [12]:

```
interact(f, x=(0.0,10.0));
```

The step size can be changed by passing a third element in the tuple.

In [13]:

```
interact(f, x=(0.0,10.0,0.01));
```

For both integer and float-valued sliders, you can pick the initial value of the widget by passing a default keyword argument to the underlying Python function. Here we set the initial value of a float slider to `5.5`.

In [14]:

```
@interact(x=(0.0,20.0,0.5))
def h(x=5.5):
    return x
```

Dropdown menus are constructed by passing a list of strings. In this case, the strings are both used as the names in the drop-down menu UI and passed to the underlying Python function.

In [15]:

```
interact(f, x=['apples','oranges']);
```

If you want a drop-down menu that passes non-string values to the Python function, you can pass a dictionary. The keys in the dictionary are used for the names in the drop-down menu UI and the values are the arguments that are passed to the underlying Python function.

In [16]:

```
interact(f, x={'one': 10, 'two': 20});
```

## Using function annotations with `interact`

You can also specify widget abbreviations using [function annotations (https://docs.python.org/3/tutorial/controlflow.html#function-annotations)](https://docs.python.org/3/tutorial/controlflow.html#function-annotations).

Define a function with a checkbox widget abbreviation for the argument `x`.

In [17]:

```
def f(x:True):  # Python 3 only
    return x
```

Then, because the widget abbreviation has already been defined, you can call `interact` with a single argument.

In [18]:

```
interact(f);
```

## interactive

In addition to `interact`, IPython provides another function, `interactive`, that is useful when you want to reuse the widgets that are produced or access the data that is bound to the UI controls.

Note that unlike `interact`, the return value of the function will not be displayed automatically, but you can display a value inside the function with `IPython.display.display`.

Here is a function that returns the sum of its two arguments and displays them. The display line may be omitted if you don't want to show the result of the function.

In [22]:

```
from IPython.display import display

def f(a, b):
    display(a + b)
    return a+b
```

Unlike `interact`, `interactive` returns a `Widget` instance rather than immediately displaying the widget.

In [23]:

```
w = interactive(f, a=10, b=20)
```

The widget is an `interactive`, a subclass of `VBox`, which is a container for other widgets.

In [24]:

```
type(w)
```

Out[24]:

```
ipywidgets.widgets.interaction.interactive
```

The children of the `interactive` are two integer-valued sliders and an output widget, produced by the widget abbreviations above.

In [25]:

```
w.children
```

Out[25]:

```
(IntSlider(value=10, description='a', max=30, min=-10),
 IntSlider(value=20, description='b', max=60, min=-20),
 Output())
```

To actually display the widgets, you can use IPython's `display` function.

In [26]:

```
display(w)
```

At this point, the UI controls work just like they would if `interact` had been used. You can manipulate them interactively and the function will be called. However, the widget instance returned by `interactive` also give you access to the current keyword arguments and return value of the underlying Python function.

Here are the current keyword arguments. If you rerun this cell after manipulating the sliders, the values will have changed.

In [27]:

```
w.kwargs
```

Out[27]:

```
{'a': 10, 'b': 20}
```

Here is the current return value of the function.

In [28]:

```
w.result
```

Out[28]:

```
30
```

# Conclusion

You should now have a basic understanding of how to use Interact in Jupyter Notebooks!