2a.Interact

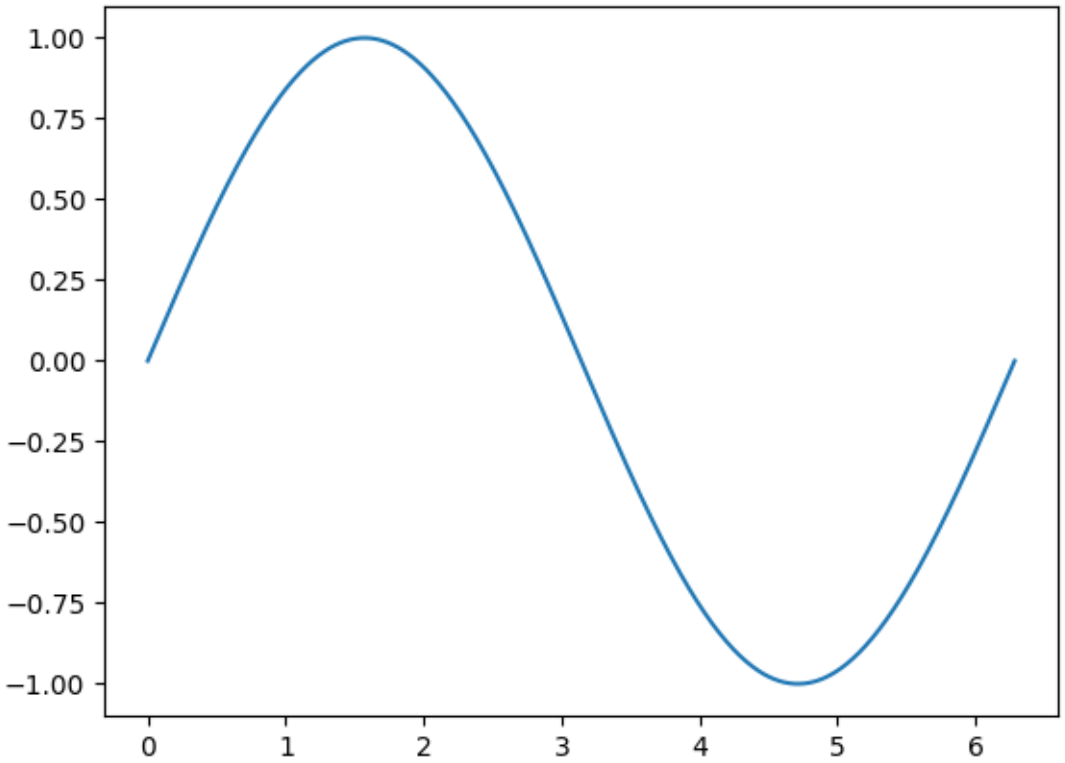
November 2, 2024

Inspired by https://blog.ouseful.info/2017/01/10/an-alternative-way-of-motivating-the-use-of-functions/

Let's build this up slowly. First, here's code to just make a simple sine plot.

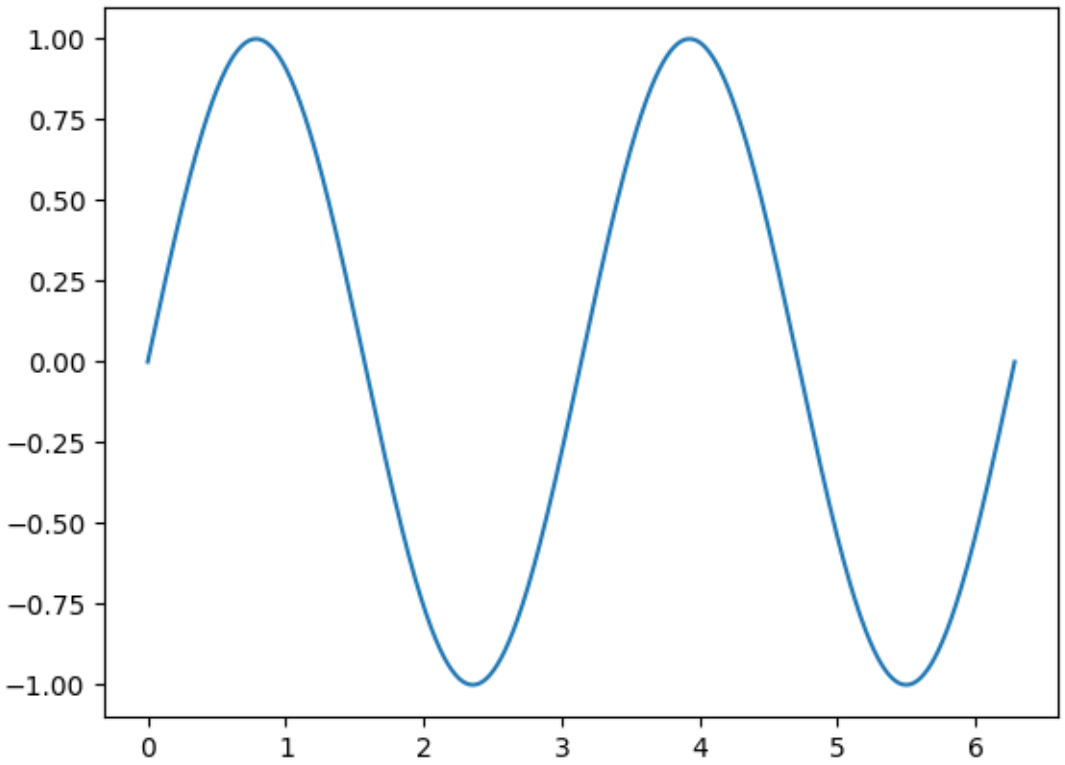
[1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[2]:
```
# create 1000 evenly spaced points
x = np.linspace(0, 2*np.pi, 1000)
y = np.sin(x)

plt.plot(x,y) ;
```



We can just copy and paste our code to see what happens if we change it to be sin(2*x) instead of sin(x).

[3]:
```
# create 1000 evenly spaced points
x = np.linspace(0, 2*np.pi, 1000)
y = np.sin(2*x)

plt.plot(x,y) ;
```

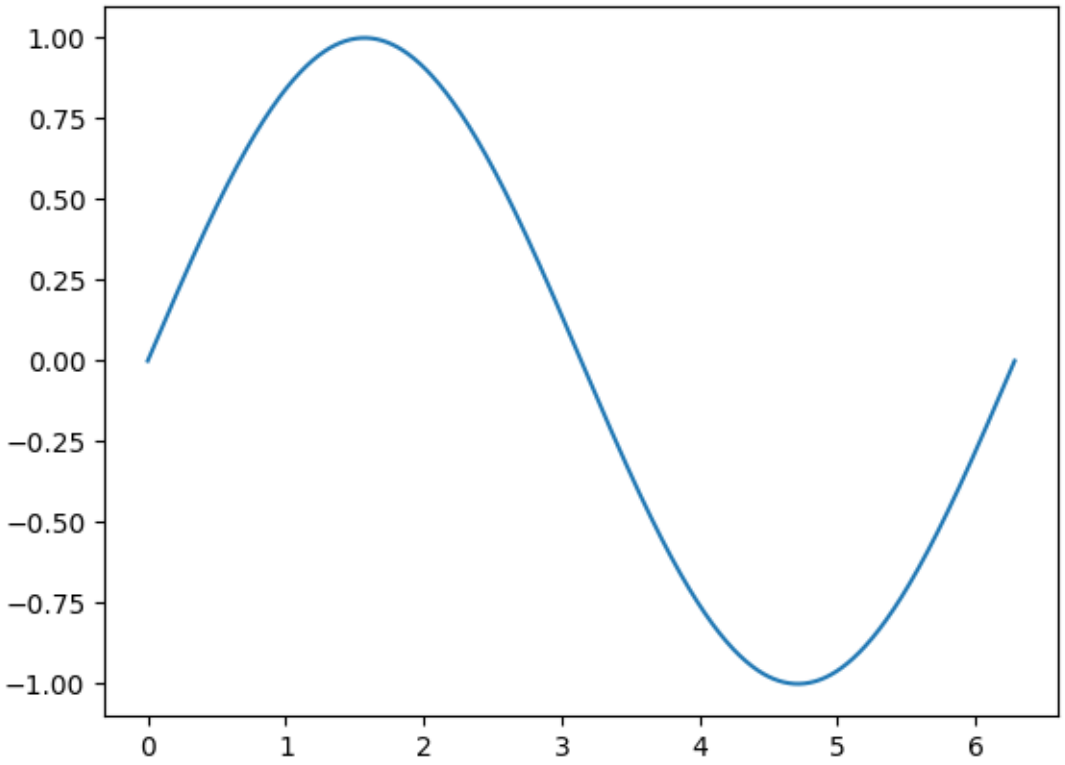


But it might make more sense to make a function that takes the frequency as an argument if we want to play around with different values here.

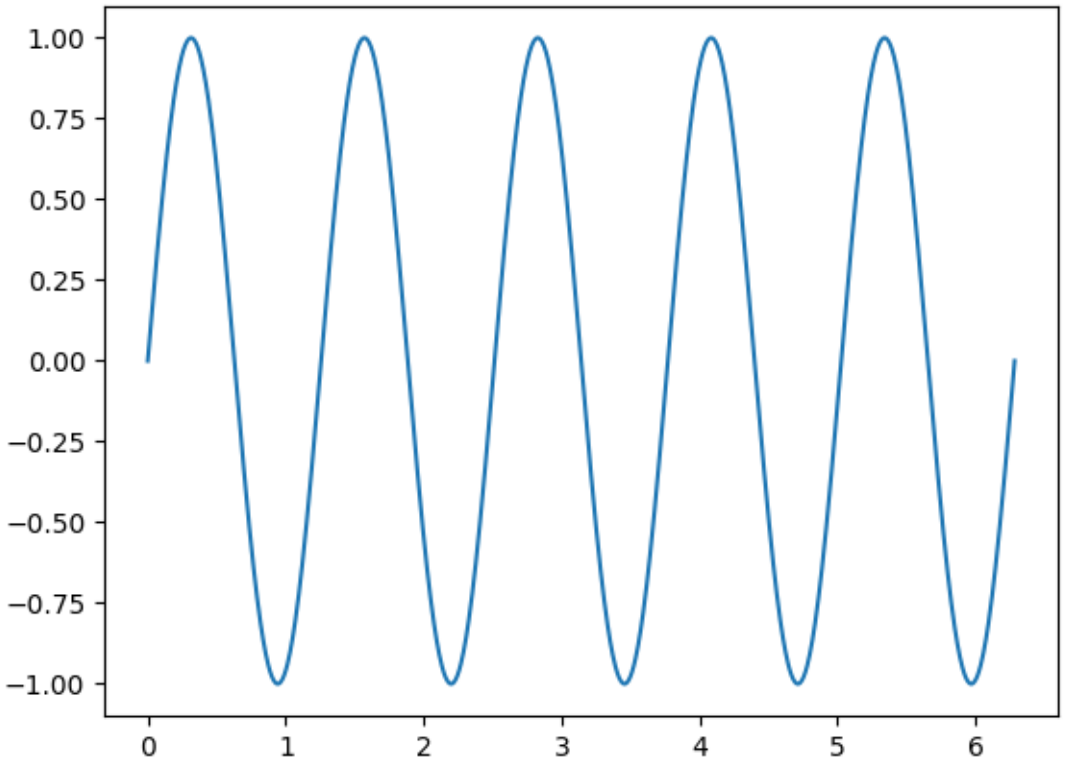
[4]:
```
def sinplot(f=1):
  x = np.linspace(0, 2*np.pi, 1000)
  y = np.sin(f*x)
  plt.plot(x,y)
  plt.show() ;
```

Now we can just rerun our function with different inputs to see what happens.

[5]:
```
sinplot()
```



[6]:
```
sinplot(5)
```



Or we can make it way cooler using interact. This way we can just use a slider to change the value of f on the fly.

[7]:
```
from ipywidgets import interact
interact(sinplot, f=(0,10)) ;
```

```
interactive(children=(IntSlider(value=1, description='f', max=10), Output()),␣
↪_dom_classes=('widget-interact',…
```

If we want to play more with interact we can just bring up the help and get some ideas of what we can do.

[8]:
```
interact?
```

Let's try out a few of the examples from the help documentation.

[9]:
```
def greeting(text="World"):
  print("Hello {}".format(text))

interact(greeting) ;
```

```
interactive(children=(Text(value='World', description='text'), Output()),␣
↪_dom_classes=('widget-interact',))
```

[10]:
```
def greeting(text="World!"):
  print("Hello {}".format(text))

interact(greeting, text="universe") ;
```

```
interactive(children=(Text(value='universe', description='text'), Output()),␣
 ↪_dom_classes=('widget-interact',)…
```

[12]:
```
def greeting(greet="hello", text="World"):
  print("{} -- {}".format(greet, text))

interact(greeting, text="universe") ;
# interact(greeting, greet="Hola") ;
```

```
interactive(children=(Text(value='hello', description='greet'),␣
 ↪Text(value='universe', description='text'), Ou…
```

This uses interact as a decorator instead. `@interact` goes on top of the function instead of using `interact(func)`.

[13]:
```
@interact
def greeting(text="World!"):
  print("Hello {}".format(text))
```

```
interactive(children=(Text(value='World!', description='text'), Output()),␣
 ↪_dom_classes=('widget-interact',))
```

[14]:
```
@interact(text="universe")
def greeting(text="World!"):
  print("Hello {}".format(text))
```

```
interactive(children=(Text(value='universe', description='text'), Output()),␣
 ↪_dom_classes=('widget-interact',)…
```

[17]:
```
@interact(greet="Hola")
def greeting(greet="hello", text="World"):
  print("{} -- {}".format(greet, text))
```

```
interactive(children=(Text(value='Hola', description='greet'),␣
 ↪Text(value='World', description='text'), Output…
```

We can also use a list with interact to limit our input options and create a dropdown that lets us pick options from the list.

[18]:
```
@interact(text = ["universe", "little bear","World!"])
def greeting(text="World!"):
  print("Hello {}".format(text))
```

```
interactive(children=(Dropdown(description='text', index=2, options=('universe',␣
 ↪'little bear', 'World!'), val…
```

Here's an example that uses a boolean input and creates a checkbox.

[19]:
```
@interact
def lin_log(log=False):
  x = np.linspace(0.1, 100, 1000)
  y = x
  if log: plt.axes(yscale='log')
  plt.plot(x,y)
  plt.show()
```

```
interactive(children=(Checkbox(value=False, description='log'), Output()),␣
↪_dom_classes=('widget-interact',))
```

[20]:
```
@interact(a=(-100,100), b=(-100, 100), c=(-100,100))
def multiply(a=1,b=-1,c=1, pos= False):
  if pos: return abs(a*b*c)
  return(a*b*c)
```

```
interactive(children=(IntSlider(value=1, description='a', min=-100),␣
↪IntSlider(value=-1, description='b', min=…
```

1 Your Turn

1. Create a function called `plot_power` that takes in two arguments: `x_max` and `power`. This function should do the following:

- Create a variable called `x` which consists of 30 equally spaced points between `-x_max` and `x_max`. (Note: `x_max` can be any positive integer)
- Create a variable called `y` which is equal to x^{power} (where `power` is the value specified by the user).
- Plot y versus x.

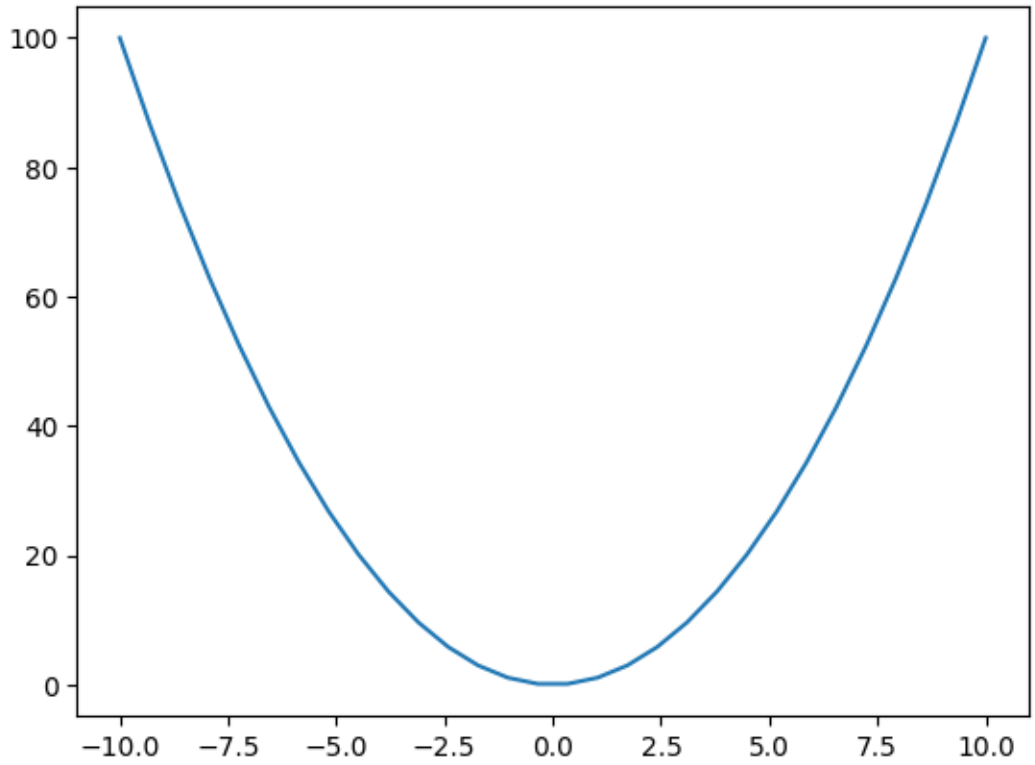
2. Use interact to create sliders with a range of values for `x_max` and `power`.

[24]:
```
# Solution
def plot_power( x_max = 0, power = 1 ):
  '''
  Plots x^power versus x.
  '''
  x = np.linspace( -x_max, x_max, 30 )
  y = x**power
  plt.plot( x, y )
  plt.show()
```

[25]:
```
plot_power
```

[25]:
```
<function __main__.plot_power(x_max=0, power=1)>
```

[26]:
```
plot_power( 10, 2 )
```



[31]:
```
interact( plot_power, x_max = ( -5, 11 ), power = [ 1, 5] ) ;
```

```
interactive(children=(IntSlider(value=0, description='x_max', max=11, min=-5),␣
↪Dropdown(description='power', o…
```

[32]:
```
# Solution
@interact( x_max = ( -5, 11 ), power = [ 1, 5] )
def plot_power( x_max = 0, power = 1 ):
  '''
  Plots x^power versus x.
  '''
  x = np.linspace( -x_max, x_max, 30 )
  y = x**power
  plt.plot( x, y )
  plt.show()
```

```
interactive(children=(IntSlider(value=0, description='x_max', max=11, min=-5),␣
↪Dropdown(description='power', o…
```

[ ]:
```

```