



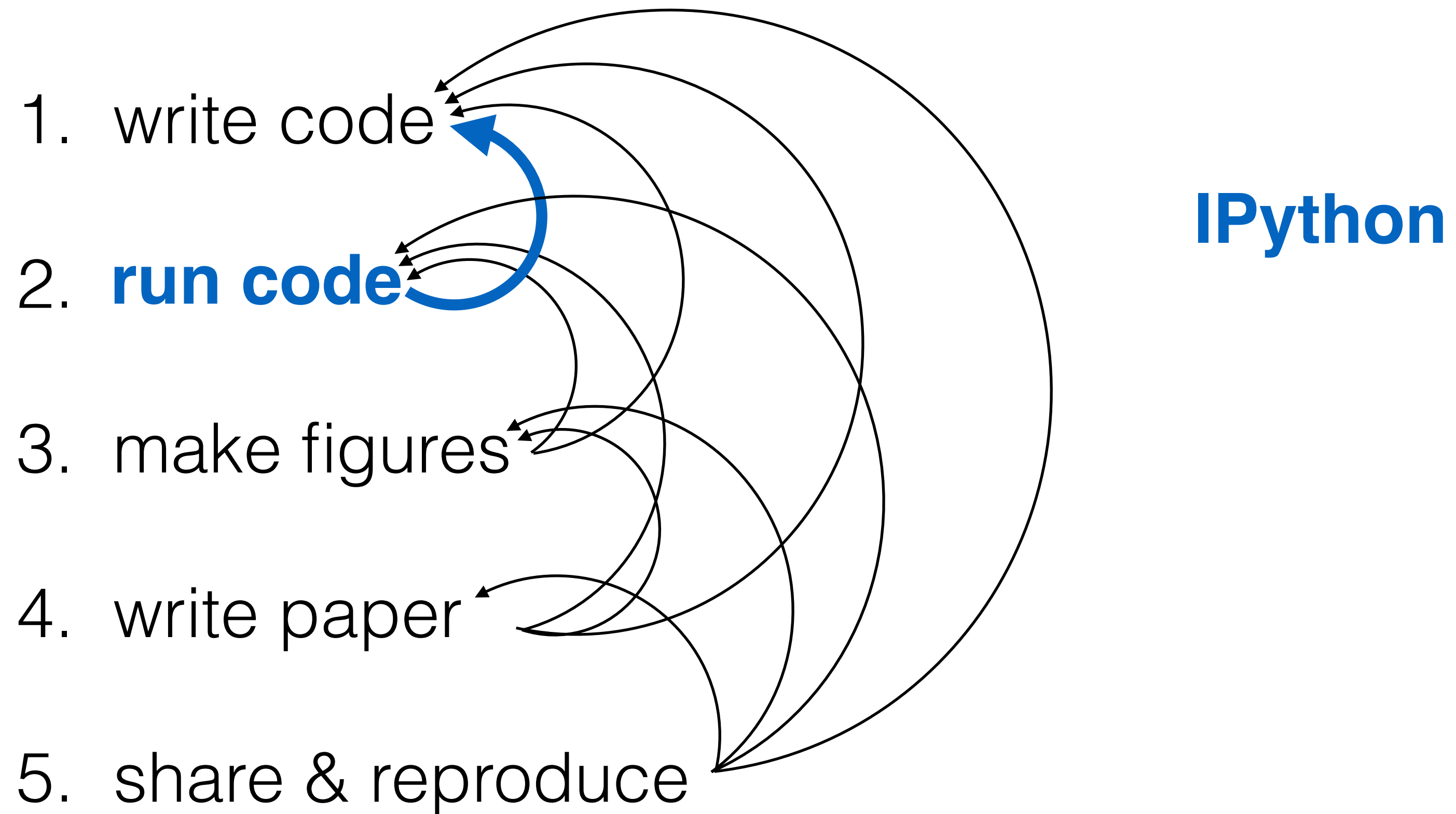
Interactive CSE with IPython and  
Jupyter

**Min Ragan-Kelley**, Simula Research Lab  
Project Jupyter Contributors



# How do we do

# computational research?





# IPython

## Interactive Python

helps run code

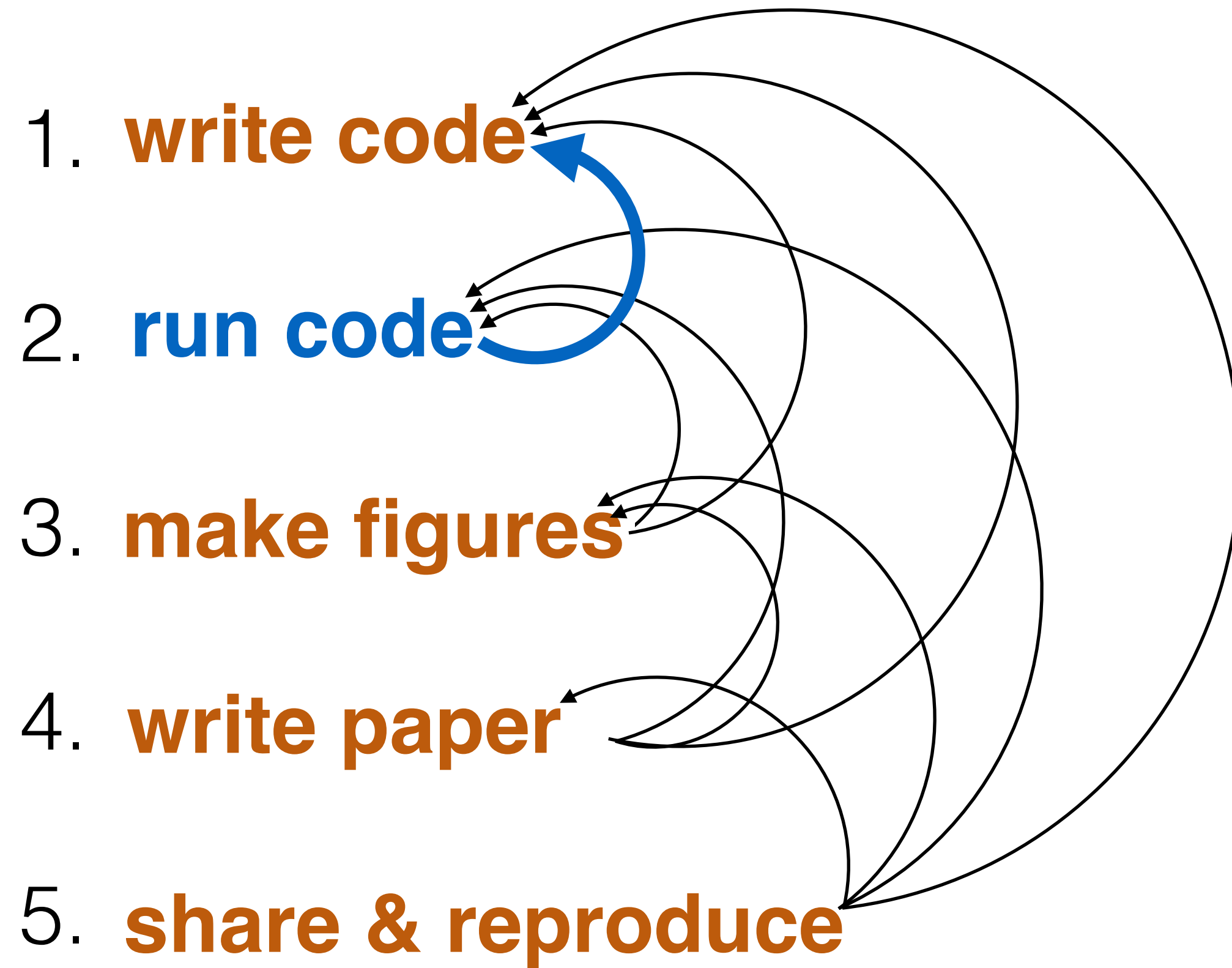
- tab completion
- introspection
- %magics

```
minrk[02:13]~/Documents/Jupyter/pres/AGU-2014 $ ipython

```



# What about Jupyter?

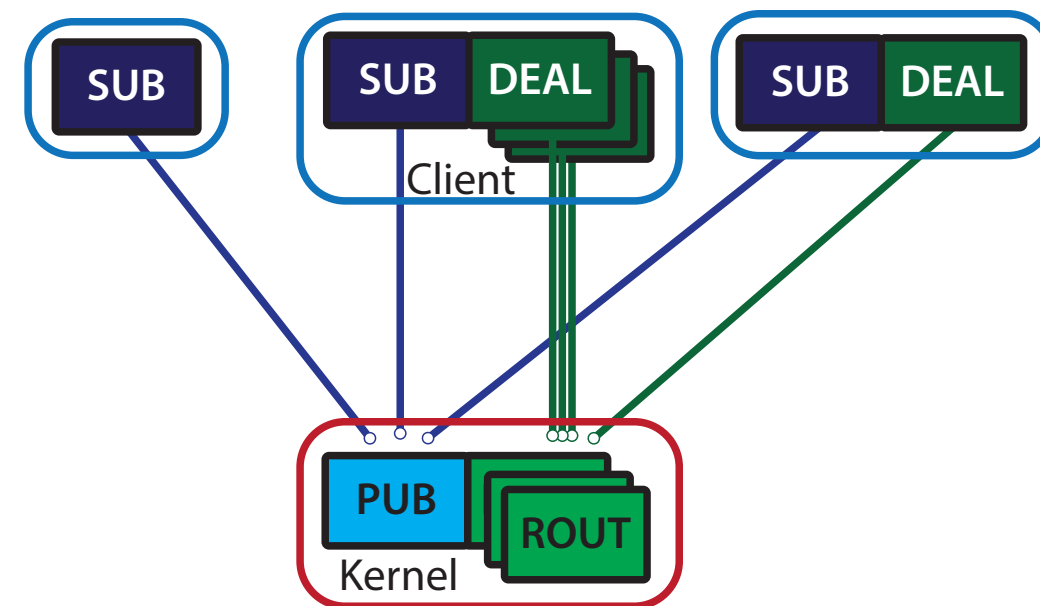




# What is Jupyter?

Rich REPL Protocol

Document Format



ØMQ + JSON

We have already computed  $P(X|A)$  above. On the other hand,  $P(X| \sim A)$  is subjective: our code can pass tests but still have a bug in it, though the probability there is a bug present is reduced. Note this is dependent on the number of tests performed, the degree of complication in the tests, etc. Let's be conservative and assign  $P(X| \sim A) = 0.5$ . Then

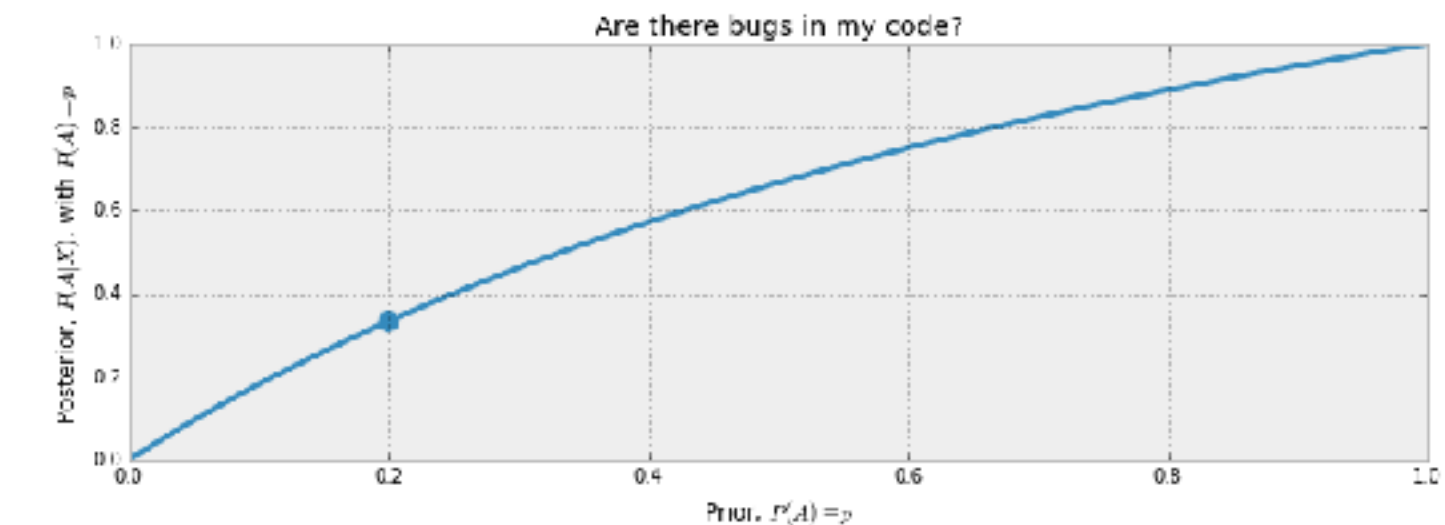
$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1 - p)}$$

$$= \frac{2p}{1 + p}$$

This is the posterior probability. What does it look like as a function of our prior,  $p \in [0, 1]$ ?

```
figsize(12.5, 4)
p = np.linspace(0, 1, 50)
plt.plot(p, 2 * p / (1 + p), color="#348ABD", lw=3)
# plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A68628"])
plt.scatter(0.2, 2 * (0.2) / 1.2, s=140, c="#348ABD")
plt.xlim(0, 1)
plt.ylim(0, 1)
plt.xlabel("Prior, P(A) = p")
plt.ylabel("Posterior, P(A|X), with P(A) = p")
plt.title("Are there bugs in my code?")
```

<matplotlib.text.Text at 0x1051de550>





# Jupyter Protocol

## REP\*L over JSON + ØMQ

- Read
- Eval
- Print\*
- Loop

```
msg_type = 'execute_request'
content = {
    'code' : """
        import pandas as pd
        df = pd.read_csv('mydata.csv')
        """
}

msg_type = 'execute_reply'
content = {
    msg_type = 'display_data'
    content = {
        'data': {
            "text/plain": "<MyDataFrame at 0x...>",
            "text/html": "<table>...</table>",
            ...
        },
        'metadata': {},
        ...
    }
}
```



# Jupyter Protocol

## supercharge the P in REP\*L

any mime-type output

- text
- svg, png, jpeg
- latex, pdf
- html, javascript
- interactive widgets

The screenshot shows a Jupyter notebook interface with three code cells. The first cell contains the code `print(df.head())` and its output is a table with columns 'cake', 'lies', and 'pie' and rows of dates from 2012-12-19 to 2012-12-21. The second cell contains LaTeX code for a function  $f(x) = \int_{-\infty}^{\infty} \hat{f}(\xi) e^{2\pi i \xi x} d\xi$ . The third cell contains an interactive widget definition for `factor_xn`. The notebook interface includes a top bar with 'In [5]:', 'In [14]:', and 'In [ ]:' labels, and a bottom bar with a timeline from Jan 2013 to Oct 2014.

```
In [5]: print(df.head())
```

	cake	lies	pie
2012-12-19	363.885981	367.826809	362.807807
2012-12-20	361.055153	368.463441	365.065045
2012-12-21	362.064454	367.768454	364.087118

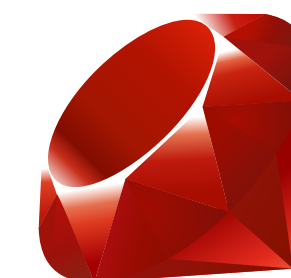
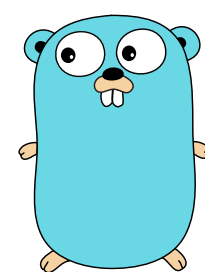
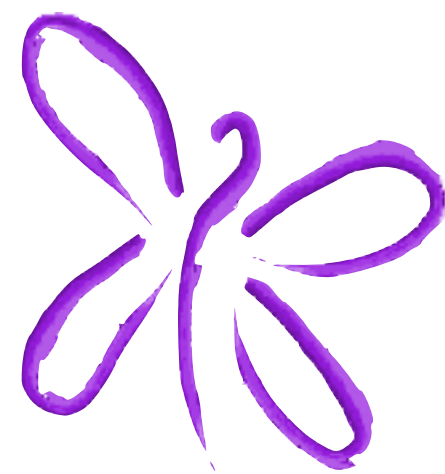
```
In [14]: Math(r'''f(x) = \int_{-\infty}^{\infty}
             \hat{f}(\xi) e^{2 \pi i \xi x},d\xi
             ''')
```

```
In [ ]: @interact
def factor_xn(n=5):
    display(Eq(x**n-1, factor(x**n-1)))
```





# Jupyter Protocol is language agnostic







# Jupyter Notebooks

- notebook = sequence of cells
- text cell = markdown + latex)
- code cell = REP (input + output)
- metadata everywhere

The screenshot shows a Jupyter Notebook window titled "Sampling\_Theorem (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", and "Help". The main content area displays a text cell with the following text:

### Investigating the Sampling Theorem

In this section, we investigate the implications of the sampling theorem. Here is the usual statement of the theorem from wikipedia:

*"If a function  $x(t)$  contains no frequencies higher than  $B$  hertz, it is completely determined by giving its ordinates at a series of points spaced  $1/(2B)$  seconds apart."*

Since a function  $x(t)$  is a function from the real line to the real line, there are uncountably many points between any two ordinates, so sampling is a massive reduction of data since it only takes a tiny number of points to completely characterize the function. This is a powerful idea worth exploring. In fact, we have seen this idea of reducing a function to a discrete set of numbers before in Fourier series expansions where (for periodic  $x(t)$ )

$$a_n = \frac{1}{T} \int_0^T x(t) \exp(-j\omega_n t) dt$$

with corresponding reconstruction as:

$$x(t) = \sum_k a_n \exp(j\omega_n t)$$

But here we are generating discrete points  $a_n$  by integrating over the **entire** function  $x(t)$ , not just evaluating it at a single point. This means we are collecting information about the entire function to compute a single discrete point  $a_n$ , whereas with sampling we are just taking individual points in isolation.