Bradley Voytek, Ph.D.

UC San Diego

Department of Cognitive Science
Neurosciences Graduate Program
Halicioglu Data Science Institute

bvoytek@ucsd.edu
@bradleyvoytek

Administrative stuff

- Waitlist…
- Everyone should be on Piazza now—I updated the list again as of this morning. If you are not, add your email to the list.
- Python3 (use Anaconda for simplicity)
- Get your GitHub account!

Administrative stuff

- Final projects will use private group repos.
- Project deliverables are an organized repo (readme and narrative notebook with visualizations)
- This provides a project deliverable for your portfolio.
- Also allows you to make project web pages using GitHub pages, Jupyter slides, etc.

Administrative stuff

Sections Details

There are 7 sections:

- Monday @ 3 pm in MANDE B-150 (Tom)
- Monday @ 4 pm in MANDE B-150 (Tom)
- Wednesday @ 12 pm in MANDE B-150 (Shuai)
- Wednesday @ 3 pm in MANDE B-150 (Shuai)
- Wednesday @ 4 pm in MANDE B-150 (Harshita)
- Friday @ 11 am in MANDE B-150 (Harshita)
- Friday @ 2 pm in CENTR 122 (Harshita)

Office Hours

Unless otherwise noted, all office hours will take place in the CSB 115, which is a computer lab.

TAs:

- Tom
  - Wednesday 1-2 pm
- Shuai
  - Thursday 4-5 pm
- Harshita
  - Friday 12-1 pm

IAs:

- Tianyu
  - Monday 2-3 pm
- Megan
  - Tuesday 4-5 pm
- Gael
  - Wednesday 11-12 am
- David
  - Thursday 3-4 pm

Professor:

- Wednesdays, 10-11 am, in CSB 169

COGS 108
Data Science in Practice

Python! (For great Data Science)

UC San Diego

Recommendations

- *Data Science from Scratch* - Joel Grus (Allen Institute for AI)
- *Python Data Science Handbook* - Jake VanderPlas (UW eScience Institute)

Why Python?

Python has emerged over the last couple decades as a first-class tool for scientific computing tasks, including the analysis and visualization of large datasets. This may have come as a surprise to early proponents of the Python language: the language itself was not specifically designed with data analysis or scientific computing in mind.

Source: VanderPlas, *Python Data Science Handbook*

Why Python?

The usefulness of Python for data science stems primarily from the large and active ecosystem of third-party packages: NumPy for manipulation of homogeneous array-based data, Pandas for manipulation of heterogeneous and labeled data, SciPy for common scientific computing tasks, Matplotlib for publication-quality visualizations, IPython [Jupyter] for interactive execution and sharing of code, Scikit-Learn for machine learning, and many more tools that will be mentioned in the following pages.

Source: VanderPlas, *Python Data Science Handbook*

Why Python?

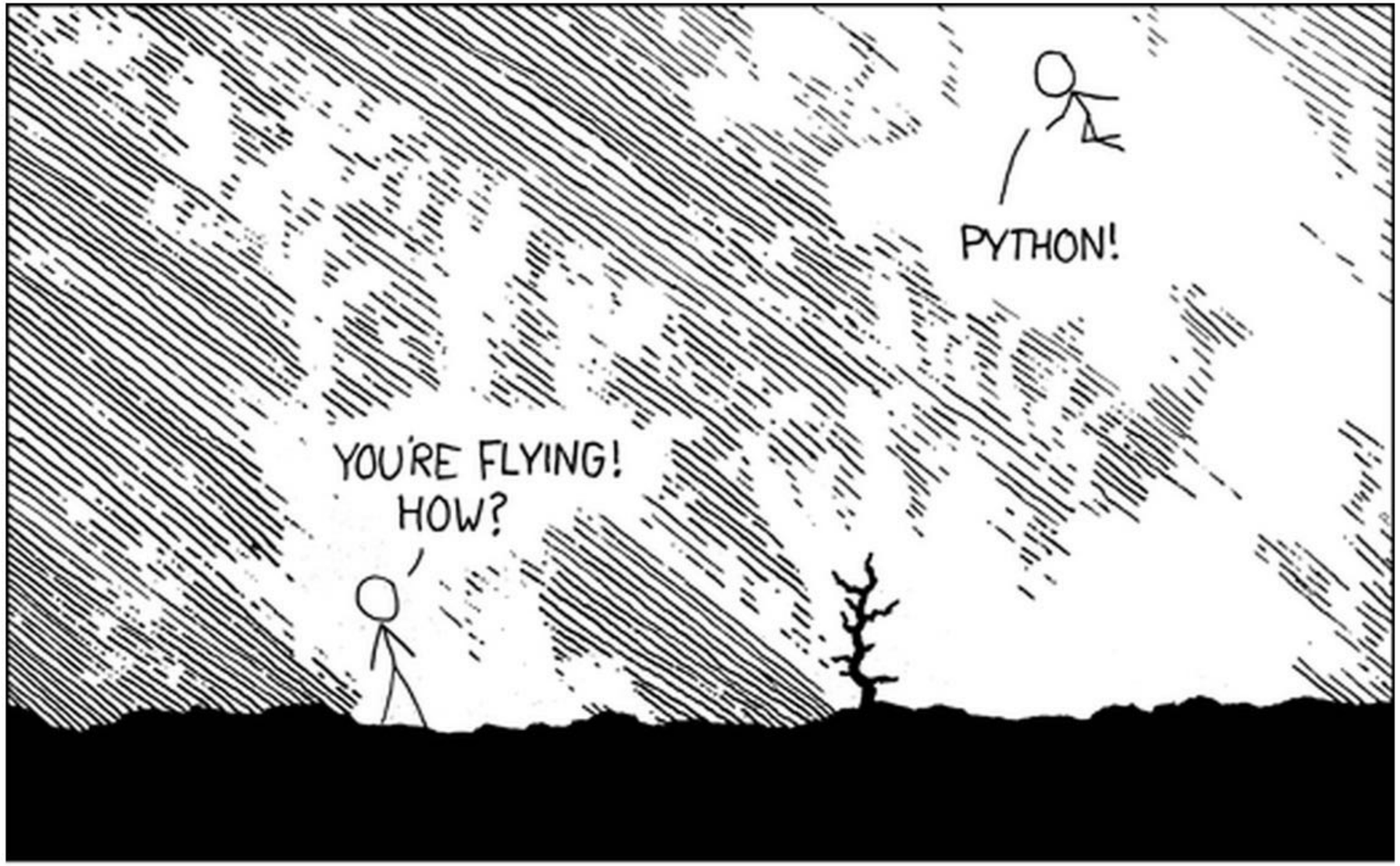
Python isn’t the best at anything.

Why Python?

Python isn't the best at anything.

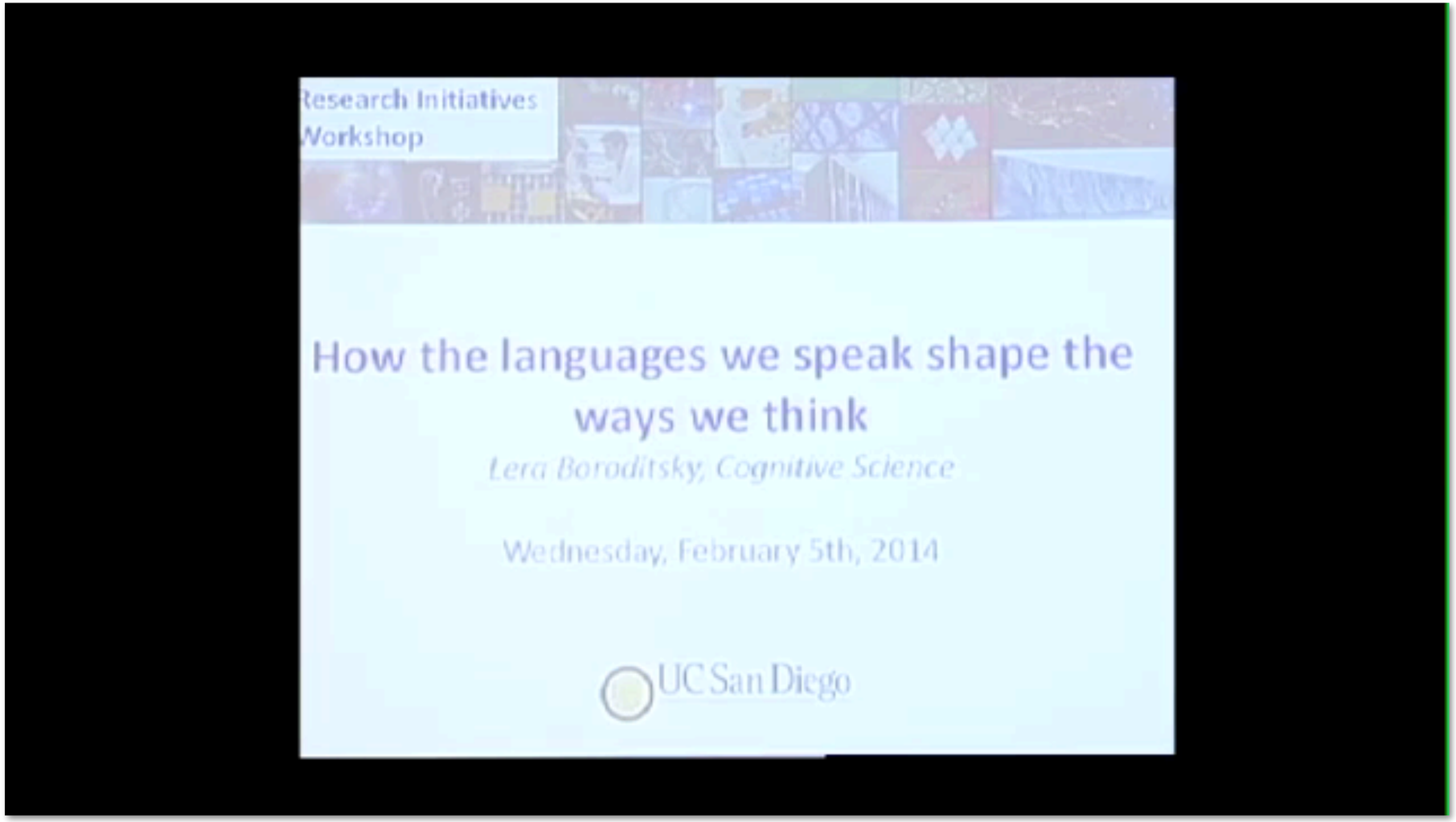
But it's the *second best* at EVERYTHING.

Why Python?



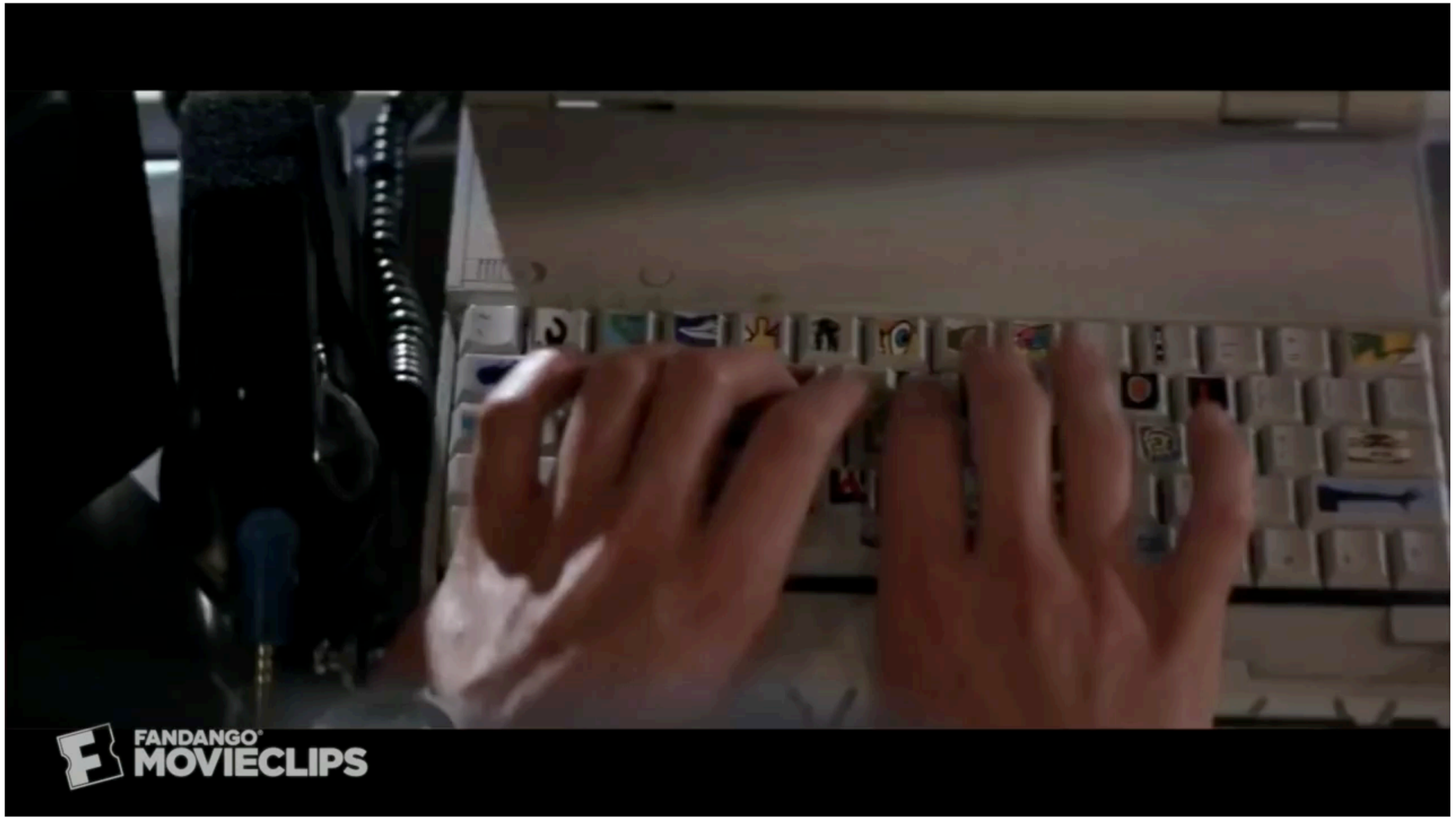
Source: *XKCD*

Languages shape thought



Source: Lera Boroditsky

Getting computers to work for you



Source: *Hackers*

Getting computers to work for you

Source: *Hackers*

The future of programming

Source: Bret Victor, *Turing Complete* 2012

Jupyter - Magics

```
In [ ]: %quickref
```

```
IPython -- An enhanced Interactive Python - Quick Reference Card
================================================================

obj?, obj??      : Get help, or more help for object (also works as
                   ?obj, ??obj).
?foo.*abc*       : List names in 'foo' containing 'abc' in them.
%magic           : Information about IPython's 'magic' % functions.

Magic functions are prefixed by % or %%, and typically take their arguments
without parentheses, quotes or even commas for convenience.  Line magics take a
single % and cell magics are prefixed with two %%.

Example magic function calls:

%alias d ls -F   : 'd' is now an alias for 'ls -F'
alias d ls -F    : Works if 'alias' not a python name
alist = %alias   : Get list of aliases to 'alist'
cd /usr/share    : Obvious. cd -<tab> to choose from visited dirs.
%cd??            : See help AND source for magic %cd
%timeit x=10     : time the 'x=10' statement with high precision.
%%timeit x=2**100
```

Jupyter - Magics

```
In [46]: %pwd

Out[46]: '/Users/Voytek'
```

Jupyter - Magics

In [47]:
```
# print object details

x = 2
x?
```

```
Type:        int
String form: 2
Docstring:
int(x=0) -> integer
int(x, base=10) -> integer

Convert a number or string to an integer, or return 0 if no arguments
are given.  If x is a number, return x.__int__().  For floating point
numbers, this truncates towards zero.

If x is not a number or if base is given, then x must be a string,
bytes, or bytearray instance representing an integer literal in the
given base.  The literal can be preceded by '+' or '-' and be surrounded
by whitespace.  The base defaults to 10.  Valid bases are 0 and 2-36.
Base 0 means to interpret the base from the string as an integer literal.
>>> int('0b100', base=0)
4
```

Jupyter - Magics

```
In [50]: # print object details

         my_string = 'hello world'
         my_string?
```

```
Type:        str
String form: hello world
Length:      11
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

Create a new string object from the given object. If encoding or
errors is specified, then the object must expose a data buffer
that will be decoded using the given encoding and error handler.
Otherwise, returns the result of object.__str__() (if defined)
or repr(object).
encoding defaults to sys.getdefaultencoding().
errors defaults to 'strict'.
```

Jupyter - Tab completion

```
In [ ]: # tab completion

        print(my_string.())
```

Jupyter - Tab completion

```
In [ ]: # tab completion

        print(my_string.())
```

```
In [ ]: # tab completion

        print(my_string.())
```

```
my_string.capitalize
my_string.casefold
my_string.center
my_string.count
my_string.encode
my_string.endswith
my_string.expandtabs
my_string.find
my_string.format
my_string.format_map
```

dropdown listing all the methods associated with my_string!

Jupyter - Tab completion

```
In [ ]: # tab completion

        print(my_string.())
```

```
In [ ]: # tab completion

        print(my_string.|())
```

```
my_string.rstrip
my_string.split
my_string.splitlines
my_string.startswith
my_string.strip
my_string.swapcase
my_string.title
my_string.translate
my_string.upper
my_string.zfill
```

```
In [ ]: # mmul

        my_lis
            'c
            't
            't
            'i
        ]
        print(
```

Jupyter - Tab completion

```
In [ ]: # tab completion

        print(my_string.())
```

```
In [ ]: # tab completion

        print(my_string.())
```

```
my_string.rstrip
my_string.split
my_string.splitlines
my_string.startswith
my_string.strip
my_string.swapcase
my_string.title
my_string.translate
my_string.upper
my_string.zfill
```

```
In [ ]: # mmul

        my_lis
            '
            '
            '
            '
        ]
        print(
```

```
In [51]: # tab completion

         print(my_string.upper())
```

```
HELLO WORLD
```

Jupyter - Multicursor support

In [52]:
```
# mmulticursor

my_list = [
    'one'
    'two'
    'three'
    'four'
]
print(my_list)
```

```
['onetwothreefour']
```

Jupyter - Multicursor support

```
In [52]: # mmulticursor

         my_list = [
             'one'
             'two'
             'three'
             'four'
         ]
         print(my_list)

         ['onetwothreefour']
```

```
In [52]: # mmulticursor

         my_list = [
             'one'
             'two'
             'three'
             'four'
         ]
         print(my_list)
```

hold down *alt* while highlighting, then press *right arrow*

Jupyter - Multicursor support

```
In [52]: # mmulticursor

         my_list = [
             'one'
             'two'
             'three'
             'four'
         ]
         print(my_list)
```

```
['onetwothreefour']
```

```
In [52]: # mmulticursor

         my_list = [
             'one',
             'two',
             'three',
             'four',
         ]
         print(my_list)
```

add a *comma* to the end of all lines, all at once!

Jupyter - Multicursor support

```
In [52]: # mmulticursor

         my_list = [
             'one'
             'two'
             'three'
             'four'
         ]
         print(my_list)

         ['onetwothreefour']
```

```
In [52]: # mmulticursor

         my_list = [
             'one',
             'two',
             'three',
             'four',
         ]
         print(my_list)
```

```
In [53]: # mmulticursor

         my_list = [
             'one',
             'two',
             'three',
             'four',
         ]
         print(my_list)

         ['one', 'two', 'three', 'four']
```

Jupyter - Magics

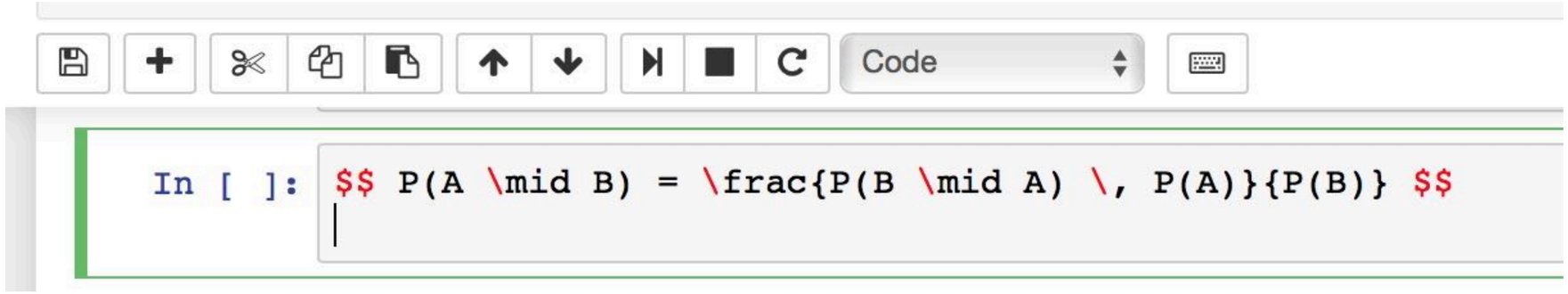
```
In [54]: %who

math      my_list          my_string
```

```
In [55]: %whos

Variable      Type       Data/Info
----------------------------------
math          module     <module 'math' from '/Use<...>h.cpython-35m-darwin.so'>
my_list       list       n=4
my_string     str        hello world
```

Jupyter - Markdown



Jupyter - Markdown

Code

```
In [ ]: $$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$
```

Markdown

```
$$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$
```

Jupyter - Markdown

Code

```
In [ ]: $$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$
```

Markdown

```
$$ P(A \mid B) = \frac{P(B \mid A) \, P(A)}{P(B)} $$
```

Markdown

$$P(A \mid B) = \frac{P(B \mid A)\, P(A)}{P(B)}$$

Bradley Voytek, Ph.D.

UC San Diego

Department of Cognitive Science
Neurosciences Graduate Program
Halicioglu Data Science Institute

bvoytek@ucsd.edu
@bradleyvoytek