Bradley Voytek, Ph.D.
UC San Diego
Cognitive and Neural Dynamics Laboratory

Department of Cognitive Science
Neurosciences Graduate Program
The Institute for Neural Computation

bvoytek@ucsd.edu
@bradleyvoytek

Administrative stuff

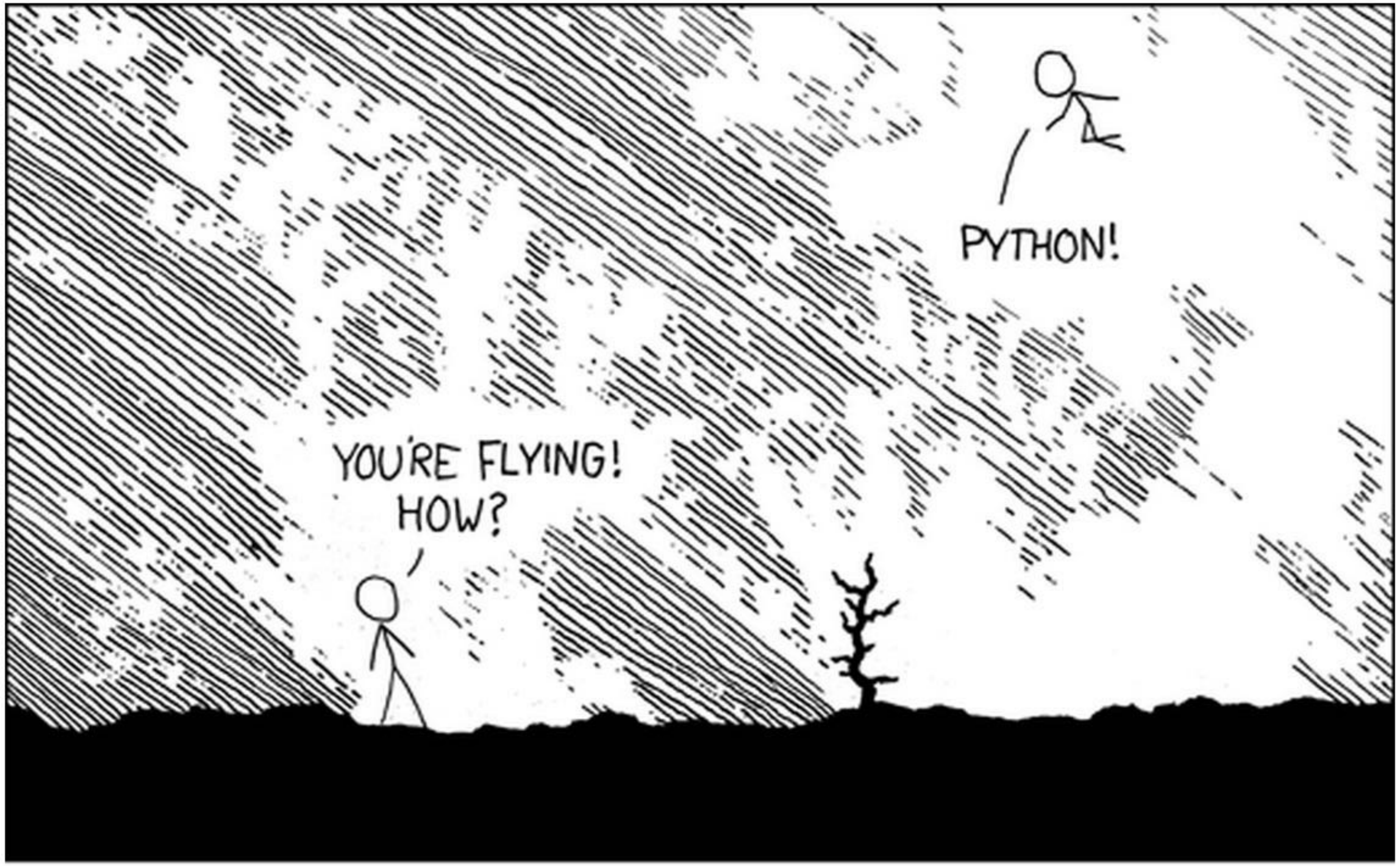
- Waitlist… The first **26** on the waitlist will get added. That's *all*.
- Final Project: More details coming this week!
- Everyone should be on Piazza now—I updated the list again as of this morning. If you are not, add your email to the list:
  - http://bit.ly/COGS108S17Pz
- Python3 (use Anaconda for simplicity!)

COGS 108
Data Science in Practice

Data Science in Python

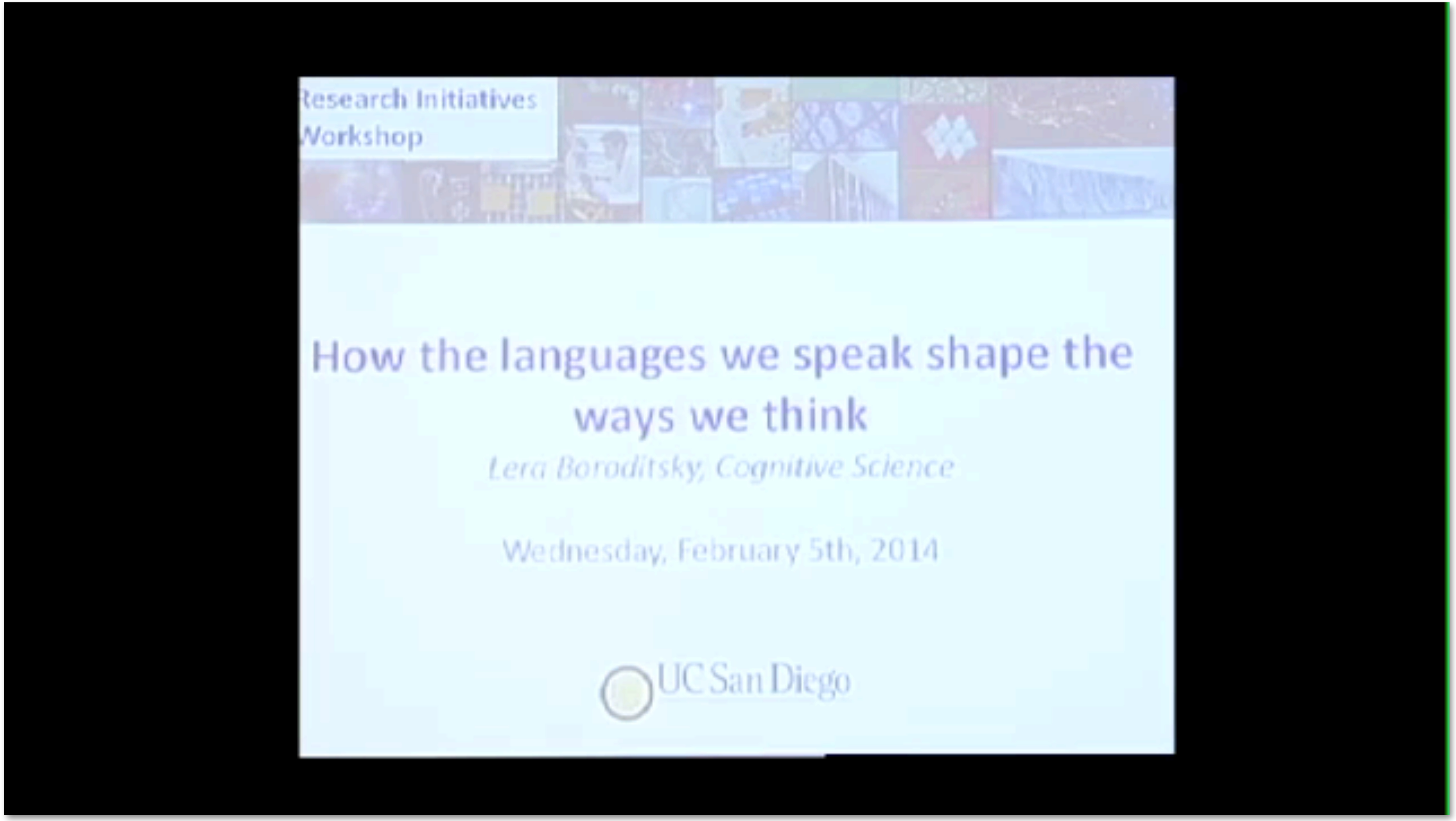
UC San Diego

Why Python?



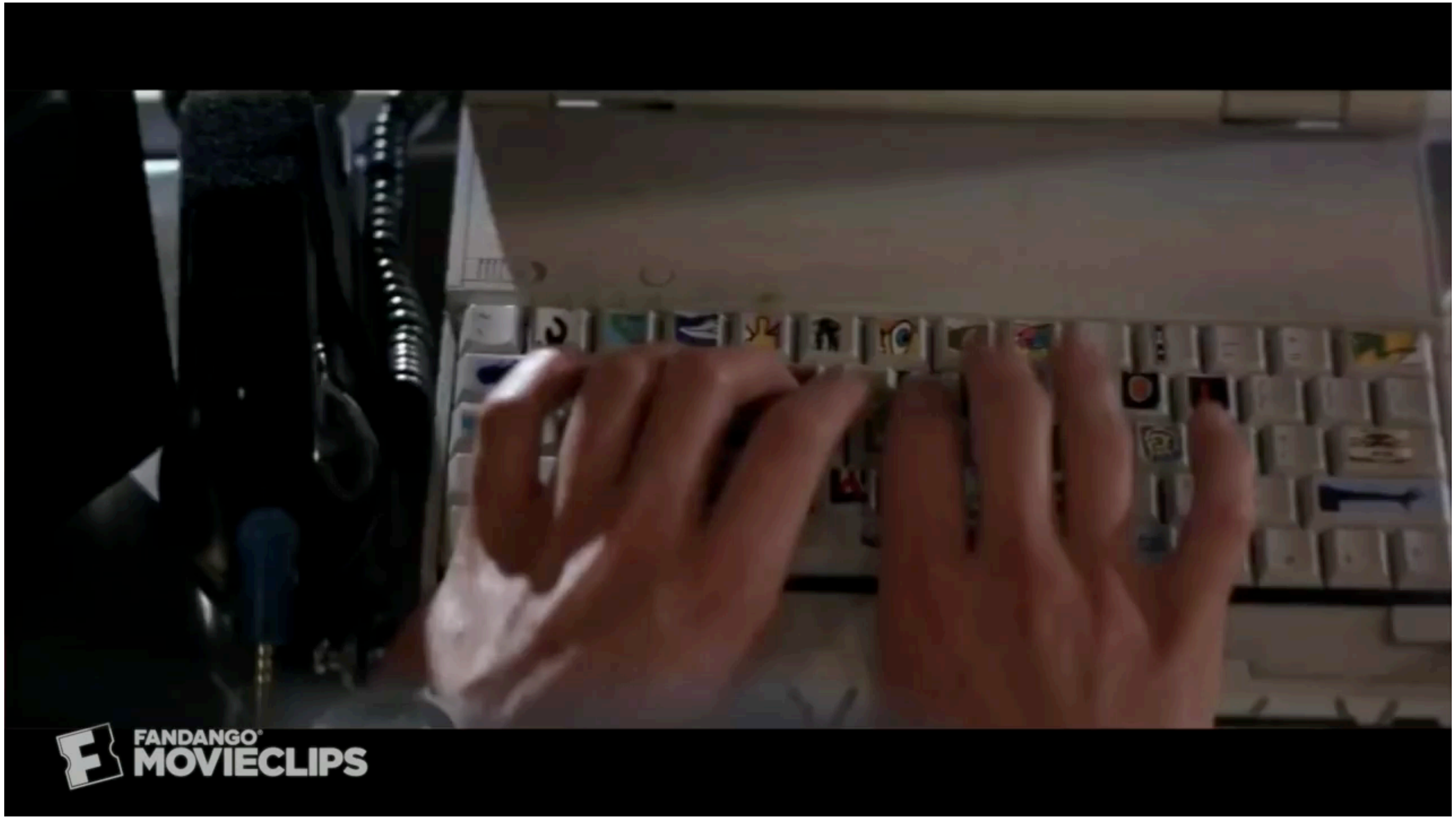
Source: *XKCD*

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

UC San Diego

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]:
```python
# 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 [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

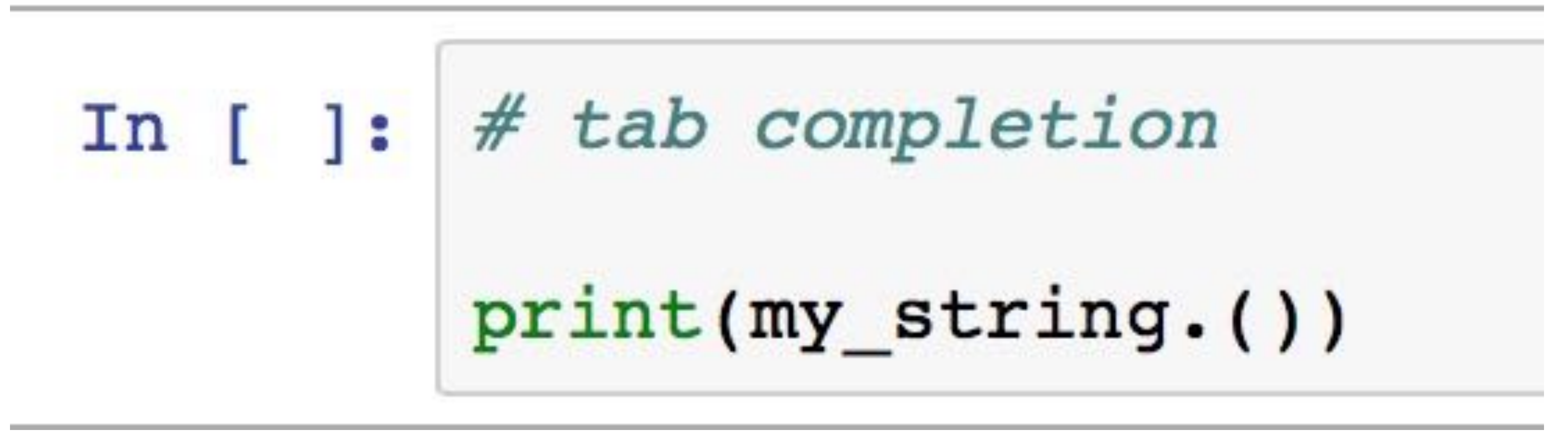
```
In [50]: # print object details

         my_string = 'hello world'
         my_string?
```

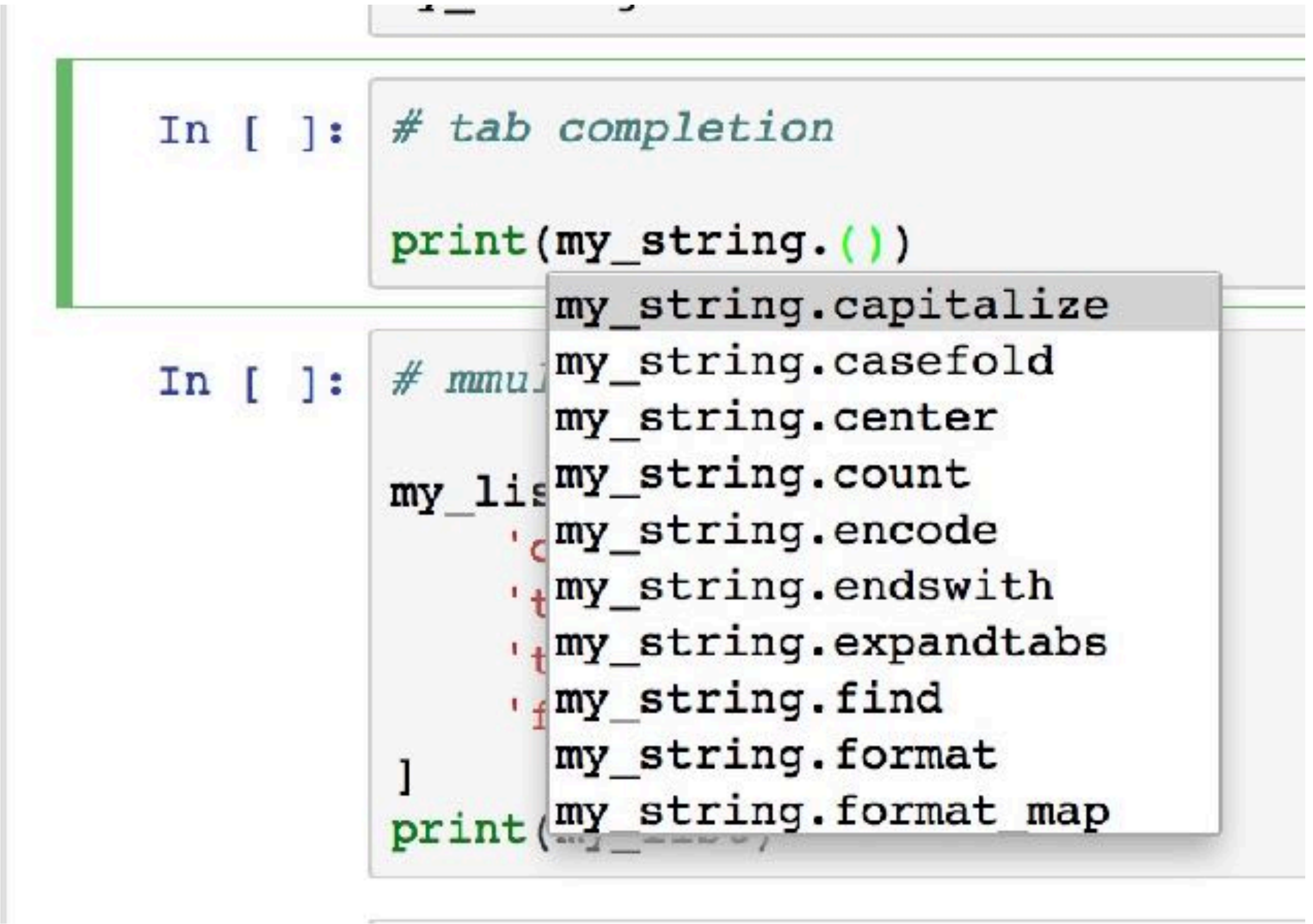
```
Type:           str
String form:    hello world
Length:         11
Docstring:
str(object='') -> str
str(bytes_or_buffer[, encoding[, errors]]) -> str

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.())
```

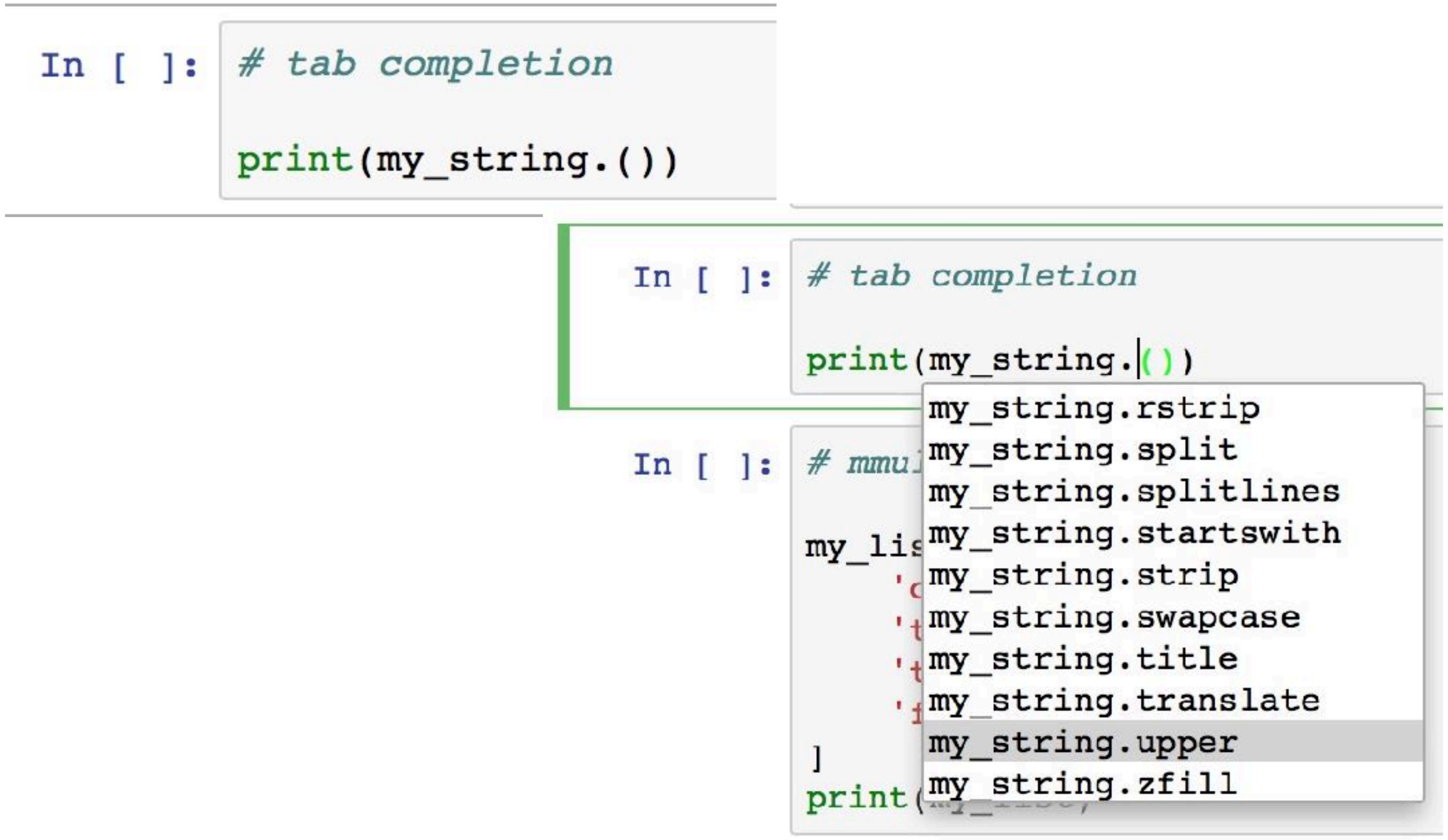
Jupyter - Tab completion





dropdown listing all the methods associated with my_string!

Jupyter - Tab completion



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 [ ]: # mmu

        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)
```

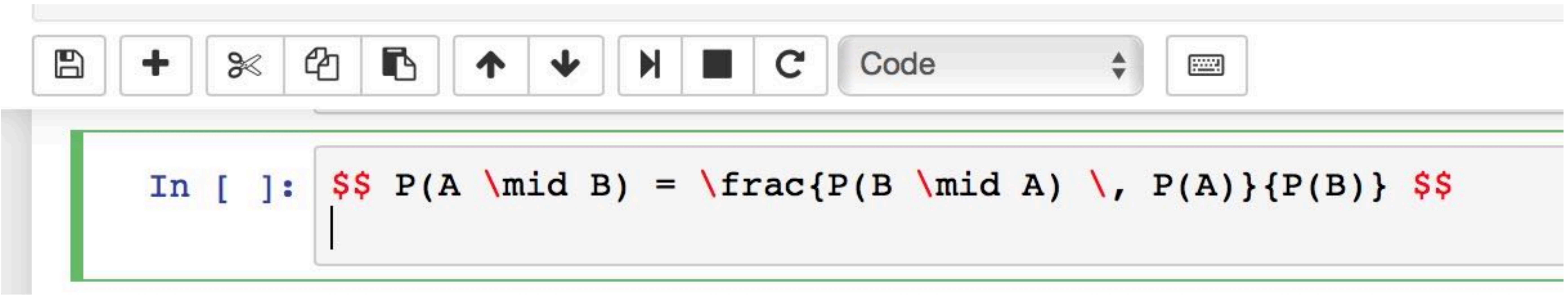
```
['one', 'two', 'three', 'four']
```

Jupyter - Magics

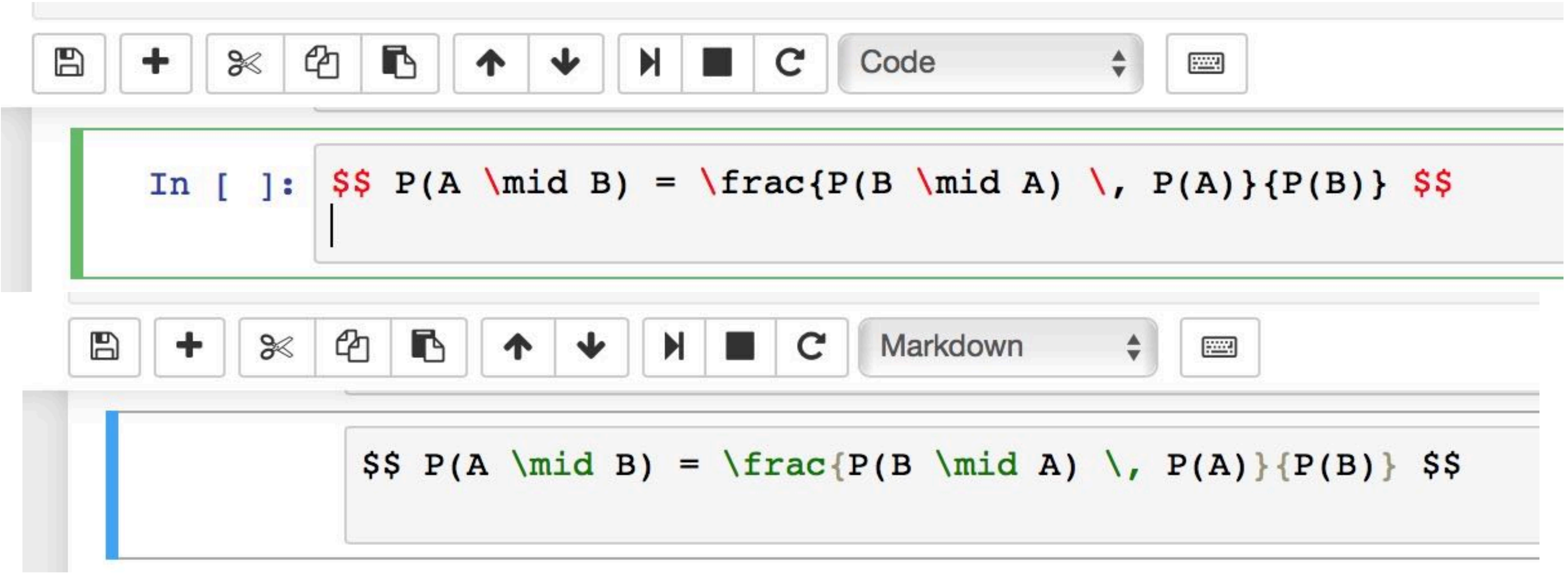
```
In [54]: %who

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
```

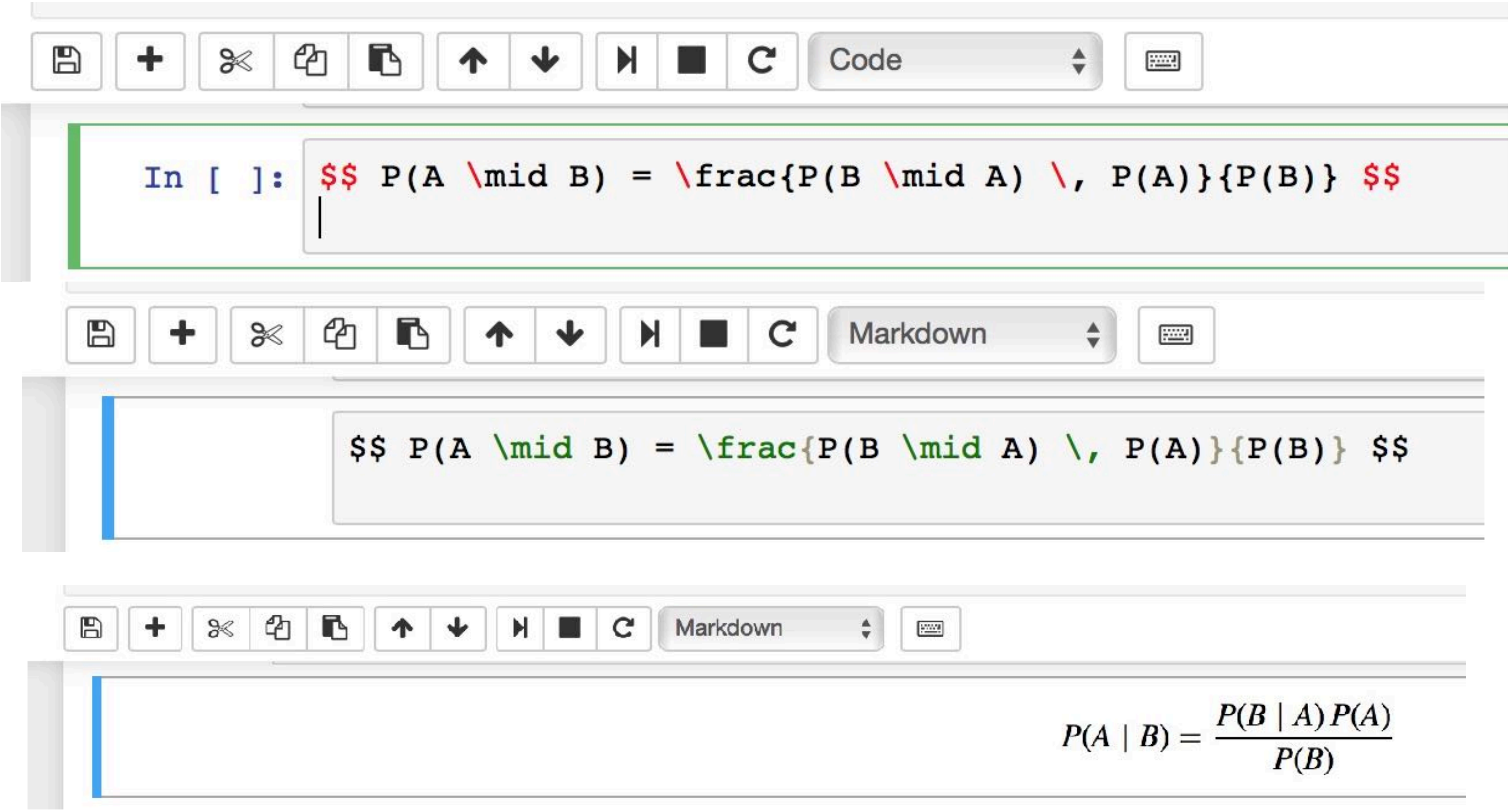
Jupyter - Markdown



Jupyter - Markdown



Jupyter - Markdown



Jupyter - Markdown

```
# Large-scale analysis of practice effects on interference across the lifespan

Behavioral data were collected from Lumosity, a web-based suite of games voluntarily played <em>ad libitum</em> by
users who pay a subscription fee to use the service. Anonymized data from the game "Lost in Migration"—a variant on
the traditional Eriksen Flanker task (see Eriksen & Eriksen, 1974)(Fig. 1)—were shared with the authors for purposes
of scientific research. Data were collected from <em>N</em> users aged 18-70, each of whom completed at least 24 game
sessions and one practice session. Lumosity's users assent to Terms of Service indicating that their anonymized data
may be used in aggregate for research purposes.

The "Lost in Migration" game (Fig. 1) is similar to the Eriksen Flanker task in that users respond to which of the
four possible directions a central bird is facing using the arrow keys on their computer keyboards. Four other birds,
each of which is facing in the same direction as one another, surround this central bird. There are two primary trial
types in this task: congruent and incongruent. In the congruent condition the central bird is facing the same
direction as the four surrounding birds; in the incongruent condition the central bird is facing in a different
direction. Each session lasts for 45 seconds. The within-subjects RT difference between incongruent and congruent
conditions was used to index interference.

<em>Note RT difference may not be right, I need help figuring out what to use here.</em>

Because of the relatively large sample size, even trivially small effects prove to be statistically significant so the
goal of this is largely model comparison and knowledge discovery.

<img src="./Lumos-TaskFigure.jpg", width=800/>
<strong>Fig. 1.</strong> Behavioral task. Examples of the two conditions included in the behavioral paradigm that
formed the basis of these analyses. In this task—a modified Flanker paradigm— subjects report the direction of the
central bird. On the left is an example of a congruent trial wherein the central target bird is facing in the same
direction as the flanking stimuli. On the right is an example of an incongruent trial wherein the central target bird
is facing in a different direction. The weighted percent difference between response times between the two trial types
gives an interference index, a measure of cognitive control.
```

Jupyter - Markdown

Large-scale analysis of practice effects on interference across the lifespan

Behavioral data were collected from Lumosity, a web-based suite of games voluntarily played *ad libitum* by users who pay a subscription fee to use the service. Anonymized data from the game "Lost in Migration"—a variant on the traditional Eriksen Flanker task (see Eriksen & Eriksen, 1974)(Fig. 1)—were shared with the authors for purposes of scientific research. Data were collected from *N* users aged 18-70, each of whom completed at least 24 game sessions and one practice session. Lumosity's users assent to Terms of Service indicating that their anonymized data may be used in aggregate for research purposes.

The "Lost in Migration" game (Fig. 1) is similar to the Eriksen Flanker task in that users respond to which of the four possible directions a central bird is facing using the arrow keys on their computer keyboards. Four other birds, each of which is facing in the same direction as one another, surround this central bird. There are two primary trial types in this task: congruent and incongruent. In the congruent condition the central bird is facing the same direction as the four surrounding birds; in the incongruent condition the central bird is facing in a different direction. Each session lasts for 45 seconds. The within-subjects RT difference between incongruent and congruent conditions was used to index interference.

Note RT difference may not be right, I need help figuring out what to use here.

Because of the relatively large sample size, even trivially small effects prove to be statistically significant so the goal of this is largely model comparison and knowledge discovery.

Fig. 1. Behavioral task. Examples of the two conditions included in the behavioral paradigm that formed the basis of these analyses. In this task—a modified Flanker paradigm— subjects report the direction of the central bird. On the left is an example of a congruent trial wherein the central target bird is facing in the same direction as the flanking stimuli. On the right is an example of an incongruent trial wherein the central target bird is facing in a different direction. The

Jupyter - Beginning an analysis

```
In [1]: % reset
        % config InlineBackend.figure_format = 'retina'
        import matplotlib.pyplot as plt
        from matplotlib import rcParams
        import numpy as np
        import scipy as sp
        import scipy.stats
        import scipy.io
        from scipy.optimize import curve_fit
        from scipy.optimize import least_squares
        % matplotlib inline
        from pylab import rcParams
        rcParams['figure.figsize'] = 8, 6
        rcParams['font.family'] = 'sans-serif'
        rcParams['font.sans-serif'] = ['Tahoma']
```

Jupyter - Beginning an analysis

magic to clear all variables

In [1]:

```
% reset
% config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from matplotlib import rcParams
import numpy as np
import scipy as sp
import scipy.stats
import scipy.io
from scipy.optimize import curve_fit
from scipy.optimize import least_squares
% matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

UC San Diego

Jupyter - Beginning an analysis

magic for high resolution figures

In [1]:
```
% reset
% config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from matplotlib import rcParams
import numpy as np
import scipy as sp
import scipy.stats
import scipy.io
from scipy.optimize import curve_fit
from scipy.optimize import least_squares
% matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

Jupyter - retina resolution

ICK

Fig. 2

Jupyter - retina resolution

ICK

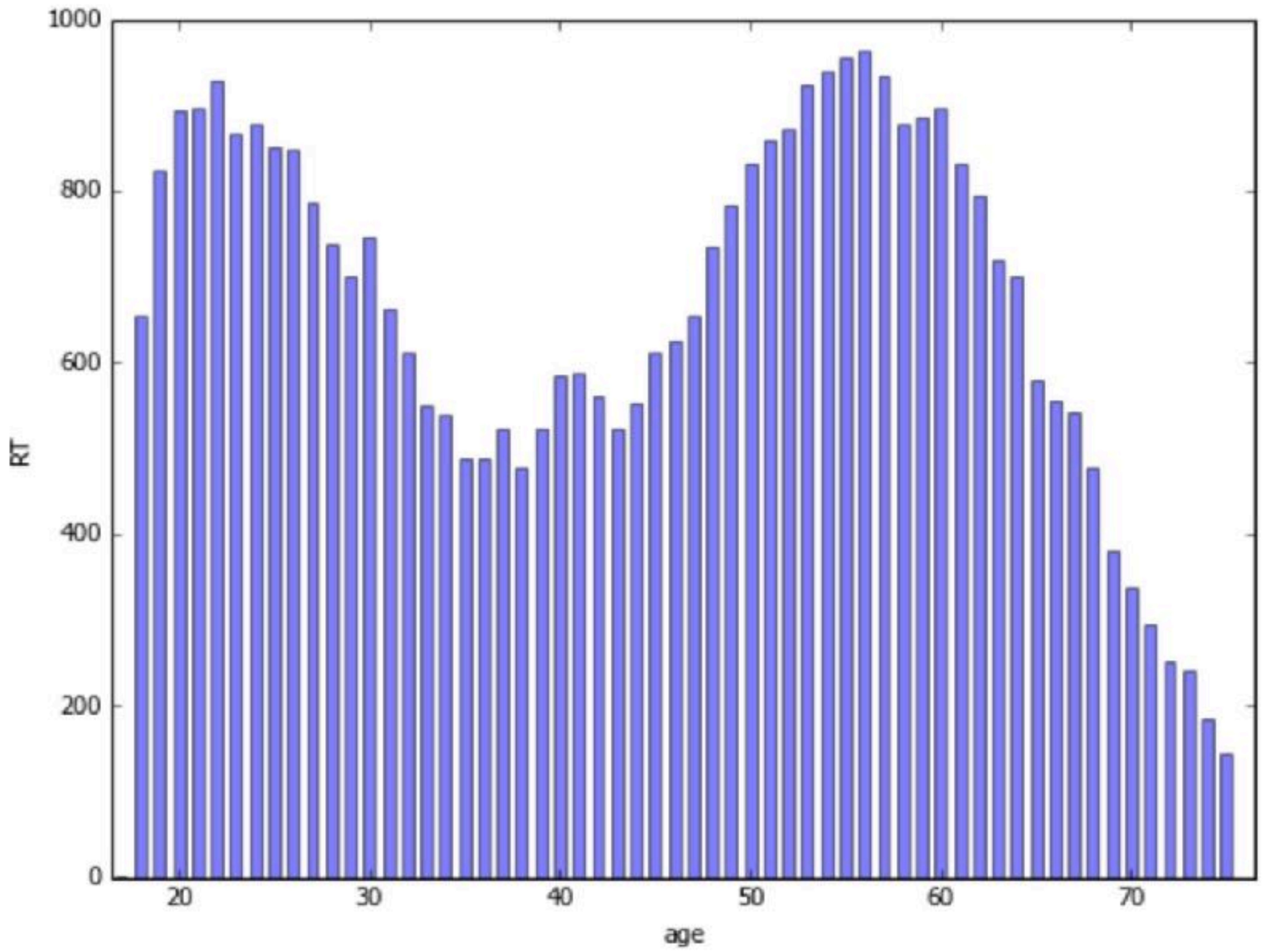


Fig. 2

YAY

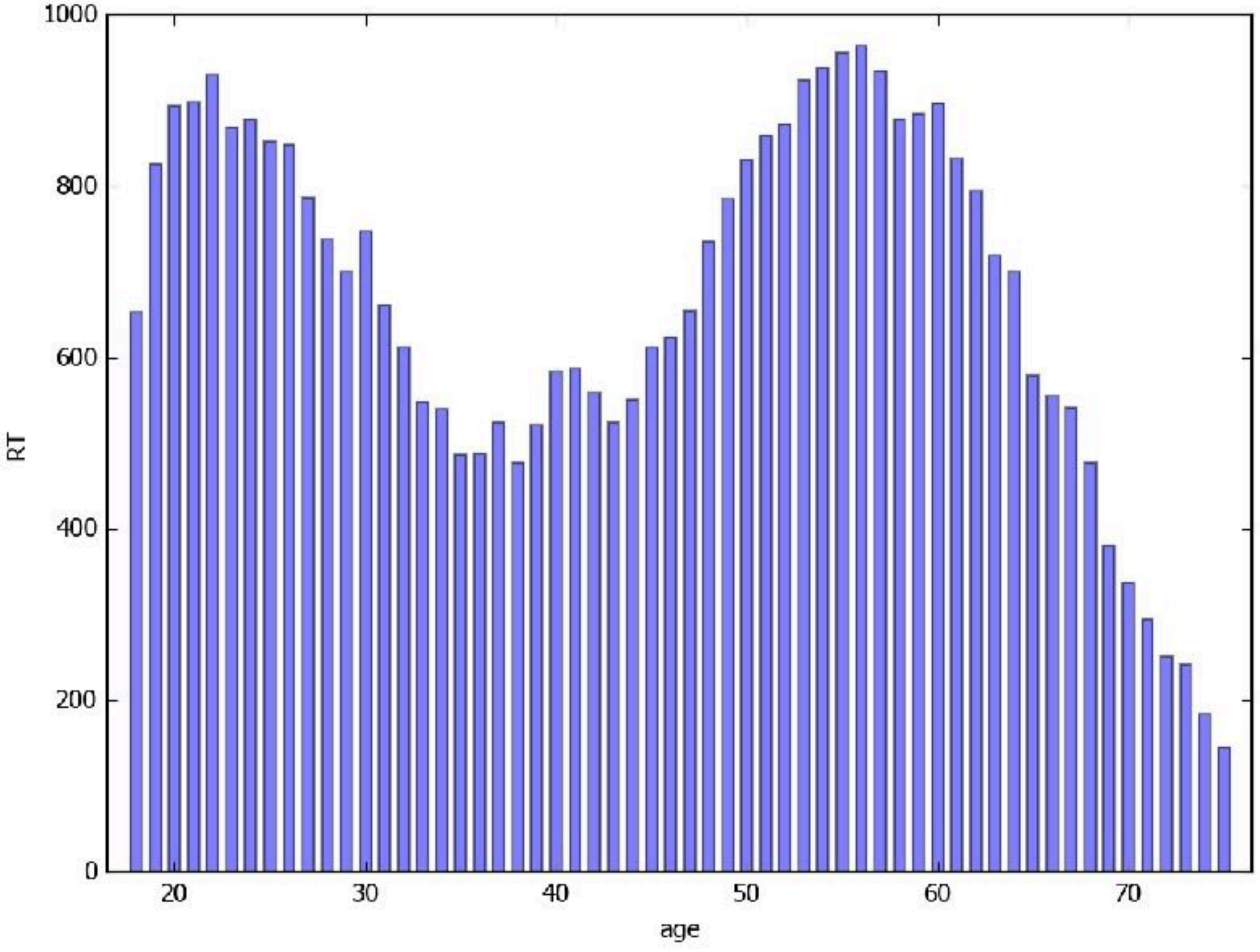


Fig. 2

Jupyter - Beginning an analysis

In [1]:

```
% reset
% config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from matplotlib import rcParams
import numpy as np
import scipy as sp
import scipy.stats
import scipy.io
from scipy.optimize import curve_fit
from scipy.optimize import least_squares
% matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

magic to allow inline plotting

Jupyter - Plots in-line!

```
plt.plot(trials, rtc_by_age[0, :], 'sb', label='20-30 years (c)')
plt.plot(trials, rti_by_age[0, :], '.b', label='20-30 years (i)')
plt.plot(trials, rtc_by_age[1, :], 'sg', label='40-50 years (c)')
plt.plot(trials, rti_by_age[1, :], '.g', label='40-50 years (i)')
plt.plot(trials, rtc_by_age[2, :], 'sr', label='60-70 years (c)')
plt.plot(trials, rti_by_age[2, :], '.r', label='60-70 years (i)')
plt.title("RT by trial")
plt.xlabel("trial")
plt.ylabel("RT")
plt.legend(loc=1)
plt.figtext(.02, .02, "Fig. 3")
plt.show()
```

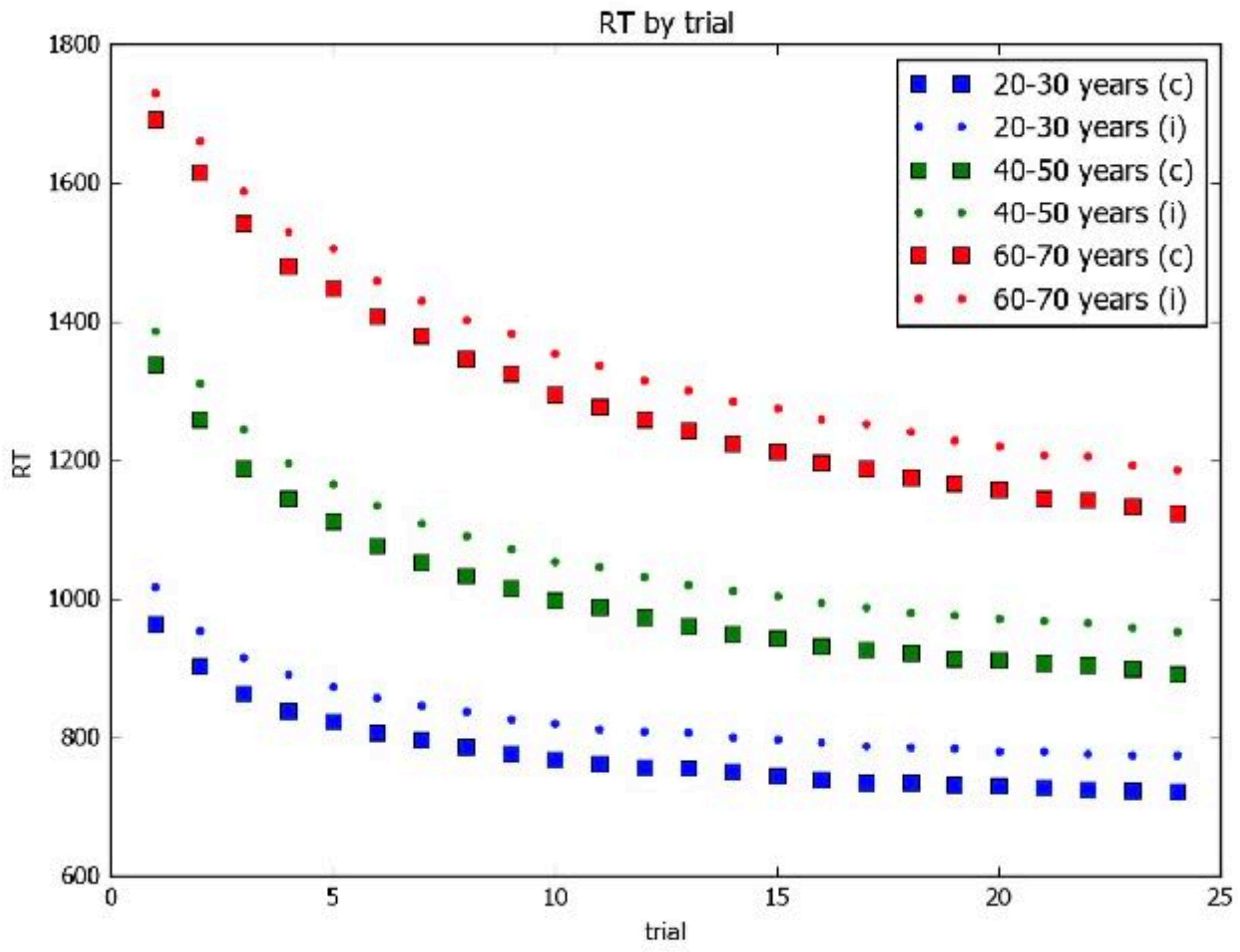


Fig. 3

Jupyter - Beginning an analysis

In [1]:

```
% reset
% config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from matplotlib import rcParams
import numpy as np
import scipy as sp
import scipy.stats
import scipy.io
from scipy.optimize import curve_fit
from scipy.optimize import least_squares
% matplotlib inline
from pylab import rcParams
rcParams['figure.figsize'] = 8, 6
rcParams['font.family'] = 'sans-serif'
rcParams['font.sans-serif'] = ['Tahoma']
```

TOTALLY REDUNDANT

Bradley Voytek, Ph.D.

UC San Diego
Cognitive and Neural Dynamics Laboratory

Department of Cognitive Science
Neurosciences Graduate Program
The Institute for Neural Computation

bvoytek@ucsd.edu
@bradleyvoytek