Notebooks are still at: https://github.com/minrk/ipython-cse17

Or follow along on JupyterHub at https://cse17.jupyter.org

Profiling and Optimising

IPython provides some tools for making it a bit easier to profile and optimise your code.

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
try:
    import seaborn as sns
except ImportError:
    print("That's okay")
```

%timeit

The main IPython tool we are going to use here is `%timeit`, a magic that automates measuring how long it

File Edit View Insert Cell Kernel Help Trusted Python 3

Notebooks are still at: https://github.com/minrk/ipython-cse17

Or follow along on JupyterHub at https://cse17.jupyter.org

Profiling and Optimising

IPython provides some tools for making it a bit easier to profile and optimise your code.

In [ ]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
```

In [ ]:
```
try:
    import seaborn as sns
except ImportError:
    print("That's okay")
```

%timeit

The main IPython tool we are going to use here is `%timeit`, a magic that automates measuring how long it

```
Size: 1000 x 1000
10 loops, best of 3: 24.9 ms per loop
Size: 2000 x 2000
1 loop, best of 3: 253 ms per loop
```

Let's look at what options `%timeit` can take.

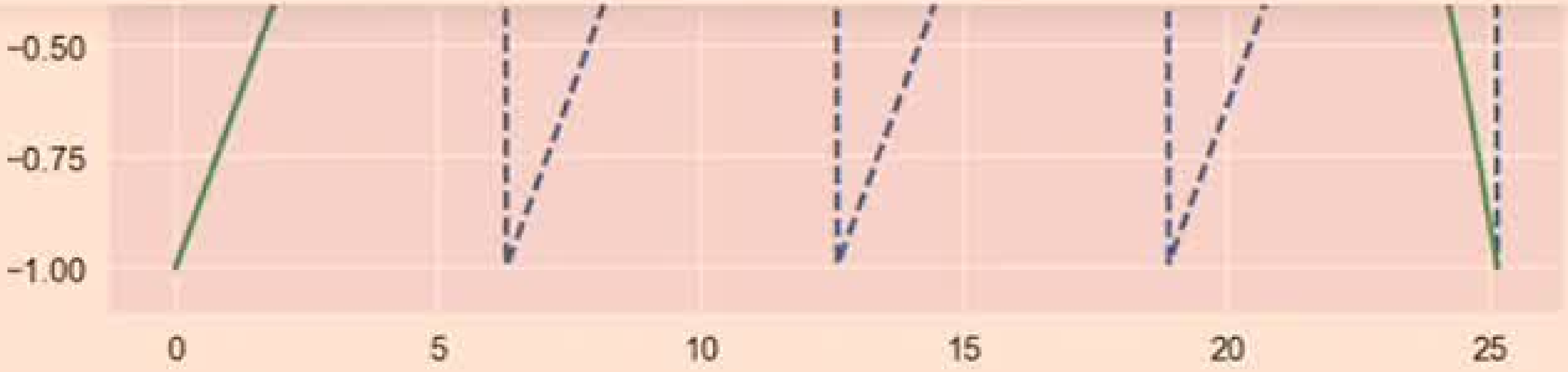
In [4]:
```
%timeit?
```

```
Docstring:
Time execution of a Python statement or expression

Usage, in line mode:
  %timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] statement
or in cell mode:
  %%timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] setup_code
  code
  code...

Time execution of a Python statement or expression using the ti
meit
module.  This function can be used both as a line and cell magi
c:

- In line mode you can time a single-line statement (though mul
tiple
  ones can be chained with using semicolons).
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
  %timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] statement
or in cell mode:
  %%timeit [-n<N> -r<R> [-t|-c] -q -p<P> -o] setup_code
  code
  code...

Time execution of a Python statement or expression using the ti
meit
module.  This function can be used both as a line and cell magi
c:

- In line mode you can time a single-line statement (though mul
tiple
  ones can be chained with using semicolons).

- In cell mode, the statement in the first line is used as setu
p code
  (executed but not timed) and the body of the cell is timed.
The cell
  body has access to any variables created in the setup code.

Options:
-n<N>: execute the given statement <N> times in a loop. If this
value
is not given, a fitting value is chosen.

-r<R>: repeat the loop iteration <R> times and take the best re
```

File Edit View Insert Cell Kernel Help Trusted Python 3

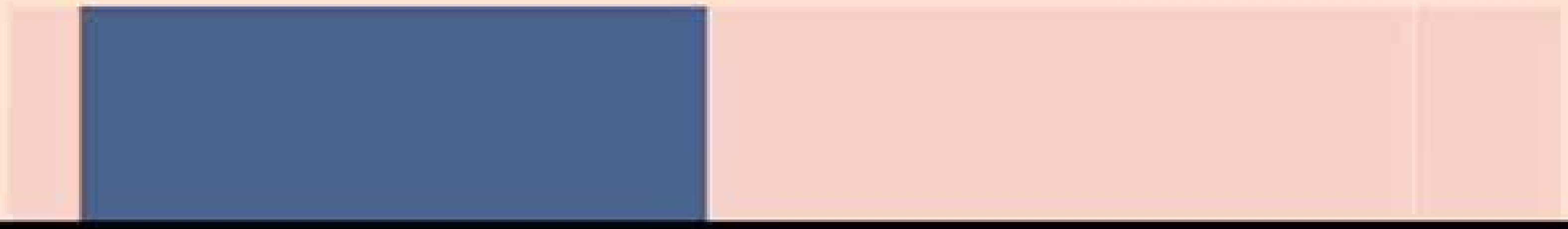


In [16]:
```
np_r = %timeit -o blur_np(x, steps)
t_np = np_r.best
```

```
100 loops, best of 3: 11.3 ms per loop
```

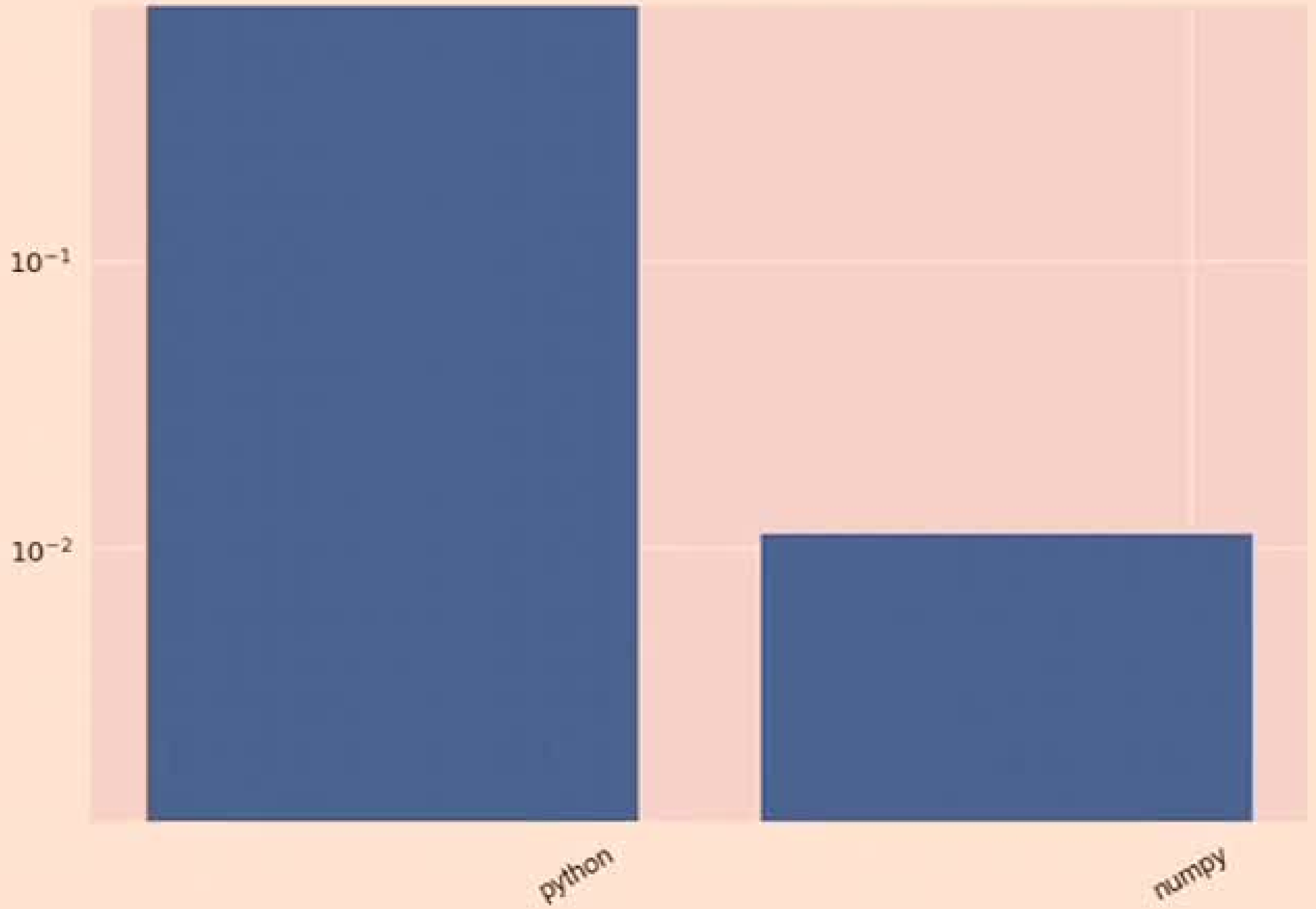
In [17]:
```
times.append(t_np)
labels.append('numpy')
```

In [18]:
```
def plot_times():
    ind = np.arange(len(times))
    plt.bar(ind, times, log=True)
    plt.xticks(ind + 0.3, labels, rotation=30)
    plt.ylim(.1 * min(times), times[0])
plot_times()
```



File Edit View Insert Cell Kernel Help Trusted Python 3

```
plot_times()
```

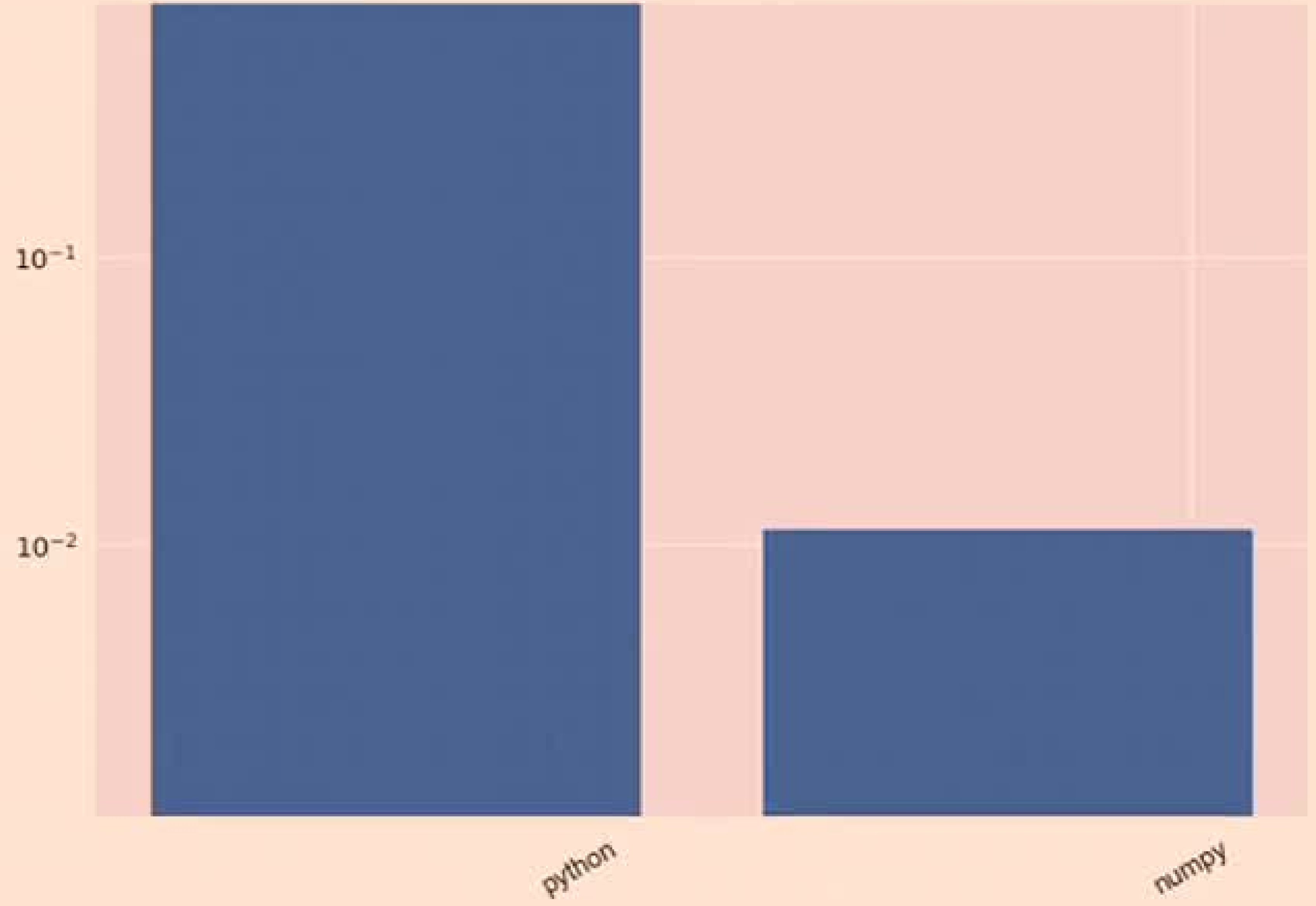


So vectorizing the inner loop brings us from 1 second to 25 milliseconds, an improvement of 40x:

```
In [ ]: t_ref / t_np
```

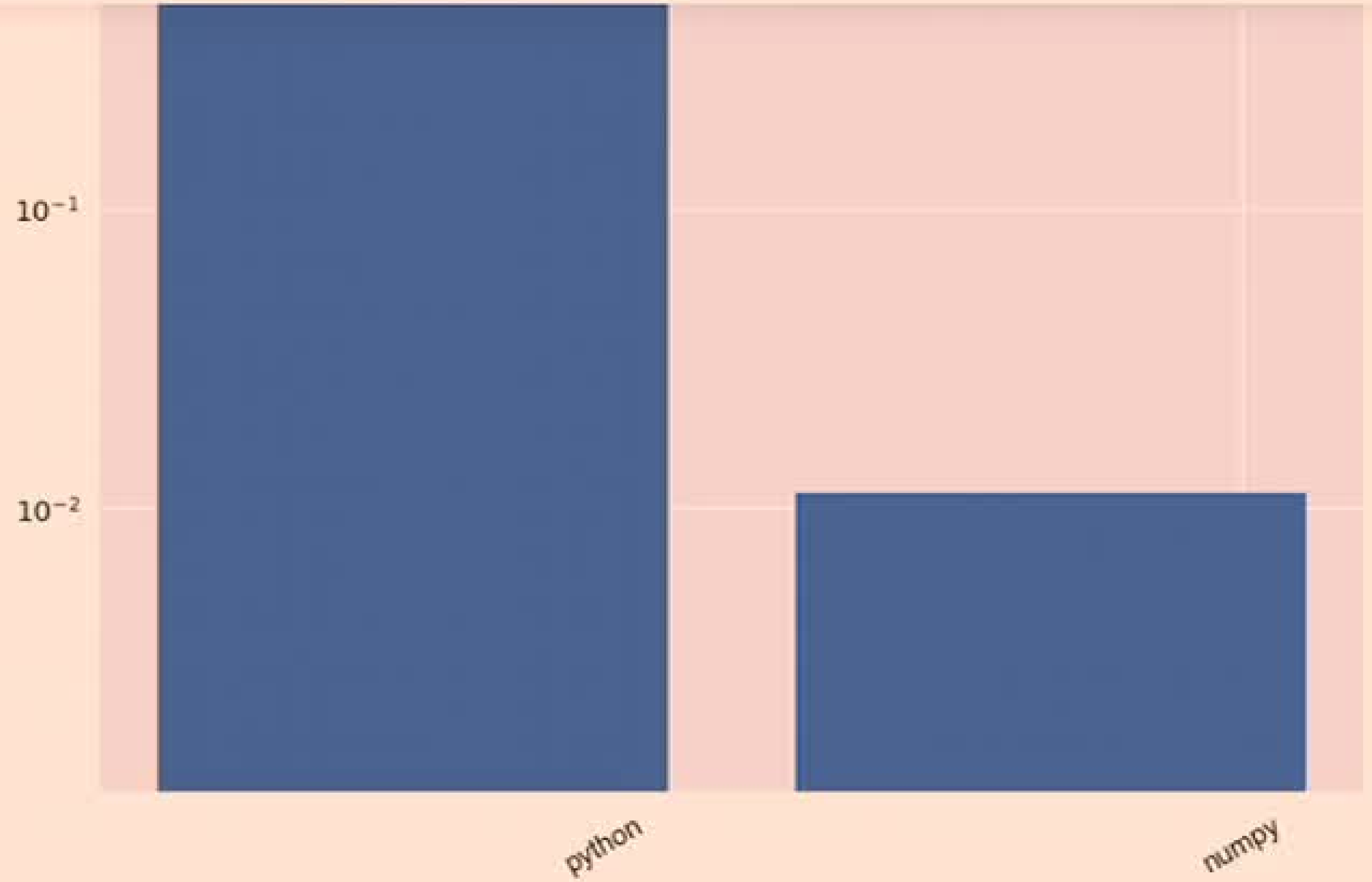
File Edit View Insert Cell Kernel Help Trusted Python 3

```
plot_times()
```



So vectorizing the inner loop brings us from 1 second to 25 milliseconds, an improvement of 40x:

File Edit View Insert Cell Kernel Help Trusted Python 3



So vectorizing the inner loop brings us from 1 second to 25 milliseconds, an improvement of 40x:

In [ ]:
```
t_ref / t_np
```

```
import numpy as np

def blur_cython(x, steps=1024):
    x = 1 * x # copy
    y = np.empty_like(x)
    y[0] = x[0]
    y[-1] = x[-1]
    for _ in range(steps):
        for i in range(1, len(x)-1):
            y[i] = .25 * ( x[i-1] + 2 * x[i] + x[i+1] )
        x, y = y, x # swap for next step
    return x
```

In [*]:
```
c1 = %timeit -o y = blur_cython(x, steps)
t_c1 = c1.best
times.append(t_c1)
labels.append("cython (no hints)")
```

In [*]:
```
plot_times()
```

Without annotations, we don't get much improvement over the pure Python version. We can note the types of the input arguments, to get some improvements:

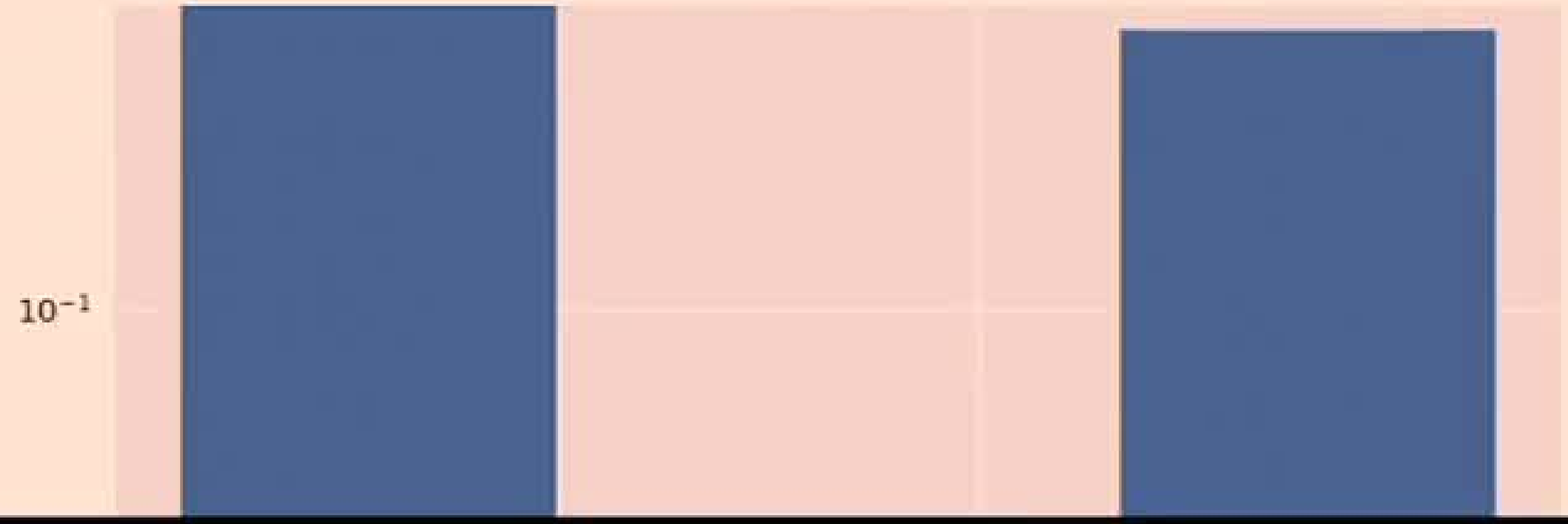
File Edit View Insert Cell Kernel Help Trusted Python 3

```
    y[0] = x[0]
    y[-1] = x[-1]
    for _ in range(steps):
        for i in range(1, len(x)-1):
            y[i] = .25 * ( x[i-1] + 2 * x[i] + x[i+1] )
        x, y = y, x # swap for next step
    return x
```

In [29]:
```
c1 = %timeit -o y = blur_cython(x, steps)
t_c1 = c1.best
times.append(t_c1)
labels.append("cython (no hints)")
```

```
1 loop, best of 3: 649 ms per loop
```

In [30]:
```
plot_times()
```

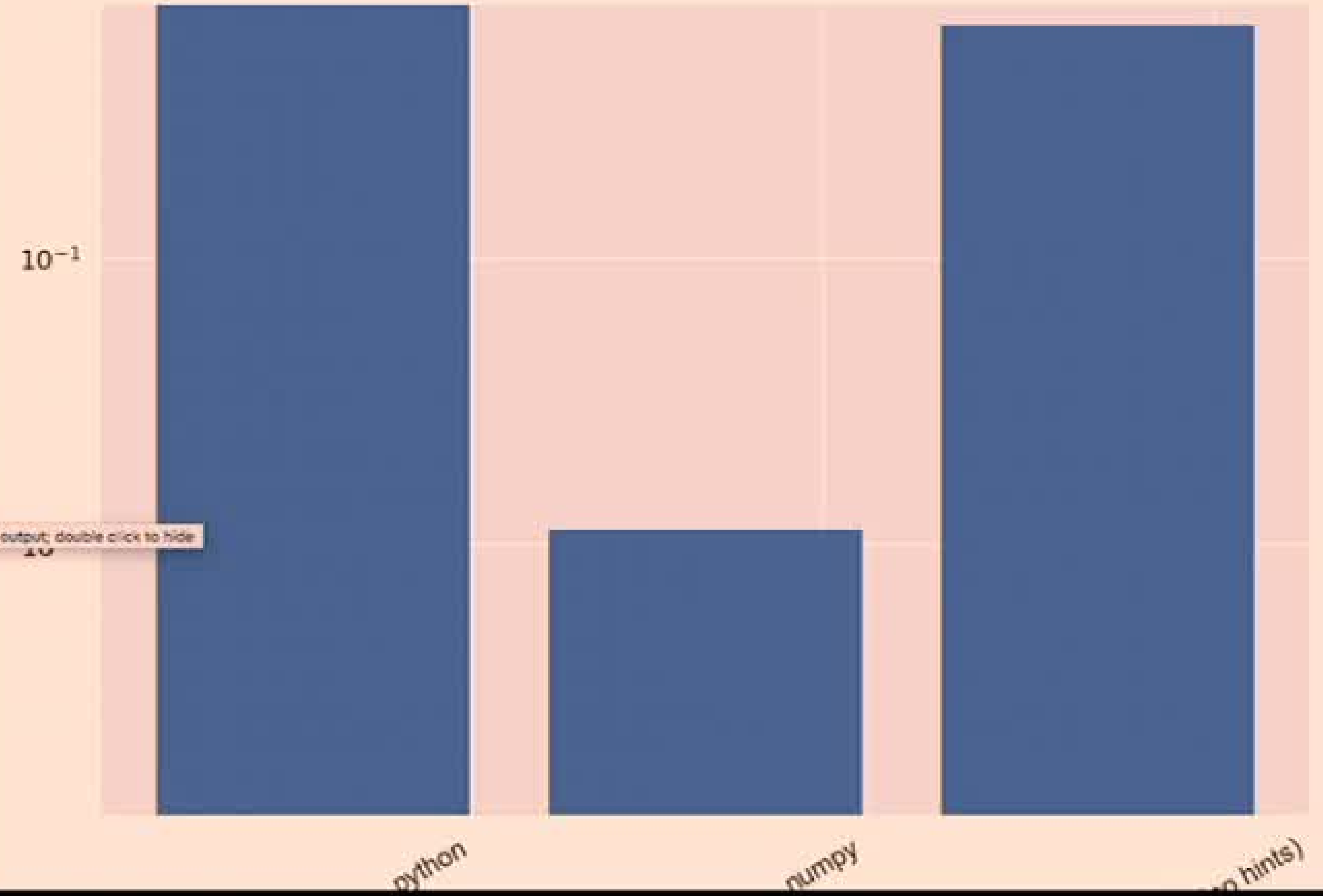


File Edit View Insert Cell Kernel Help Trusted Python 3

```
t_c1 = c1.best
times.append(t_c1)
labels.append("cython (no hints)")
```

1 loop, best of 3: 649 ms per loop

In [30]:
```
plot_times()
```



File Edit View Insert Cell Kernel Help Trusted Python 3

```
    y = np.empty_like(x)
    y[0] = x[0]
    y[-1] = x[-1]
    cdef int i, N = len(x)
    for _ in range(steps):
        for i in range(1, N-1):
            y[i] = .25 * ( x[i-1] + 2 * x[i] + x[i+1] )
        x, y = y, x # swap for next step
    return x
```

In [*]:

```
c2 = %timeit -o blur_cython2(x, steps)
t_c2 = c2.best
times.append(t_c2)
labels.append("cython (loops)")
plot_times()
```

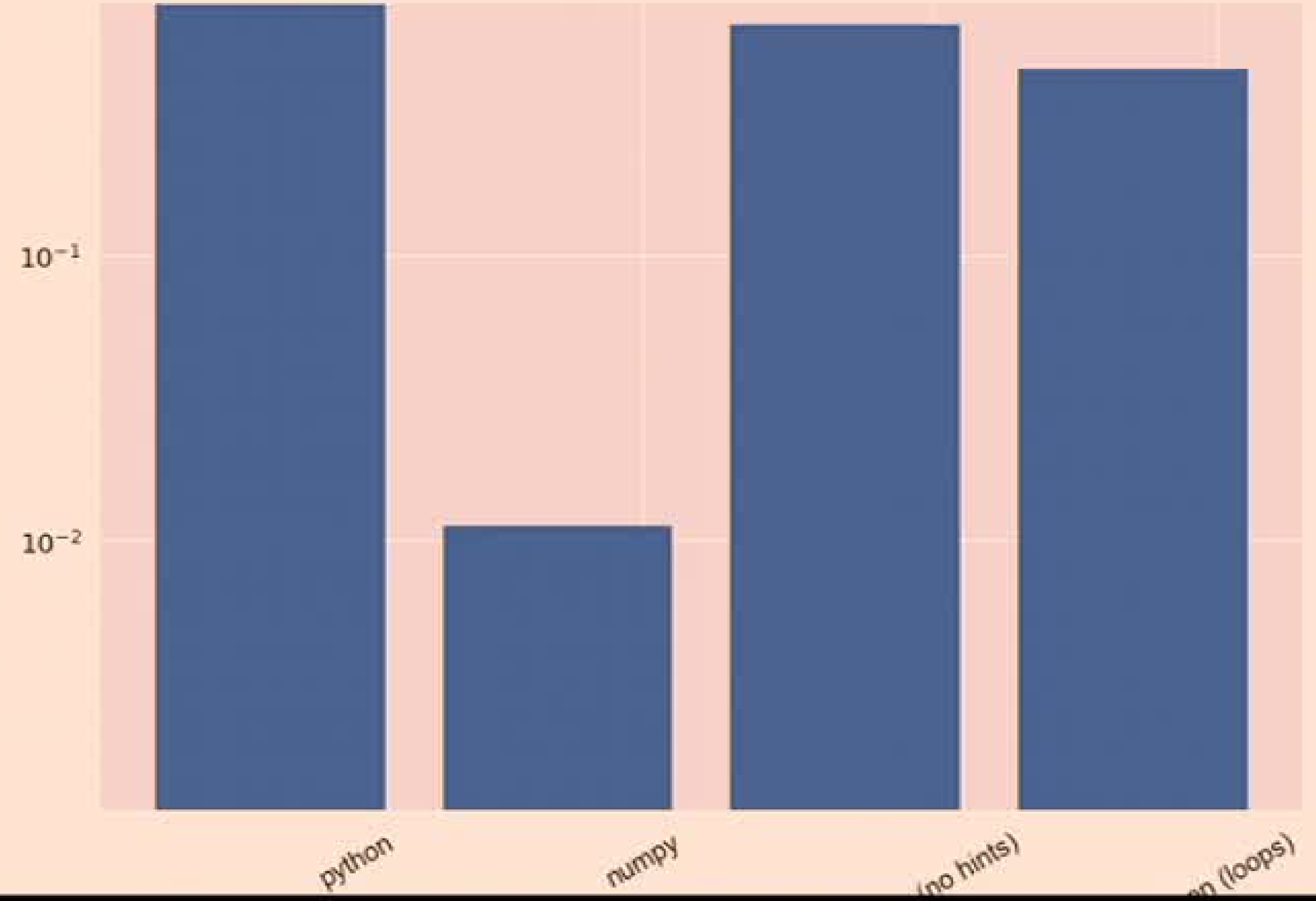
Just by making sure the iteration variables are defined as integers, we can save about 25% of the time.

The biggest key to optimizing with Cython is getting that yellow out of your loops. The more deeply nested a bit of code is within a loop, the more often it is called, and the more value you can get out of making it fast. In Cython, fast means avoiding Python (getting rid of yellow). To get rid of Python calls, we need to tell Python about the numpy arrays **x** and **y**:

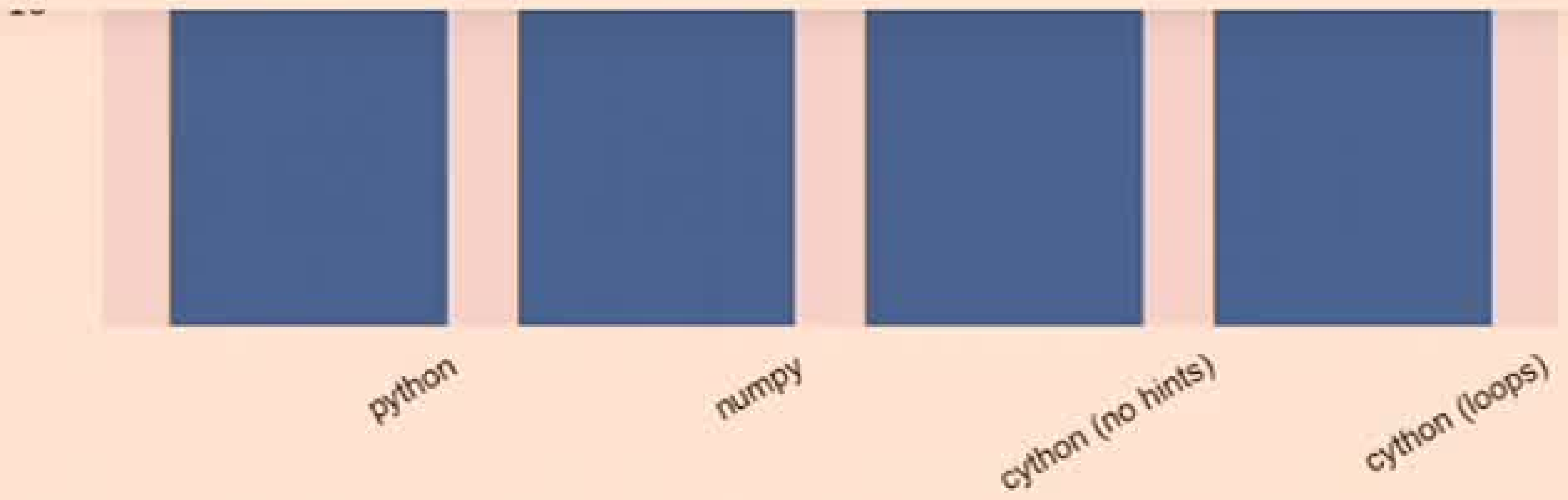
File Edit View Insert Cell Kernel Help Trusted Python 3

In [32]:
```python
c2 = %timeit -o blur_cython2(x, steps)
t_c2 = c2.best
times.append(t_c2)
labels.append("cython (loops)")
plot_times()
```

```
1 loop, best of 3: 453 ms per loop
```



File Edit View Insert Cell Kernel Help Trusted Python 3



Just by making sure the iteration variables are defined as integers, we can save about 25% of the time.

The biggest key to optimizing with Cython is getting that yellow out of your loops. The more deeply nested a bit of code is within a loop, the more often it is called, and the more value you can get out of making it fast. In Cython, fast means avoiding Python (getting rid of yellow). To get rid of Python calls, we need to tell Python about the numpy arrays **x** and **y**:

In [ ]:

```
%%cython -a

import numpy as np
cimport numpy as np

def blur_cython_typed(np.ndarray[double, ndim=1] x_, int steps=1
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
    x = 1 * x_
    y = np.empty_like(x_)
    y[0] = x[0]
    y[-1] = x[-1]
    for _ in range(steps):
        for i in range(1, N-1):
            y[i] = .25 * ( y[i-1] + 2 * y[i] + y[i+1] )
        x, y = y, x # swap for next step
    return x
```

In [*]:
```
ct = %timeit -o y = blur_cython_typed(x, steps)
t_ct = ct.best

times.append(t_ct)
labels.append("cython (types)")
plot_times()
```

```
100 loops, best of 3: 7.27 ms per loop
```

We can furter optimize with Cython macros, which disable bounds checking and negative indexing, and avoiding the Python variable swaping by using indices into a single array:

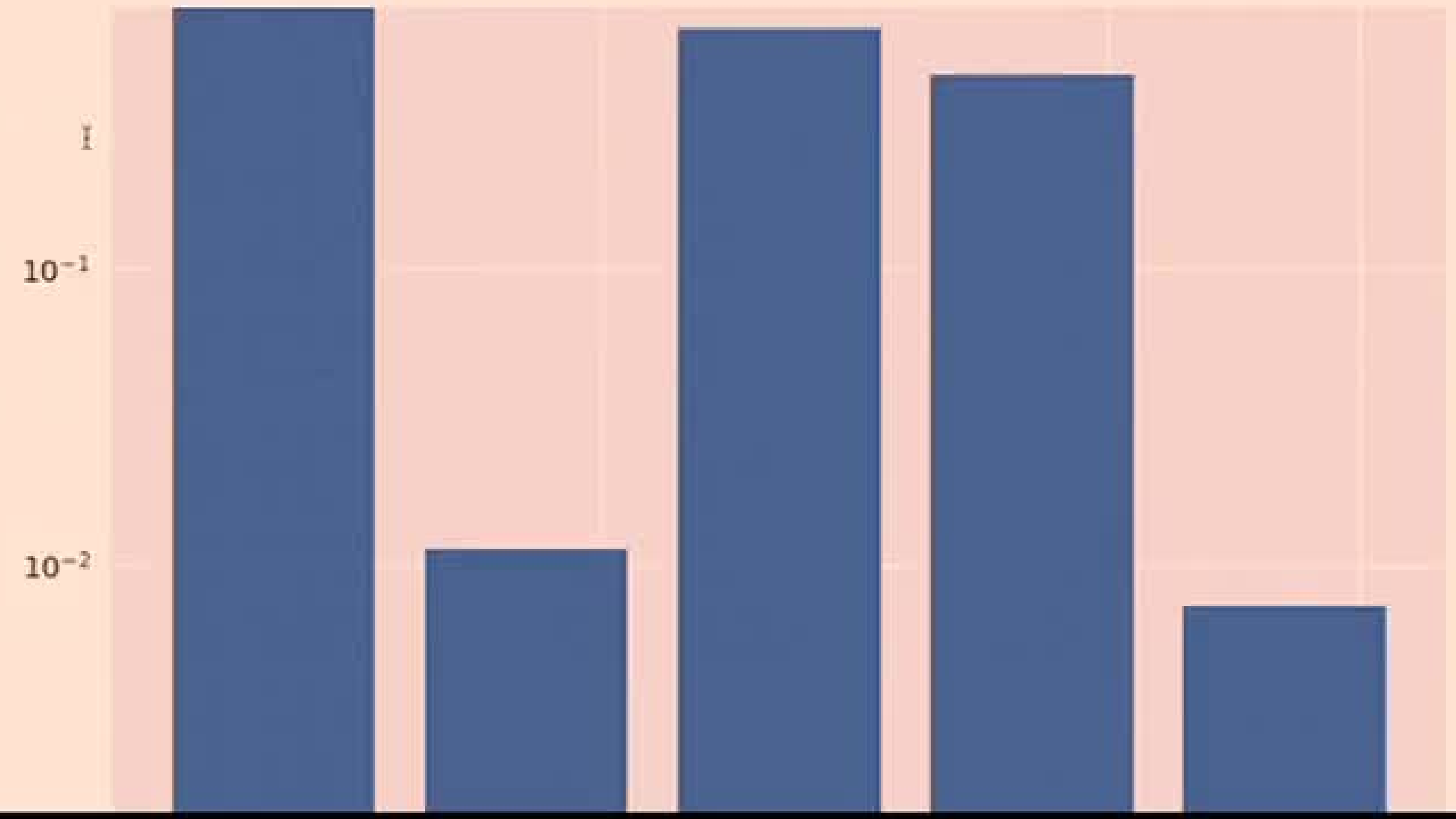
In [ ]:
```
%%cython -a
```

File Edit View Insert Cell Kernel Help Trusted Python 3

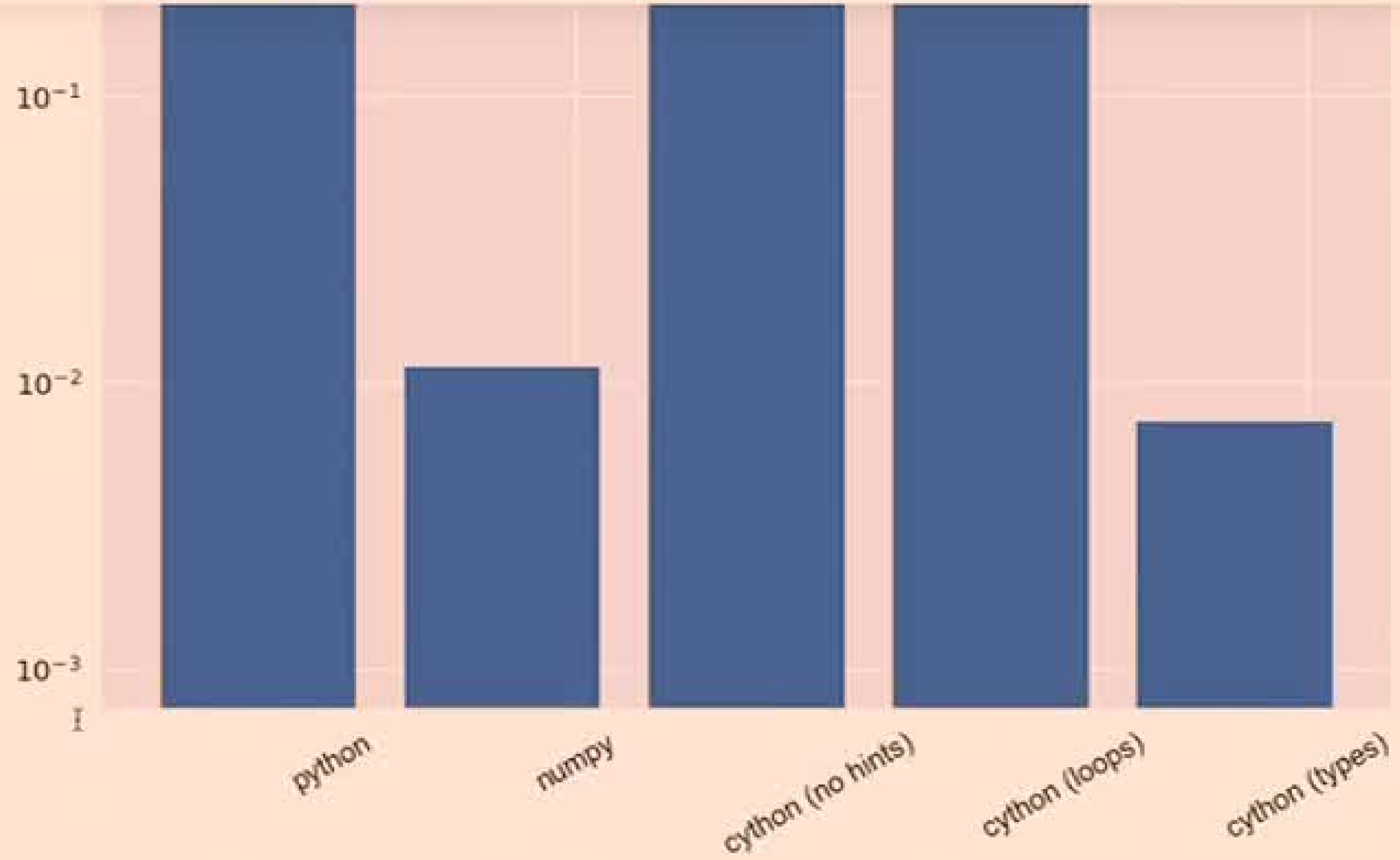
```
+17:         x, y = y, x # swap for next step
+18:     return x
```

In [34]:
```
ct = %timeit -o y = blur_cython_typed(x, steps)
t_ct = ct.best

times.append(t_ct)
labels.append("cython (types)")
plot_times()
```

```
100 loops, best of 3: 7.27 ms per loop
```

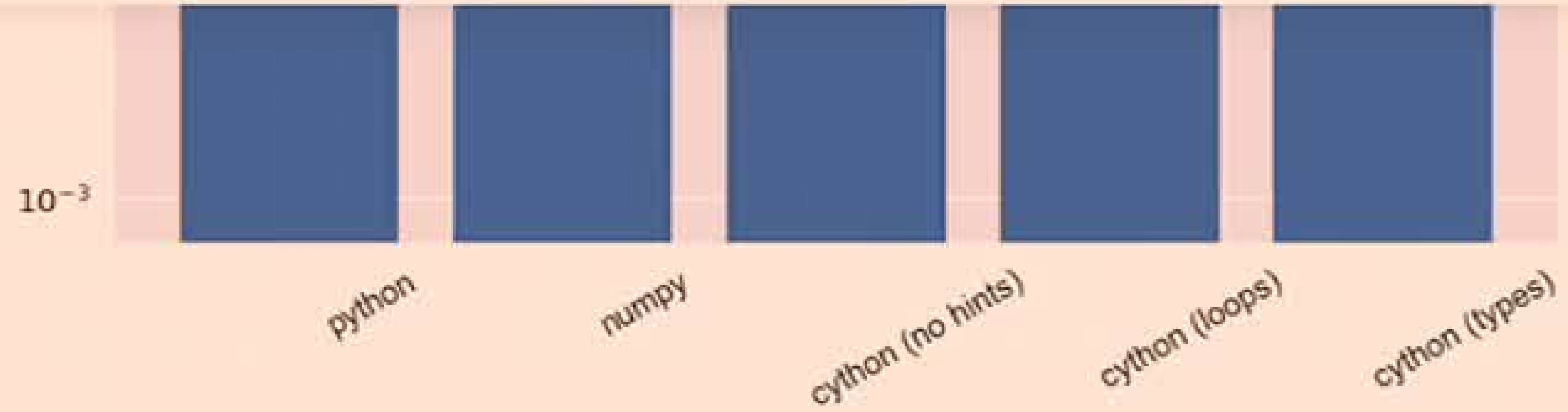


File Edit View Insert Cell Kernel Help Trusted Python 3



We can furter optimize with Cython macros, which disable bounds checking and negative indexing, and avoiding the Python variable swaping by using indices into a single array:

In [ ]:
```
%%cython -a
#cython: boundscheck=False
#cython: wraparound=False
```



We can furter optimize with Cython macros, which disable bounds checking and negative indexing, and avoiding the Python variable swaping by using indices into a single array:

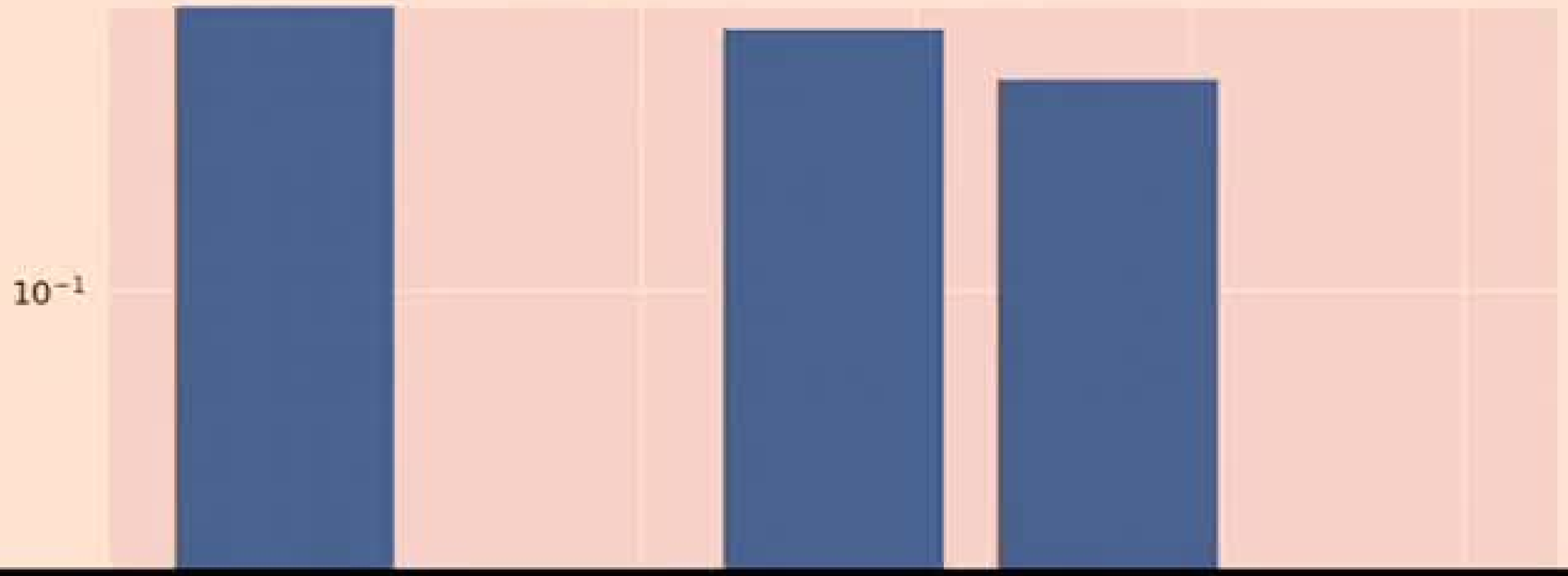
In [ ]:
```
%%cython -a
#cython: boundscheck=False
#cython: wraparound=False

import numpy as np
cimport numpy as np

def blur_cython_optimized(np.ndarray[double, ndim=1] x, int step
    cdef size_t N = x.shape[0]
    cdef np.ndarray[double, ndim=2] y
    y = np.empty((2, N), dtype=np.float64)
    y[0,:] = x
    y[1,0] = x[0]
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
    y[-1] = x[-1]
    for _ in range(steps):
        for i in range(1, N-1):
            y[i] = .25 * ( y[i-1] + 2 * y[i] + y[i+1] )
        x, y = y, x # swap for next step
    return x
```

In [34]:
```
ct = %timeit -o y = blur_cython_typed(x, steps)
t_ct = ct.best

times.append(t_ct)
labels.append("cython (types)")
plot_times()
```

```
100 loops, best of 3: 7.27 ms per loop
```

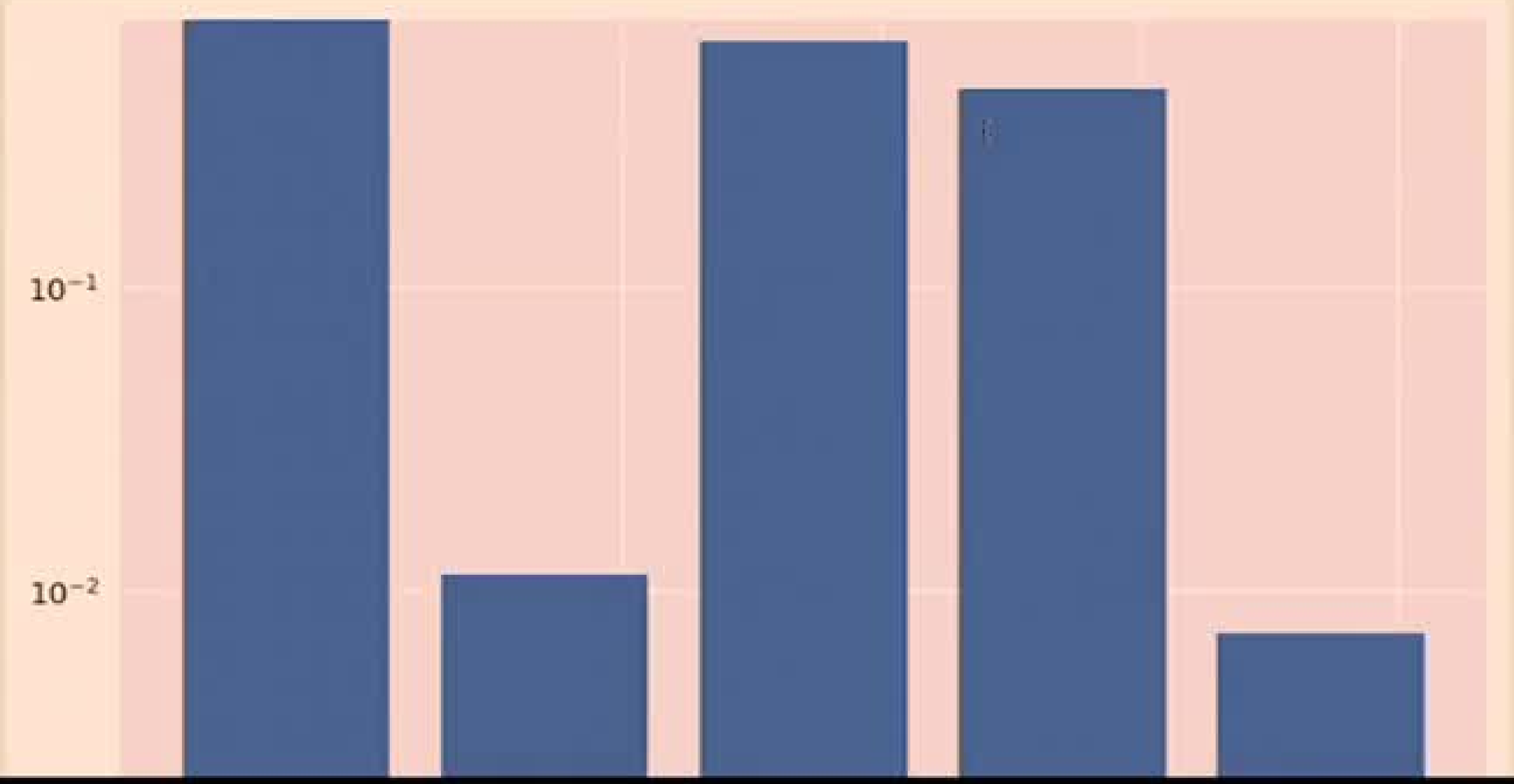


File Edit View Insert Cell Kernel Help Trusted Python 3

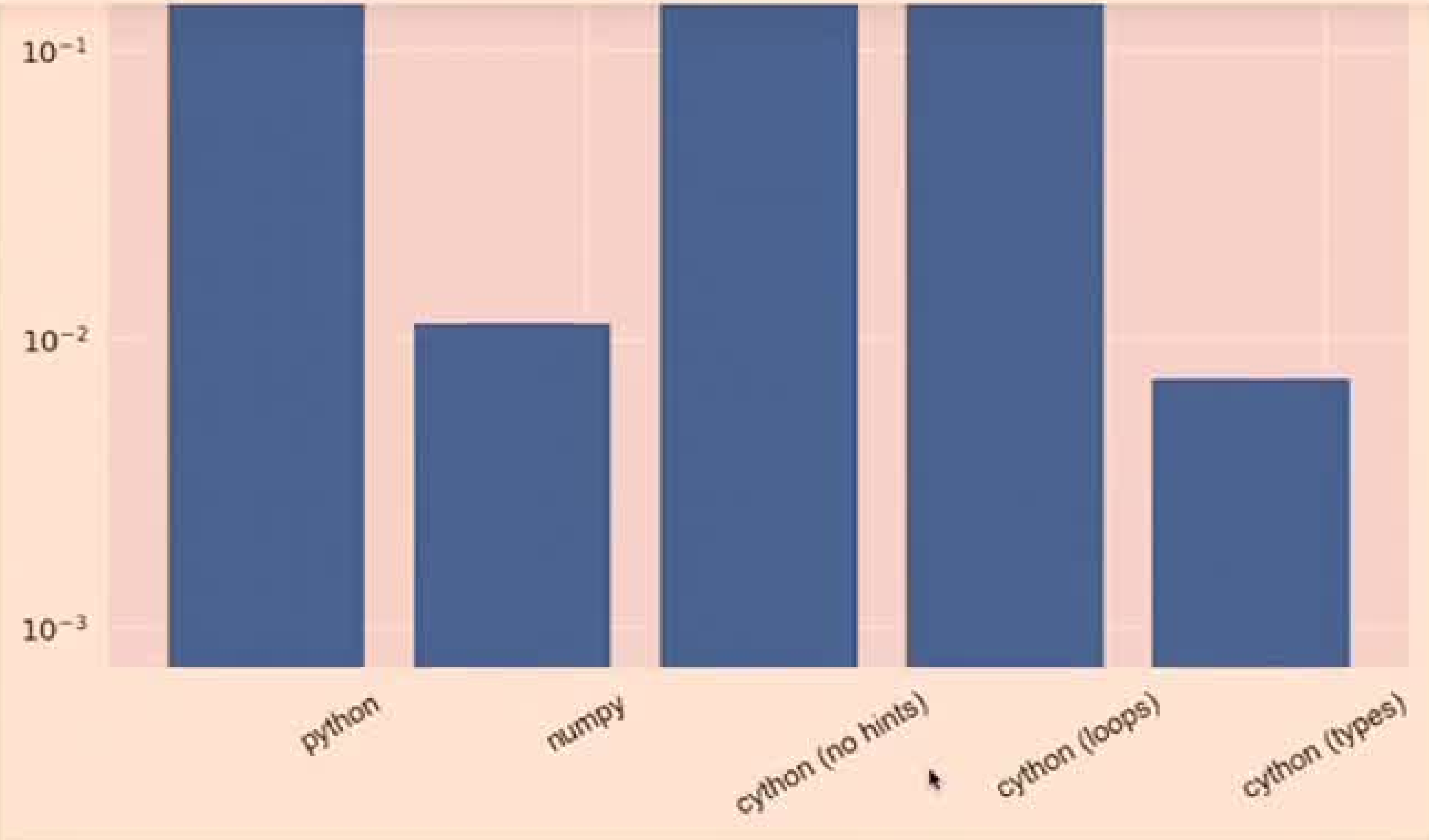
```
+17:         x, y = y, x # swap for next step
+18:     return x
```

In [34]:
```
ct = %timeit -o y = blur_cython_typed(x, steps)
t_ct = ct.best

times.append(t_ct)
labels.append("cython (types)")
plot_times()
```

```
100 loops, best of 3: 7.27 ms per loop
```



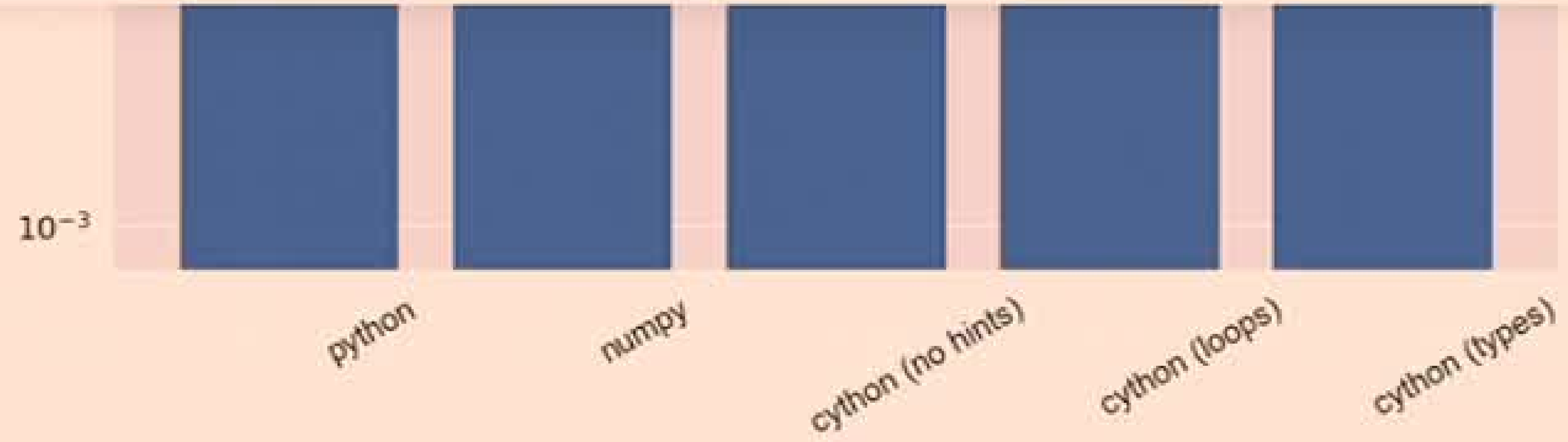
File Edit View Insert Cell Kernel Help Trusted Python 3



We can furter optimize with Cython macros, which disable bounds checking and negative indexing, and avoiding the Python variable swaping by using indices into a single array:

In [ ]:
```
cython -a
ython: boundscheck=False
ython: wraparound=False
```

File Edit View Insert Cell Kernel Help Trusted Python 3



We can furter optimize with Cython macros, which disable bounds checking and negative indexing, and avoiding the Python variable swaping by using indices into a single array:

In [ ]:

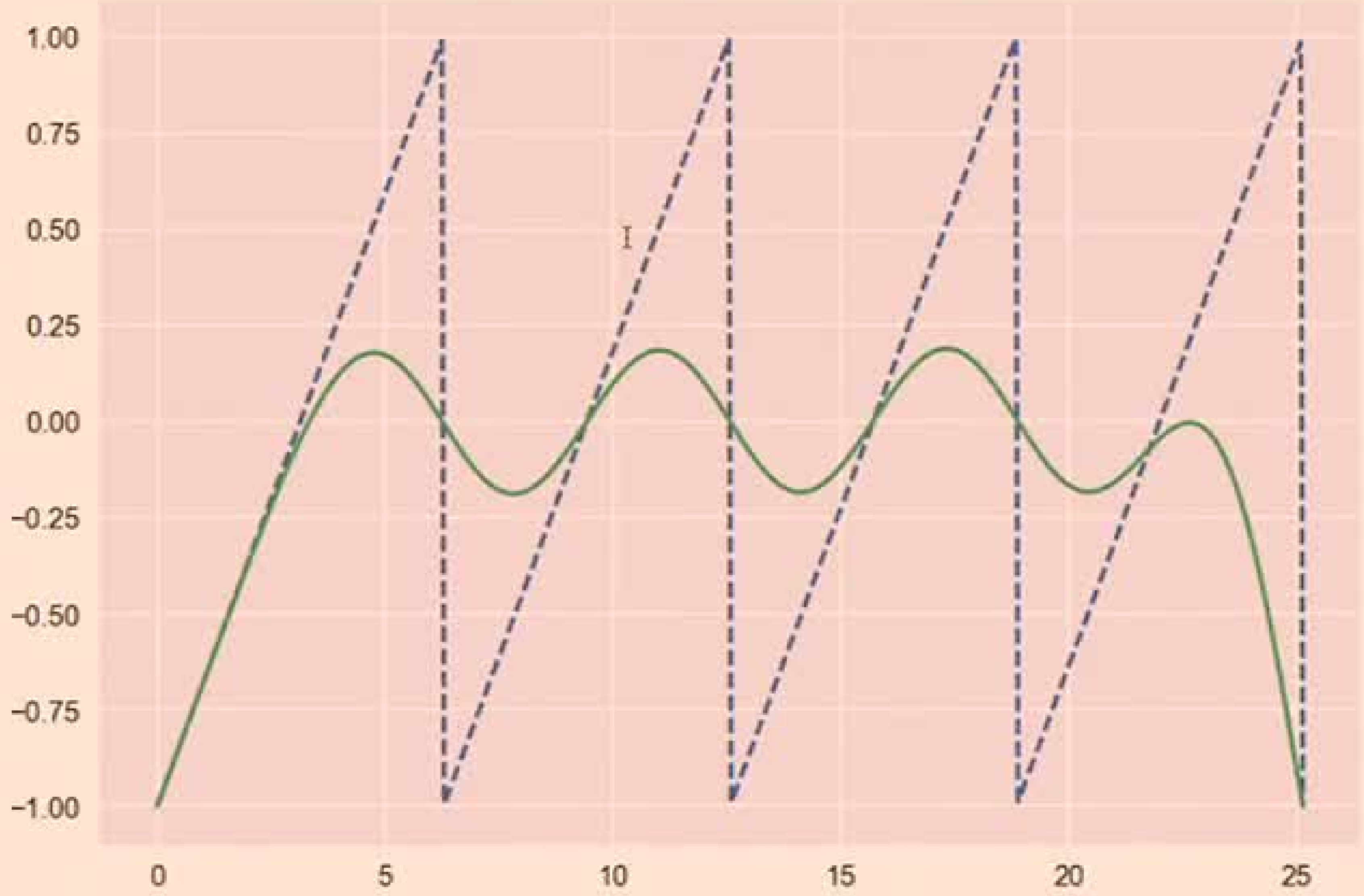
```
%%cython -a
#cython: boundscheck=False
#cython: wraparound=False

import numpy as np
cimport numpy as np

def blur_cython_optimized(np.ndarray[double, ndim=1] x, int step
    cdef size_t N = x.shape[0]
    cdef np.ndarray[double, ndim=2] y
    y = np.empty((2, N), dtype=np.float64)
    y[0,:] = x
    y[1,0] = x[0]
```

File Edit View Insert Cell Kernel Help

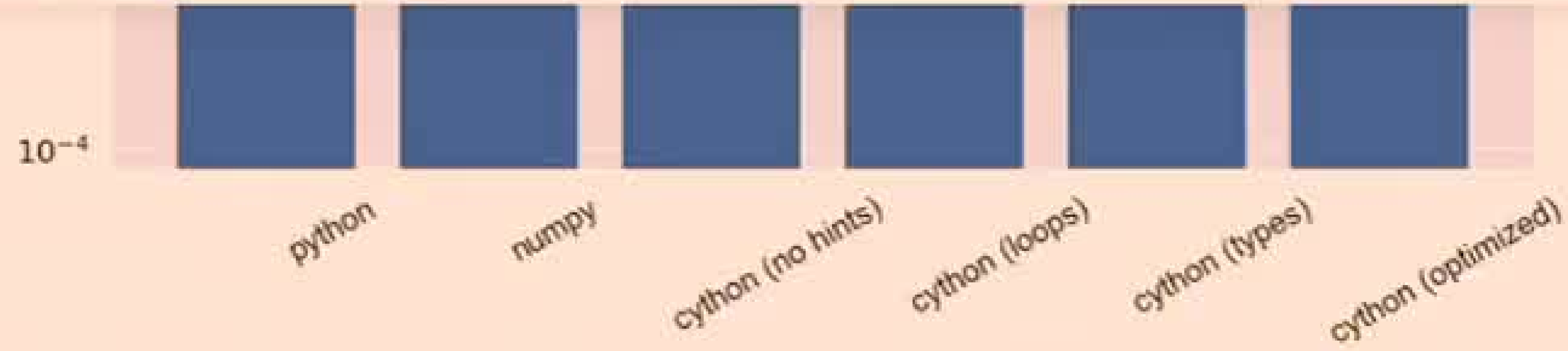
Trusted | Python 3

In [37]:
```python
y = blur_cython_optimized(x, steps)
plt.plot(t, x, '--')
plt.plot(t, y)
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x11ff6b358>]
```

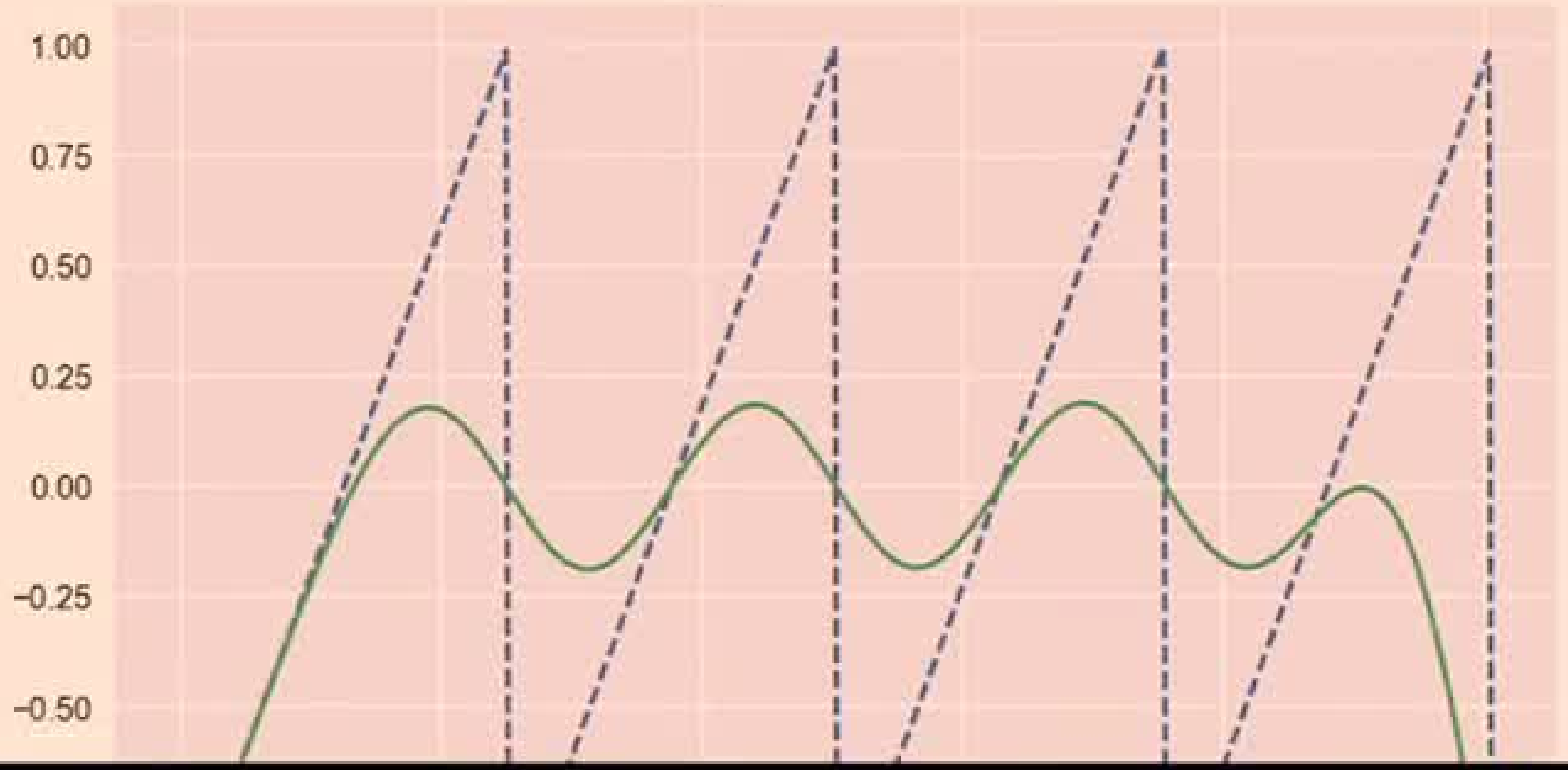


File Edit View Insert Cell Kernel Help Trusted Python 3



In [37]:
```
y = blur_cython_optimized(x, steps)
plt.plot(t, x, '--')
plt.plot(t, y)
```

Out[37]: [<matplotlib.lines.Line2D at 0x11ff6b358>]



File Edit View Insert Cell Kernel Help Trusted Python 3

Generated by Cython 0.25.2

Yellow lines hint at Python interaction.
Click on a line that starts with a "+" to see the C code that Cython generated for it.

```
+01: #cython: boundscheck=False
 02: #cython: wraparound=False
 03:
+04: import numpy as np
 05: cimport numpy as np
 06:
+07: def blur_cython_optimized(np.ndarray[double, ndim=1] x, in
t steps=1024):
+08:     cdef size_t N = x.shape[0]
 09:     cdef np.ndarray[double, ndim=2] y
+10:     y = np.empty((2, N), dtype=np.float64)
+11:     y[0,:] = x
+12:     y[1,0] = x[0]
+13:     y[1,N-1] = x[N-1]
 14:
+15:     cdef size_t _, i, j=0, k=1
+16:     for _ in range(steps):
+17:         j = _ % 2
+18:         k = 1 - j
+19:         for i in range(1, N-1):
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
    y[0,:] = x
    y[1,0] = x[0]
    y[1,N-1] = x[N-1]

    cdef size_t _, i, j=0, k=1
    for _ in range(steps):
        j = _ % 2
        k = 1 - j
        for i in range(1, N-1):
            y[k,i] = .25 * ( y[j,i-1] + 2 * y[j,i] + y[j,i+1] )
    return y[k]
```

Note how there is now zero yellow called in any of the loops, only in the initial copy of the input array.

In [36]:

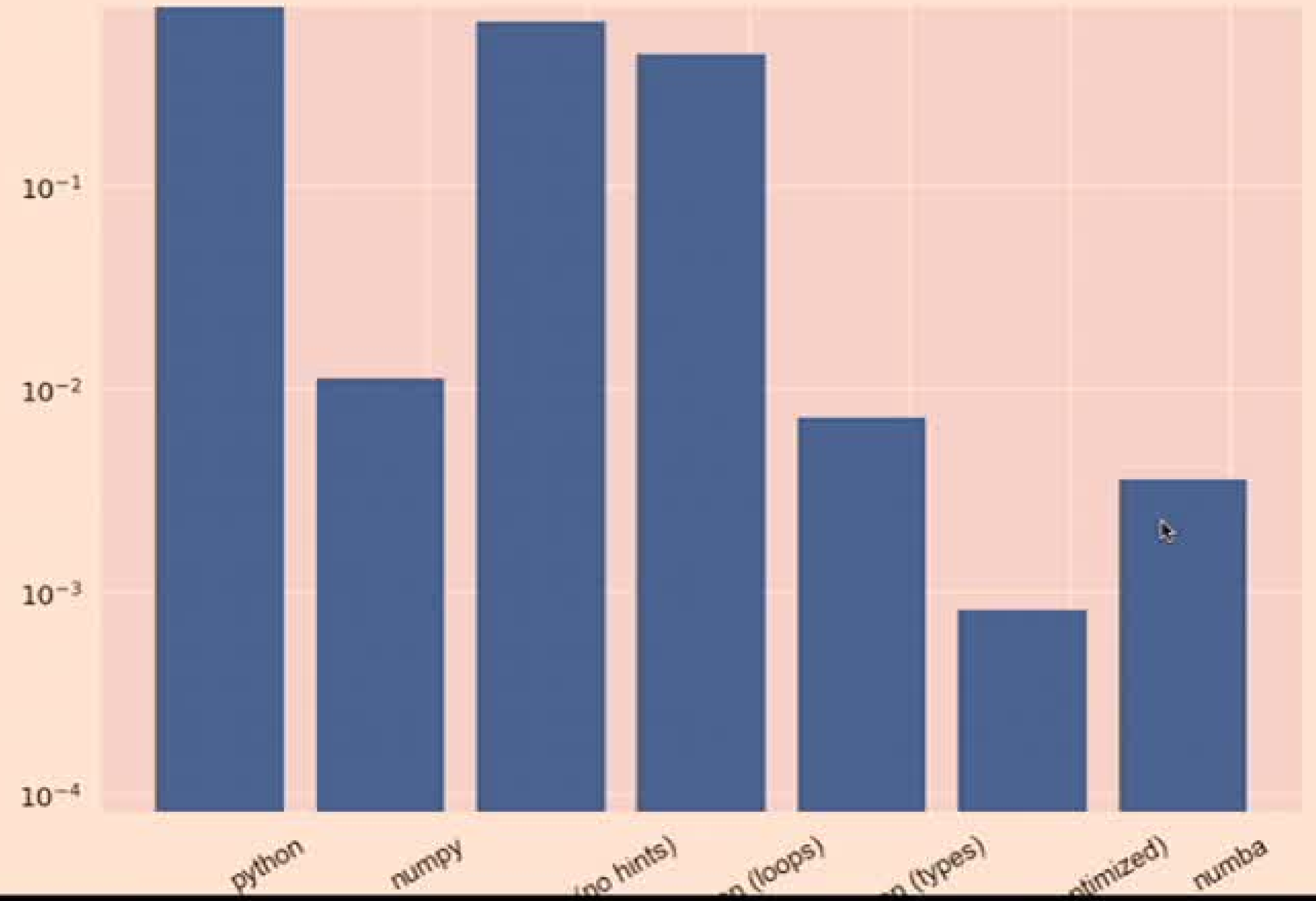
```
copt = %timeit -o y = blur_cython_optimized(x, steps)
t_copt = copt.best
times.append(t_copt)
labels.append("cython (optimized)")
plot_times()
```

```
1000 loops, best of 3: 820 µs per loop
```

File Edit View Insert Cell Kernel Help Trusted Python 3

In [39]:
```
nb = %timeit -o blur_numba(x, steps)
t_nb = nb.best
times.append(t_nb)
labels.append("numba")
plot_times()
```

```
100 loops, best of 3: 3.62 ms per loop
```



File Edit View Insert Cell Kernel Help Trusted Python 3

```
import os
import glob
list(os.walk('/tmp'))
```

```
Overwriting profileme.py
```

In [41]:
```
!python -m cProfile profileme.py
```

```
         931 function calls (906 primitive calls) in 0.004 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:197(_call_with_frames_removed)
       34    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:208(_verbose_message)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:293(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:297(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:304(__exit__)
        8    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:307(<genexpr>)
        2    0.000    0.000    0.000    0.000 <frozen importlib
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:197(_call_with_frames_removed)
       34    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:208(_verbose_message)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:293(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:297(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:304(__exit__)
        8    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:307(<genexpr>)
        2    0.000    0.000    0.000    0.000 <frozen importlib
```

File Edit View Insert Cell Kernel Help Trusted Python 3

In [41]:
```
!python -m cProfile profileme.py
```

```
         931 function calls (906 primitive calls) in 0.004 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:197(_call_with_frames_removed)
       34    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:208(_verbose_message)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:293(__init__)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

In [41]:
```
!python -m cProfile profileme.py
```

```
         931 function calls (906 primitive calls) in 0.004 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:197(_call_with_frames_removed)
       34    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:208(_verbose_message)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:293(__init__)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

In [40]:
```
%%writefile profileme.py
import os
import glob
list(os.walk('/tmp'))
```

```
Overwriting profileme.py
```

In [41]:
```
!python -m cProfile profileme.py
```

```
         931 function calls (906 primitive calls) in 0.004 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

In [40]:
```
%%writefile profileme.py
import os
import glob
list(os.walk('/tmp'))
```

```
Overwriting profileme.py
```

In [41]:
```
!python -m cProfile profileme.py
```

```
         931 function calls (906 primitive calls) in 0.004 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:102(release)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:142(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:146(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:153(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:159(_get_module_lock)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:173(cb)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:197(_call_with_frames_removed)
       34    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:208(_verbose_message)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:293(__init__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:297(__enter__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:304(__exit__)
        8    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:307(<genexpr>)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:35(_new_module)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:355(__init__)
        4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:389(cached)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:402(parent)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:410(has_location)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:493(_init_module_attrs)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:553(module_from_spec)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:355(__init__)
        4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:389(cached)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:402(parent)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:410(has_location)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:493(_init_module_attrs)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:553(module_from_spec)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:57(__init__)
      2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:641(_load_unlocked)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:698(find_spec)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:77(acquire)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:771(find_spec)
        6    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:834(__enter__)
        6    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:838(__exit__)
        2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap>:861(_find_spec)
```

File Edit View Insert Cell Kernel Help Notebook saved Trusted Python 3

```
        2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:931(_find_and_load_unlocked)
        2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap>:958(_find_and_load)
          8    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:1080(_path_importer_cache)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:1117(_get_spec)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:1149(find_spec)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:1228(_get_spec)
          6    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:1233(find_spec)
          4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:263(cache_from_source)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:361(_get_cached)
          6    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:37(_relax_case)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:393(_check_name_wrapper)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:430(_validate_bytecode_header)
          2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:485(_compile_bytecode)
          4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:52(_r_long)
```

File Edit View Insert Cell Kernel Help Notebook saved Trusted Python 3

```
._bootstrap_external>:1233(find_spec)
         4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:263(cache_from_source)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:361(_get_cached)
         6    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:37(_relax_case)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:393(_check_name_wrapper)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:430(_validate_bytecode_header)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:485(_compile_bytecode)
         4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:52(_r_long)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:524(spec_from_file_location)
        32    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:57(_path_join)
        32    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:59(<listcomp>)
         4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:63(_path_split)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:669(create_module)
       2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap_external>:672(exec_module)
         2    0.000    0.000    0.000    0.000 <frozen importlib
```

File Edit View Insert Cell Kernel Help Notebook saved Trusted Python 3

```
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:393(_check_name_wrapper)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:430(_validate_bytecode_header)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:485(_compile_bytecode)
         4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:52(_r_long)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:524(spec_from_file_location)
        32    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:57(_path_join)
        32    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:59(<listcomp>)
         4    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:63(_path_split)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:669(create_module)
       2/1    0.000    0.000    0.001    0.001 <frozen importlib
._bootstrap_external>:672(exec_module)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:743(get_code)
        10    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:75(_path_stat)
         2    0.000    0.000    0.000    0.000 <frozen importlib
._bootstrap_external>:800(__init__)
         2    0.000    0.000    0.000    0.000 <frozen importlib
```

```
         8    0.000    0.000    0.000    0.000 enum.py:265(__cal
l__)
         8    0.000    0.000    0.000    0.000 enum.py:515(__new
__)
         4    0.000    0.000    0.000    0.000 enum.py:797(__and
__)
         1    0.000    0.000    0.000    0.000 fnmatch.py:11(<mo
dule>)
         1    0.000    0.000    0.000    0.000 functools.py:44(u
pdate_wrapper)
         1    0.000    0.000    0.000    0.000 functools.py:449(
lru_cache)
         1    0.000    0.000    0.000    0.000 functools.py:480(
decorating_function)
         1    0.000    0.000    0.001    0.001 glob.py:1(<module
>)
      14/6    0.000    0.000    0.003    0.000 os.py:277(walk)
         4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
         4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
         4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
         1    0.000    0.000    0.004    0.004 profileme.py:1(<m
odule>)
         2    0.000    0.000    0.000    0.000 re.py:231(compile
)
         2    0.000    0.000    0.000    0.000 re.py:286(_compil
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        2    0.000    0.000    0.000    0.000 sre_parse.py:223(
__init__)
       16    0.000    0.000    0.000    0.000 sre_parse.py:232(
__next)
       16    0.000    0.000    0.000    0.000 sre_parse.py:248(
match)
       12    0.000    0.000    0.000    0.000 sre_parse.py:253(
get)
        8    0.000    0.000    0.000    0.000 sre_parse.py:285(
tell)
      4/2    0.000    0.000    0.000    0.000 sre_parse.py:407(
_parse_sub)
      4/2    0.000    0.000    0.000    0.000 sre_parse.py:469(
_parse)
        2    0.000    0.000    0.000    0.000 sre_parse.py:76(_
_init__)
        8    0.000    0.000    0.000    0.000 sre_parse.py:81(g
roups)
        2    0.000    0.000    0.000    0.000 sre_parse.py:829(
fix_flags)
        2    0.000    0.000    0.000    0.000 sre_parse.py:84(o
pengroup)
        2    0.000    0.000    0.000    0.000 sre_parse.py:845(
parse)
        2    0.000    0.000    0.000    0.000 sre_parse.py:96(c
losegroup)
        2    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
e' of 'bytearray' objects}
         1    0.000    0.000    0.000    0.000 {method 'update'
of 'dict' objects}
```

In [ ]:
```
import os
import cProfile
cProfile.run("list(os.walk('/tmp'))")
```

In [ ]:
```
%prun list(os.walk('/tmp'))
```

In [ ]:
```
%load_ext snakeviz
```

In [ ]:
```
%snakeviz list(os.walk('/usr/local'))
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
cProfile.run( 'list(os.walk('/tmp'))' )
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
        1    0.000    0.000    0.002    0.002 {built-in method
builtins.exec}
        4    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        9    0.000    0.000    0.000    0.000 {built-in method
posix.fspath}
        4    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
        1    0.000    0.000    0.002    0.002 {built-in method
builtins.exec}
        4    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        9    0.000    0.000    0.000    0.000 {built-in method
posix.fspath}
        4    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
        1    0.000    0.000    0.002    0.002 {built-in method
builtins.exec}
        4    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        9    0.000    0.000    0.000    0.000 {built-in method
posix.fspath}
        4    0.000    0.000    0.000    0.000 {built-in method
posix.lstat}
        5    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
posix.scandir}
        22    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
         1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
         4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
        22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

```
In [ ]: %prun list(os.walk('/tmp'))

In [ ]: %load_ext snakeviz

In [ ]: %snakeviz list(os.walk('/usr/local'))
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
         22    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
          1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
          4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
         22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
          4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

In [43]:
```
%prun list(os.walk('/tmp'))
```

```
          134 function calls (126 primitive calls) in 0.001 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.001    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
e' of 'bytearray' objects}
        1    0.000    0.000    0.000    0.000 {method 'update'
of 'dict' objects}
```

In [42]:

```python
import os
import cProfile
cProfile.run("list(os.walk('/tmp')")
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
e' of 'bytearray' objects}
         1    0.000    0.000    0.000    0.000 {method 'update'
of 'dict' objects}
```

In [42]:

```
import os
import cProfile
cProfile.run("list(os.walk('/tmp')")
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        22    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
         1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
         4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
        22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

In [43]:

```
%prun list(os.walk('/tmp'))
```

```
         134 function calls (126 primitive calls) in 0.001 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.001    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 {built-in method
```

File Edit View Insert Cell Kernel Help Trusted | Python 3

```
posix.scandir}
       22    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
        4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
       22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
        4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

In [44]:
```
%prun list(os.walk('/tmp'))
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

In [44]:

```
%prun list(os.walk('/tmp'))
```

```
          134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 {built-in method
posix.lstat}
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        1    0.000    0.000    0.002    0.002 {built-in method
builtins.exec}
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
       22    0.000    0.000    0.000    0.000 {method 'is_dir'
```

File Edit View Insert Cell Kernel Help Notebook saved Trusted Python 3

```
      4   0.000   0.000   0.000   0.000 {method 'startswi
th' of 'str' objects}
```

In [44]:

```
%prun list(os.walk('/tmp'))
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 {built-in method
posix.lstat}
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
        1    0.000    0.000    0.002    0.002 {built-in method
builtins.exec}
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
       22    0.000    0.000    0.000    0.000 {method 'is_dir'
```

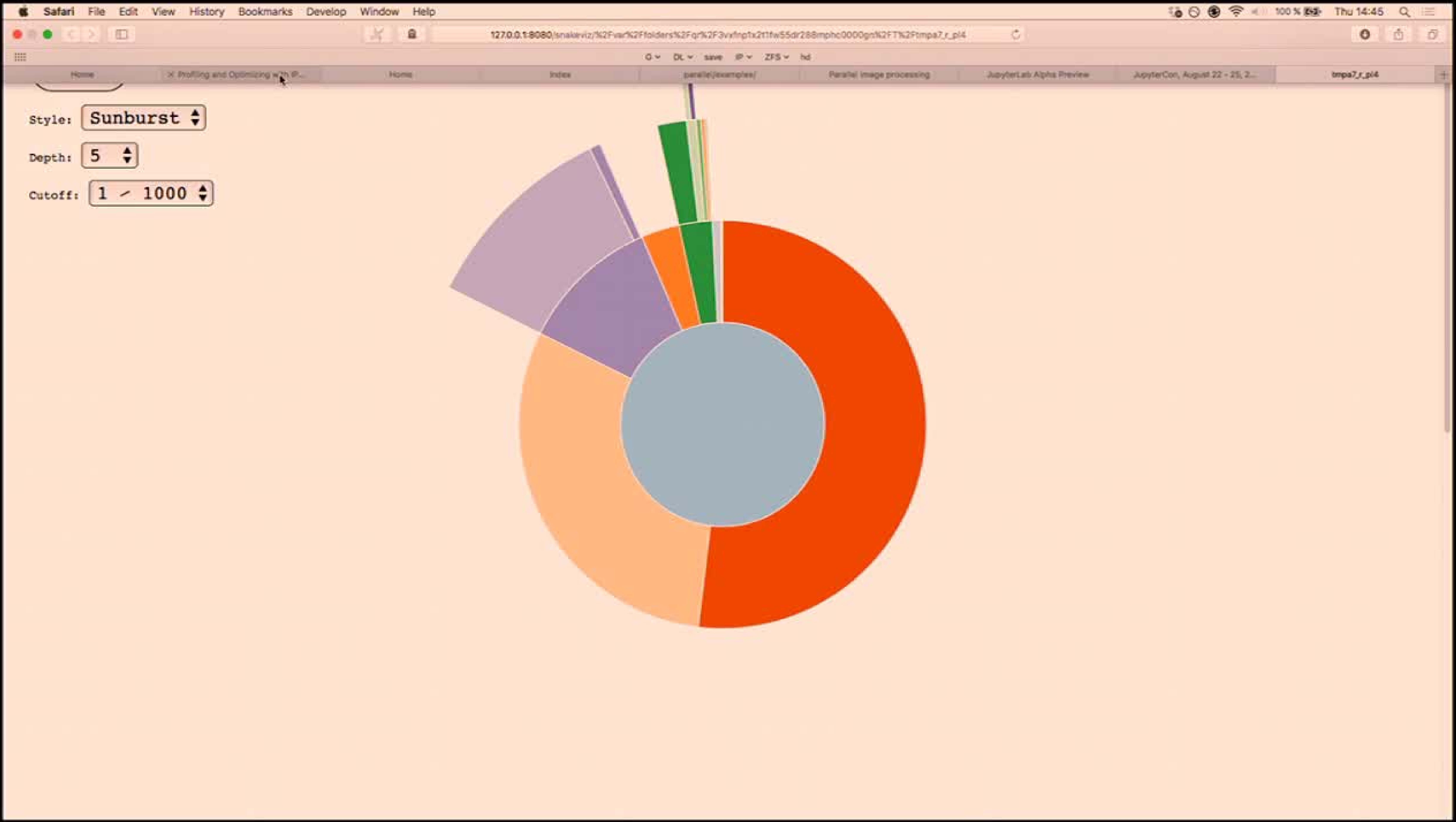
File Edit View Insert Cell Kernel Help Trusted Python 3

```
of 'list' objects}
        4    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
        9    0.000    0.000    0.000    0.000 {built-in method
posix.fspath}
        4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
        4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
        1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
```

```
In [45]: %load_ext snakeviz
```

```
In [*]: %snakeviz list(os.walk('/usr/local'))
```

```
In [ ]:
```


Safari File Edit View History Bookmarks Develop Window Help
Thu 14:45
Home
Profiling and Optimizing with IP...
Index
parallel/examples/
Parallel image processing
JupyterLab Alpha Preview
tmpa7_r_pl4
Style: Sunburst
Depth: 5
Cutoff: 1 / 1000

```
of 'list' objects}
          4    0.000    0.000    0.000    0.000 {built-in method
builtins.isinstance}
          9    0.000    0.000    0.000    0.000 {built-in method
posix.fspath}
          4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
          4    0.000    0.000    0.000    0.000 {built-in method
_stat.S_ISLNK}
          1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
```

In [45]:
```
%load_ext snakeviz
```

In [46]:
```
%snakeviz list(os.walk('/usr/local'))
```

```
*** Profile stats marshalled to file '/var/folders/qr/3vxfnp1x2
t1fw55dr288mphc0000gn/T/tmpa7_r_pl4'.
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
of '_lsprof.Profiler' objects}
        4    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
       22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
        4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
```

In [44]:
```
%prun list(os.walk('/tmp'))
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: internal time

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
       27    0.001    0.000    0.001    0.000 {built-in method
builtins.next}
        5    0.000    0.000    0.000    0.000 {built-in method
posix.scandir}
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 {built-in method
posix.lstat}
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
e' of 'bytearray' objects}
         1    0.000    0.000    0.000    0.000 {method 'update'
of 'dict' objects}
```

In [click to scroll output; double click to hide

```python
s
import cProfile
cProfile.run("list(os.walk('/tmp)")
```

```
          134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
        4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
        4    0.000    0.000    0.000    0.000 posixpath.py:73(j
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        18    0.000    0.000    0.000    0.000 {method 'rpartiti
on' of 'str' objects}
        68    0.000    0.000    0.000    0.000 {method 'rstrip'
of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
e' of 'bytearray' objects}
         1    0.000    0.000    0.000    0.000 {method 'update'
of 'dict' objects}
```

click to scroll output; double click to hide

In [42]:
```
import os
import cProfile
cProfile.run("list(os.walk('/tmp)")
```

```
         134 function calls (126 primitive calls) in 0.002 seco
nds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(f
unction)
        1    0.000    0.000    0.002    0.002 <string>:1(<modul
e>)
     14/6    0.000    0.000    0.002    0.000 os.py:277(walk)
        4    0.000    0.000    0.000    0.000 posixpath.py:166(
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        10    0.000    0.000    0.000    0.000 {built-in method
posix.stat}
        72    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
         1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
         6    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'extend'
of 'list' objects}
        10    0.000    0.000    0.000    0.000 {method 'find' of
'bytearray' objects}
        22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
         2    0.000    0.000    0.000    0.000 {method 'items' o
f 'dict' objects}
        36    0.000    0.000    0.000    0.000 {method 'join' of
'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'read' of
'_io.FileIO' objects}
        18    0.000    0.000    0.000    0.000 {method 'rpartiti
on' of 'str' objects}
        68    0.000    0.000    0.000    0.000 {method 'rstrip'
of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
e' of 'bytearray' objects}
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
        10    0.000    0.000    0.000    0.000 {built-in method
posix.stat}
        72    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
         1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
         6    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'extend'
of 'list' objects}
        10    0.000    0.000    0.000    0.000 {method 'find' of
'bytearray' objects}
        22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
         2    0.000    0.000    0.000    0.000 {method 'items' o
f 'dict' objects}
        36    0.000    0.000    0.000    0.000 {method 'join' of
'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'read' of
'_io.FileIO' objects}
        18    0.000    0.000    0.000    0.000 {method 'rpartiti
on' of 'str' objects}
        68    0.000    0.000    0.000    0.000 {method 'rstrip'
of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
posix.scandir}
        10    0.000    0.000    0.000    0.000 {built-in method
posix.stat}
        72    0.000    0.000    0.000    0.000 {method 'append'
of 'list' objects}
         1    0.000    0.000    0.000    0.000 {method 'disable'
of '_lsprof.Profiler' objects}
         6    0.000    0.000    0.000    0.000 {method 'endswith
' of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'extend'
of 'list' objects}
        10    0.000    0.000    0.000    0.000 {method 'find' of
'bytearray' objects}
        22    0.000    0.000    0.000    0.000 {method 'is_dir'
of 'posix.DirEntry' objects}
         2    0.000    0.000    0.000    0.000 {method 'items' o
f 'dict' objects}
        36    0.000    0.000    0.000    0.000 {method 'join' of
'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'read' of
'_io.FileIO' objects}
        18    0.000    0.000    0.000    0.000 {method 'rpartiti
on' of 'str' objects}
        68    0.000    0.000    0.000    0.000 {method 'rstrip'
of 'str' objects}
         4    0.000    0.000    0.000    0.000 {method 'startswi
th' of 'str' objects}
         2    0.000    0.000    0.000    0.000 {method 'translat
```

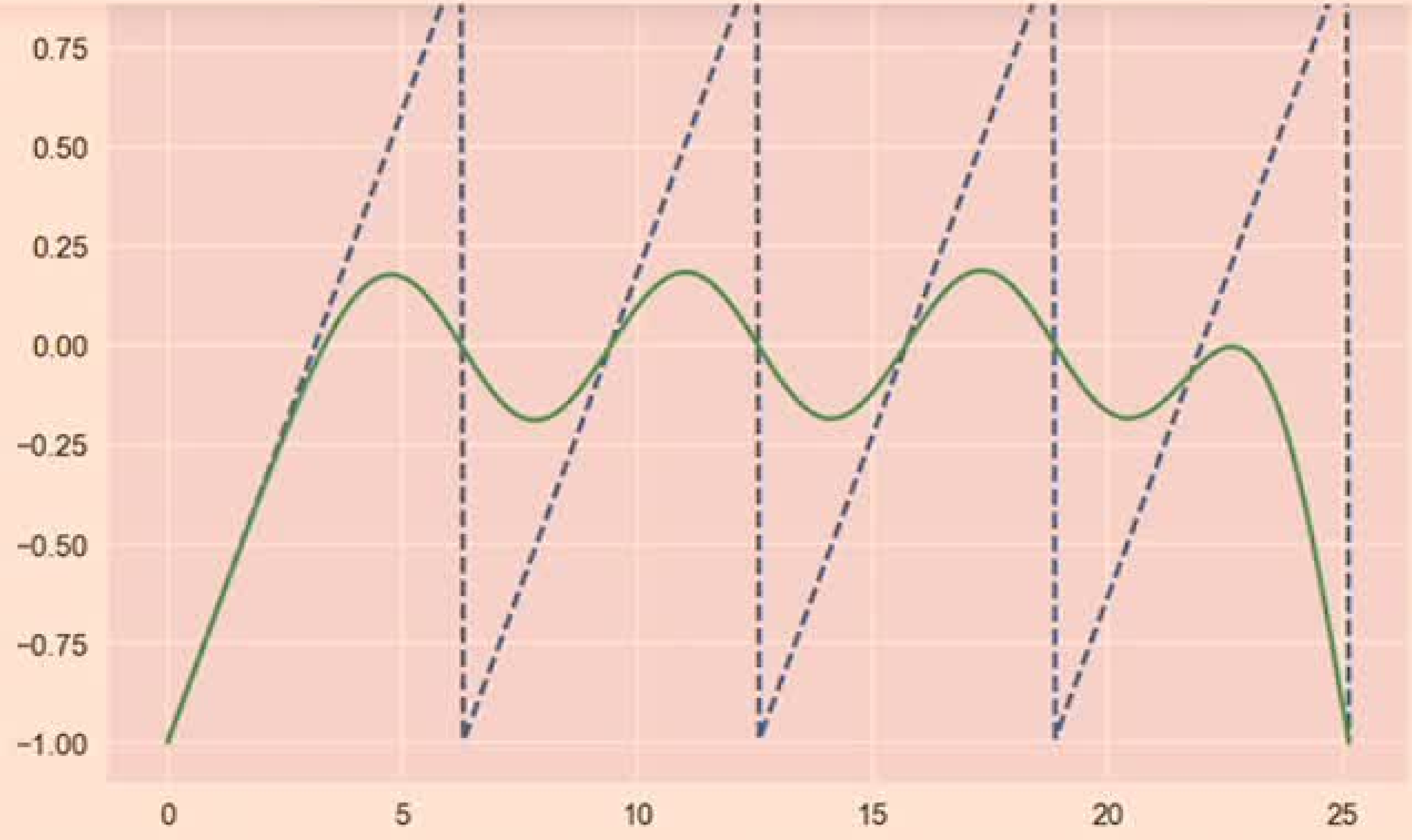
File Edit View Insert Cell Kernel Help   Trusted   Python 3

```
           4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
           4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
           1    0.000    0.000    0.004    0.004 profileme.py:1(<m
odule>)
           2    0.000    0.000    0.000    0.000 re.py:231(compile
)
           2    0.000    0.000    0.000    0.000 re.py:286(_compil
e)
           2    0.000    0.000    0.000    0.000 sre_compile.py:22
3(_compile_charset)
           2    0.000    0.000    0.000    0.000 sre_compile.py:25
0(_optimize_charset)
           2    0.000    0.000    0.000    0.000 sre_compile.py:37
6(_mk_bitmap)
           2    0.000    0.000    0.000    0.000 sre_compile.py:37
8(<listcomp>)
         4/2    0.000    0.000    0.000    0.000 sre_compile.py:41
4(_get_literal_prefix)
           2    0.000    0.000    0.000    0.000 sre_compile.py:44
1(_get_charset_prefix)
           2    0.000    0.000    0.000    0.000 sre_compile.py:48
2(_compile_info)
           4    0.000    0.000    0.000    0.000 sre_compile.py:53
9(isstring)
           2    0.000    0.000    0.000    0.000 sre_compile.py:54
```

File Edit View Insert Cell Kernel Help Trusted Python 3

```
pdate_wrapper)
           1    0.000    0.000    0.000    0.000 functools.py:449(
lru_cache)
           1    0.000    0.000    0.000    0.000 functools.py:480(
decorating_function)
           1    0.000    0.000    0.001    0.001 glob.py:1(<module
>)
        14/6    0.000    0.000    0.003    0.000 os.py:277(walk)
           4    0.000    0.000    0.000    0.000 posixpath.py:166(
islink)
           4    0.000    0.000    0.000    0.000 posixpath.py:39(_
get_sep)
           4    0.000    0.000    0.000    0.000 posixpath.py:73(j
oin)
           1    0.000    0.000    0.004    0.004 profileme.py:1(<m
odule>)
           2    0.000    0.000    0.000    0.000 re.py:231(compile
)
           2    0.000    0.000    0.000    0.000 re.py:286(_compil
e)
           2    0.000    0.000    0.000    0.000 sre_compile.py:22
3(_compile_charset)
           2    0.000    0.000    0.000    0.000 sre_compile.py:25
0(_optimize_charset)
           2    0.000    0.000    0.000    0.000 sre_compile.py:37
6(_mk_bitmap)
           2    0.000    0.000    0.000    0.000 sre_compile.py:37
8(<listcomp>)
```

click to scroll output; double click to hide



numba

numba is a library that attempts to automatically do type-based optimizations like we did with Cython. To use numba, you decorate functions with `@autojit`.

In [38]:
```
import numba

@numba.autojit
```

Kernel ready | Saving every 120s | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

Notebook file format

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- **`nbformat`** & **`nbformat_minor`**: The version of the format this notebook is stored in. The current version is 4.1.

Loading [MathJax]/extensions/Safe.js

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- **`nbformat`** & **`nbformat_minor`**: The version of the format this notebook is stored in. The current version is 4.1.
- **`metadata`**: Information about the notebook, like the language it's written in.
- **`cells`**: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The [notebook format documentation]() has the answers. It describes the structure of notebook files in detail.

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

- **nbformat** & **nbformat_minor**: The version of the format this notebook is stored in. The current version is 4.1.
- **metadata**: Information about the notebook, like the language it's written in.
- **cells**: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code ¶

The **IPython.nbformat** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [ ]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The **`IPython.nbformat`** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [5]:
```python
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

Out[5]:
```
{'cells': [{'cell_type': 'markdown',
   'metadata': {},
   'source': '# Notebook file format\n\nNotebooks are stored on
disk as JSON files. JSON is a really simple way of representing
data: it looks exactly like Python lists and dictionaries, so y
ou already know how to read it.\n\n```javascript\n{\n  "key": "
value",\n  "ultimate answer": 42,\n  "lists": ["like", "this"]\
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Manipulating notebooks in Python code

The **IPython.nbformat** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [5]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

Out[5]:
```
{'cells': [{'cell_type': 'markdown',
   'metadata': {},
   'source': '# Notebook file format\n\nNotebooks are stored on
disk as JSON files. JSON is a really simple way of representing
data: it looks exactly like Python lists and dictionaries, so y
ou already know how to read it.\n\n```javascript\n{\n  "key": "
value",\n  "ultimate answer": 42,\n  "lists": ["like", "this"]\
n}\n```\n\nAt the top level of the notebook file there are four
fields:\n\n* `nbformat` & `nbformat_minor`: The version of the
format this notebook is stored in. The current version is 4.1.\
n* `metadata`: Information about the notebook, like the languag
e it\'s written in.\n* `cells`: List of cells with the notebook
content  '},
  {'cell_type': 'markdown',
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Manipulating notebooks in Python code

The **IPython.nbformat** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [5]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

Out[5]:

```
{'cells': [{'cell_type': 'markdown',
   'metadata': {},
   'source': '# Notebook file format\n\nNotebooks are stored on
disk as JSON files. JSON is a really simple way of representing
data: it looks exactly like Python lists and dictionaries, so y
ou already know how to read it.\n\n```javascript\n{\n  "key": "
value",\n  "ultimate answer": 42,\n  "lists": ["like", "this"]\
n}\n```\n\nAt the top level of the notebook file there are four
fields:\n\n* `nbformat` & `nbformat_minor`: The version of the
format this notebook is stored in. The current version is 4.1.\
n* `metadata`: Information about the notebook, like the languag
e it\'s written in.\n* `cells`: List of cells with the notebook
content  '},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Challenges\n\nOpen the notebooks in this direc
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Manipulating notebooks in Python code

The **IPython.nbformat** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [5]:
```python
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

Out[5]:

```
{'cells': [{'cell_type': 'markdown',
   'metadata': {},
   'source': '# Notebook file format\n\nNotebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.\n\n```javascript\n{\n  "key": "value",\n  "ultimate answer": 42,\n  "lists": ["like", "this"]\n}\n```\n\nAt the top level of the notebook file there are four fields:\n\n* `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.\n* `metadata`: Information about the notebook, like the language it\'s written in.\n* `cells`: List of cells with the notebook content  '},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Challenges\n\nOpen the notebooks in this direc
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [5]:
```python
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

Out[5]:

```
{'cells': [{'cell_type': 'markdown',
   'metadata': {},
   'source': '# Notebook file format\n\nNotebooks are stored on
disk as JSON files. JSON is a really simple way of representing
data: it looks exactly like Python lists and dictionaries, so y
ou already know how to read it.\n\n```javascript\n{\n  "key": "
value",\n  "ultimate answer": 42,\n  "lists": ["like", "this"]\
n}\n```\n\nAt the top level of the notebook file there are four
fields:\n\n* `nbformat` & `nbformat_minor`: The version of the
format this notebook is stored in. The current version is 4.1.\
n* `metadata`: Information about the notebook, like the languag
e it\'s written in.\n* `cells`: List of cells with the notebook
content  '},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Challenges\n\nOpen the notebooks in this direc
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
n}\n```\n\nAt the top level of the notebook file there are four
fields:\n\n* `nbformat` & `nbformat_minor`: The version of the
format this notebook is stored in. The current version is 4.1.\
n* `metadata`: Information about the notebook, like the languag
e it\'s written in.\n* `cells`: List of cells with the notebook
content  '},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Challenges\n\nOpen the notebooks in this direc
tory in your text editor and look at the structure.\n\n1. What
distinguishes a markdown cell from a code cell?\n2. How many di
fferent kinds of output can you see?\n\nThe [notebook format do
cumentation](http://ipython.org/ipython-doc/3/notebook/nbformat
.html) has the answers. It describes the structure of notebook
files in detail.'},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3


```
n}\n```\n\nAt the top level of the notebook file there are four
fields:\n\n* `nbformat` & `nbformat_minor`: The version of the
format this notebook is stored in. The current version is 4.1.\
n* `metadata`: Information about the notebook, like the languag
e it\'s written in.\n* `cells`: List of cells with the notebook
content  '},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Challenges\n\nOpen the notebooks in this direc
tory in your text editor and look at the structure.\n\n1. What
distinguishes a markdown cell from a code cell?\n2. How many di
fferent kinds of output can you see?\n\nThe [notebook format do
cumentation](http://ipython.org/ipython-doc/3/notebook/nbformat
.html) has the answers. It describes the structure of notebook
files in detail.'},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
  {'cell_type': 'code',
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
.html) has the answers. It describes the structure of notebook
files in detail.'},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': 'print(nb.cells[2].source)'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "# Run this, then reload the page to see the chang
e\nnb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at
me!**'))\nnbformat.write(nb, 'Notebook file format.ipynb')"},
  {'cell_type': 'code',
   'execution_count': None,
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3


```
files in detail.'},
  {'cell_type': 'markdown',
   'metadata': {},
   'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': 'print(nb.cells[2].source)'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': "# Run this, then reload the page to see the chang
e\nnb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at
me!**'))\nnbformat.write(nb, 'Notebook file format.ipynb')"},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3


```
 {'cell_type': 'markdown',
  'metadata': {},
  'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': 'print(nb.cells[2].source)'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "# Run this, then reload the page to see the chang
e\nnb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at
me!**'))\nnbformat.write(nb, 'Notebook file format.ipynb')"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
```

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

```
 {'cell_type': 'markdown',
  'metadata': {},
  'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': 'print(nb.cells[2].source)'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "# Run this, then reload the page to see the chang
e\nnb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at
me!**'))\nnbformat.write(nb, 'Notebook file format.ipynb')"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3


```
 {'cell_type': 'markdown',
  'metadata': {},
  'source': '## Manipulating notebooks in Python code\n\nThe `
IPython.nbformat` package has functions to load and save notebo
oks, convert between different versions of the format, and vali
date notebooks against the specification.'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "import nbformat\nnb = nbformat.read('Notebook fil
e format.ipynb', as_version=4)\nnb"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': 'print(nb.cells[2].source)'},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
  'source': "# Run this, then reload the page to see the chang
e\nnb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at
me!**'))\nnbformat.write(nb, 'Notebook file format.ipynb')"},
 {'cell_type': 'code',
  'execution_count': None,
  'metadata': {'collapsed': True},
  'outputs': [],
```

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3


```
  'source': '!open "Notebook file format.html"'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': 'from nbconvert import get_export_names\nget_expor
t_names()'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': '!jupyter nbconvert --to script "nbval.ipynb"\n%py
cat nbval.py'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': '!jupyter nbconvert --to pdf "nbval.ipynb"'},
  {'cell_type': 'code',
   'execution_count': None,
   'metadata': {'collapsed': True},
   'outputs': [],
   'source': '!open nbval.pdf'}],
 'metadata': {'kernelspec': {'display_name': 'Python 3',
   'language': 'python',
   'name': 'python3'},
  'language_info': {'codemirror_mode': {'name': 'ipython', 'ver
sion': 3},
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
sion': 3},
    'file_extension': '.py',
    'mimetype': 'text/x-python',
    'name': 'python',
    'nbconvert_export
    'pygments_lexer':
    'version': '3.6.0
 'nbformat': 4,
 'nbformat_minor': 1}
```

From "localhost":

Are you sure you want to leave this page?

Stay on Page Leave Page

In [6]:
```
print(nb.cells[2].source)
```

```
## Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save n
otebooks, convert between different versions of the format, and
validate notebooks against the specification.
```

In [7]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!*
nbformat.write(nb, 'Notebook file format.ipynb')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file format.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

File Edit View Insert Cell Kernel Help Saving every 120s Python 3

Markdown CellToolbar

Look at me!

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

Loading [MathJax]/extensions/Safe.js

The `IPython.nbformat` package has functions to load and save notebooks, Saving every 120s Not Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help

Look at me!

Notebook file format

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [ ]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

In [ ]:
```
print(nb.cells[2].source)
```

In [ ]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!*
nbformat.write(nb, 'Notebook file format.ipynb')
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

Manipulating notebooks in Python code

The **`IPython.nbformat`** package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [ ]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

In [ ]:
```
print(nb.cells[2].source)
```

In [ ]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!*
nbformat.write(nb, 'Notebook file format.ipynb')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file format.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

convert between different versions of the format, and validate notebooks against the specification.

In [ ]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

In [ ]:
```
print(nb.cells[2].source)
```

In [ ]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!*
nbformat.write(nb, 'Notebook file format.ipynb')
```

In [8]:
```
!jupyter nbconvert --to html "Notebook file format.ipynb"
```

```
[NbConvertApp] Converting notebook Notebook file format.ipynb t
o html
[NbConvertApp] Writing 259945 bytes to Notebook file format.htm
l
```

In [*]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

Look at me!

Notebook file format ¶

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
    "value",
```

Loading [Contrib]/a11y/accessibility-menu.js

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

At the top level of the notebook file there are four fields:

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

```
nbformat.write(nb, 'Notebook file format.ipyn
b')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file f
ormat.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

In [ ]:
```
!jupyter nbconvert --to script "nbval.ipynb"
%pycat nbval.py
```

In [ ]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

In [ ]:
```
!open nbval.pdf
```

In [ ]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!**'))
nbformat.write(nb, 'Notebook file format.ipynb')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file format.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

In [ ]:
```
!jupyter nbconvert --to script "nbval.ipynb"
%pycat nbval.py
```

In [ ]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
nbformat.write(nb, 'Notebook file format.ipyn
b')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file f
ormat.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

In [ ]:
```
!jupyter nbconvert --to script "nbval.ipynb"
%pycat nbval.py
```

In [ ]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# $ py.test --nbval nbval.ipynb --sanitize-with doc_sanitize.cf
g
# ```

# ### Examples of plugin behaviour

# The following examples demonstrate how the plugin behaves dur
ing testing. Test this notebook yourself to see the validation
in action!

# These two imports produce no output as standard, if any **war
nings** are printed out the cell will fail. Under normal operat
ing conditions they will pass.

# In[2]:

import numpy as np
import time


# If python doesn't consistently print 7, then something has go
ne terribly wrong. **Deterministic cells** are expected to pass
everytime

# In[3]:

print(5+2)
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# coding: utf-8

# # IPython Notebook Validation for py.test - Documentation

# One of the powerful uses of the IPython notebook is for docu
mentation purposes, here we use a notebook to demonstrate the
behaviour and usage of the IPython Notebook Validation plugin
for py.test. The IPython notebook format `.ipynb` stores outpu
ts as well as inputs. Validating the notebook means to rerun t
he notebook and make sure that it is generating the same outpu
t as has been stored.
#
# Therefore, the **user MUST make the following the distinctio
n**:
#
# 1. Running a notebook manually will likely change the output
```

```
In [ ]: !jupyter nbconvert --to pdf "nbval.ipynb"
```

```
In [ ]: !open nbval.pdf
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
get_export_names()
```

Out[10]:
```
['asciidoc',
 'custom',
 'html',
 'latex',
 'markdown',
 'notebook',
 'pdf',
 'python',
 'rst',
 'script',
 'slides']
```

In [11]:
```
!jupyter nbconvert --to script "nbval.ipynb"
%pycat nbval.py
```

```
[NbConvertApp] Converting notebook nbval.ipynb to script
[NbConvertApp] Writing 7081 bytes to nbval.py


# coding: utf-8

# # IPython Notebook Validation for py.test - Documentation

# One of the powerful uses of the IPython notebook is for docu
mentation purposes, here we use a notebook to demonstrate the
behaviour and usage of the IPython Notebook Validation plugin
for py.test. The IPython notebook format `.ipynb` stores outpu
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# coding: utf-8

# # IPython Notebook Validation for py.test - Documentation

# One of the powerful uses of the IPython notebook is for docu
mentation purposes, here we use a notebook to demonstrate the
behaviour and usage of the IPython Notebook Validation plugin
for py.test. The IPython notebook format `.ipynb` stores outpu
ts as well as inputs. Validating the notebook means to rerun t
he notebook and make sure that it is generating the same outpu
t as has been stored.
#
# Therefore, the **user MUST make the following the distinctio
n**:
#
# 1. Running a notebook manually will likely change the output
```

In [*]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
```

In [ ]:
```
!open nbval.pdf
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
for py.test. The IPython notebook format `.ipynb` stores outpu
ts as well as inputs. Validating the notebook means to rerun t
he notebook and make sure that it is generating the same outpu
t as has been stored.
#
# Therefore, the **user MUST make the following the distinctio
n**:
#
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [ ]:
```
!open nbval.pdf
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [*]:
```
!open nbval.pdf
```

In [ ]:

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

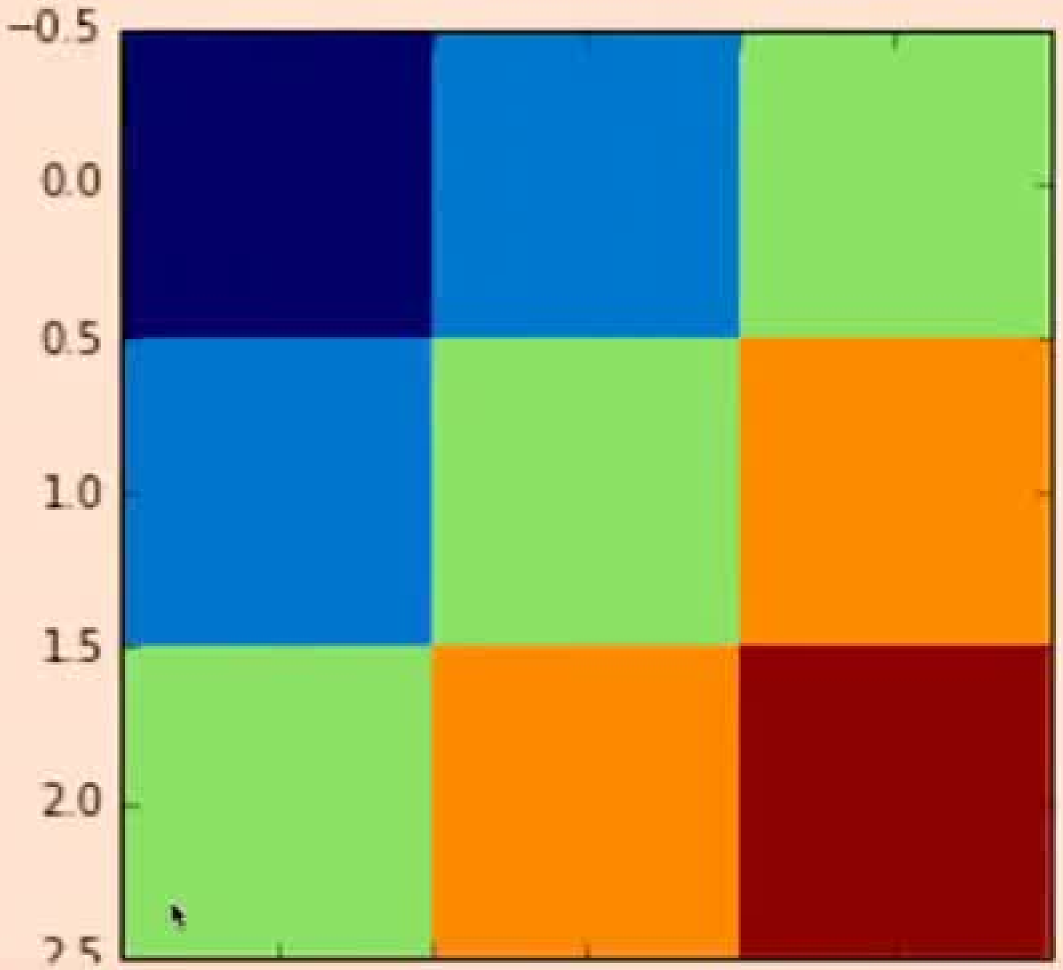
```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [13]:
```
!open nbval.pdf
```

In [ ]:

```
In [10]: plt.imshow(np.array([[i + j for i in range(3)]
                              for j in range(3)]),
                    interpolation='None'
                    )
```

```
Out[10]: <matplotlib.image.AxesImage at 0x7f2cb3374198>
```



If the raised exception doesn't match the stored exception, we get a failure

---

4


```
RuntimeError                              Traceback (most recent call last)

<ipython-input-3-32dcc1c70a4e> in <module>()
  1 # NBVAL_RAISES_EXCEPTION
  2 print("If the raised exception doesn't match the stored exception, we get a failure"
----> 3 raise RuntimeError("Foo")


RuntimeError: Foo
```

```
In [2]: # NBVAL_IGNORE_OUTPUT
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

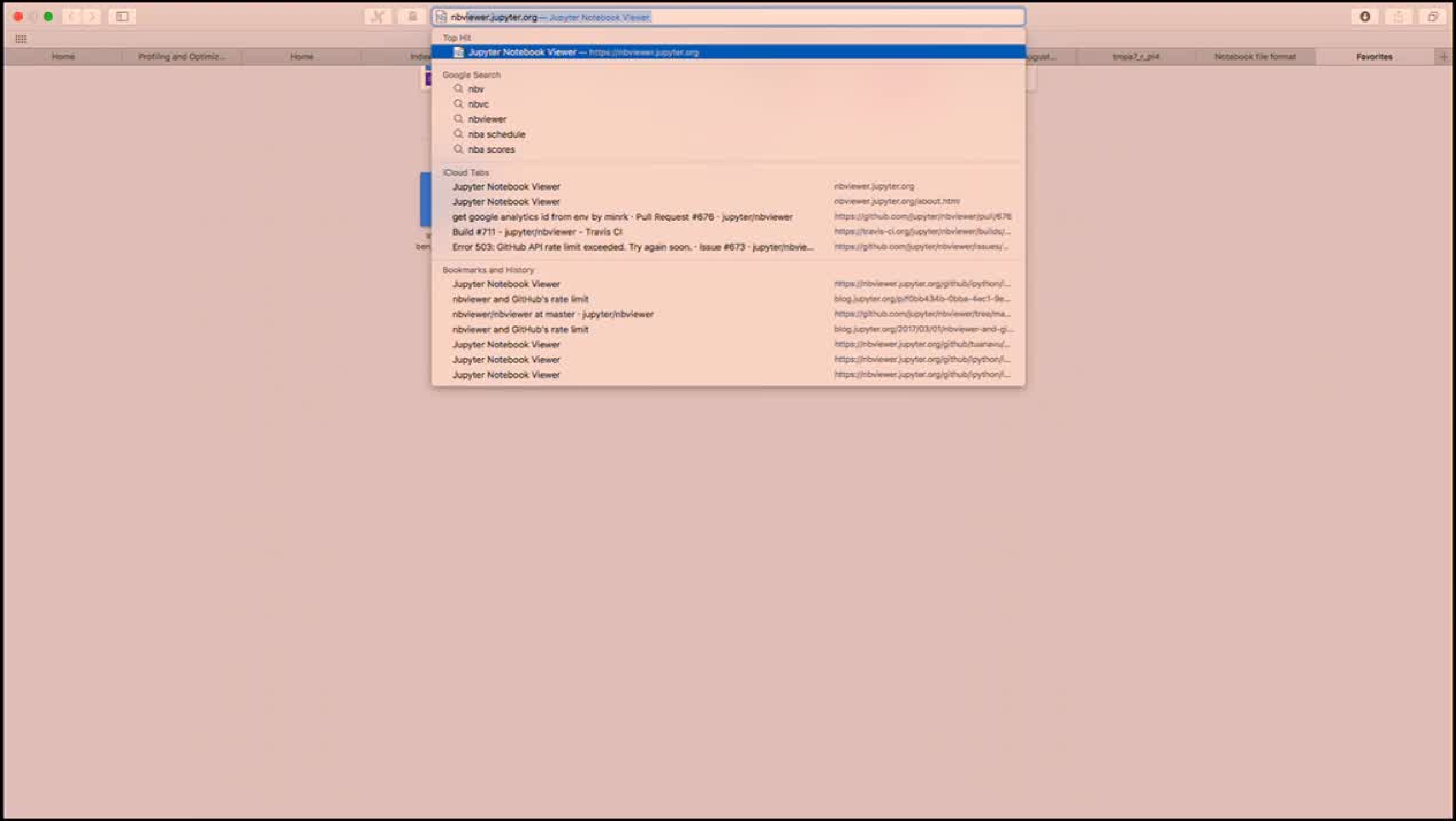
```
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [13]:
```
!open nbval.pdf
```

In [ ]:


nbviewer.jupyter.org — Jupyter Notebook Viewer
Home
Profiling and Optimiz...
Home
Notebook file format
Favorites
Top Hit
Jupyter Notebook Viewer — https://nbviewer.jupyter.org
Google Search
nbv
nbvc
nbviewer
nba schedule
nba scores
iCloud Tabs
Jupyter Notebook Viewer
nbviewer.jupyter.org
Jupyter Notebook Viewer
nbviewer.jupyter.org/about.html
get google analytics id from env by minrk · Pull Request #676 · jupyter/nbviewer
Build #711 - jupyter/nbviewer - Travis CI
Error 503: GitHub API rate limit exceeded. Try again soon. · Issue #673 · jupyter/nbvie...
Bookmarks and History
Jupyter Notebook Viewer
nbviewer and GitHub's rate limit
nbviewer/nbviewer at master · jupyter/nbviewer
nbviewer and GitHub's rate limit
Jupyter Notebook Viewer
Jupyter Notebook Viewer
Jupyter Notebook Viewer

jupyter
nbviewer

JUPYTER FAQ

nbviewer

A simple way to share Jupyter Notebooks

minrk Go!

Programming Languages

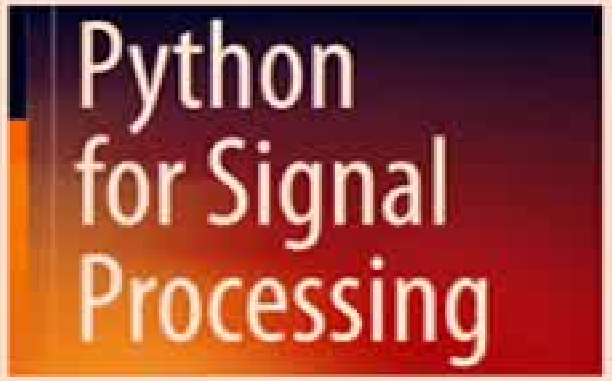
IPython

IRuby

IJulia

Books

Python for Signal Processing



O'Reilly Book

Probabilistic Programming

Misc

Data Visualization with Lightning

Interactive data visualization with Bokeh

Interactive plots with Plotly

Notebook file format

Notebooks are stored on disk as JSON files. JSON is a really simple way of representing data: it looks exactly like Python lists and dictionaries, so you already know how to read it.

```
{
  "key": "value",
  "ultimate answer": 42,
  "lists": ["like", "this"]
}
```

At the top level of the notebook file there are four fields:

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

- `nbformat` & `nbformat_minor`: The version of the format this notebook is stored in. The current version is 4.1.
- `metadata`: Information about the notebook, like the language it's written in.
- `cells`: List of cells with the notebook content

Challenges

Open the notebooks in this directory in your text editor and look at the structure.

1. What distinguishes a markdown cell from a code cell?
2. How many different kinds of output can you see?

The notebook format documentation has the answers. It describes the structure of notebook files in detail.

Manipulating notebooks in Python code

The `IPython.nbformat` package has functions to load and save notebooks, convert between different versions of the format, and validate notebooks against the specification.

In [ ]:
```
import nbformat
nb = nbformat.read('Notebook file format.ipynb', as_version=4)
nb
```

In [ ]:
```
print(nb.cells[2].source)
```

In [ ]:
```
# Run this, then reload the page to see the change
nb.cells.insert(0, nbformat.v4.new_markdown_cell('**Look at me!**'))
nbformat.write(nb, 'Notebook file format.ipynb')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file format.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

IPython: beyond plain Python

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
```

jupyter nbviewer   JUPYTER   FAQ

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
```

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
```

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------

* Access to the standard Python help with object docstrings and the Python
  manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.
```

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.
```

IPython: beyond plain Python

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```
```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```
```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.
```

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
```

When executing code in IPython, all valid Python syntax works as-is, but IPython provides a number of features designed to make the interactive experience more fluid and efficient.

First things first: running code, getting help

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------
```

In the notebook, to run a cell of code, hit Shift-Enter. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

In [1]:
```
print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

In [2]:
```
?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------

* Access to the standard Python help with object docstrings and the Python
  manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.

* System command aliases, via the %alias command or the configuration file(s).

* Dynamic object information:

  Typing ?word or word? prints detailed information about an object. Certain
```

In the notebook, to run a cell of code, hit `Shift-Enter`. This executes the cell and puts the cursor in the next cell below, or makes a new one if you are at the end. Alternately, you can use:

- `Alt-Enter` to force the creation of a new cell unconditionally (useful when inserting new content in the middle of an existing notebook).
- `Control-Enter` executes the cell and keeps the cursor in the same cell, useful for quick experimentation of snippets that you don't need to keep permanently.

```
In [1]: print("Hi")
```

```
Hi
```

To get help use the question mark `?`. This will bring out the pager, use `esc` or `q` to close it with the keybard, or click on the close icon ton the top right of the pager.

```
In [2]: ?
```

```
IPython -- An enhanced Interactive Python
=========================================

IPython offers a fully compatible replacement for the standard Python
interpreter, with convenient shell features, special commands, command
history mechanism and output results caching.

At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------

* Access to the standard Python help with object docstrings and the Python
  manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.

* System command aliases, via the %alias command or the configuration file(s).

* Dynamic object information:

  Typing ?word or word? prints detailed information about an object. Certain
```

```
At your system command line, type 'ipython -h' to see the command line
options available. This document only describes interactive features.

MAIN FEATURES
-------------

* Access to the standard Python help with object docstrings and the Python
  manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.

* System command aliases, via the %alias command or the configuration file(s).

* Dynamic object information:

  Typing ?word or word? prints detailed information about an object. Certain
  long strings (code, etc.) get snipped in the center for brevity.

  Typing ??word or word?? gives access to the full information without
  snipping long strings. Strings that are longer than the screen are printed
  through the less pager.

  The ?/?? system gives access to the full source code for any object (if
  available), shows function prototypes and other useful information.

  If you just want to see an object's docstring, type '%pdoc object' (without
  quotes, and without % if you have automagic on).

* Tab completion in the local namespace:

  At any time, hitting tab will complete any available python commands or
  variable names, and show you a list of the possible completions if there's
  no unambiguous one. It will also complete filenames in the current directory.

* Search previous command history in multiple ways:

  - Start typing, and then use arrow keys up/down or (Ctrl-p/Ctrl-n) to search
    through the history items that match what you've typed so far.

  - Hit Ctrl-r: opens a search prompt. Begin typing and the system searches
    your history for lines that match what you've typed so far, completing as
    much as it can.
```

At your system command line, type 'ipython -h' to see the command line options available. This document only describes interactive features.

MAIN FEATURES

* Access to the standard Python help with object docstrings and the Python manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.

* System command aliases, via the %alias command or the configuration file(s).

* Dynamic object information:

  Typing ?word or word? prints detailed information about an object. Certain long strings (code, etc.) get snipped in the center for brevity.

  Typing ??word or word?? gives access to the full information without snipping long strings. Strings that are longer than the screen are printed through the less pager.

  The ?/?? system gives access to the full source code for any object (if available), shows function prototypes and other useful information.

  If you just want to see an object's docstring, type '%pdoc object' (without quotes, and without % if you have automagic on).

* Tab completion in the local namespace:

  At any time, hitting tab will complete any available python commands or variable names, and show you a list of the possible completions if there's no unambiguous one. It will also complete filenames in the current directory.

* Search previous command history in multiple ways:

  - Start typing, and then use arrow keys up/down or (Ctrl-p/Ctrl-n) to search through the history items that match what you've typed so far.

  - Hit Ctrl-r: opens a search prompt. Begin typing and the system searches your history for lines that match what you've typed so far, completing as much as it can.

```
* Access to the standard Python help with object docstrings and the Python
  manuals. Simply type 'help' (no quotes) to invoke it.

* Magic commands: type %magic for information on the magic subsystem.

* System command aliases, via the %alias command or the configuration file(s).

* Dynamic object information:

  Typing ?word or word? prints detailed information about an object. Certain
  long strings (code, etc.) get snipped in the center for brevity.

  Typing ??word or word?? gives access to the full information without
  snipping long strings. Strings that are longer than the screen are printed
  through the less pager.

  The ?/?? system gives access to the full source code for any object (if
  available), shows function prototypes and other useful information.

  If you just want to see an object's docstring, type '%pdoc object' (without
  quotes, and without % if you have automagic on).

* Tab completion in the local namespace:

  At any time, hitting tab will complete any available python commands or
  variable names, and show you a list of the possible completions if there's
  no unambiguous one. It will also complete filenames in the current directory.

* Search previous command history in multiple ways:

  - Start typing, and then use arrow keys up/down or (Ctrl-p/Ctrl-n) to search
    through the history items that match what you've typed so far.

  - Hit Ctrl-r: opens a search prompt. Begin typing and the system searches
    your history for lines that match what you've typed so far, completing as
    much as it can.

  - %hist: search history by index.

* Persistent command history across sessions.

* Logging of input with the ability to save and restore a working session.
```

```
  long strings (code, etc.) get snipped in the center for brevity.

  Typing ??word or word?? gives access to the full information without
  snipping long strings. Strings that are longer than the screen are printed
  through the less pager.

  The ?/?? system gives access to the full source code for any object (if
  available), shows function prototypes and other useful information.

  If you just want to see an object's docstring, type '%pdoc object' (without
  quotes, and without % if you have automagic on).

* Tab completion in the local namespace:

  At any time, hitting tab will complete any available python commands or
  variable names, and show you a list of the possible completions if there's
  no unambiguous one. It will also complete filenames in the current directory.

* Search previous command history in multiple ways:

  - Start typing, and then use arrow keys up/down or (Ctrl-p/Ctrl-n) to search
    through the history items that match what you've typed so far.

  - Hit Ctrl-r: opens a search prompt. Begin typing and the system searches
    your history for lines that match what you've typed so far, completing as
    much as it can.

  - %hist: search history by index.

* Persistent command history across sessions.

* Logging of input with the ability to save and restore a working session.

* System shell with !. Typing !ls will run 'ls' in the current directory.

* The reload command does a 'deep' reload of a module: changes made to the
  module since you imported will actually be available without having to exit.
```

```
  long strings (code, etc.) get snipped in the center for brevity.

  Typing ??word or word?? gives access to the full information without
  snipping long strings. Strings that are longer than the screen are printed
  through the less pager.

  The ?/?? system gives access to the full source code for any object (if
  available), shows function prototypes and other useful information.

  If you just want to see an object's docstring, type '%pdoc object' (without
  quotes, and without % if you have automagic on).

* Tab completion in the local namespace:

  At any time, hitting tab will complete any available python commands or
  variable names, and show you a list of the possible completions if there's
  no unambiguous one. It will also complete filenames in the current directory.

* Search previous command history in multiple ways:

  - Start typing, and then use arrow keys up/down or (Ctrl-p/Ctrl-n) to search
    through the history items that match what you've typed so far.

  - Hit Ctrl-r: opens a search prompt. Begin typing and the system searches
    your history for lines that match what you've typed so far, completing as
    much as it can.

  - %hist: search history by index.

* Persistent command history across sessions.

* Logging of input with the ability to save and restore a working session.

* System shell with !. Typing !ls will run 'ls' in the current directory.

* The reload command does a 'deep' reload of a module: changes made to the
  module since you imported will actually be available without having to exit.
```

* Output caching system:

  For output that is returned from actions, a system similar to the input cache exists but using _ instead of _i. Only actions that produce a result (NOT assignments, for example) are cached. If you are familiar with Mathematica, IPython's _ variables behave exactly like Mathematica's % variables.

  The following GLOBAL variables always exist (so don't overwrite them!):
  _ (one underscore): previous output.
  __ (two underscores): next previous.
  ___ (three underscores): next-next previous.

  Global variables named _<n> are dynamically created (<n> being the prompt counter), such that the result of output <n> is always available as _<n>.

  Finally, a global dictionary named _oh exists with entries for all lines which generated output.

* Directory history:

  Your history of visited directories is kept in the global list _dh, and the magic %cd command can be used to go to any entry in that list.

* Auto-parentheses and auto-quotes (adapted from Nathan Gray's LazyPython)

  1. Auto-parentheses

     Callable objects (i.e. functions, methods, etc) can be invoked like this (notice the commas between the arguments)::

     ```
     In [1]: callable_ob arg1, arg2, arg3
     ```

     and the input will be translated to this::

     ```
     callable_ob(arg1, arg2, arg3)
     ```

     This feature is off by default (in rare cases it can produce undesirable side-effects), but you can activate it at the command-line by starting IPython with `--autocall 1`, set it permanently in your configuration file, or turn on at runtime with `%autocall 1`.

     You can force auto-parentheses by using '/' as the first character of a line. For example::

```
In [1]: /globals             # becomes 'globals()'
```

Note that the '/' MUST be the first character on the line! This won't work::

```
In [2]: print /globals    # syntax error
```

In most cases the automatic algorithm should work, so you should rarely need to explicitly invoke /. One notable exception is if you are trying to call a function with a list of tuples as arguments (the parenthesis will confuse IPython)::

```
In [1]: zip (1,2,3),(4,5,6)  # won't work
```

but this will work::

```
In [2]: /zip (1,2,3),(4,5,6)
------> zip ((1,2,3),(4,5,6))
Out[2]= [(1, 4), (2, 5), (3, 6)]
```

IPython tells you that it has altered your command line by displaying the new command line preceded by -->. e.g.::

```
In [18]: callable list
-------> callable (list)
```

2. Auto-Quoting

   You can force auto-quoting of a function's arguments by using ',' as the first character of a line. For example::

```
In [1]: ,my_function /home/me   # becomes my_function("/home/me")
```

   If you use ';' instead, the whole argument is quoted as a single string (while ',' splits on whitespace)::

```
In [2]: ,my_function a b c   # becomes my_function("a","b","c")
In [3]: ;my_function a b c   # becomes my_function("a b c")
```

   Note that the ',' MUST be the first character on the line! This won't work::

```
    In [2]: /zip (1,2,3),(4,5,6)
    ------> zip ((1,2,3),(4,5,6))
    Out[2]= [(1, 4), (2, 5), (3, 6)]

IPython tells you that it has altered your command line by
displaying the new command line preceded by -->.  e.g.::

    In [18]: callable list
    -------> callable (list)

2. Auto-Quoting

You can force auto-quoting of a function's arguments by using ',' as
the first character of a line.  For example::

    In [1]: ,my_function /home/me   # becomes my_function("/home/me")

If you use ';' instead, the whole argument is quoted as a single
string (while ',' splits on whitespace)::

    In [2]: ,my_function a b c   # becomes my_function("a","b","c")
    In [3]: ;my_function a b c   # becomes my_function("a b c")

Note that the ',' MUST be the first character on the line!  This
won't work::

    In [4]: x = ,my_function /home/me    # syntax error
```

Typing `object_name?` will print all sorts of details about any object, including docstrings, function definition lines (for call arguments) and constructor details for classes.

In [3]:

```python
import pandas as pd
pd.DataFrame?
```

```
Init signature: pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
Docstring:
Two-dimensional size-mutable, potentially heterogeneous tabular data
structure with labeled axes (rows and columns). Arithmetic operations
align on both row and column labels. Can be thought of as a dict-like
container for Series objects. The primary pandas data structure
```

```
>>> d = {'col1': ts1, 'col2': ts2}
>>> df = DataFrame(data=d, index=index)
>>> df2 = DataFrame(np.random.randn(10, 5))
>>> df3 = DataFrame(np.random.randn(10, 5),
...                 columns=['a', 'b', 'c', 'd', 'e'])

See also
--------
DataFrame.from_records : constructor from tuples, also record arrays
DataFrame.from_dict : from dicts of Series, arrays, or dicts
DataFrame.from_items : from sequence of (key, value) pairs
pandas.read_csv, pandas.read_table, pandas.read_clipboard
File:           ~/conda/lib/python3.6/site-packages/pandas/core/frame.py
Type:           type
```

Using two question marks will try to find the source code for the given object.

In [4]:
```
pd.DataFrame??
```

```
Init signature: pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
Source:
class DataFrame(NDFrame):
    """ Two-dimensional size-mutable, potentially heterogeneous tabular data
    structure with labeled axes (rows and columns). Arithmetic operations
    align on both row and column labels. Can be thought of as a dict-like
    container for Series objects. The primary pandas data structure

    Parameters
    ----------
    data : numpy ndarray (structured or homogeneous), dict, or DataFrame
        Dict can contain Series, arrays, constants, or list-like objects
    index : Index or array-like
        Index to use for resulting frame. Will default to np.arange(n) if
        no indexing information part of input data and no index provided
    columns : Index or array-like
        Column labels to use for resulting frame. Will default to
        np.arange(n) if no column labels are provided
    dtype : dtype, default None
        Data type to force, otherwise infer
    copy : boolean, default False
        Copy data from inputs. Only affects DataFrame / 2d ndarray input
```

```
pandas.read_csv, pandas.read_table, pandas.read_clipboard
File:           ~/conda/lib/python3.6/site-packages/pandas/core/frame.py
Type:           type
```

Using two question marks will try to find the source code for the given object.

In [4]:
```
pd.DataFrame??
```

```
Init signature: pd.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)
Source:
class DataFrame(NDFrame):
    """ Two-dimensional size-mutable, potentially heterogeneous tabular data
    structure with labeled axes (rows and columns). Arithmetic operations
    align on both row and column labels. Can be thought of as a dict-like
    container for Series objects. The primary pandas data structure

    Parameters
    ----------
    data : numpy ndarray (structured or homogeneous), dict, or DataFrame
        Dict can contain Series, arrays, constants, or list-like objects
    index : Index or array-like
        Index to use for resulting frame. Will default to np.arange(n) if
        no indexing information part of input data and no index provided
    columns : Index or array-like
        Column labels to use for resulting frame. Will default to
        np.arange(n) if no column labels are provided
    dtype : dtype, default None
        Data type to force, otherwise infer
    copy : boolean, default False
        Copy data from inputs. Only affects DataFrame / 2d ndarray input

    Examples
    --------
    >>> d = {'col1': ts1, 'col2': ts2}
    >>> df = DataFrame(data=d, index=index)
    >>> df2 = DataFrame(np.random.randn(10, 5))
    >>> df3 = DataFrame(np.random.randn(10, 5),
    ...                 columns=['a', 'b', 'c', 'd', 'e'])

    See also
    --------
```

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.


```
est --nbval-lax my_notebook.ipynb
```

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

REGEX Output sanitizing

Since all output is captured by the IPython notebook, some pesky messages and prompts (with time-stamped messages, for example) may fail tests always, which might be expected. The plugin allows the user to specify a sanitizing file at the command prompt using the following flag:

```
$ py.test --nbval my_notebook.ipynb --sanitize-with my_sanitize_file
```

This sanitize file contains a number of REGEX replacements. It is recommended, when removing output for the tests, that you replace the removed output with some sort of marker, this helps with debugging. The following file is written to the folder of this notebook and can be used to santize its outputs:

In [1]:
```
%%writefile doc_sanitize.cfg
[regex1]
regex: \d{1,2}/\d{1,2}/\d{2,4}
replace: DATE-STAMP

[regex2]
regex: \d{2}:\d{2}:\d{2}
replace: TIME-STAMP
```

```
$ cd /path/to/this/notebook
$ py.test --nbval nbval.ipynb --sanitize-with doc_sanitize.cfg
```

Examples of plugin behaviour

The following examples demonstrate how the plugin behaves during testing. Test this notebook yourself to see the validation in action!

These two imports produce no output as standard, if any **warnings** are printed out the cell will fail. Under normal operating conditions they will pass.

In [2]:
```python
import numpy as np
import time
```

If python doesn't consistently print 7, then something has gone terribly wrong. **Deterministic cells** are expected to pass everytime

In [3]:
```python
print(5+2)
```

```
7
```

Random outputs will always fail.

In [4]:
```python
print([np.random.rand() for i in range(4)])
print([np.random.rand() for i in range(4)])
```

```
[0.36133679016382714, 0.5043774697891126, 0.23281910875007927, 0.2713065513128683]
[0.5512421277985322, 0.02592706358897756, 0.05036036771084684, 0.7515926759190724]
```

Inconsistent number of lines of output will cause an error to be thrown.

In [5]:
```python
for i in range(np.random.randint(1, 8)):
    print(1)
```

```
1
```

```
$ cd /path/to/this/notebook
$ py.test --nbval nbval.ipynb --sanitize-with doc_sanitize.cfg
```

Examples of plugin behaviour

The following examples demonstrate how the plugin behaves during testing. Test this notebook yourself to see the validation in action!

These two imports produce no output as standard, if any **warnings** are printed out the cell will fail. Under normal operating conditions they will pass.

In [2]:
```
import numpy as np
import time
```

If python doesn't consistently print 7, then something has gone terribly wrong. **Deterministic cells** are expected to pass everytime

In [3]:
```
print(5+2)
```

```
7
```

Random outputs will always fail.

In [4]:
```
print([np.random.rand() for i in range(4)])
print([np.random.rand() for i in range(4)])
```

```
[0.36133679016382714, 0.5043774697891126, 0.23281910875007927, 0.2713065513128683]
[0.5512421277985322, 0.02592706358897756, 0.05036036771084684, 0.7515926759190724]
```

Inconsistent number of lines of output will cause an error to be thrown.

In [5]:
```
for i in range(np.random.randint(1, 8)):
    print(1)
```

```
1
```

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction:**

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

Branch: master ▾ ipython-cse17 / nbval.ipynb

Find file Cop

minrk add nbval, nbformat 16a9059 39 minut

1 contributor

557 lines (556 sloc) 21.6 KB

Raw Blame History

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

REGEX Output sanitizing

Since all output is captured by the IPython notebook, some pesky messages and prompts (with time-stamped messages, for example) may fail tests always, which might be expected. The plugin allows the user to specify a sanitizing file at the command

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# IPython Notebook Validation for py.test - Documentation"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.\n",
    "\n",
    "Therefore, the **user MUST make the following the distinction**:\n",
    "\n",
    "1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)\n",
    "2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file\n",
    "\n",
    "The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Command line usage"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.\n",
    "\n",
    "```\n",
    "$ py.test --nbval my_notebook.ipynb\n",
    "```\n",
    "\n",
```

```
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# IPython Notebook Validation for py.test - Documentation"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.\n",
    "\n",
    "Therefore, the **user MUST make the following the distinction**:\n",
    "\n",
    "1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)\n",
    "2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file\n",
    "\n",
    "The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this."
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "### Command line usage"
   ]
```

minrk / ipython-cse17

Unwatch 1 | Star 2 | For

<> Code | Issues 0 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

Branch: master | ipython-cse17 / nbval.ipynb | Find file | Cop

minrk add nbval, nbformat | 16a9059 40 minut

1 contributor

557 lines (556 sloc) | 21.6 KB | Raw | Blame | History

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

minrk / ipython-cse17

Unwatch 1 Star 2 For

<> Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

Branch: master ipython-cse17 / nbval.ipynb

Find file Cop

minrk add nbval, nbformat 16a9059 40 minut

1 contributor

557 lines (556 sloc) 21.6 KB

Raw Blame History

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

REGEX Output sanitizing

Since all output is captured by the IPython notebook, some pesky messages and prompts (with time-stamped messages, for example) may fail tests always, which might be expected. The plugin allows the user to specify a sanitizing file at the command prompt using the following flag:

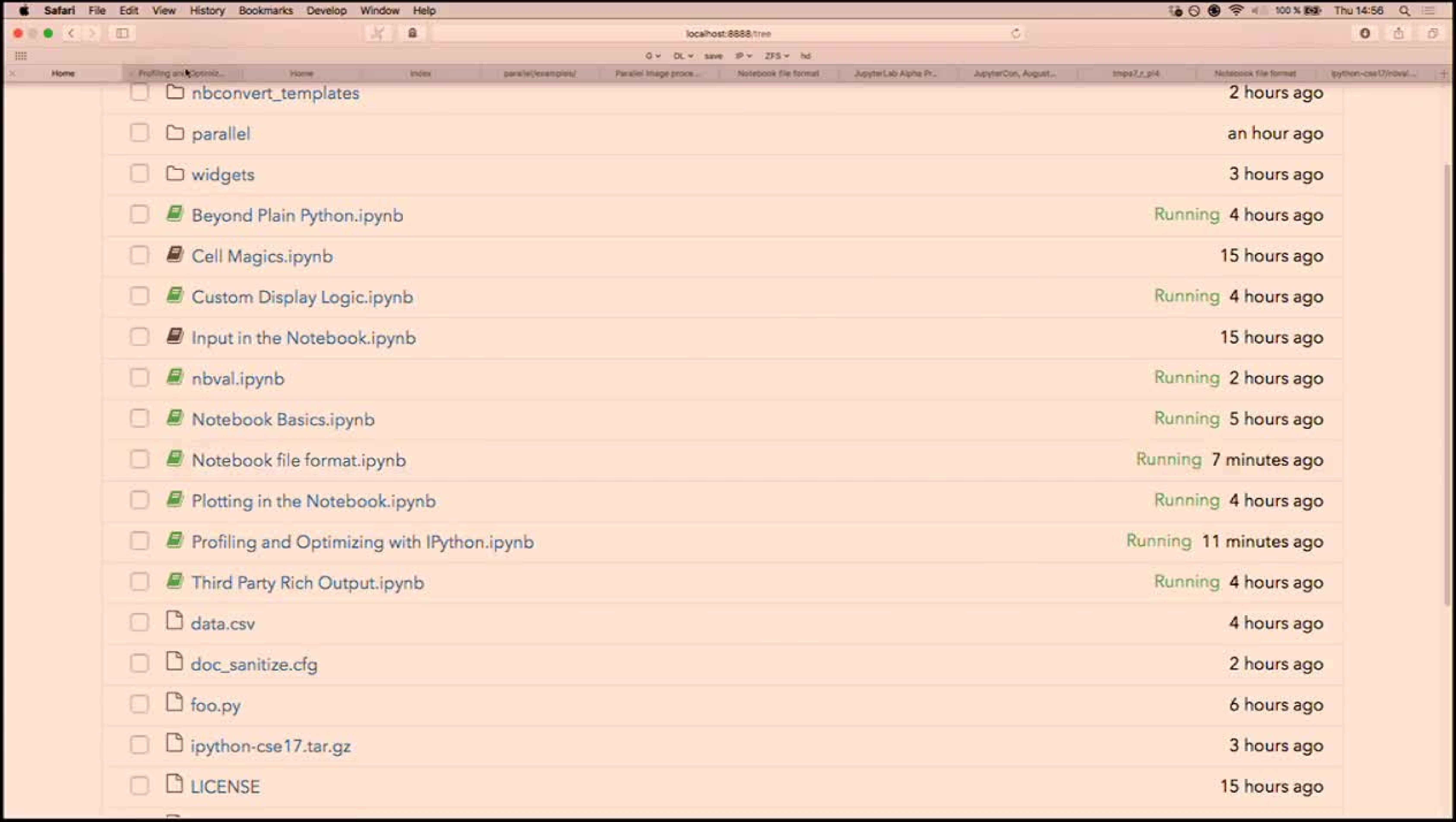
```
$ py.test --nbval my_notebook.ipynb --sanitize-with my_sanitize_file
```

This sanitize file contains a number of REGEX replacements. It is recommended, when removing output for the tests, that you replace the removed output with some sort of marker, this helps with debugging. The following file is written to the folder of this notebook and can be used to santize its outputs:

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction:**


Safari File Edit View History Bookmarks Develop Window Help
Thu 14:56
localhost:8888/tree
Home
nbconvert_templates 2 hours ago
parallel an hour ago
widgets 3 hours ago
Beyond Plain Python.ipynb Running 4 hours ago
Cell Magics.ipynb 15 hours ago
Custom Display Logic.ipynb Running 4 hours ago
Input in the Notebook.ipynb 15 hours ago
nbval.ipynb Running 2 hours ago
Notebook Basics.ipynb Running 5 hours ago
Notebook file format.ipynb Running 7 minutes ago
Plotting in the Notebook.ipynb Running 4 hours ago
Profiling and Optimizing with IPython.ipynb Running 11 minutes ago
Third Party Rich Output.ipynb Running 4 hours ago
data.csv 4 hours ago
doc_sanitize.cfg 2 hours ago
foo.py 6 hours ago
ipython-cse17.tar.gz 3 hours ago
LICENSE 15 hours ago

Safari File Edit View History Bookmarks Develop Window Help

localhost:8888/tree

Name	Status	Last Modified
nbconvert_templates		2 hours ago
parallel		an hour ago
widgets		3 hours ago
Beyond Plain Python.ipynb	Running	4 hours ago
Cell Magics.ipynb		15 hours ago
Custom Display Logic.ipynb	Running	4 hours ago
Input in the Notebook.ipynb		15 hours ago
nbval.ipynb	Running	2 hours ago
Notebook Basics.ipynb	Running	5 hours ago
Notebook file format.ipynb	Running	7 minutes ago
Plotting in the Notebook.ipynb	Running	4 hours ago
Profiling and Optimizing with IPython.ipynb	Running	11 minutes ago
Third Party Rich Output.ipynb	Running	4 hours ago
data.csv		4 hours ago
doc_sanitize.cfg		2 hours ago
foo.py		6 hours ago
ipython-cse17.tar.gz		3 hours ago
LICENSE		15 hours ago

```
nbformat.write(nb, 'Notebook file format.ipyn
b')
```

In [ ]:
```
!jupyter nbconvert --to html "Notebook file f
ormat.ipynb"
```

In [ ]:
```
!open "Notebook file format.html"
```

In [ ]:
```
from nbconvert import get_export_names
get_export_names()
```

In [ ]:
```
!jupyter nbconvert --to script "nbval.ipynb"
%pycat nbval.py
```

In [ ]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction:**

jupyter
nbviewer

JUPYTER FAQ

nbviewer

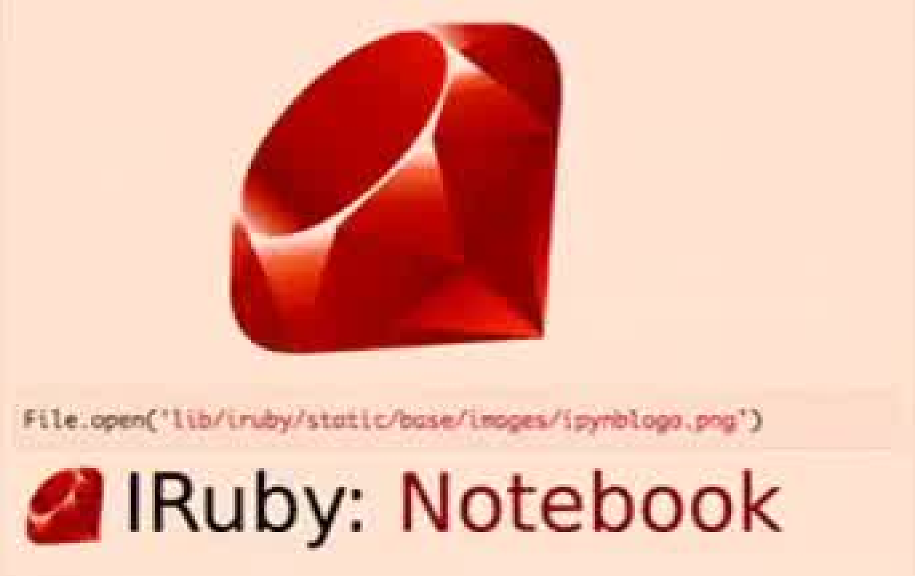
A simple way to share Jupyter Notebooks

URL | GitHub username | GitHub username/repo | Gist ID Go!

Programming Languages

IPython

IRuby



IJulia

Books

jupyter nbviewer

JUPYTER FAQ

nbviewer

A simple way to share Jupyter Notebooks

URL | GitHub username | GitHub username/repo | Gist ID Go!

Programming Languages

IPython

IRuby

IJulia

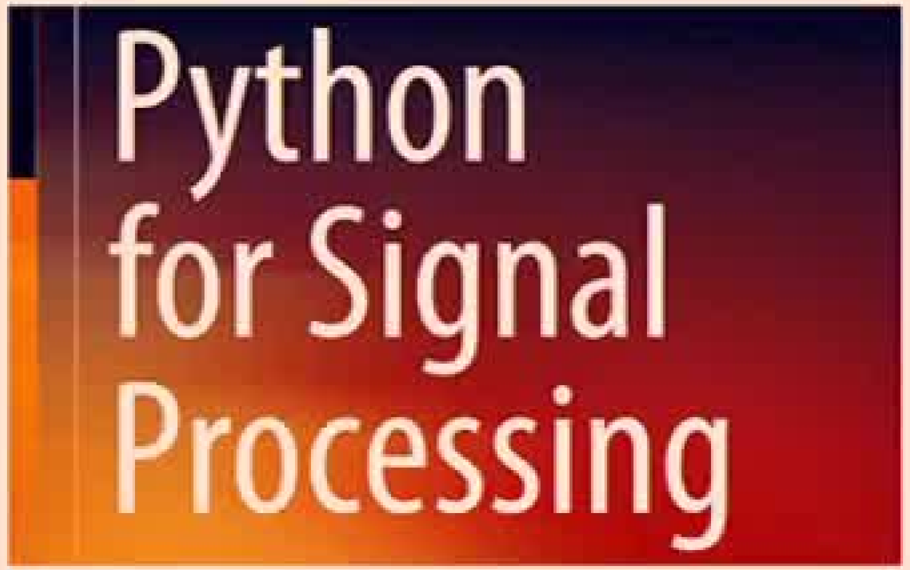
Books

jupyter
nbviewer

JUPYTER FAQ

Books

Python for Signal Processing



O'Reilly Book

Probabilistic Programming

Misc

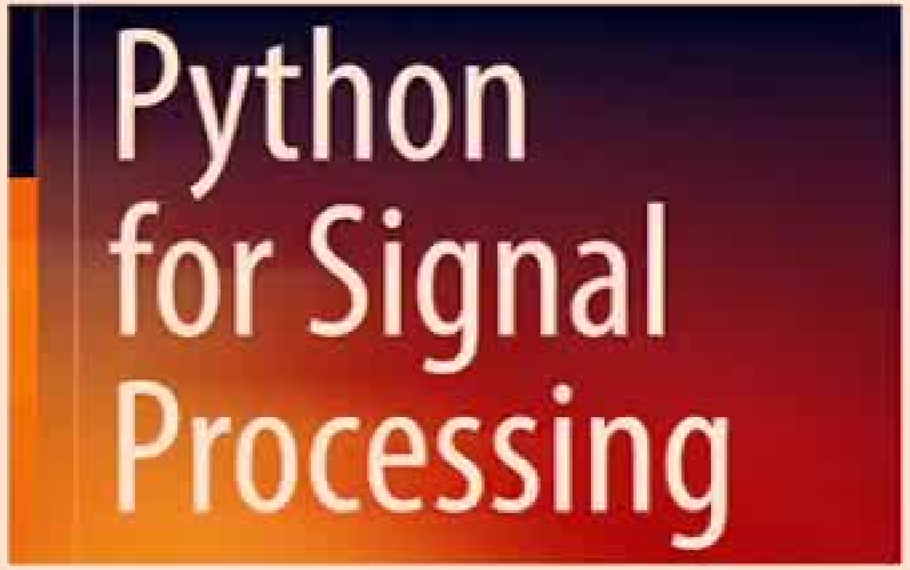
Data Visualization with Lightning

Interactive data visualization with Bokeh

Interactive plots with Plotly

Books

Python for Signal Processing

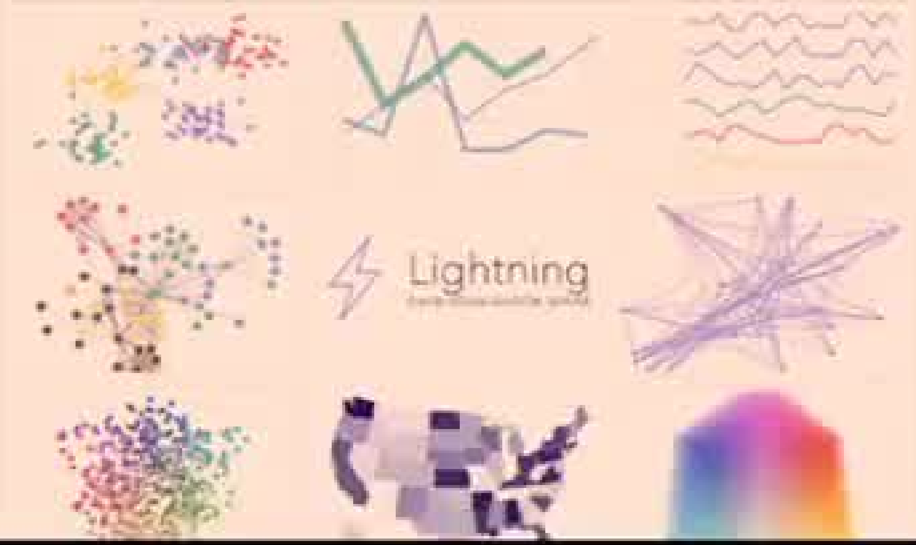


O'Reilly Book

Probabilistic Programming

Misc

Data Visualization with Lightning

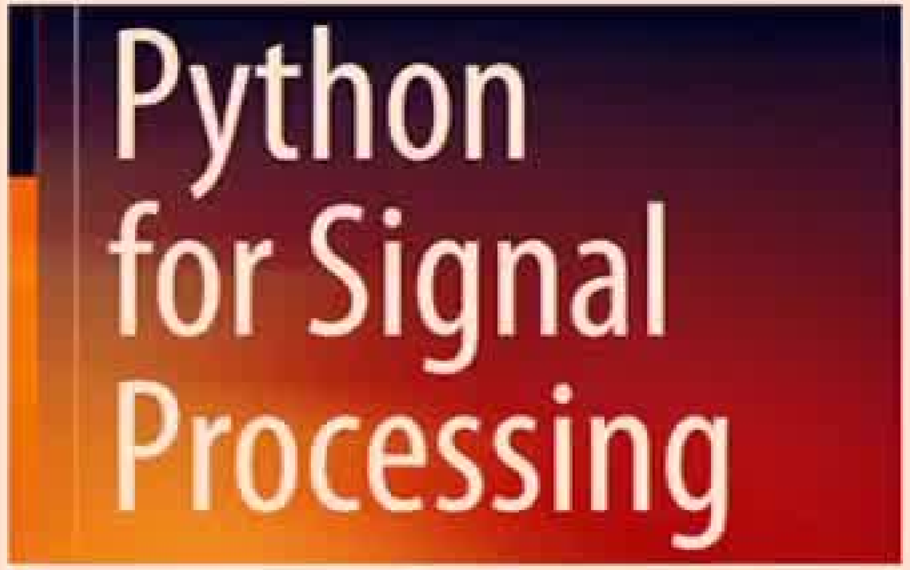


Interactive data visualization with Bokeh

Interactive plots with Plotly

Books

Python for Signal Processing



O'Reilly Book

Probabilistic Programming

Misc

Data Visualization with Lightning

Interactive data visualization with Bokeh

Interactive plots with Plotly

Probabilistic Programming

and Bayesian Methods for Hackers

Version 0.1

`Original content created by Cam Davidson-Pilon`

`Ported to Python 3 and PyMC3 by Max Margenot (@clean_utensils) and Thomas Wiecki (@twiecki) at Quantopian (@quantopian)`

Welcome to *Bayesian Methods for Hackers*. The full Github repository is available at github/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers. The other chapters can be found on the project's homepage. We hope you enjoy the book, and we encourage any contributions!

Chapter 1

The Philosophy of Bayesian Inference

> You are a skilled programmer, but bugs still slip into your code. After a particularly difficult implementation of an algorithm, you decide to test your code on a trivial example. It passes. You test the code on a harder problem. It passes once again. And it passes the next, *even more difficult*, test too! You are starting to believe that there may be no bugs in this code...

Probabilistic Programming

and Bayesian Methods for Hackers

Version 0.1

`Original content created by Cam Davidson-Pilon`

`Ported to Python 3 and PyMC3 by Max Margenot (@clean_utensils) and Thomas Wiecki (@twiecki) at Quantopian (@quantopian)`

Welcome to *Bayesian Methods for Hackers*. The full Github repository is available at github/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers. The other chapters can be found on the project's homepage. We hope you enjoy the book, and we encourage any contributions!

Chapter 1

Probabilistic Programming

and Bayesian Methods for Hackers

Version 0.1

`Original content created by Cam Davidson-Pilon`

`Ported to Python 3 and PyMC3 by Max Margenot (@clean_utensils) and Thomas Wiecki (@twiecki) at Quantopian (@quantopian)`

Welcome to *Bayesian Methods for Hackers*. The full Github repository is available at github/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers. The other chapters can be found on the project's homepage. We hope you enjoy the book, and we encourage any contributions!

Chapter 1

If you think this way, then congratulations, you already are thinking Bayesian! Bayesian inference is simply updating your beliefs after considering new evidence. A Bayesian can rarely be certain about a result, but he or she can be very confident. Just like in the example above, we can never be 100% sure that our code is bug-free unless we test it on every possible problem; something rarely possible in practice. Instead, we can test it on a large number of problems, and if it succeeds we can feel more *confident* about our code, but still not certain. Bayesian inference works identically: we update our beliefs about an outcome; rarely can we be absolutely sure unless we rule out all other alternatives.

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of

If you think this way, then congratulations, you already are thinking Bayesian! Bayesian inference is simply updating your beliefs after considering new evidence. A Bayesian can rarely be certain about a result, but he or she can be very confident. Just like in the example above, we can never be 100% sure that our code is bug-free unless we test it on every possible problem; something rarely possible in practice. Instead, we can test it on a large number of problems, and if it succeeds we can feel more *confident* about our code, but still not certain. Bayesian inference works identically: we update our beliefs about an outcome; rarely can we be absolutely sure unless we rule out all other alternatives.

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of

If you think this way, then congratulations, you already are thinking Bayesian! Bayesian inference is simply updating your beliefs after considering new evidence. A Bayesian can rarely be certain about a result, but he or she can be very confident. Just like in the example above, we can never be 100% sure that our code is bug-free unless we test it on every possible problem; something rarely possible in practice. Instead, we can test it on a large number of problems, and if it succeeds we can feel more *confident* about our code, but still not certain. Bayesian inference works identically: we update our beliefs about an outcome; rarely can we be absolutely sure unless we rule out all other alternatives.

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of

be 100% sure that our code is bug-free unless we test it on every possible problem; something rarely possible in practice. Instead, we can test it on a large number of problems, and if it succeeds we can feel more *confident* about our code, but still not certain. Bayesian inference works identically: we update our beliefs about an outcome; rarely can we be absolutely sure unless we rule out all other alternatives.

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of occurrences defines the probability.

Bayesians, on the other hand, have a more intuitive approach. Bayesians interpret a probability as measure of *belief*, or confidence, of an event occurring. Simply, a probability is a summary of

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of occurrences defines the probability.

Bayesians, on the other hand, have a more intuitive approach. Bayesians interpret a probability as measure of *belief*, or confidence, of an event occurring. Simply, a probability is a summary of an opinion. An individual who assigns a belief of 0 to an event has no confidence that the event will occur; conversely, assigning a belief of 1 implies that the individual is absolutely certain of an event occurring. Beliefs between 0 and 1 allow for weightings of other outcomes. This definition agrees with the probability of a plane accident example, for having observed the frequency of

The Bayesian state of mind

Bayesian inference differs from more traditional statistical inference by preserving *uncertainty*. At first, this sounds like a bad statistical technique. Isn't statistics all about deriving *certainty* from randomness? To reconcile this, we need to start thinking like Bayesians.

The Bayesian world-view interprets probability as measure of *believability in an event*, that is, how confident we are in an event occurring. In fact, we will see in a moment that this is the natural interpretation of probability.

For this to be clearer, we consider an alternative interpretation of probability: *Frequentist*, known as the more *classical* version of statistics, assume that probability is the long-run frequency of events (hence the bestowed title). For example, the *probability of plane accidents* under a frequentist philosophy is interpreted as the *long-term frequency of plane accidents*. This makes logical sense for many probabilities of events, but becomes more difficult to understand when events have no long-term frequency of occurrences. Consider: we often assign probabilities to outcomes of presidential elections, but the election itself only happens once! Frequentists get around this by invoking alternative realities and saying across all these realities, the frequency of occurrences defines the probability.

Bayesians, on the other hand, have a more intuitive approach. Bayesians interpret a probability as measure of *belief*, or confidence, of an event occurring. Simply, a probability is a summary of an opinion. An individual who assigns a belief of 0 to an event has no confidence that the event will occur; conversely, assigning a belief of 1 implies that the individual is absolutely certain of an event occurring. Beliefs between 0 and 1 allow for weightings of other outcomes. This definition agrees with the probability of a plane accident example, for having observed the frequency of plane accidents, an individual's belief should be equal to that frequency, excluding any outside an opinion. An individual who assigns a belief of 0 to an event has no confidence that the event will occur; conversely, assigning a belief of 1 implies that the individual is absolutely certain of an event occurring. Beliefs between 0 and 1 allow for weightings of other outcomes. This definition agrees with the probability of a plane accident example, for having observed the frequency of plane accidents, an individual's belief should be equal to that frequency, excluding any outside information. Similarly, under this definition of probability being equal to beliefs, it is meaningful to speak about probabilities (beliefs) of presidential election outcomes: how confident are you candidate *A* will win?

Notice in the paragraph above, I assigned the belief (probability) measure to an *individual*, not to Nature. This is very interesting, as this definition leaves room for conflicting beliefs between individuals. Again, this is appropriate for what naturally occurs: different individuals have different beliefs of events occurring, because they possess different *information* about the world. The existence of different beliefs does not imply that anyone is wrong. Consider the following examples demonstrating the relationship between individual beliefs and probabilities:

- I flip a coin, and we both guess the result. We would both agree, assuming the coin is fair, that the probability of Heads is 1/2. Assume, then, that I peek at the coin. Now I know for certain what the result is: I assign probability 1.0 to either Heads or Tails (whichever it is). Now what is *your* belief that the coin is Heads? My knowledge of the outcome has not changed the coin's results. Thus we assign different probabilities to the result.
- Your code either has a bug in it or not, but we do not know for certain which is true, though we have a belief about the presence or absence of a bug.
- A medical patient is exhibiting symptoms x, y and z. There are a number of diseases that could be causing all of them, but only a single disease is present. A doctor has beliefs about which disease, but a second doctor may have slightly different beliefs.

information. Similarly, under this definition of probability being equal to beliefs, it is meaningful to speak about probabilities (beliefs) of presidential election outcomes: how confident are you candidate *A* will win?

Notice in the paragraph above, I assigned the belief (probability) measure to an *individual*, not to Nature. This is very interesting, as this definition leaves room for conflicting beliefs between individuals. Again, this is appropriate for what naturally occurs: different individuals have different beliefs of events occurring, because they possess different *information* about the world. The existence of different beliefs does not imply that anyone is wrong. Consider the following examples demonstrating the relationship between individual beliefs and probabilities:

- I flip a coin, and we both guess the result. We would both agree, assuming the coin is fair, that the probability of Heads is 1/2. Assume, then, that I peek at the coin. Now I know for certain what the result is: I assign probability 1.0 to either Heads or Tails (whichever it is). Now what is *your* belief that the coin is Heads? My knowledge of the outcome has not changed the coin's results. Thus we assign different probabilities to the result.
- Your code either has a bug in it or not, but we do not know for certain which is true, though we have a belief about the presence or absence of a bug.
- A medical patient is exhibiting symptoms x, y and z. There are a number of diseases that could be causing all of them, but only a single disease is present. A doctor has beliefs about which disease, but a second doctor may have slightly different beliefs.

This philosophy of treating beliefs as probability is natural to humans. We employ it constantly as we interact with the world and only see partial truths, but gather evidence to form beliefs. Alternatively, you have to be *trained* to think like a frequentist.

To align ourselves with traditional probability notation, we denote our belief about event A as individuals. Again, this is appropriate for what naturally occurs: different individuals have different beliefs of events occurring, because they possess different *information* about the world. The existence of different beliefs does not imply that anyone is wrong. Consider the following examples demonstrating the relationship between individual beliefs and probabilities:

- I flip a coin, and we both guess the result. We would both agree, assuming the coin is fair, that the probability of Heads is 1/2. Assume, then, that I peek at the coin. Now I know for certain what the result is: I assign probability 1.0 to either Heads or Tails (whichever it is). Now what is *your* belief that the coin is Heads? My knowledge of the outcome has not changed the coin's results. Thus we assign different probabilities to the result.
- Your code either has a bug in it or not, but we do not know for certain which is true, though we have a belief about the presence or absence of a bug.
- A medical patient is exhibiting symptoms x, y and z. There are a number of diseases that could be causing all of them, but only a single disease is present. A doctor has beliefs about which disease, but a second doctor may have slightly different beliefs.

This philosophy of treating beliefs as probability is natural to humans. We employ it constantly as we interact with the world and only see partial truths, but gather evidence to form beliefs. Alternatively, you have to be *trained* to think like a frequentist.

To align ourselves with traditional probability notation, we denote our belief about event A as $P(A)$. We call this quantity the *prior probability*.

John Maynard Keynes, a great economist and thinker, said "When the facts change, I change my mind. What do you do, sir?" This quote reflects the way a Bayesian updates his or her beliefs after seeing evidence. Even — especially — if the evidence is counter to what was initially believed, the evidence cannot be ignored. We denote our updated belief as $P(A|X)$, interpreted

It's clear that in each example we did not completely discard the prior belief after seeing new evidence X, but we *re-weighted the prior* to incorporate the new evidence (i.e. we put more weight, or confidence, on some beliefs versus others).

By introducing prior uncertainty about events, we are already admitting that any guess we make is potentially very wrong. After observing data, evidence, or other information, we update our beliefs, and our guess becomes *less wrong*. This is the alternative side of the prediction coin, where typically we try to be *more right*.

Bayesian Inference in Practice

If frequentist and Bayesian inference were programming functions, with inputs being statistical problems, then the two would be different in what they return to the user. The frequentist inference function would return a number, representing an estimate (typically a summary statistic like the sample average etc.), whereas the Bayesian function would return *probabilities*.

For example, in our debugging problem above, calling the frequentist function with the argument "My code passed all X tests; is my code bug-free?" would return a *YES*. On the other hand, asking our Bayesian function "Often my code has bugs. My code passed all X tests; is my code bug-free?" would return something very different: probabilities of *YES* and *NO*. The function might return:

> *YES*, with probability 0.8; *NO*, with probability 0.2

This is very different from the answer the frequentist function returned. Notice that the Bayesian function accepted an additional argument: "*Often my code has bugs*". This parameter is the *prior*. By including the prior parameter, we are telling the Bayesian function to include our belief about this belief.

Denote N as the number of instances of evidence we possess. As we gather an *infinite* amount of evidence, say as $N \rightarrow \infty$, our Bayesian results (often) align with frequentist results. Hence for large N, statistical inference is more or less objective. On the other hand, for small N, inference is much more *unstable*: frequentist estimates have more variance and larger confidence intervals. This is where Bayesian analysis excels. By introducing a prior, and returning probabilities (instead of a scalar estimate), we *preserve the uncertainty* that reflects the instability of statistical inference of a small N dataset.

One may think that for large N, one can be indifferent between the two techniques since they offer similar inference, and might lean towards the computationally-simpler, frequentist methods. An individual in this position should consider the following quote by Andrew Gelman (2005)[1], before making such a decision:

> Sample sizes are never large. If N is too small to get a sufficiently-precise estimate, you need to get more data (or make more assumptions). But once N is "large enough," you can start subdividing the data to learn more (for example, in a public opinion poll, once you have a good estimate for the entire country, you can estimate among men and women, northerners and southerners, different age groups, etc.). N is never enough because if it were "enough" you'd already be on to the next problem for which you need more data.

Are frequentist methods incorrect then?

No.

Frequentist methods are still useful or state-of-the-art in many areas. Tools such as least squares linear regression, LASSO regression, and expectation-maximization algorithms are all powerful this belief.

Denote N as the number of instances of evidence we possess. As we gather an *infinite* amount of evidence, say as $N \rightarrow \infty$, our Bayesian results (often) align with frequentist results. Hence for large N, statistical inference is more or less objective. On the other hand, for small N, inference is much more *unstable*: frequentist estimates have more variance and larger confidence intervals. This is where Bayesian analysis excels. By introducing a prior, and returning probabilities (instead of a scalar estimate), we *preserve the uncertainty* that reflects the instability of statistical inference of a small N dataset.

One may think that for large N, one can be indifferent between the two techniques since they offer similar inference, and might lean towards the computationally-simpler, frequentist methods. An individual in this position should consider the following quote by Andrew Gelman (2005)[1], before making such a decision:

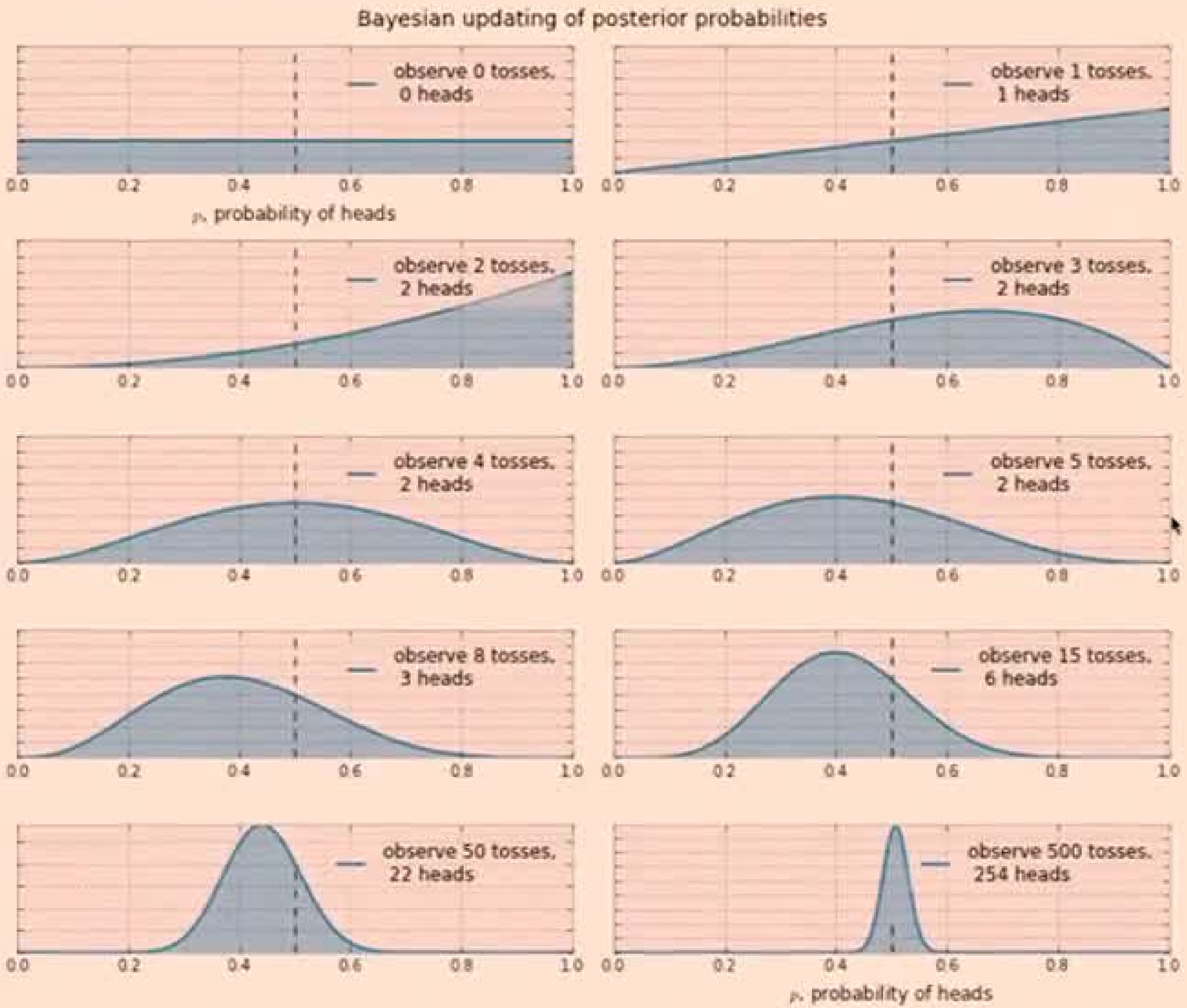
> Sample sizes are never large. If N is too small to get a sufficiently-precise estimate, you need to get more data (or make more assumptions). But once N is "large enough," you can start subdividing the data to learn more (for example, in a public opinion poll, once you have a good estimate for the entire country, you can estimate among men and women, northerners and southerners, different age groups, etc.). N is never enough because if it were "enough" you'd already be on to the next problem for which you need more data.

Are frequentist methods incorrect then?

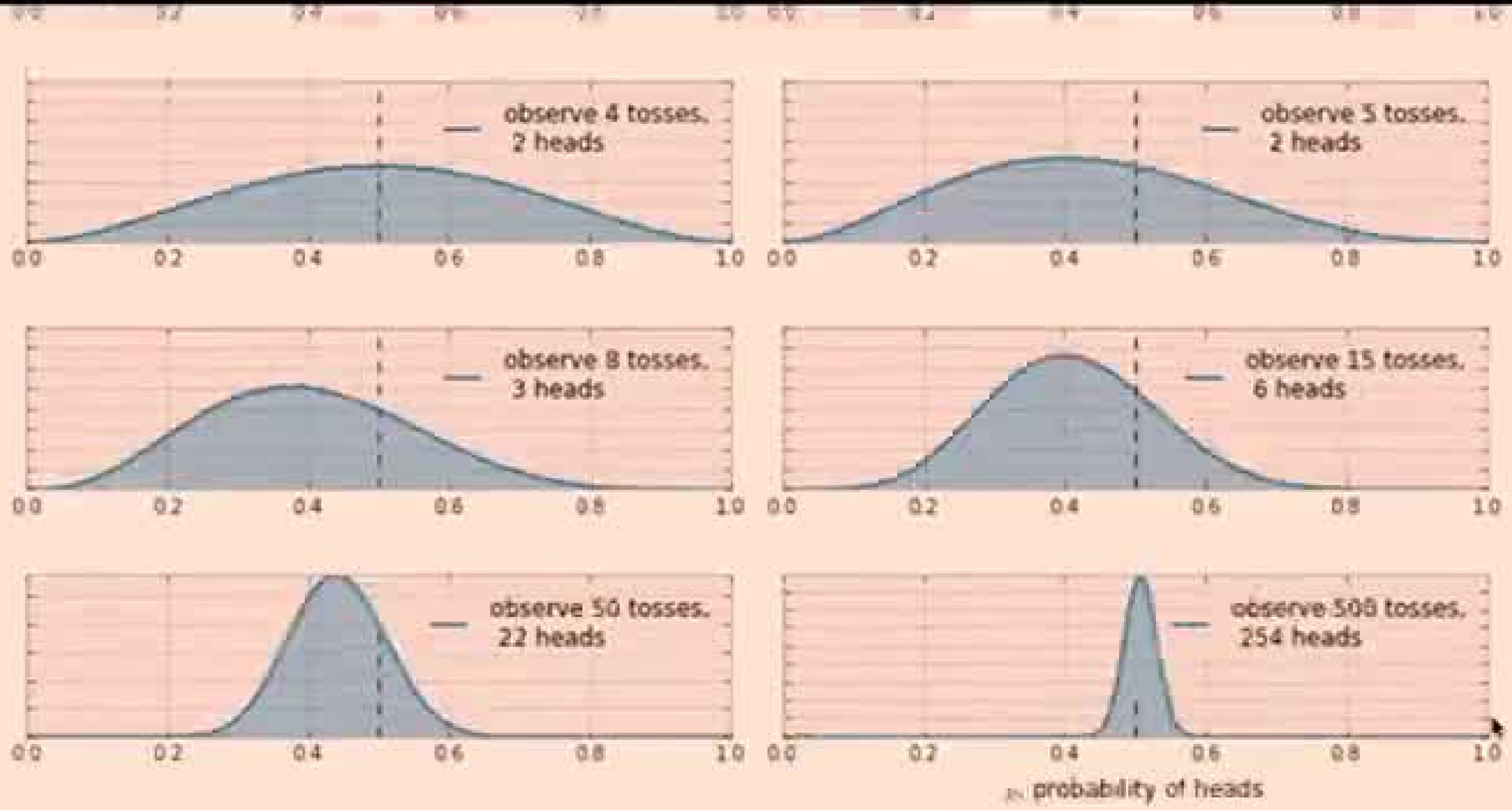
No.

Frequentist methods are still useful or state-of-the-art in many areas. Tools such as least squares linear regression, LASSO regression, and expectation-maximization algorithms are all powerful

```
plt.tight_layout()
```



The posterior probabilities are represented by the curves, and our uncertainty is proportional to the width of the curve. As the plot above shows, as we start to observe data our posterior probabilities start to shift and move around. Eventually, as we observe more and more data (coin-flips), our probabilities will tighten closer and closer around the true value of $p = 0.5$ (marked by



The posterior probabilities are represented by the curves, and our uncertainty is proportional to the width of the curve. As the plot above shows, as we start to observe data our posterior probabilities start to shift and move around. Eventually, as we observe more and more data (coin-flips), our probabilities will tighten closer and closer around the true value of $p = 0.5$ (marked by a dashed line).

Notice that the plots are not always *peaked* at 0.5. There is no reason it should be: recall we assumed we did not have a prior opinion of what p is. In fact, if we observe quite extreme data, say 8 flips and only 1 observed heads, our distribution would look very biased *away* from lumping around 0.5 (with no prior opinion, how confident would you feel betting on a fair coin after observing 8 tails and 1 head). As more data accumulates, we would see more and more probability being assigned at $p = 0.5$, though never all of it.

The next example is a simple demonstration of the mathematics of Bayesian inference.

The posterior probabilities are represented by the curves, and our uncertainty is proportional to the width of the curve. As the plot above shows, as we start to observe data our posterior probabilities start to shift and move around. Eventually, as we observe more and more data (coin-flips), our probabilities will tighten closer and closer around the true value of $p = 0.5$ (marked by a dashed line).

Notice that the plots are not always *peaked* at 0.5. There is no reason it should be: recall we assumed we did not have a prior opinion of what p is. In fact, if we observe quite extreme data, say 8 flips and only 1 observed heads, our distribution would look very biased *away* from lumping around 0.5 (with no prior opinion, how confident would you feel betting on a fair coin after observing 8 tails and 1 head). As more data accumulates, we would see more and more probability being assigned at $p = 0.5$, though never all of it.

The next example is a simple demonstration of the mathematics of Bayesian inference.

Example: Bug, or just sweet, unintended feature?

Let A denote the event that our code has **no bugs** in it. Let X denote the event that the code passes all debugging tests. For now, we will leave the prior probability of no bugs as a variable, i.e. $P(A) = p$.

We are interested in $P(A|X)$, i.e. the probability of no bugs, given our debugging tests X. To use the formula above, we need to compute some quantities.

What is $P(X|A)$, i.e., the probability that the code passes X tests *given* there are no bugs? Well, it is equal to 1, for a code with no bugs will pass all tests.

observing 8 tails and 1 head). As more data accumulates, we would see more and more probability being assigned at $p = 0.5$, though never all of it.

The next example is a simple demonstration of the mathematics of Bayesian inference.

Example: Bug, or just sweet, unintended feature? ¶

Let A denote the event that our code has **no bugs** in it. Let X denote the event that the code passes all debugging tests. For now, we will leave the prior probability of no bugs as a variable, i.e. $P(A) = p$.

We are interested in $P(A|X)$, i.e. the probability of no bugs, given our debugging tests X. To use the formula above, we need to compute some quantities.

What is $P(X|A)$, i.e., the probability that the code passes X tests *given* there are no bugs? Well, it is equal to 1, for a code with no bugs will pass all tests.

$P(X)$ is a little bit trickier: The event X can be divided into two possibilities, event X occurring even though our code *indeed has* bugs (denoted $\sim A$, spoken *not* A), or event X without bugs (A). $P(X)$ can be represented as:

$$P(X) = P(X \text{ and } A) + P(X \text{ and } \sim A) \tag{4}$$

$$\tag{5}$$

$$= P(X|A)P(A) + P(X| \sim A)P(\sim A) \tag{6}$$

$$\tag{7}$$

$$= P(X|A)p + P(X| \sim A)(1 - p) \tag{8}$$

Let A denote the event that our code has **no bugs** in it. Let X denote the event that the code passes all debugging tests. For now, we will leave the prior probability of no bugs as a variable, i.e. $P(A) = p$.

We are interested in $P(A|X)$, i.e. the probability of no bugs, given our debugging tests X. To use the formula above, we need to compute some quantities.

What is $P(X|A)$, i.e., the probability that the code passes X tests *given* there are no bugs? Well, it is equal to 1, for a code with no bugs will pass all tests.

$P(X)$ is a little bit trickier: The event X can be divided into two possibilities, event X occurring even though our code *indeed has* bugs (denoted $\sim A$, spoken *not* A), or event X without bugs (A). $P(X)$ can be represented as:

$$P(X) = P(X \text{ and } A) + P(X \text{ and } \sim A) \tag{4}$$
$$\tag{5}$$
$$= P(X|A)P(A) + P(X| \sim A)P(\sim A) \tag{6}$$
$$\tag{7}$$
$$= P(X|A)p + P(X| \sim A)(1 - p) \tag{8}$$

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

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1 - p)} \tag{9}$$

passes all debugging tests. For now, we will leave the prior probability of no bugs as a variable, i.e. $P(A) = p$.

We are interested in $P(A|X)$, i.e. the probability of no bugs, given our debugging tests X. To use the formula above, we need to compute some quantities.

What is $P(X|A)$, i.e., the probability that the code passes X tests *given* there are no bugs? Well, it is equal to 1, for a code with no bugs will pass all tests.

$P(X)$ is a little bit trickier: The event X can be divided into two possibilities, event X occurring even though our code *indeed has* bugs (denoted $\sim A$, spoken *not* A), or event X without bugs (A). $P(X)$ can be represented as:

$$P(X) = P(X \text{ and } A) + P(X \text{ and } \sim A) \tag{4}$$

$$\tag{5}$$

$$= P(X|A)P(A) + P(X| \sim A)P(\sim A) \tag{6}$$

$$\tag{7}$$

$$= P(X|A)p + P(X| \sim A)(1 - p) \tag{8}$$

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

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1 - p)} \tag{9}$$

$$\tag{10}$$

passes all debugging tests. For now, we will leave the prior probability of no bugs as a variable, i.e. $P(A) = p$.

We are interested in $P(A|X)$, i.e. the probability of no bugs, given our debugging tests X. To use the formula above, we need to compute some quantities.

What is $P(X|A)$, i.e., the probability that the code passes X tests *given* there are no bugs? Well, it is equal to 1, for a code with no bugs will pass all tests.

$P(X)$ is a little bit trickier: The event X can be divided into two possibilities, event X occurring even though our code *indeed has* bugs (denoted $\sim A$, spoken *not* A), or event X without bugs (A). $P(X)$ can be represented as:

$$P(X) = P(X \text{ and } A) + P(X \text{ and } \sim A) \tag{4}$$

$$\tag{5}$$

$$= P(X|A)P(A) + P(X| \sim A)P(\sim A) \tag{6}$$

$$\tag{7}$$

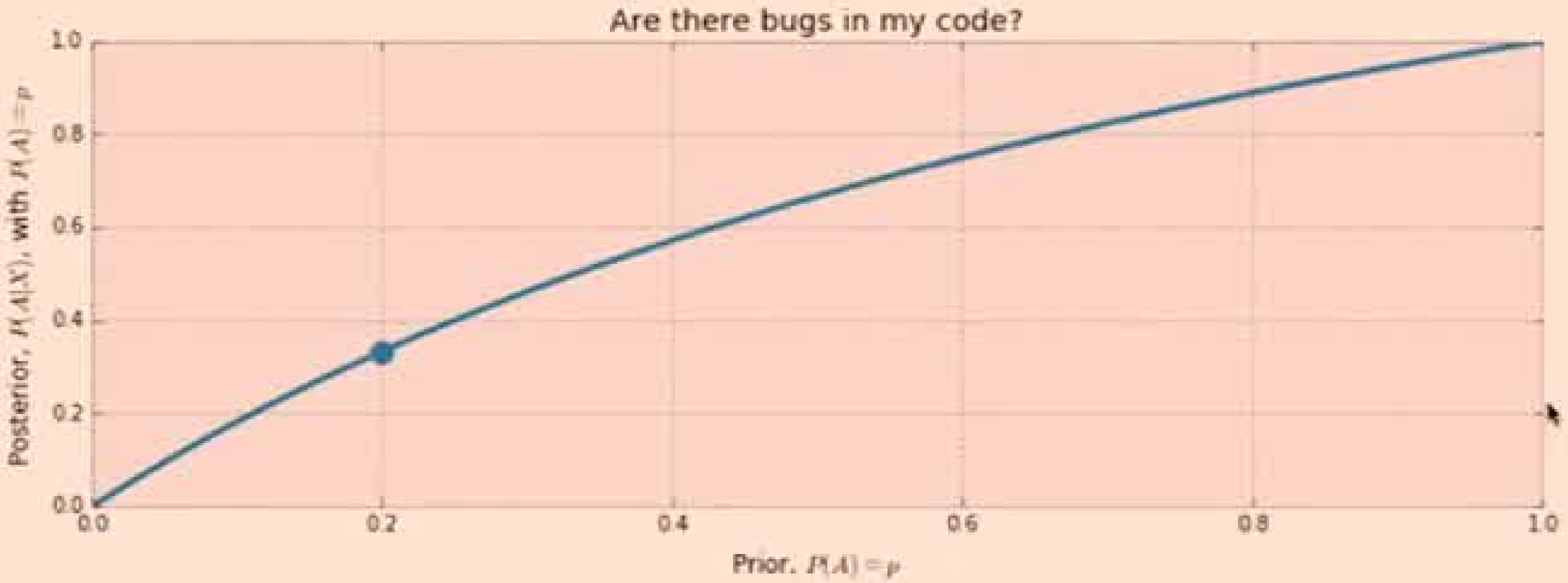
$$= P(X|A)p + P(X| \sim A)(1 - p) \tag{8}$$

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

$$P(A|X) = \frac{1 \cdot p}{1 \cdot p + 0.5(1 - p)} \tag{9}$$

$$\tag{10}$$

```
plt.plot(p, 2*p/(1+p), color="#348ABD", lw=3)
#plt.fill_between(p, 2*p/(1+p), alpha=.5, facecolor=["#A60628"])
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?");
```

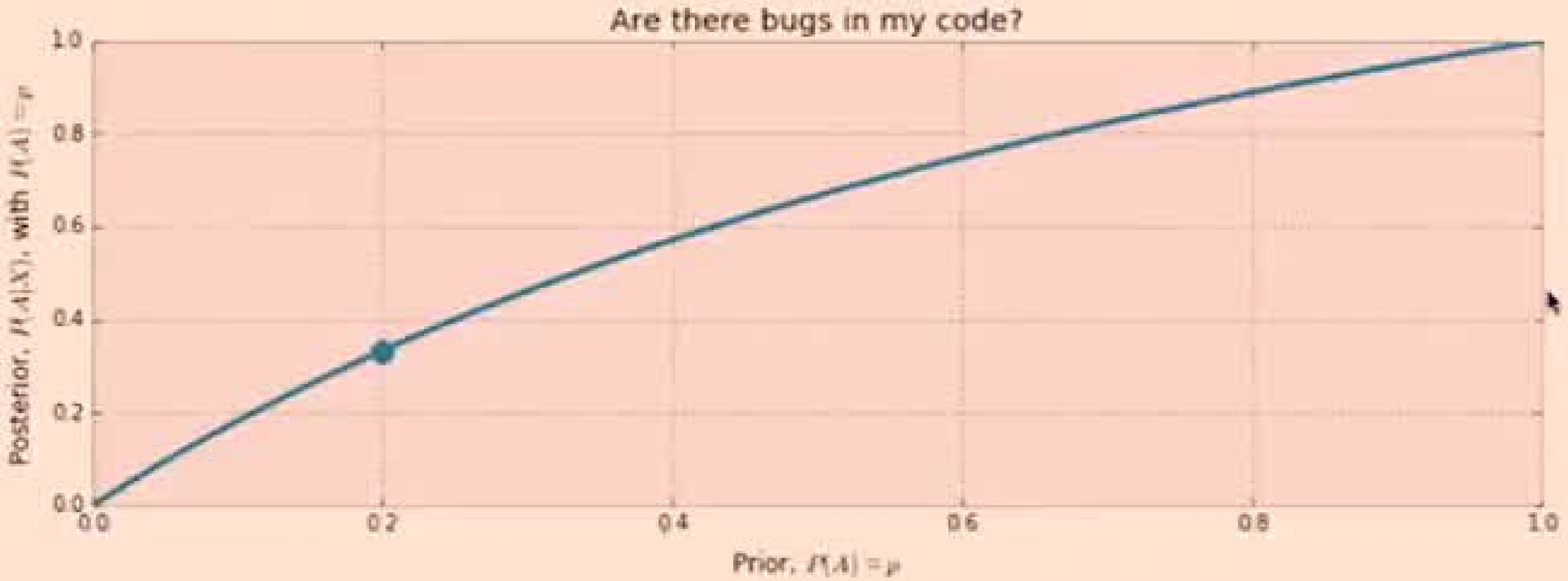


We can see the biggest gains if we observe the X tests passed when the prior probability, p, is low. Let's settle on a specific value for the prior. I'm a strong programmer (I think), so I'm going to give myself a realistic prior of 0.20, that is, there is a 20% chance that I write code bug-free. To be more realistic, this prior should be a function of how complicated and large the code is, but let's pin it at 0.20. Then my updated belief that my code is bug-free is 0.33.

Recall that the prior is a probability: p is the prior probability that there *are no bugs*, so $1 - p$ is the prior probability that there *are bugs*.

Similarly, our posterior is also a probability, with $P(A|X)$ the probability there is no bug *given we saw all tests pass*, hence $1 - P(A|X)$ is the probability there is a bug *given all tests passed*. What does our posterior probability look like? Below is a chart of both the prior and the posterior

```
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=["#A60628"])
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?");
```



We can see the biggest gains if we observe the X tests passed when the prior probability, p, is low. Let's settle on a specific value for the prior. I'm a strong programmer (I think), so I'm going to give myself a realistic prior of 0.20, that is, there is a 20% chance that I write code bug-free. To be more realistic, this prior should be a function of how complicated and large the code is, but let's pin it at 0.20. Then my updated belief that my code is bug-free is 0.33.

Recall that the prior is a probability: p is the prior probability that there *are no bugs*, so $1 - p$ is the prior probability that there *are bugs*.

Similarly, our posterior is also a probability, with $P(A|X)$ the probability there is no bug *given we*

Kernel ready | Saving every 120s | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

Notebooks are still at: https://github.com/minrk/ipython-cse17

Or follow along on JupyterHub at https://cse17.jupyter.org

Profiling and Optimising

IPython provides some tools for making it a bit easier to profile and optimise your code.

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

In [2]:
```
try:
    import seaborn as sns
```

Notebooks are still at: https://github.com/minrk/ipython-cse17

Or follow along on JupyterHub at https://cse17.jupyter.org

Profiling and Optimising

IPython provides some tools for making it a bit easier to profile and optimise your code.

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

In [2]:
```
try:
    import seaborn as sns
except ImportError:
    print("That's okay")
```

```
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [13]:
```
!open nbval.pdf
```

In [ ]:

```
# 1. Running a notebook manually will likely change the output
```

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [13]:
```
!open nbval.pdf
```

In [ ]:

Counting Words.ipynb	Running	an hour ago
DAG Dependencies.ipynb		an hour ago
MC Options.ipynb		an hour ago
memmap.ipynb		an hour ago
Monitoring MPI.ipynb	Running	an hour ago
MPI Broadcast.ipynb	Running	an hour ago
Parallel face detection.ipynb		an hour ago
Parallel image processing.ipynb	Running	an hour ago
cathat.txt		an hour ago
davinci.txt		an hour ago
davinci0.txt		an hour ago
davinci1.txt		an hour ago
davinci2.txt		an hour ago
davinci3.txt		an hour ago
davinci4.txt		an hour ago
davinci5.txt		an hour ago
davinci6.txt		an hour ago
davinci7.txt		an hour ago
davinci8.txt		an hour ago
davinci9.txt		an hour ago
gutenberg.zip		an hour ago

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

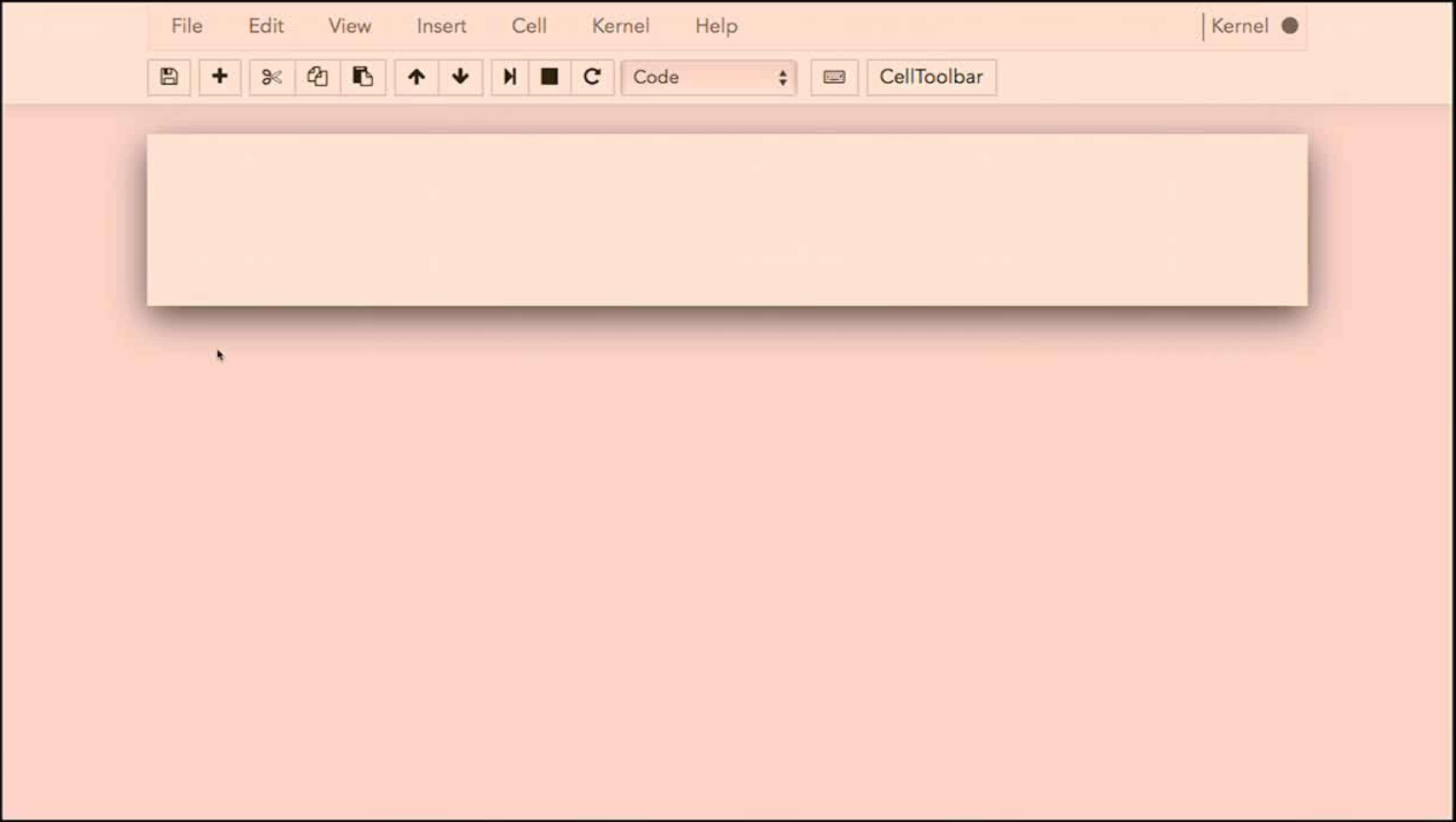
For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

Optional dependencies: I will use NetworkX for one demo, and **`scikit-image`** for another, but they are not critical. Both packages are in in Anaconda.

For the image-related demos, all you need are some images on your computer. The notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline


File
Edit
View
Insert
Cell
Kernel
Help
Kernel
Code
CellToolbar

Saving every 120s | Not Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

IPython Notebook Validation for py.test - Documentation

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

One of the powerful uses of the IPython notebook is for documentation purposes, here we use a notebook to demonstrate the behaviour and usage of the IPython Notebook Validation plugin for py.test. The IPython notebook format `.ipynb` stores outputs as well as inputs. Validating the notebook means to rerun the notebook and make sure that it is generating the same output as has been stored.

Therefore, the **user MUST make the following the distinction**:

1. Running a notebook manually will likely change the output stored in the associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

associated .ipynb file. These outputs will be used as references for the tests (i.e. the outputs from the last time you ran the notebook)
2. Validating with py.test plugin - these tests run your notebook code seperately without storing the information, the outputs generated will be compared against those in the .ipynb file

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

The purpose of the testing module is to ensure that the notebook is behaving as expected and that changes to underlying source code, haven't affected the results of an IPython notebook. For example, for documentation purposes - such as this.

Command line usage

The py.test program doesn't usually collect notebooks for testing; by passing the `--nbval` flag at the command line, the IPython Notebook Validation plugin will collect and test notebook cells, comparing their outputs with those saved in the file.

```
$ py.test --nbval my_notebook.ipynb
```

There is also an option `--nbval-lax`, which collects notebooks and runs them, failing if there is an error. This mode does not check the output of cells unless they are marked with a special `#NBVAL_CHECK_OUTPUT` comment.

```
$ py.test --nbval-lax my_notebook.ipynb
```

REGEX Output sanitizing

Since all output is captured by the IPython notebook, some pesky messages and prompts (with time-stamped messages, for example) may fail tests always, which

Saving every 30s Not Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help

```
Overwriting doc_sanitize.cfg
```

The first replacement finds dates in the given format replaces them with the label 'DATE-STAMP', likewise for strings that look like time. These will prevent the tests from failing due to time differences.

Validate this notebook

You can validate this notebook yourself, as shown below; the outputs that you see here are stored in the ipynb file. If your system produces different outputs, the testing process will fail. Just use the following commands:

```
$ cd /path/to/this/notebook
$ py.test --nbval nbval.ipynb --sanitize-with doc_saniti
ze.cfg
```

Examples of plugin behaviour

The following examples demonstrate how the plugin behaves during testing. Test this notebook yourself to see the validation in action!

```python
# NBVAL_SKIP
print("Entering infinite loop...")
while True:
    pass
```

Checking exceptions

Sometimes, we might want to allow a notebook cell to raise an exception, and check that the traceback is as we expect. By annotating the cell with the comment **# NBVAL_RAISES_EXCEPTION** you can indicate that the cell is expected to raise an exception. The full traceback is not compared, but rather just that the raised exception is the same as the stored exception.

In [3]:

```python
# NBVAL_RAISES_EXCEPTION
print("This exception will be tested")
raise RuntimeError("Foo")
```

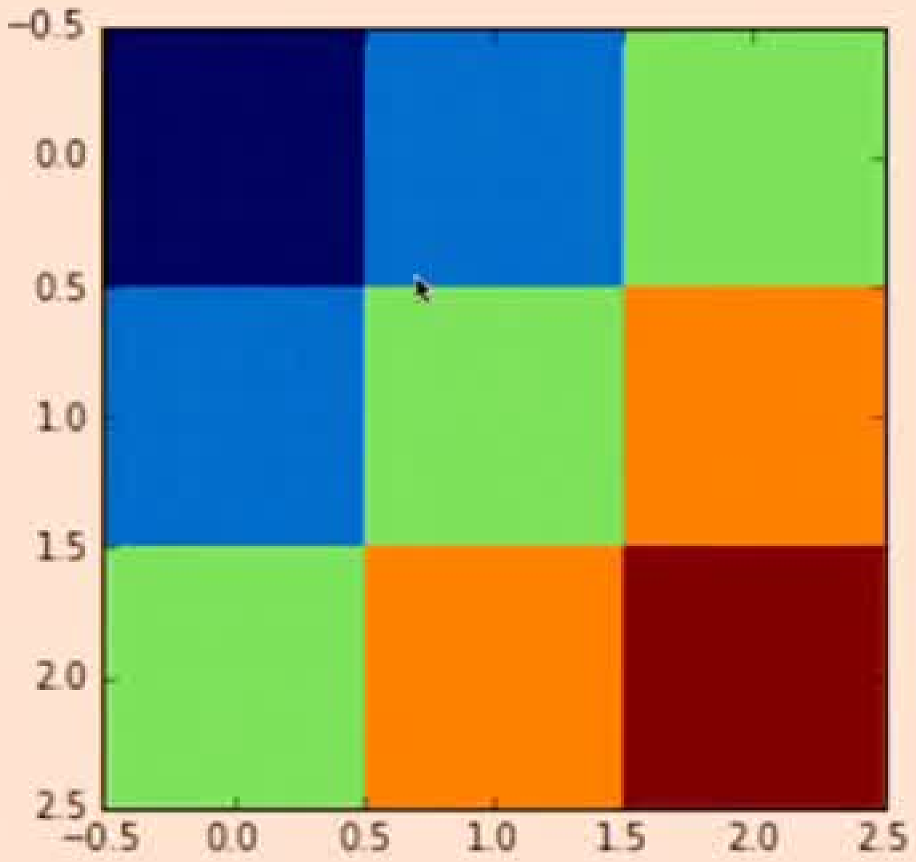
```
This exception will be tested

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recen
t call last)
<ipython-input-1-b97c0d501d6a> in <module>()
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

possible to modify the plugin to allow comparison of the image whole string.

In [10]:
```
plt.imshow(np.array([[i + j for i in range(3)]
                     for j in range(3)]),
           interpolation='None'
           )
```

Out[10]: `<matplotlib.image.AxesImage at 0x7f2cb3374198>`



In [ ]:

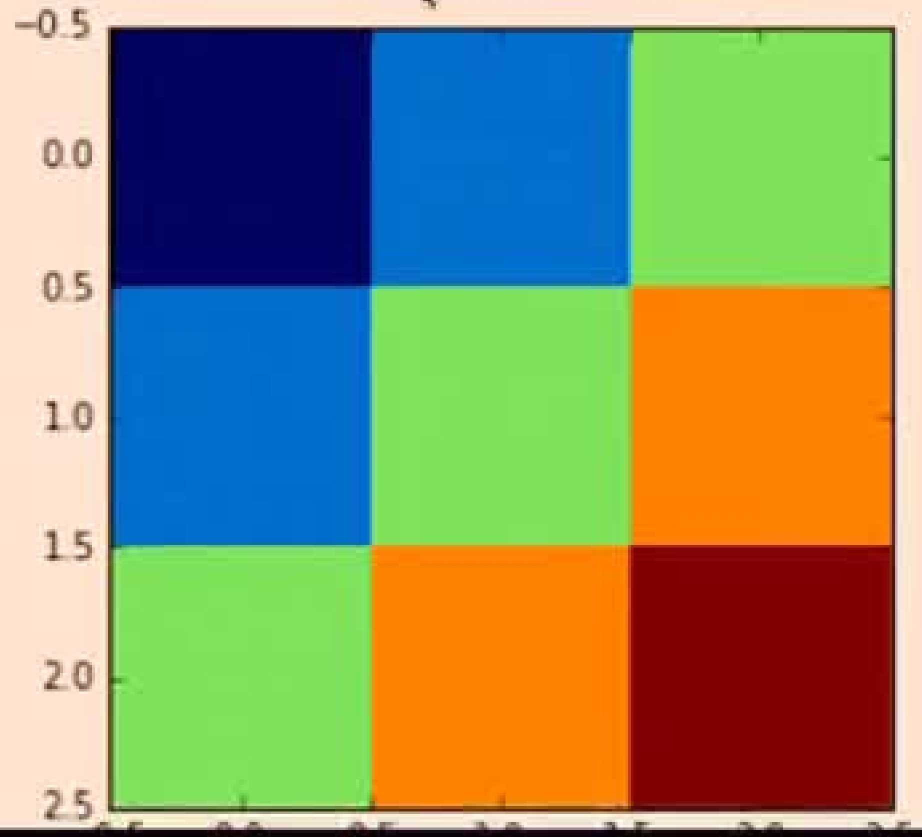
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

In [9]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

Currently, only the matplotlib text output of the Figure is compared, but it is possible to modify the plugin to allow comparison of the image whole string.

In [10]:
```
plt.imshow(np.array([[i + j for i in range(3)]
                     for j in range(3)]),
           interpolation='None'
           )
```

Out[10]: <matplotlib.image.AxesImage at 0x7f2cb3374198>



File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
raise SyntaxError("Foo")

Traceback: mismatch 'ename'
<<<<<<<<<<<< Reference output from ipynb file:
RuntimeError
============ disagrees with newly computed (test) output:
SyntaxError
>>>>>>>>>>>>
________________________________ cell 12 __________________
_______________
Notebook cell execution failed
Cell 12: Cell outputs differ

Input:
plt.imshow(np.array([[i + j for i in range(3)]
                     for j in range(3)]),
           interpolation='None'
           )

Traceback: mismatch 'text/plain'
<<<<<<<<<<<< Reference output from ipynb file:
<matplotlib.image.AxesImage at 0x7f2cb3374198><matplotlib.figur
e.Figure at 0x7f2cb564a550>
============ disagrees with newly computed (test) output:
<matplotlib.image.AxesImage at 0x116e547f0><matplotlib.figure.F
igure at 0x116d4cfd0>
>>>>>>>>>>>>
==================== 4 failed, 10 passed in 3.08 seconds =====
```

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3

```
______________

Notebook cell execution failed
Cell 3: Cell outputs differ

Input:
print([np.random.rand() for i in range(4)])
print([np.random.rand() for i in range(4)])

Traceback: mismatch 'text'
<<<<<<<<<<<< Reference output from ipynb file:
[0.36133679016382714, 0.5043774697891126, 0.23281910875007927,
0.271306551312868з]
[0.551242127798532, 0.02592706358897756, 0.05036036771084684,
0.7515926759190724]

============ disagrees with newly computed (test) output:
[0.12348198835905, 0.8191013186977266, 0.6102236226498696, 0.60
99335157815623]
[0.7834888338674239, 0.4372787069426135, 0.06627007941476448, 0
.716097615524O207]

>>>>>>>>>>>>
__________________________________ cell 4 __________________

______________

Notebook cell execution failed
Cell 4: Cell outputs differ

Input:
```

Favorites

G

DL

save

IP

ZFS

nd

Frequently Visited

Inbox (4) - benjaminrk@g...

nbdime – diffing and merging of Jupyter Notebooks

Version: 0.3.0.dev

nbdime provides tools for diffing and merging Jupyter notebooks.

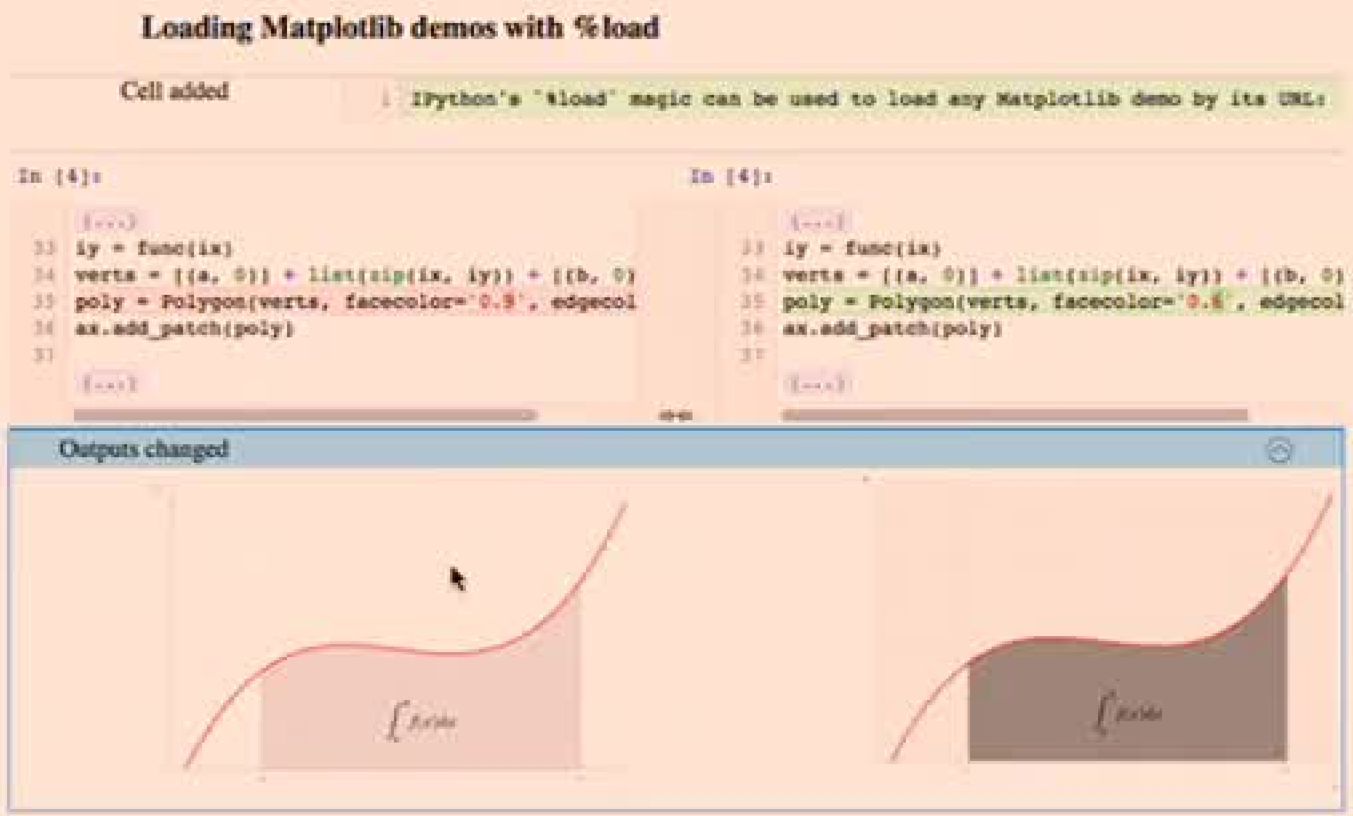


Figure: nbdime example

Why is nbdime needed?

Jupyter notebooks are useful, rich media documents stored in a plain text JSON format. This format is relatively easy to parse. However, primitive line-based diff and merge tools do not handle well the logical structure of notebook documents. These tools yield diffs like this:

```
$ diff a.ipynb b.ipynb
76,77d75
<     "plt.rc('axes', grid=False)\n",
<     "plt.rc('axes', facecolor='white')\n",
```

nbdime – diffing and merging of Jupyter Notebooks

Version: 0.3.0.dev

nbdime provides tools for diffing and merging Jupyter notebooks.

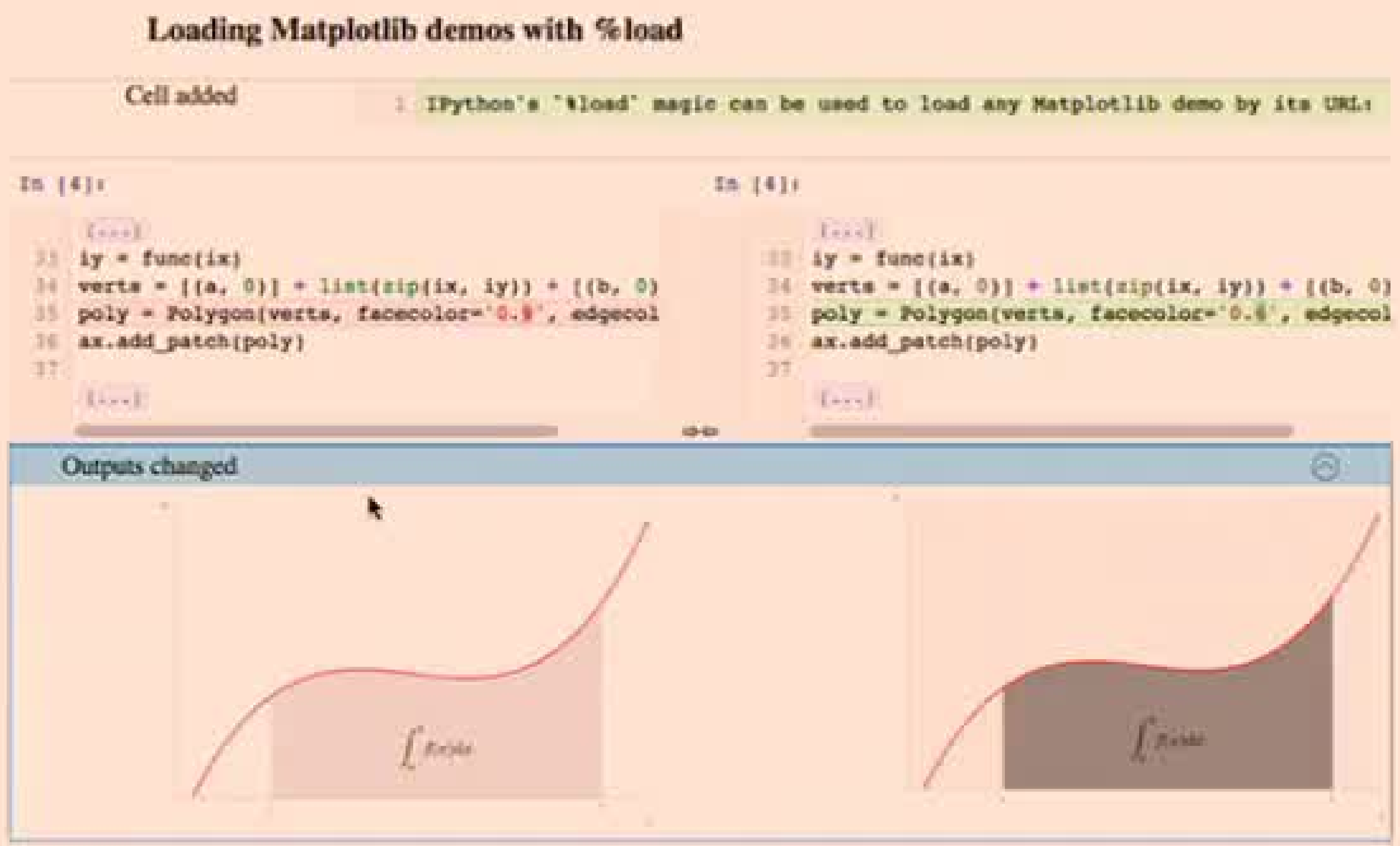


Figure: nbdime example

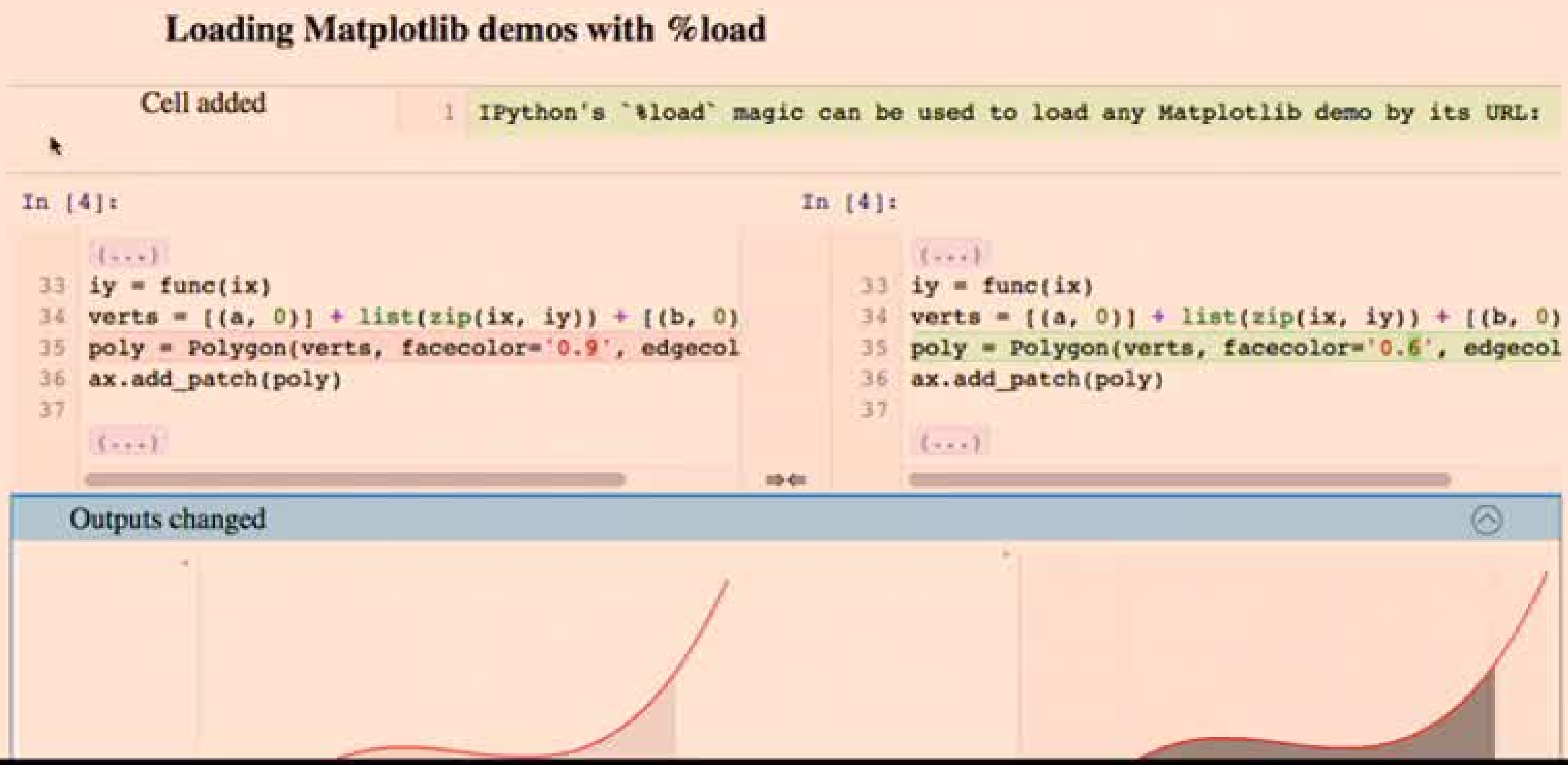
Why is nbdime needed?

Jupyter notebooks are useful, rich media documents stored in a plain text JSON format. This format is relatively easy to parse. However, primitive line-based diff and merge tools do not handle well the logical structure of notebook documents. These tools yield diffs like this:

nbdime – diffing and merging of Jupyter Notebooks

Version: 0.3.0.dev

nbdime provides tools for diffing and merging Jupyter notebooks.



nbdime – diffing and merging of Jupyter Notebooks

Version: 0.3.0.dev

nbdime provides tools for diffing and merging Jupyter notebooks.

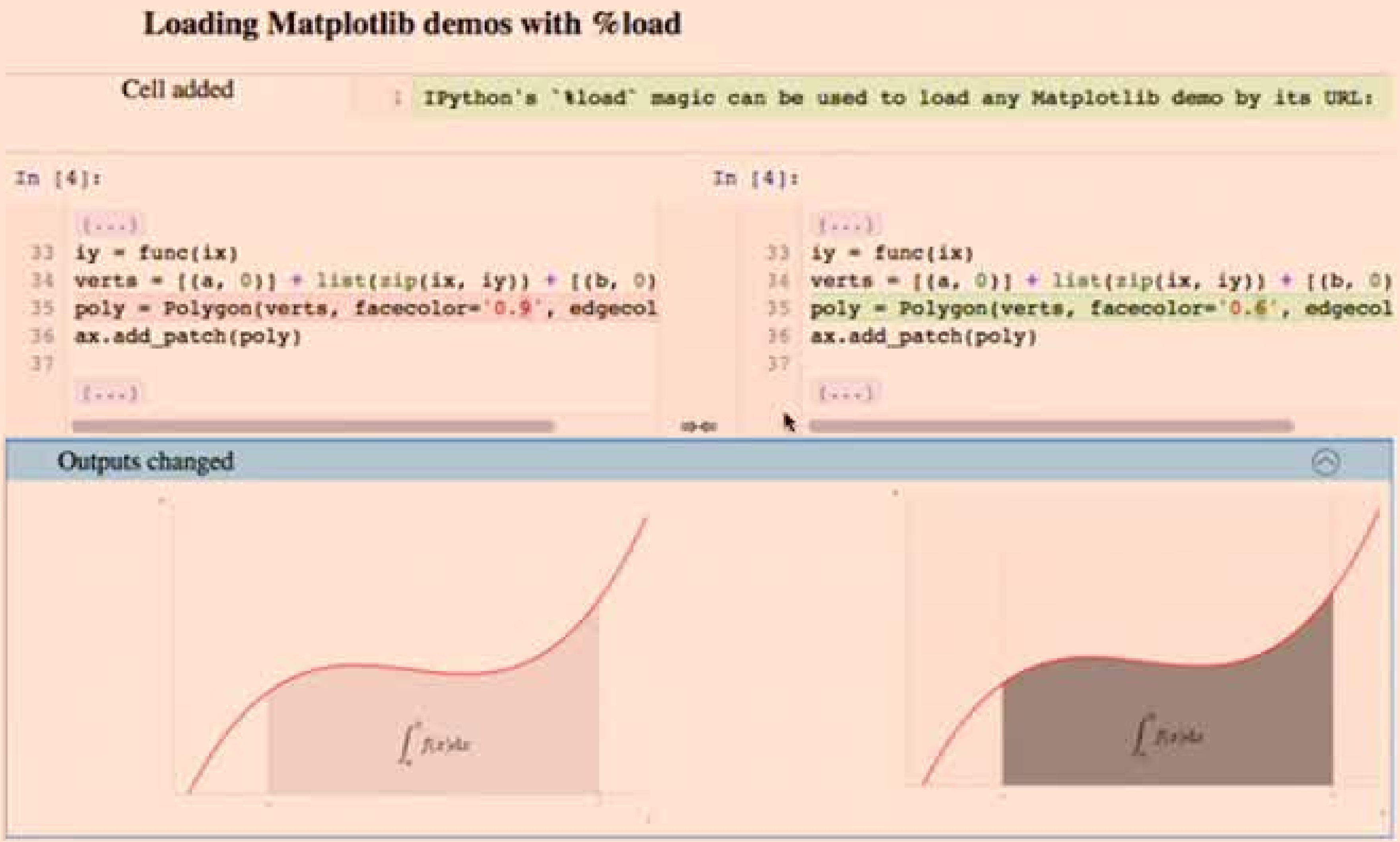


Figure: nbdime example

Why is nbdime needed?

iTerm2 Shell Edit View Profiles Toolbelt Window Help

IPython: Users/minrk... ⌘5

```
images.zip        images_common.py
minrk[14:15]~/dev/jpy/pres/ipython-cse17 (master) $ git rm parallel/images.zip
error: the following file has changes staged in the index:
    parallel/images.zip
(use --cached to keep the file, or -f to force removal)
minrk[14:15]~/dev/jpy/pres/ipython-cse17 (master) $ git rm parallel/images.zip
minrk[14:16]~/dev/jpy/pres/ipython-cse17 (master) $ gitgit
minrk[14:16]~/dev/jpy/pres/ipython-cse17 (master) $ git push
Counting objects: 89, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (88/88), done.
Writing objects: 100% (89/89), 10.94 MiB | 391.00 KiB/s, done.
Total 89 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local objects.
To github.com:minrk/ipython-cse17.git
   1a6a644..16a9059  master -> master
minrk[14:17]~/dev/jpy/pres/ipython-cse17 (master) $ ls
Beyond Plain Python.ipynb        Notebook Basics.ipynb            Profiling and Optimizing with IPython.ipynb doc_sanitize.cfg        mod.py                 parallel
Cell Magics.ipynb                Notebook file format.html        README.md                                   foo.py                  nbconvert_templates    profileme.py
Custom Display Logic.ipynb       Notebook file format.ipynb       Third Party Rich Output.ipynb               images                  nbval.ipynb            test.txt
Input in the Notebook.ipynb      Notebook file format.py          __pycache__                                 ipython-cse17.tar.gz    nbval.pdf              widgets
LICENSE                          Plotting in the Notebook.ipynb   data.csv                                    log.py                  nbval.py
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $ du -hs Plotting\ in\ the\ Notebook.ipynb
288K    Plotting in the Notebook.ipynb
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $
```

```
RuntimeError                              Traceback (most recent call last)

<ipython-input-3-32dcc1c70a4e> in <module>()
      1 # NBVAL_RAISES_EXCEPTION
      2 print("If the raised exception doesn't match the stored exception, we get a failure"
----> 3 raise RuntimeError("Foo")


RuntimeError: Foo


In [2]: # NBVAL_IGNORE_OUTPUT
```

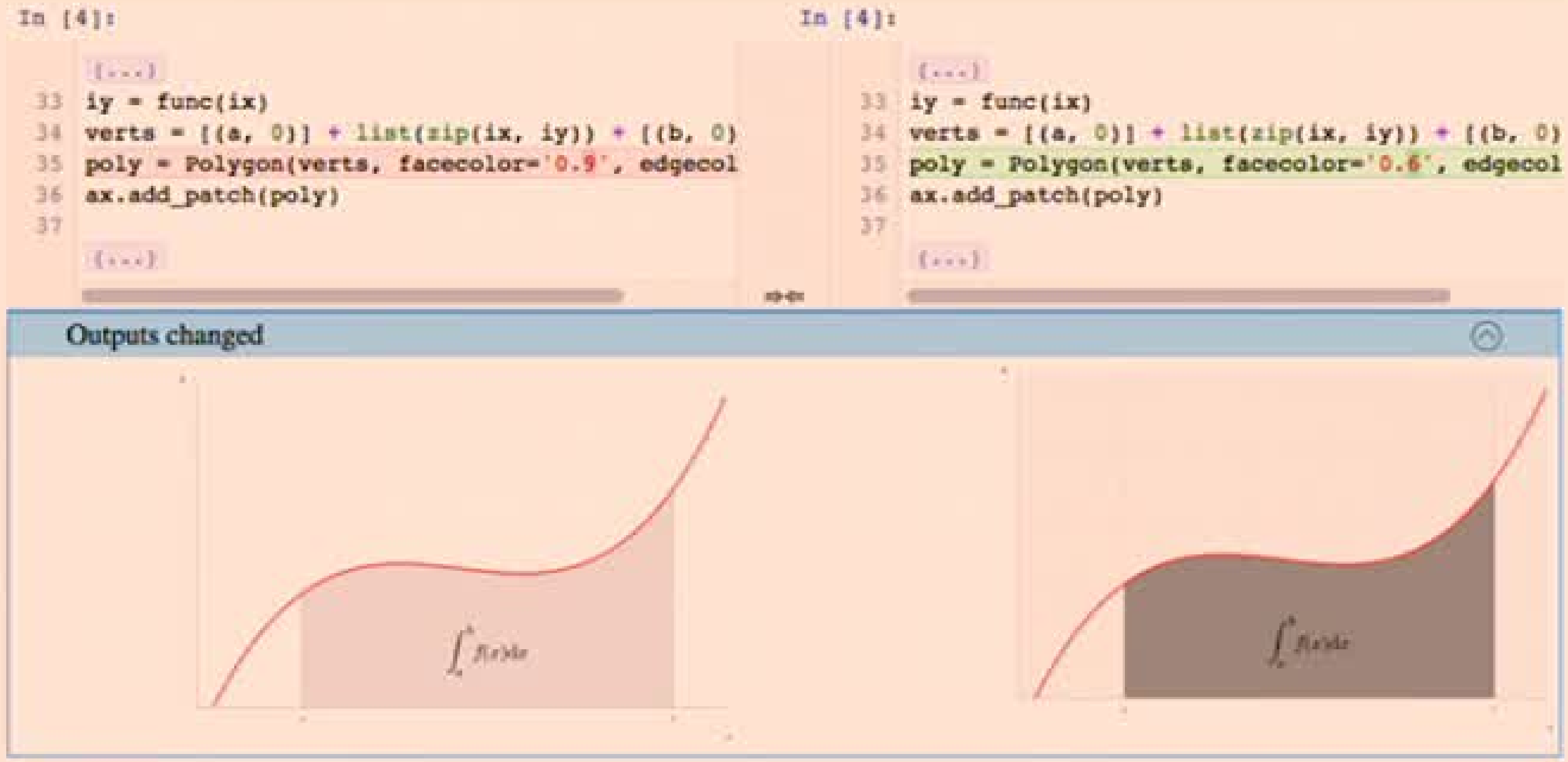


Figure: nbdime's content-aware diff

Quickstart

To get started with nbdime, install with pip:

```
pip install nbdime
```

And you can be off to the races by diffing notebooks in your terminal with **nbdiff**:

```
nbdiff notebook_1.ipynb notebook_2.ipynb
```

or viewing a rich web-based rendering of the diff with **nbdiff-web**:

Console commands

nbdime provides the following CLI commands:

```
nbshow
nbdiff
nbdiff-web
nbmerge
nbmerge-web
mergetool
config-git
```

Pass `--help` to each command to see help text for the command's usage.

Additional commands are available for Git integration.

nbshow

nbshow gives you a nice, terminal-optimized summary view of a notebook. You can use it to quickly peek at notebooks without launching the full notebook web application.

```
$ nbshow -s -o c.ipynb
markdown cell 0:
  source:
    # Plotting with Matplotlib

    IPython works with the [Matplotlib](http://matplotlib.org/) plotting library,
```

nbmerge merges two notebooks with a common parent. If there are conflicts, they are stored in metadata of the destination file. **nbmerge** will exit with nonzero status if there are any unresolved conflicts.

nbmerge writes the output to `stdout` by default, so you can use pipes to send the result to a file, or the `-o, --output` argument to specify a file in which to save the merged notebook.

Because there are several categories of data in a notebook (such as input, output, and metadata), nbmerge has several ways to deal with conflicts, and can take different actions based on the type of data with the conflict.

Important

Conflict-resolution in nbmerge is under active development and is subject to change.

The `-m, --merge-strategy` option lets you select a global strategy to use. The following options are currently implemented:

inline
: This is the default. Conflicts in input and output are recorded with conflict markers, while conflicts on metadata are stored in the appropriate metadata (actual values are kept as their base values).

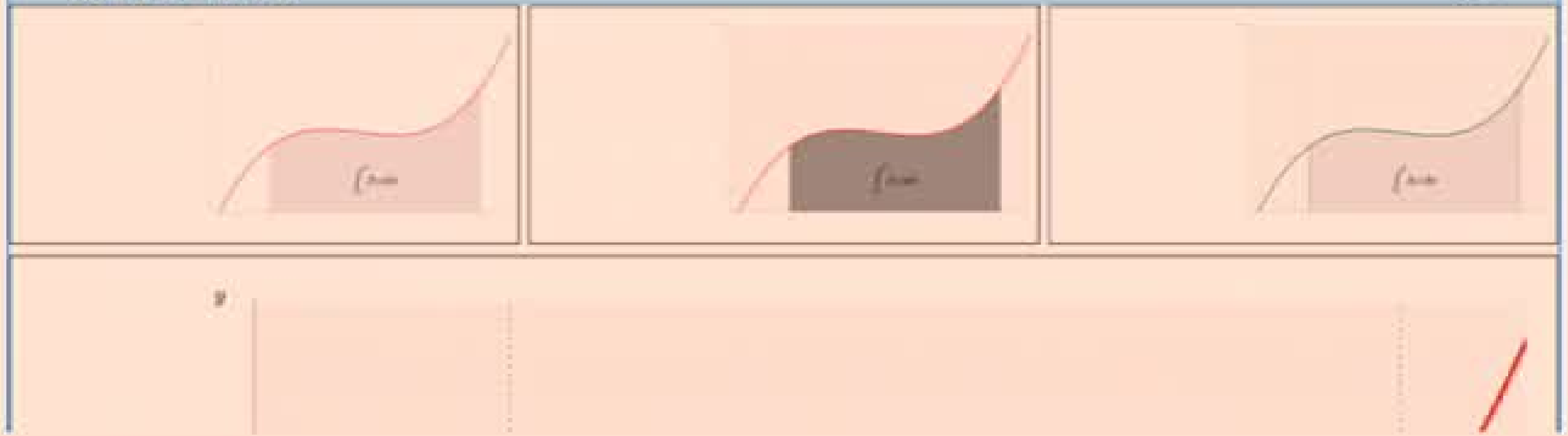
  This gives you a valid notebook that you can open in your usual notebook editor and resolve conflicts by hand, just like you might for a regular source file in your text editor.

use-base
: When a conflict is encountered, use the value from the base notebook.

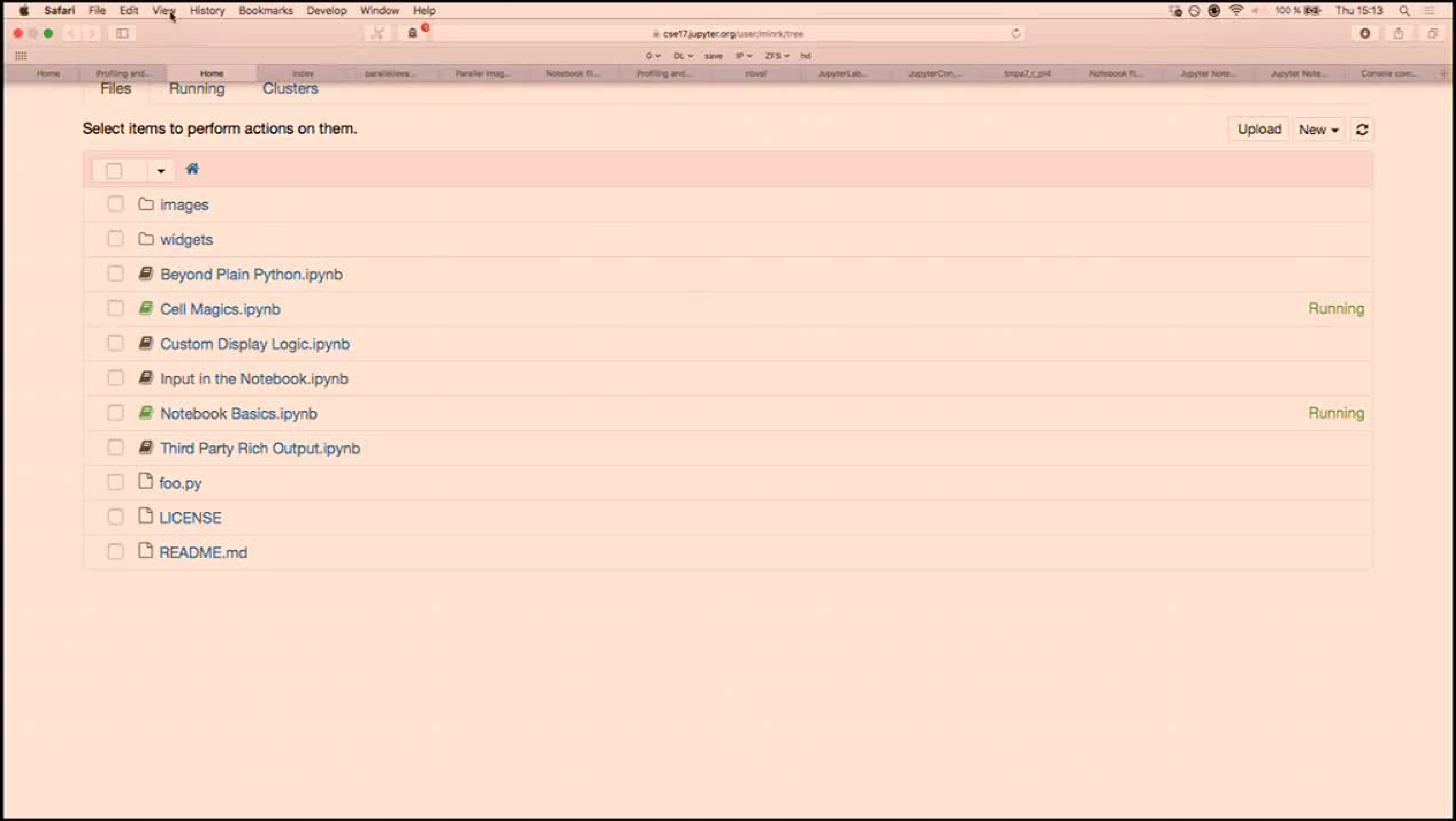
use-local
: When a conflict is encountered, use the value from the local notebook.

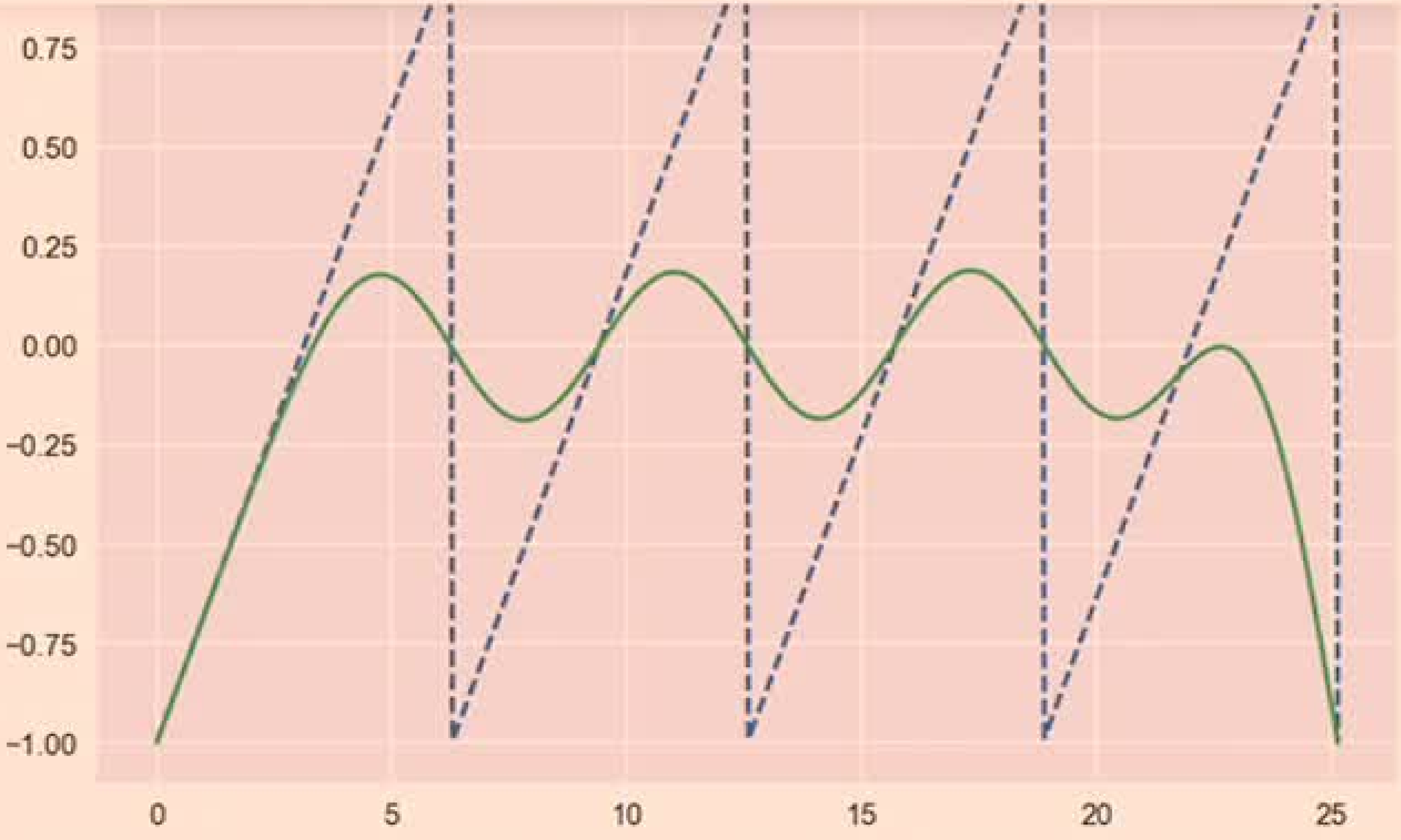
```
ix = np.linspace(a, b)
iy = func(ix)
verts = [(a, 0)] + list(zip(ix, iy)) + [(b, 0)]
poly = Polygon(verts, facecolor='0.6', edgecolor='0.5')
ax.add_patch(poly)

(...)
plt.figtext(0.9, 0.05, '$x$')
plt.figtext(0.1, 0.9, '$y$')

ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
(...)

plt.show()
```

Outputs changed



Previous   Next


Safari File Edit View History Bookmarks Develop Window Help
Thu 15:13
cse17.jupyter.org
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
images
widgets
Beyond Plain Python.ipynb
Cell Magics.ipynb
Running
Custom Display Logic.ipynb
Input in the Notebook.ipynb
Notebook Basics.ipynb
Running
Third Party Rich Output.ipynb
foo.py
LICENSE
README.md



numba

numba is a library that attempts to automatically do type-based optimizations like we did with Cython. To use numba, you decorate functions with `@autojit`.

In [38]:
```
import numba

@numba.autojit
```

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

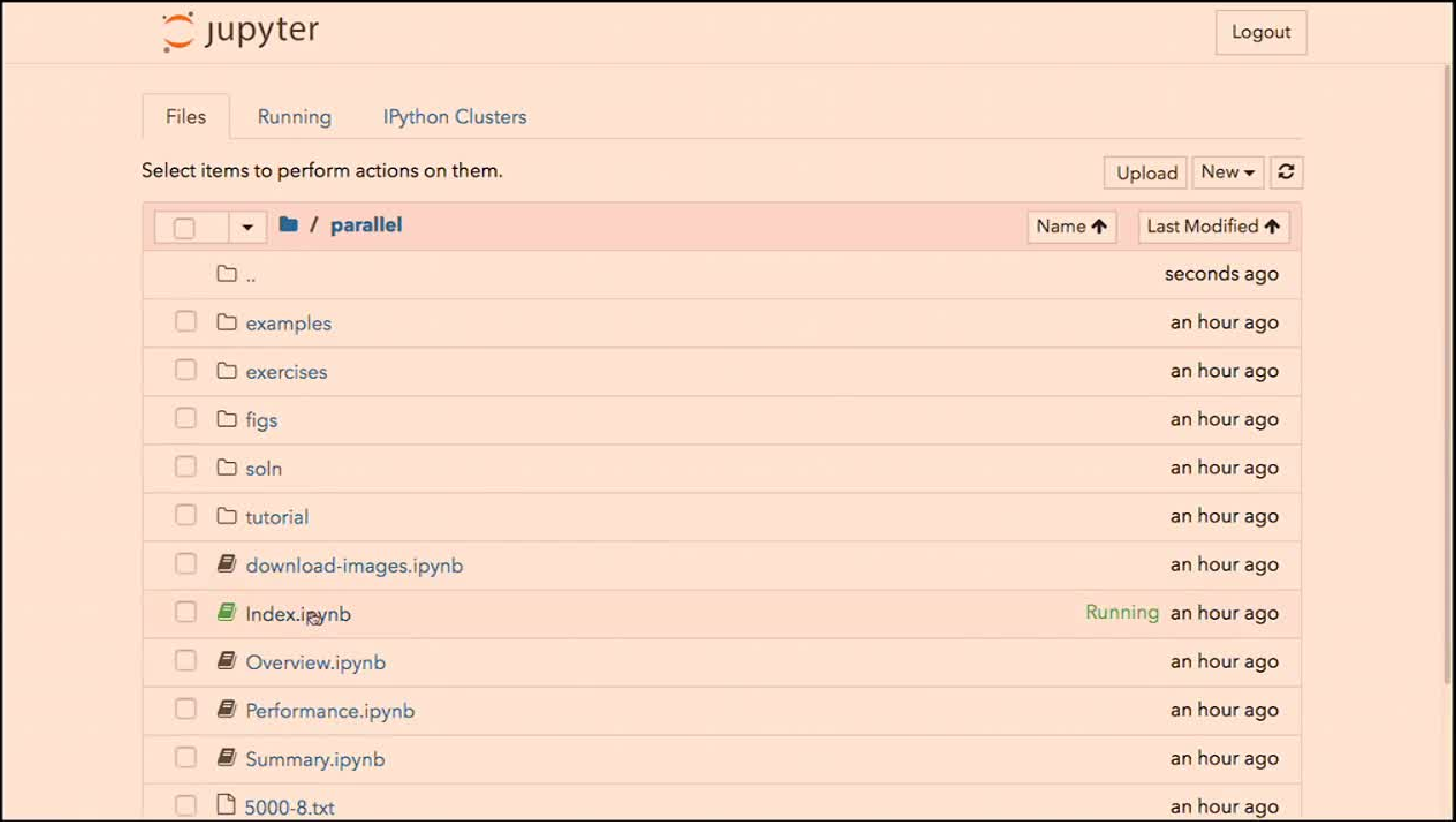
For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

Optional dependencies: I will use NetworkX for one demo, and **`scikit-image`** for another, but they are not critical. Both packages are in in Anaconda.

For the image-related demos, all you need are some images on your computer. The notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline


jupyter
Logout
Files
Running
IPython Clusters
Select items to perform actions on them.
Upload
New
/ parallel
Name
Last Modified
..
seconds ago
examples
an hour ago
exercises
an hour ago
figs
an hour ago
soln
an hour ago
tutorial
an hour ago
download-images.ipynb
an hour ago
Index.ipynb
Running
an hour ago
Overview.ipynb
an hour ago
Performance.ipynb
an hour ago
Summary.ipynb
an hour ago
5000-8.txt
an hour ago

Kernel ready | Saving every 120s | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

Interactive (parallel) Python

Installation and dependencies

You will need ipyparallel >= 5.x, and pyzmq ≥ 13. To use the demo notebooks, you will also need tornado ≥ 4. I will also make use of numpy and matplotlib. If you have Canopy or Anaconda, you already have all of these.

Quick one-line install for IPython and its dependencies:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

Loading [MathJax]/extensions/Safe.js

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline

- Motivating Example
- Overview
- Tutorial
    - Remote Execution
    - Multiplexing
    - Load-Balancing
    - Both!
    - Parallel Magics
- Examples
- Exercises

Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [ ]:

```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display
```

Connected | Saving every 120s | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [ ]:

```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display
```

Loading [MathJax]/extensions/Safe.js

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [ ]:
```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/ima
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [ ]:
```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/ima


def download_images():
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
            for chunk in r.iter_content(chunk_size=8192):
                p.value += len(chunk)
                f.write(chunk)


if not os.path.exists(images):
    images_zip = images + '.zip'
    if os.path.exists(images_zip):
        try:
            zf = ZipFile(images_zip)
        except BadZipFile:
            os.remove(images_zip)
        else:
            zf.close()
    if not os.path.exists(images_zip):
        download_images()

    ZipFile(images_zip).extractall('..')
```

In [*]:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

In [ ]:

```
import sys,os,re,time
import urllib

import numpy as np
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
        for chunk in r.iter_content(chunk_size=8192):
            p.value += len(chunk)
            f.write(chunk)


if not os.path.exists(images):
    images_zip = images + '.zip'
    if os.path.exists(images_zip):
        try:
            zf = ZipFile(images_zip)
        except BadZipFile:
            os.remove(images_zip)
        else:
            zf.close()
    if not os.path.exists(images_zip):
        download_images()

    ZipFile(images_zip).extractall('..')
```

In [*]:

```python
%matplotlib inline
import matplotlib.pyplot as plt
```

In [ ]:

```python
import sys,os,re,time
import urllib

import numpy as np
```

iTerm2 Shell Edit View Profiles Toolbelt Window Help

IPython: Users/minrk... ⌘5

```
images.zip        images_common.py
minrk[14:15]~/dev/jpy/pres/ipython-cse17 (master) $ git rm parallel/images.zip
error: the following file has changes staged in the index:
    parallel/images.zip
(use --cached to keep the file, or -f to force removal)
minrk[14:15]~/dev/jpy/pres/ipython-cse17 (master) $ git rm parallel/images.zip
minrk[14:16]~/dev/jpy/pres/ipython-cse17 (master) $ gitgit
minrk[14:16]~/dev/jpy/pres/ipython-cse17 (master) $ git push
Counting objects: 89, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (88/88), done.
Writing objects: 100% (89/89), 10.94 MiB | 391.00 KiB/s, done.
Total 89 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local objects.
To github.com:minrk/ipython-cse17.git
   1a6a644..16a9059  master -> master
minrk[14:17]~/dev/jpy/pres/ipython-cse17 (master) $ ls
Beyond Plain Python.ipynb      Notebook Basics.ipynb          Profiling and Optimizing with IPython.ipynb doc_sanitize.cfg        mod.py                 parallel
Cell Magics.ipynb              Notebook file format.html      README.md                    foo.py                  nbconvert_templates    profileme.py
Custom Display Logic.ipynb     Notebook file format.ipynb     Third Party Rich Output.ipynb images                  nbval.ipynb            test.txt
Input in the Notebook.ipynb    Notebook file format.py        __pycache__                  ipython-cse17.tar.gz    nbval.pdf              widgets
LICENSE                        Plotting in the Notebook.ipynb data.csv                     log.py                  nbval.py
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $ du -hs Plotting\ in\ the\ Notebook.ipynb
288K    Plotting in the Notebook.ipynb
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $
```

```
        > jupyter nbconvert myslides.ipynb --to slides --post serve

        Multiple notebooks can be given at the command line in a couple of
        different ways:

        > jupyter nbconvert notebook*.ipynb
        > jupyter nbconvert notebook1.ipynb notebook2.ipynb

        or you can specify the notebooks list in a config file, containing::

            c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

        > jupyter nbconvert --config mycfg.py
        """.format(get_export_names()))

    # Writer specific variables
    writer = Instance('nbconvert.writers.base.WriterBase',
                      help="""Instance of the writer class used to write the
                      results of the conversion.""", allow_none=True)
    writer_class = DottedObjectName('FilesWriter',
                                    help="""Writer class used to write the
                                    results of the conversion""").tag(config=True)
    writer_aliases = {'fileswriter': 'nbconvert.writers.files.FilesWriter',
                      'debugwriter': 'nbconvert.writers.debug.DebugWriter',
                      'stdoutwriter': 'nbconvert.writers.stdout.StdoutWriter'}
    writer_factory = Type(allow_none=True)
```

Line: 155:17-155:24

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [*]:

```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/ima


def download_images():
    r = requests.get(images_url, stream=True)
    content_length = r.headers.get('content-length')
    print("Downloading images")
    sys.stdout.flush()
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [*]:
```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/ima


def download_images():
    r = requests.get(images_url, stream=True)
    content_length = r.headers.get('content-length')
    print("Downloading images")
    sys.stdout.flush()
    p = IntProgress(max=content_length)
    display(p)
    with open(images_zip, 'wb') as f:
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

You can download images with this notebook, or get a zip here, or find any images on your computer.

In [*]:

```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/ima


def download_images():
    r = requests.get(images_url, stream=True)
    content_length = r.headers.get('content-length')
    print("Downloading images")
    sys.stdout.flush()
    p = IntProgress(max=content_length)
    display(p)
    with open(images_zip, 'wb') as f:
```

Safari File Edit View History Bookmarks Develop Window Help

100 % Thu 15:15

```
    which will convert mynotebook.ipynb to the default format (probably HTML).

    You can specify the export format with `--to`.
    Options include {0}

    > jupyter nbconvert --to latex mynotebook.ipynb

    Both HTML and LaTeX support multiple output templates. LaTeX includes
    'base', 'article' and 'report'.  HTML includes 'basic' and 'full'. You
    can specify the flavor of the format used.

    > jupyter nbconvert --to html --template basic mynotebook.ipynb

    You can also pipe the output to stdout, rather than a file

    > jupyter nbconvert mynotebook.ipynb --stdout

    PDF is generated via latex

    > jupyter nbconvert mynotebook.ipynb --to pdf

    You can get (and serve) a Reveal.js-powered slideshow

    > jupyter nbconvert myslides.ipynb --to slides --post serve

    Multiple notebooks can be given at the command line in a couple of 
    different ways:

    > jupyter nbconvert notebook*.ipynb
    > jupyter nbconvert notebook1.ipynb notebook2.ipynb

    or you can specify the notebooks list in a config file, containing::

        c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

    > jupyter nbconvert --config mycfg.py
    """.format(get_export_names()))

    # Writer specific variables
    writer = Instance('nbconvert.writers.base.WriterBase',  
                      help="""Instance of the writer class used to write the 
                      results of the conversion.""", allow_none=True)
    writer_class = DottedObjectName('FilesWriter',
                                    help="""Writer class used to write the 
                                    results of the conversion""").tag(config=True)
    writer_aliases = {'fileswriter': 'nbconvert.writers.files.FilesWriter',
                      'debugwriter': 'nbconvert.writers.debug.DebugWriter',
                      'stdoutwriter': 'nbconvert.writers.stdout.StdoutWriter'}
    writer_factory = Type(allow_none=True)
```

Line: 155:17-155:24

Find: include

Replace: minrk

Options: Regular Expression | Ignore Whitespace | Ignore Case | ✓ Wrap Around

In: Document matching

Found "include" at line 155, column 17.

Find All | Replace All | Replace | Replace & Find | Previous | Next

Interrupting kernel | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

In [3]:
```
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [*]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recen
t call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

Interrupting kernel Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help

In [3]:
```
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [*]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

iTerm2 Shell Edit View Profiles Toolbelt Window Help 100 % Thu 15:16

python3.6 ⌘1 bash ⌘2 bash ⌘3 python3.6 ⌘4 IPython: Users/minrk... ⌘5

```
minrk[14:16]~/dev/jpy/pres/ipython-cse17 (master) $ git push
Counting objects: 89, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (88/88), done.
Writing objects: 100% (89/89), 10.94 MiB | 391.00 KiB/s, done.
Total 89 (delta 8), reused 0 (delta 0)
remote: Resolving deltas: 100% (8/8), completed with 1 local objects.
To github.com:minrk/ipython-cse17.git
   1a6a644..16a9059  master -> master
minrk[14:17]~/dev/jpy/pres/ipython-cse17 (master) $ ls
Beyond Plain Python.ipynb      Notebook Basics.ipynb        Profiling and Optimizing with IPython.ipynb doc_sanitize.cfg        mod.py                 parallel
Cell Magics.ipynb              Notebook file format.html    README.md                   foo.py                  nbconvert_templates    profileme.py
Custom Display Logic.ipynb     Notebook file format.ipynb   Third Party Rich Output.ipynb images                 nbval.ipynb            test.txt
Input in the Notebook.ipynb    Notebook file format.py      __pycache__                 ipython-cse17.tar.gz    nbval.pdf              widgets
LICENSE                        Plotting in the Notebook.ipynb data.csv                  log.py                  nbval.py
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $ du -hs Plotting\ in\ the\ Notebook.ipynb
288K    Plotting in the Notebook.ipynb
minrk[15:07]~/dev/jpy/pres/ipython-cse17 (master) $ ls
Beyond Plain Python.ipynb      Notebook Basics.ipynb        Profiling and Optimizing with IPython.ipynb doc_sanitize.cfg        mod.py                 parallel
Cell Magics.ipynb              Notebook file format.html    README.md                   foo.py                  nbconvert_templates    profileme.py
Custom Display Logic.ipynb     Notebook file format.ipynb   Third Party Rich Output.ipynb images                 nbval.ipynb            test.txt
Input in the Notebook.ipynb    Notebook file format.py      __pycache__                 ipython-cse17.tar.gz    nbval.pdf              widgets
LICENSE                        Plotting in the Notebook.ipynb data.csv                  log.py                  nbval.py
minrk[15:15]~/dev/jpy/pres/ipython-cse17 (master) $ conda install -y scikit-image
Fetching package metadata ........
```

```python
        > jupyter nbconvert myslides.ipynb --to slides --post serve

        Multiple notebooks can be given at the command line in a couple of
        different ways:

        > jupyter nbconvert notebook*.ipynb
        > jupyter nbconvert notebook1.ipynb notebook2.ipynb

        or you can specify the notebooks list in a config file, containing::

            c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

        > jupyter nbconvert --config mycfg.py
        """.format(get_export_names()))

    # Writer specific variables
    writer = Instance('nbconvert.writers.base.WriterBase',
                      help="""Instance of the writer class used to write the
                      results of the conversion.""", allow_none=True)
    writer_class = DottedObjectName('FilesWriter',
                                    help="""Writer class used to write the
                                    results of the conversion""").tag(config=True)
    writer_aliases = {'fileswriter': 'nbconvert.writers.files.FilesWriter',
                      'debugwriter': 'nbconvert.writers.debug.DebugWriter',
                      'stdoutwriter': 'nbconvert.writers.stdout.StdoutWriter'}
    writer_factory = Type(allow_none=True)
```

Line: 155:17-155:24

In [3]:
```python
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [5]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```python
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recen
t call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

Define a function to find the corners of the image:

notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline

- Motivating Example
- Overview
- Tutorial
  - Remote Execution
  - Multiplexing
  - Load-Balancing
  - Both!
  - Parallel Magics
- Examples
- Exercises

Interactive (parallel) Python

Installation and dependencies

You will need ipyparallel >= 5.x, and pyzmq ≥ 13. To use the demo notebooks, you will also need tornado ≥ 4. I will also make use of numpy and matplotlib. If you have Canopy or Anaconda, you already have all of these.

Quick one-line install for IPython and its dependencies:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

File Edit View Insert Cell Kernel Help Saving every 120s Python 3

Markdown CellToolbar

Interactive monitoring of a parallel MPI simulation with the IPython Notebook

In [3]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from IPython.display import display
from ipyparallel import Client, error

cluster = Client(profile="mpi")
view = cluster[:]
view.block = True
e0 = cluster[0]
e0.activate('0')
```

In [4]:

```
cluster.ids
```

Saving every 120s Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help

Interactive monitoring of a parallel MPI simulation with the IPython Notebook

In [3]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from IPython.display import display
from ipyparallel import Client, error

cluster = Client(profile="mpi")
view = cluster[:]
view.block = True
e0 = cluster[0]
e0.activate('0')
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Interactive monitoring of a parallel MPI simulation with the IPython Notebook

In [22]:
```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from IPython.display import display
from ipyparallel import Client, error

cluster = Client(profile="mpi")
view = cluster[:]
view.block = True
e0 = cluster[0]
e0.activate('0')
```

In [4]:
```
cluster.ids
```

Out[4]:
```
[0, 1, 2, 3]
```

Now, we load the MPI libraries into the engine namespaces, and do a simple

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Out[23]: [0, 1, 2, 3]

Now, we load the MPI libraries into the engine namespaces, and do a simple printing of their MPI rank information to verify that all nodes are operational and they match our cluster's real capacity.

Here, we are making use of IPython's special **%%px** cell magic, which marks the entire cell for parallel execution. This means that the code below will not run in this notebook's kernel, but instead will be sent to *all* engines for execution there. In this way, IPython makes it very natural to control your entire cluster from within the notebook environment:

In [5]:
```
%%px
# MPI initialization, library imports and sanity checks on all e
from mpi4py import MPI
import numpy as np
import time

mpi = MPI.COMM_WORLD
bcast = mpi.bcast
barrier = mpi.barrier
rank = mpi.rank
print("MPI rank: %i/%i" % (mpi.rank,mpi.size))
```

```
[stdout:0] MPI rank: 0/4
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
[stdout:0] MPI rank: 2/4
[stdout:1] MPI rank: 0/4
[stdout:2] MPI rank: 1/4
[stdout:3] MPI rank: 3/4
```

We write a utility that reorders a list according to the mpi ranks of the engines, since all gather operations will return data in engine id order, not in MPI rank order. We'll need this later on when we want to reassemble in IPython data structures coming from all the engines: IPython will collect the data ordered by engine ID, but our code creates data structures based on MPI rank, so we need to map from one indexing scheme to the other. This simple function does the job:

In [25]:
```python
ranks = view['rank']
rank_indices = np.argsort(ranks)

def mpi_order(seq):
    """Return elements of a sequence ordered by MPI rank.

    The input sequence is assumed to be ordered by engine ID."""
    return [seq[x] for x in rank_indices]
```

MPI simulation example

This is our 'simulation', a toy example that computes $\sin(f(x^2 + y^2))$ for a slowly

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
    # remotely for interactive introspection
    global j, Z, nx, nyt
    freqs = np.linspace(0.6, 1, nsteps)
    for j in range(nsteps):
        nx, ny = 2+j//4, 2+j//2//mpi.size
        nyt = mpi.size*ny
        Xax = np.linspace(xmin, xmax, nx)
        Yax = np.linspace(ymin+rank*dy, ymin+(rank+1)*dy, ny, en
        X, Y = np.meshgrid(Xax, Yax)
        f = freqs[j]
        Z = np.cos(f*(X**2 + Y**2))
        # We add a small delay to simulate that a real-world com
        # would take much longer, and we ensure all nodes are sy
        time.sleep(delay)
        # The stop flag can be set remotely via IPython, allowin
        # cleanly stopped from the outside
        if stop:
            break
```

IPython tools to interactively monitor and plot the MPI results

We now define a local (to this notebook) plotting function that fetches data from the engines' global namespace. Once it has retrieved the current state of the relevant

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

```python
        if stop:
            break
```

IPython tools to interactively monitor and plot the MPI results

We now define a local (to this notebook) plotting function that fetches data from the engines' global namespace. Once it has retrieved the current state of the relevant variables, it produces and returns a figure:

In [14]:
```python
from IPython.display import clear_output

def plot_current_results(in_place=True):
    """Makes a blocking call to retrieve remote data and display
    as a contour plot.

    Parameters
    ----------
    in_place : bool
        By default it calls clear_output so that new plots repla
        to False to allow keeping of all previous outputs.
    """

    # We make a blocking call to load the remote data from the s
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```python
        msg = 'Simulation completed!'
    tmon = dt.datetime.now() - t0
    if plots_in_place and fig is not None:
        clear_output(wait=True)
        plt.close('all')
        display(fig)
    print(msg)
    print('Monitored for: %s.' % tmon)
```

Making a simulation object that can be monitored interactively

In [17]:

```python
%%px
from threading import Thread
stop = False
nsteps = 100
delay=0.5
# Create a thread wrapper for the simulation. The target must be
# function so we wrap the call to 'simulation' in a simple lambd
simulation_thread = Thread(target = lambda : simulation())
# Now we actually start the simulation
simulation_thread.start()
```

In [18]:

```python
monitor_simulation(refresh=1);
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    if plots_in_place and fig is not None:
        clear_output(wait=True)
        plt.close('all')
        display(fig)
    print(msg)
    print('Monitored for: %s.' % tmon)
```

Making a simulation object that can be monitored interactively

In [17]:
```
%%px
from threading import Thread
stop = False
nsteps = 100
delay=0.5
# Create a thread wrapper for the simulation. The target must be
# function so we wrap the call to 'simulation' in a simple lambd
simulation_thread = Thread(target = lambda : simulation())
# Now we actually start the simulation
simulation_thread.start()
```

In [18]:
```
monitor_simulation(refresh=1);
```

Mesh: 17 x 36, step 61/100

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
        if true, every new figure replaces the last one, producin
        animation effect in the notebook.  If false, all frames a
        in sequence and appended in the output area.
    """
    import datetime as dt, time

    if not simulation_alive():
        plot_current_results(in_place=plots_in_place)
        plt.close('all')
        print('Simulation has already finished, no monitoring to
        return

    t0 = dt.datetime.now()
    fig = None
    try:
        while simulation_alive():
            fig = plot_current_results(in_place=plots_in_place)
            plt.close('all') # prevent re-plot of old figures
            time.sleep(refresh) # so we don't hammer the server
    except (KeyboardInterrupt, error.TimeoutError):
        msg = 'Monitoring interrupted, simulation is ongoing!'
    else:
        msg = 'Simulation completed!'
    tmon = dt.datetime.now() - t0
    if plots_in_place and fig is not None:
        clear_output(wait=True)
        plt.close('all')
        display(fig)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    If true, every new figure replaces the last one, producin
    animation effect in the notebook.  If false, all frames a
    in sequence and appended in the output area.
"""
import datetime as dt, time

if not simulation_alive():
    plot_current_results(in_place=plots_in_place)
    plt.close('all')
    print('Simulation has already finished, no monitoring to
    return

t0 = dt.datetime.now()
fig = None
try:
    while simulation_alive():
        fig = plot_current_results(in_place=plots_in_place)
        plt.close('all') # prevent re-plot of old figures
        time.sleep(refresh) # so we don't hammer the server
except (KeyboardInterrupt, error.TimeoutError):
    msg = 'Monitoring interrupted, simulation is ongoing!'
else:
    msg = 'Simulation completed!'
tmon = dt.datetime.now() - t0
if plots_in_place and fig is not None:
    clear_output(wait=True)
    plt.close('all')
    display(fig)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    except (KeyboardInterrupt, error.TimeoutError):
        msg = 'Monitoring interrupted, simulation is ongoing!'
    else:
        msg = 'Simulation completed!'
    tmon = dt.datetime.now() - t0
    if plots_in_place and fig is not None:
        clear_output(wait=True)
        plt.close('all')
        display(fig)
    print(msg)
    print('Monitored for: %s.' % tmon)
```

Making a simulation object that can be monitored interactively

In [17]:
```
x
m threading import Thread
p = False
eps = 100
ay=0.5
reate a thread wrapper for the simulation. The target must be an
unction so we wrap the call to 'simulation' in a simple lambda:
ulation_thread = Thread(target = lambda : simulation())
ow we actually start the simulation
ulation_thread.start()
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
        except (KeyboardInterrupt, error.TimeoutError):
            msg = 'Monitoring interrupted, simulation is ongoing!'
        else:
            msg = 'Simulation completed!'
        tmon = dt.datetime.now() - t0
        if plots_in_place and fig is not None:
            clear_output(wait=True)
            plt.close('all')
            display(fig)
        print(msg)
        print('Monitored for: %s.' % tmon)
```

Making a simulation object that can be monitored interactively

```
In [17]: x
m threading import Thread
p = False
eps = 100
ay=0.5
reate a thread wrapper for the simulation. The target must be an
unction so we wrap the call to 'simulation' in a simple lambda:
ulation_thread = Thread(target = lambda : simulation())
ow we actually start the simulation
```

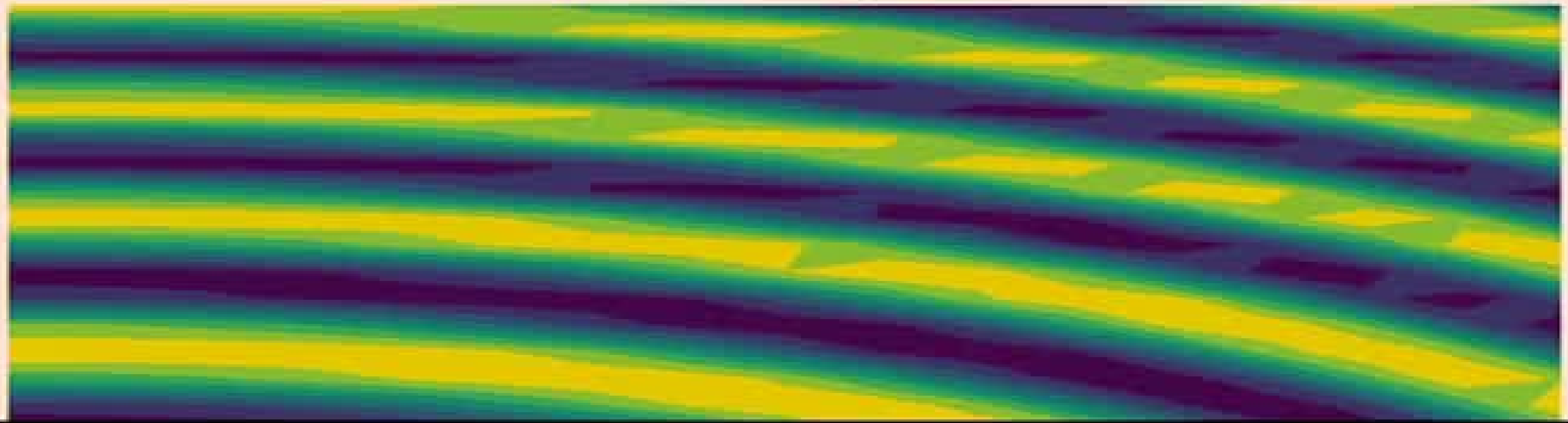
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Making a simulation object that can be monitored interactively

In [17]:
```
%%px
from threading import Thread
stop = False
nsteps = 100
delay=0.5
# Create a thread wrapper for the simulation. The target must be
# function so we wrap the call to 'simulation' in a simple lambd
simulation_thread = Thread(target = lambda : simulation())
# Now we actually start the simulation
simulation_thread.start()
```

In [18]:
```
monitor_simulation(refresh=1);
```

Mesh: 17 x 36, step 61/100

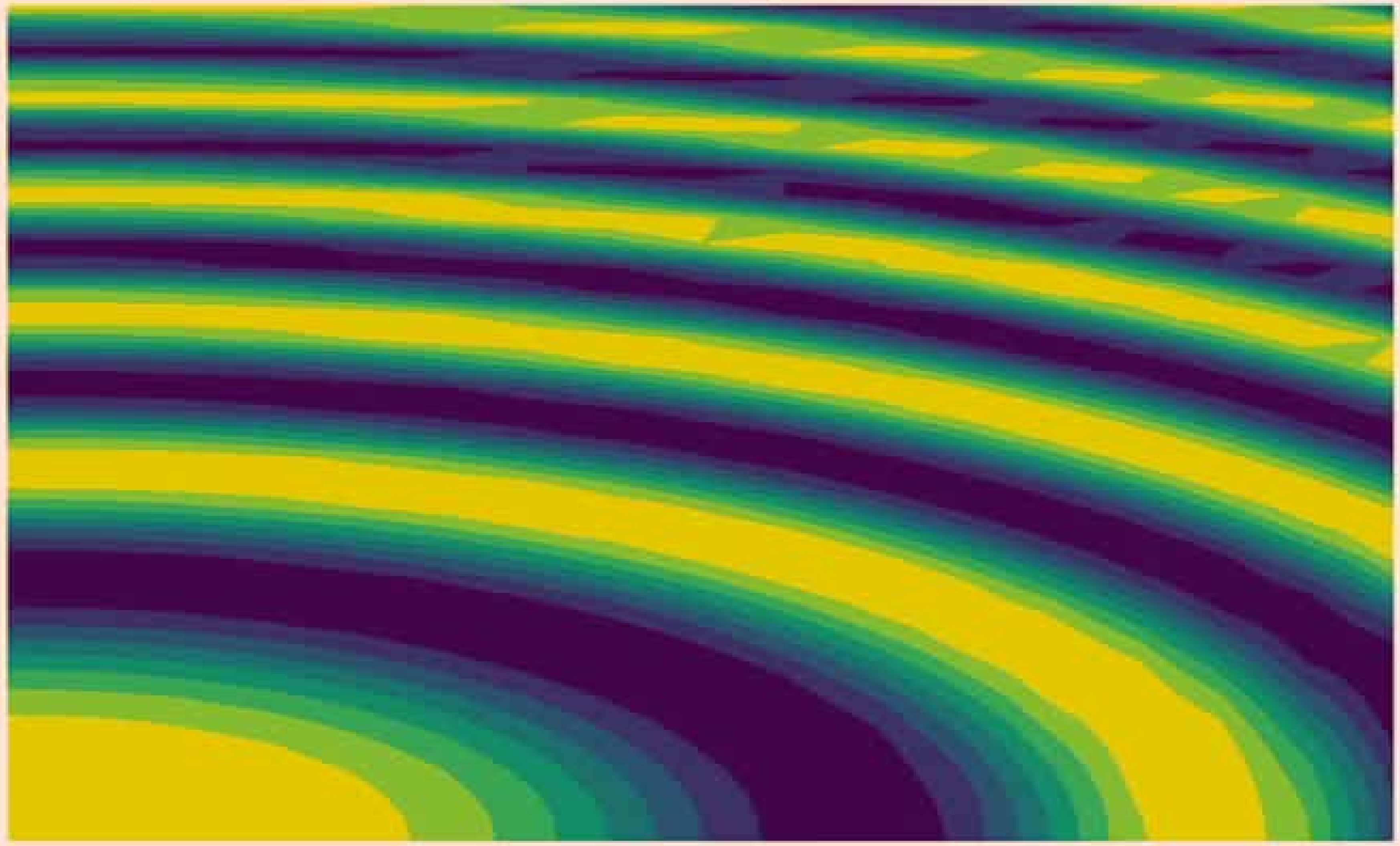


File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
simulation_thread.start()
```

In [18]:
```
monitor_simulation(refresh=1);
```

Mesh: 17 x 36, step 61/100



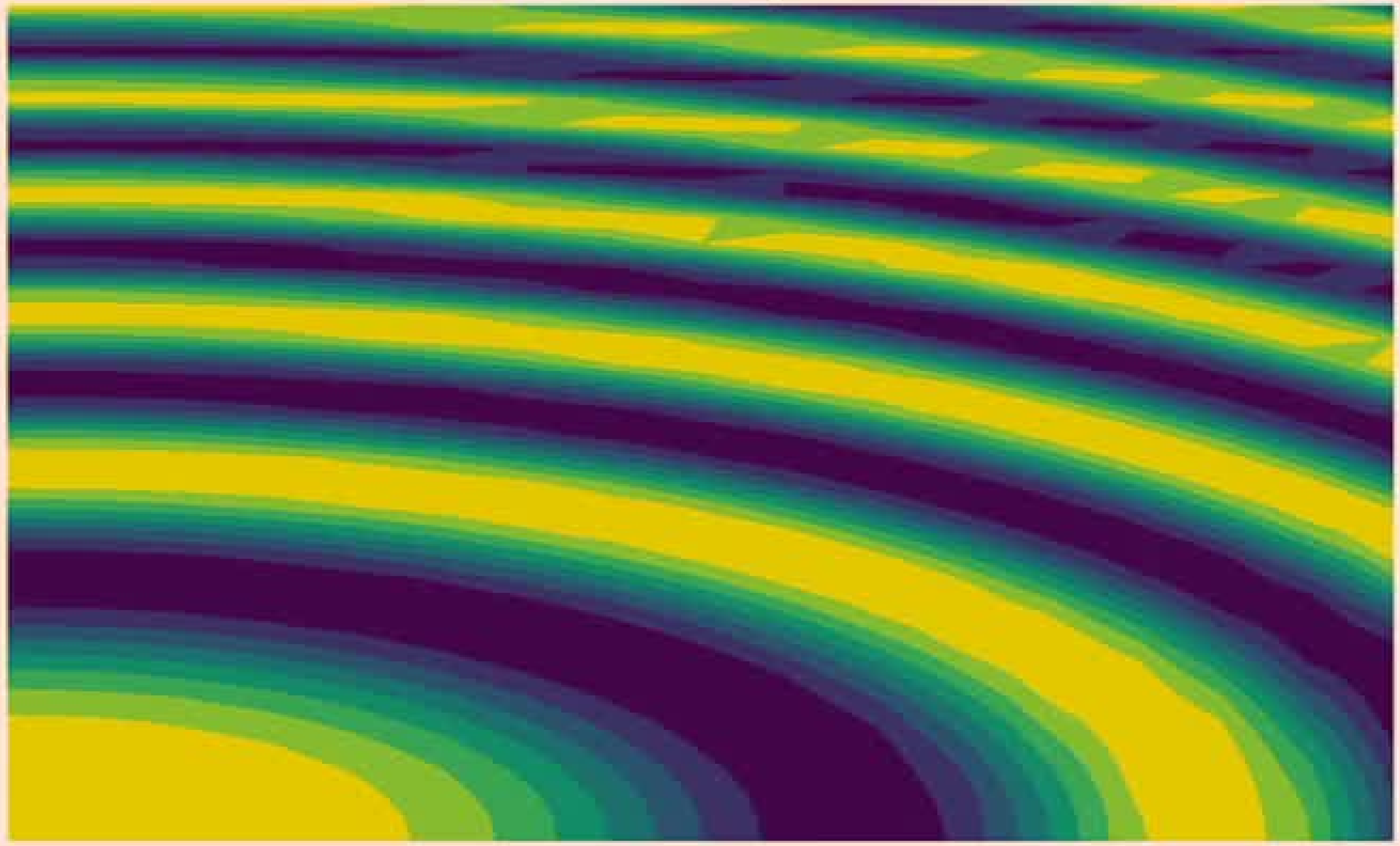
```
Monitoring interrupted, simulation is ongoing!
Monitored for: 0:00:29.828907.
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
simulation_thread.start()
```

In [18]:
```
monitor_simulation(refresh=1);
```

Mesh: 17 x 36, step 61/100



```
Monitoring interrupted, simulation is ongoing!
Monitored for: 0:00:29.828907.
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

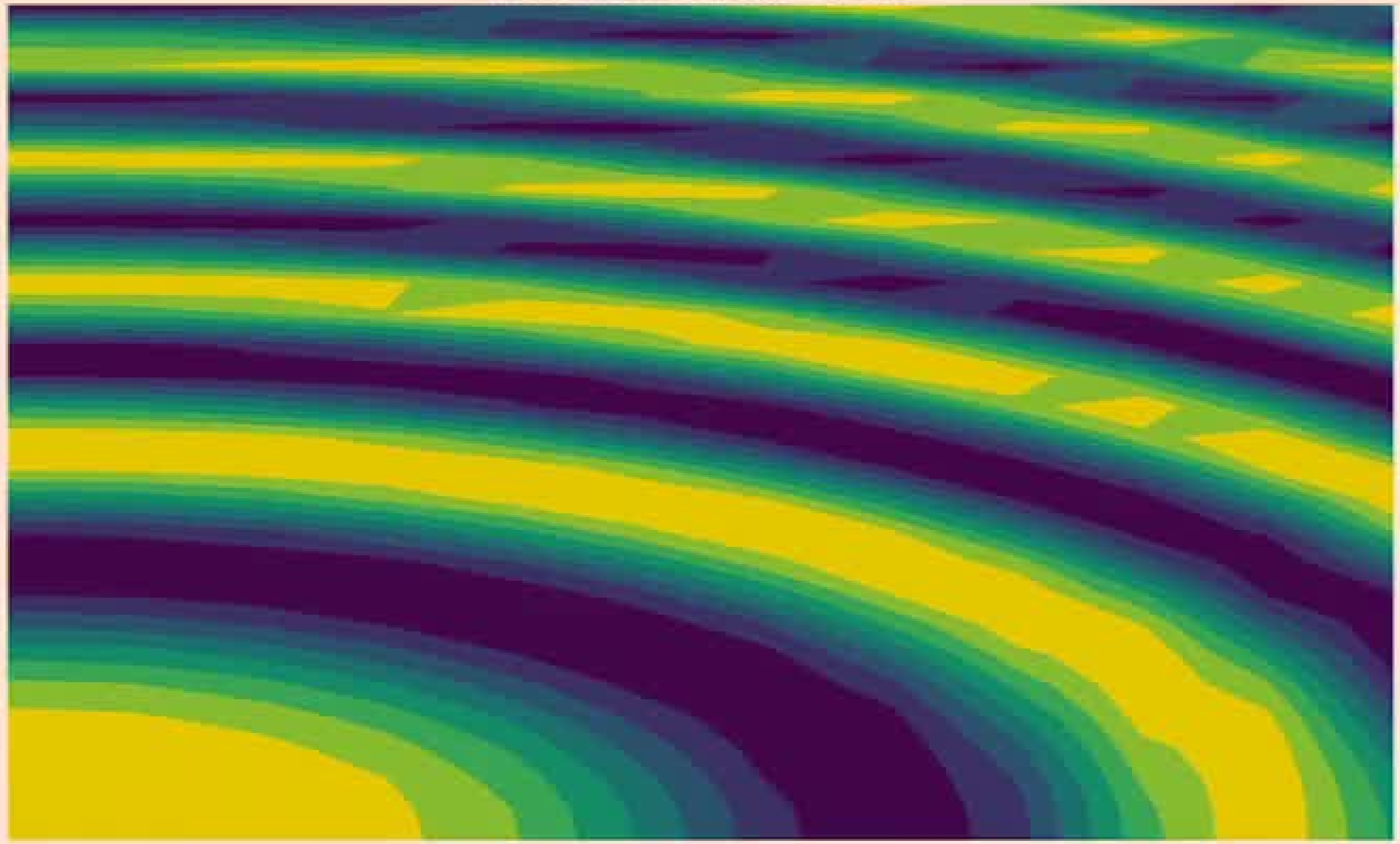
In [*]:
```
monitor_simulation(refresh=1);
```

If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [*]:
```
monitor_simulation(refresh=1);
```

Mesh: 12 x 28, step 44/100



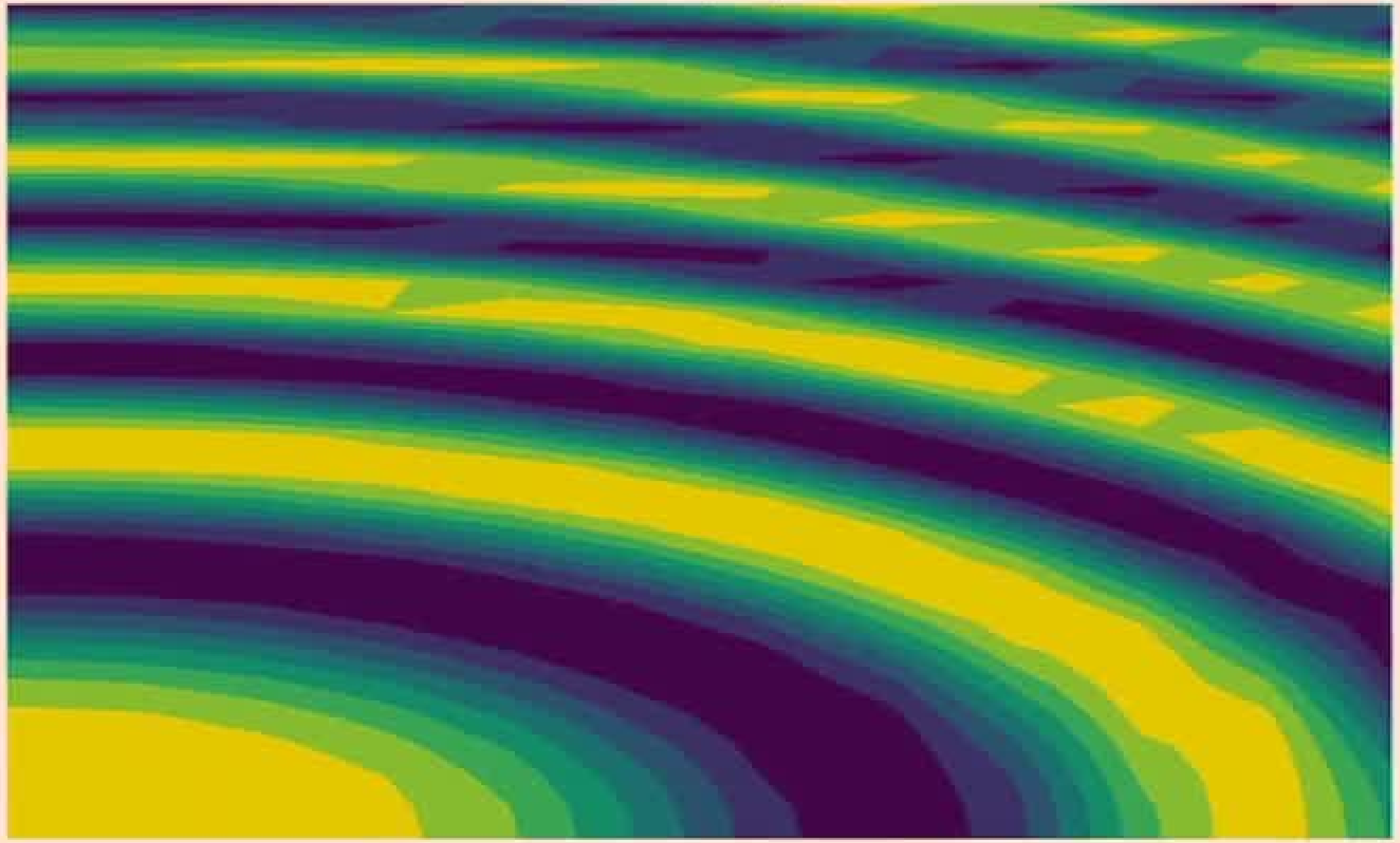
If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

Interrupting kernel | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

In [*]:
```
monitor_simulation(refresh=1);
```

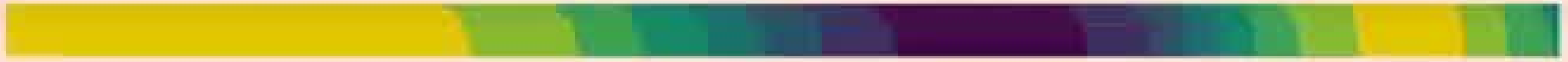
Mesh: 12 x 28, step 44/100



If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

Interrupting kernel Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-b619ab205d86> in <module>()
----> 1 monitor_simulation(refresh=1);

<ipython-input-29-ae2365e1441d> in monitor_simulation(refresh, plots_in_place)
     29     try:
     30         while simulation_alive():
---> 31             fig = plot_current_results(in_place=plots_in_place)
     32             plt.close('all') # prevent re-plot of old figures
     33             time.sleep(refresh) # so we don't hammer the server too fast

<ipython-input-27-b77fff73a570> in plot_current_results(in_place)
     24         return ax.figure
     25
---> 26     nx, nyt, j, nsteps = view.pull(['nx', 'nyt', 'j', '
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
TypeError: AsyncResults with a single result are not iterable.
```

If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

In [32]:
```
view['stop'] = True
```

In [20]:
```
%px0 from IPython.parallel import bind_kernel; bind_kernel()
```

Out[20]:
```
<AsyncResult: execute>
```

In [21]:
```
%px0 %qtconsole
```

Out[21]:
```
<AsyncResult: execute>
```

In [ ]:

```
result are not iterable.")
    383         try:
    384             rlist = self.get(0)

TypeError: AsyncResults with a single result are not iterable.
```

If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

In [32]:
```
view['stop'] = True
```

In [33]:
```
%px0 from ipyparallel import bind_kernel; bind_kernel()
```

Out[33]: <AsyncResult: execute>

In [34]:
```
%px0 %qtconsole
```

Out[34]: <AsyncResult: execute>

In [ ]:

python File Edit View Kernel Window Help

If the raised exception doesn't match the stored exception, we get a failure

```
------------------------------------------------------------------------

4
```

Jupyter QtConsole

```
ipdb> bt
> <ipython-
input-16-05c9758a9c21>(1)<module>(
)
----> 1 1/0


ipdb> exit

In [18]:
```

```
RuntimeError                              Traceback (most recent call last)

<ipython-input-3-32dcc1c70a4e> in <module>()
  1 # NBVAL_RAISES_EXCEPTION
  2 print("If the raised exception doesn't match the stored exception, we get a failure"
----> 3 raise RuntimeError("Foo")


RuntimeError: Foo
```

```
In [2]: # NBVAL_IGNORE_OUTPUT
```

If the raised exception doesn't match the stored exception, we get a failure

---

4

```
ipdb> bt
> <ipython-
input-16-05c9758a9c21>(1)<module>(
)
----> 1 1/0


ipdb> exit

In [18]:
```

```
RuntimeError                              Traceback (most recent call last)

<ipython-input-3-32dcc1c70a4e> in <module>()
  1 # NBVAL_RAISES_EXCEPTION
  2 print("If the raised exception doesn't match the stored exception, we get a failure"
----> 3 raise RuntimeError("Foo")


RuntimeError: Foo
```

```
In [2]: # NBVAL_IGNORE_OUTPUT
```

```
result are not iterable.")
    383         try:
    384             rlist = self.get(0)

TypeError: AsyncResults with a single result are not iterable.
```

If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

In [32]:
```
view['stop'] = True
```

In [33]:
```
%px0 from ipyparallel import bind_kernel; bind_kernel()
```

Out[33]:
```
<AsyncResult: execute>
```

In [34]:
```
%px0 %qtconsole
```

Out[34]:
```
<AsyncResult: execute>
```

In [ ]:

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

In [3]:
```
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [5]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

Define a function to find the corners of the image:

iTerm2 Shell Edit View Profiles Toolbelt Window Help 100 % Thu 15:23

python3.6 ⌘1 bash ⌘2 bash ⌘3 python3.6 ⌘4 IPython: Users/minrk... ⌘5

```
    conda.exceptions.CondaRuntimeError: Runtime error: RuntimeError: Runtime error: Could not open '/Users/minrk/conda/pkgs/scikit-image-0.12.3-np112py36_1.tar.bz2.part' for writing (HTTPSConnectionPool(host='binstar-cio-packages-prod.s3.amazonaws.com', port=443): Read timed out.).

minrk[15:20]~/dev/jpy/pres/ipython-cse17 (master) $ conda install -y scikit-image
Fetching package metadata .........
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0   2% |######                                                | ETA:  0:01:48 168.95 kB/s
```

Safari File Edit View History Bookmarks Develop Window Help

100 % Thu 15:23

```
    conda.exceptions.CondaRuntimeError: Runtime error; RuntimeError: Runtime error: Could not open '/Users/minrk/conda/pkgs/scikit-image-0.12.3-np112py36_1.tar.bz2.part' for writing (HTTPSConnectionPool(host='binstar-cio-packages-prod.s3.amazonaws.com', port=443): Read timed out.).

minrk[15:20]~/dev/jpy/pres/ipython-cse17 (master) $ conda install -y scikit-image
Fetching package metadata .........
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0   3% |#########                                      | ETA:  0:01:56 155.29 kB/s
```

In [3]:
```python
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [5]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```python
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

Define a function to find the corners of the image:

```
result are not iterable.")
    383         try:
    384             rlist = self.get(0)

TypeError: AsyncResults with a single result are not iterable.
```

If you execute the following cell before the MPI code is finished running, it will stop the simulation at that point, which you can verify by calling the monitoring again:

```
In [32]: view['stop'] = True
```

```
In [33]: %px0 from ipyparallel import bind_kernel; bind_kernel()
Out[33]: <AsyncResult: execute>
```

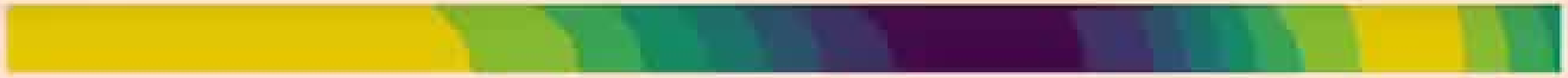
```
In [34]: %px0 %qtconsole
Out[34]: <AsyncResult: execute>
```

```
In [ ]:
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
plots_in_place)
     29     try:
     30         while simulation_alive():
---> 31             fig = plot_current_results(in_place=plots_i
n_place)
     32             plt.close('all') # prevent re-plot of old f
igures
     33             time.sleep(refresh) # so we don't hammer th
e server too fast

<ipython-input-27-b77fff73a570> in plot_current_results(in_plac
e)
     24         return ax.figure
     25
---> 26     nx, nyt, j, nsteps = view.pull(['nx', 'nyt', 'j', '
nsteps'], targets=0)
     27     fig, ax = plt.subplots()
     28     ax.contourf(Z)

/Users/minrk/conda/lib/python3.6/site-packages/ipyparallel/clie
nt/asyncresult.py in __iter__(self)
    380     def __iter__(self):
    381         if self._single_result:
--> 382             raise TypeError("AsyncResults with a single
result are not iterable.")
    383         try:
    384             rlist = self.get(0)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3



```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-31-b619ab205d86> in <module>()
----> 1 monitor_simulation(refresh=1);

<ipython-input-29-ae2365e1441d> in monitor_simulation(refresh, plots_in_place)
     29     try:
     30         while simulation_alive():
---> 31             fig = plot_current_results(in_place=plots_in_place)
     32             plt.close('all') # prevent re-plot of old figures
     33             time.sleep(refresh) # so we don't hammer the server too fast

<ipython-input-27-b77fff73a570> in plot_current_results(in_place)
     24         return ax.figure
     25 
---> 26     nx, nyt, j, nsteps = view.pull(['nx', 'nyt', 'j', 'nsteps'], targets=0)
     27     fig, ax = plt.subplots()
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
    in_place : bool
        By default it calls clear_output so that new plots replac
        to False to allow keeping of all previous outputs.
    """

    # We make a blocking call to load the remote data from the s
    # variables we can read from the engine namespaces
    #view.apply_sync(load_simulation_globals)
    # And now we can use the view to read these variables from a
    # concatenate all of them into single arrays for local plott
    try:
        Z = np.concatenate(mpi_order(view['Z']))
    except ValueError:
        print("dimension mismatch in Z, not plotting")
        ax = plt.gca()
        return ax.figure

    nx, nyt, j, nsteps = view.pull(['nx', 'nyt', 'j', 'nsteps'],
    fig, ax = plt.subplots()
    ax.contourf(Z)
    ax.set_title('Mesh: %i x %i, step %i/%i' % (nx, nyt, j+1, ns
    plt.axis('off')
    # We clear the notebook output before plotting this if in-pl
    if in_place:
        clear_output(wait=True)
    display(fig)
    return fig
```

Interactive monitoring of a parallel MPI simulation with the IPython Notebook

In [22]:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from IPython.display import display
from ipyparallel import Client, error

cluster = Client(profile="mpi")
view = cluster[:]
view.block = True
e0 = cluster[0]
e0.activate('0')
```

In [23]:

```
cluster.ids
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [3]:
```
import sys,os,re,time
import urllib

import numpy as np

import ipyparallel as parallel
```

In [5]:
```
!conda install scikit-image
```

```
Fetching package metadata ........^C
```

In [4]:
```
from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-4-12197010217e> in <module>()
----> 1 from skimage.io import imread
      2 from skimage.feature import corner_harris, corner_peaks

ModuleNotFoundError: No module named 'skimage'
```

Define a function to find the corners of the image:

iTerm2 Shell Edit View Profiles Toolbelt Window Help

```
    conda.exceptions.CondaRuntimeError: Runtime error: RuntimeError: Runtime error: Could not open '/Users/minrk/conda/pkgs/scikit-image-0.12.3-np112py36_1.tar.bz2.part' for writing (HTTPSConnectionPool(host='binstar-cio-packages-prod.s3.amazonaws.com', port=443): Read timed out.).

minrk[15:20]~/dev/jpy/pres/ipython-cse17 (master) $ conda install -y scikit-image
Fetching package metadata .........
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0  14% |################################                | ETA:  0:01:48 148.36 kB/s
```

Interactive (parallel) Python

Installation and dependencies

You will need ipyparallel >= 5.x, and pyzmq ≥ 13. To use the demo notebooks, you will also need tornado ≥ 4. I will also make use of numpy and matplotlib. If you have Canopy or Anaconda, you already have all of these.

Quick one-line install for IPython and its dependencies:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

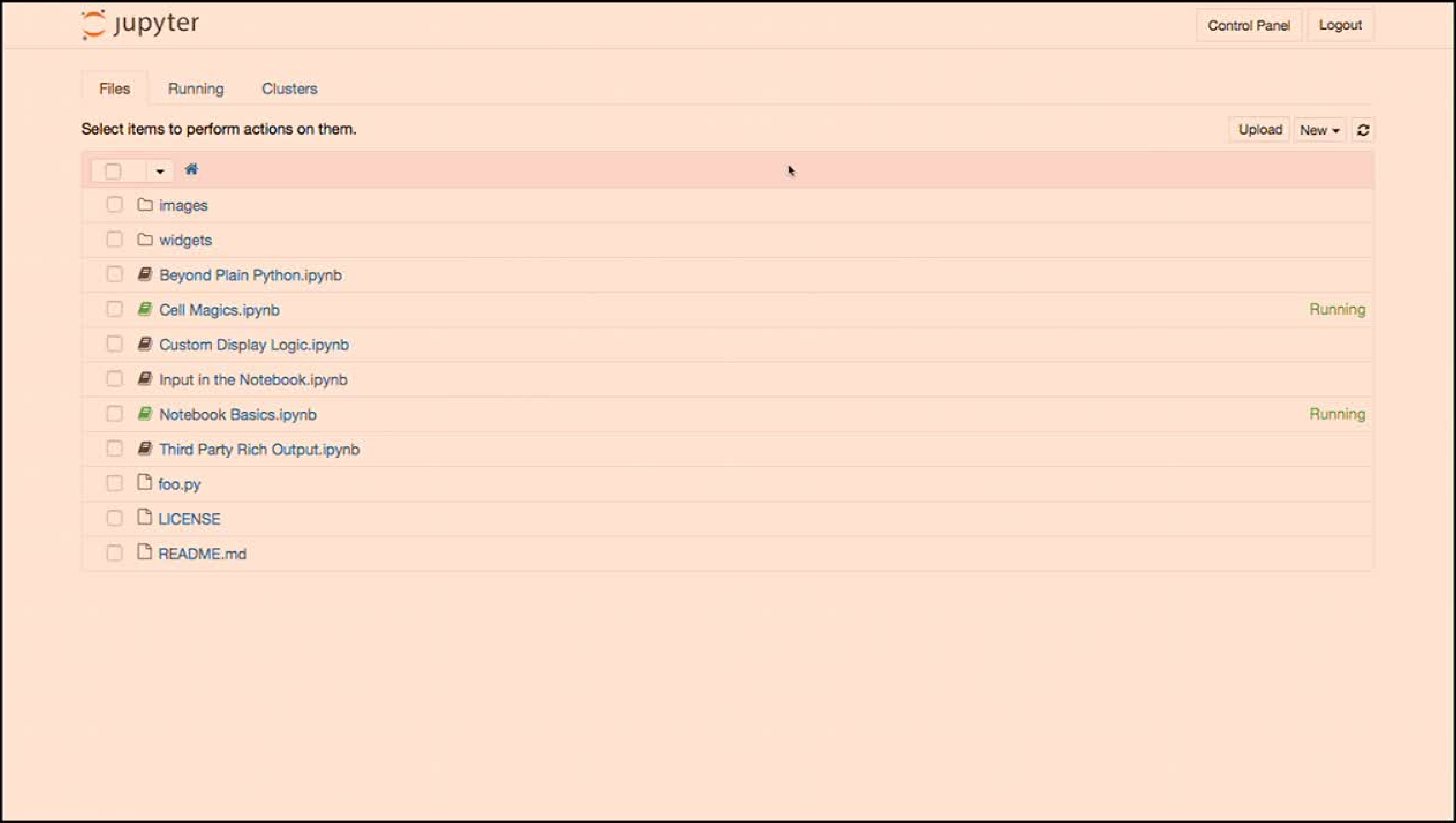
Motivating example: Parallel image processing with scikit-image

To get a sense of what IPython.parallel might be used for, we start with an example of some batch processing of image files with scikit-image. We will revisit pieces of this example as we learn about the different components of IPython.

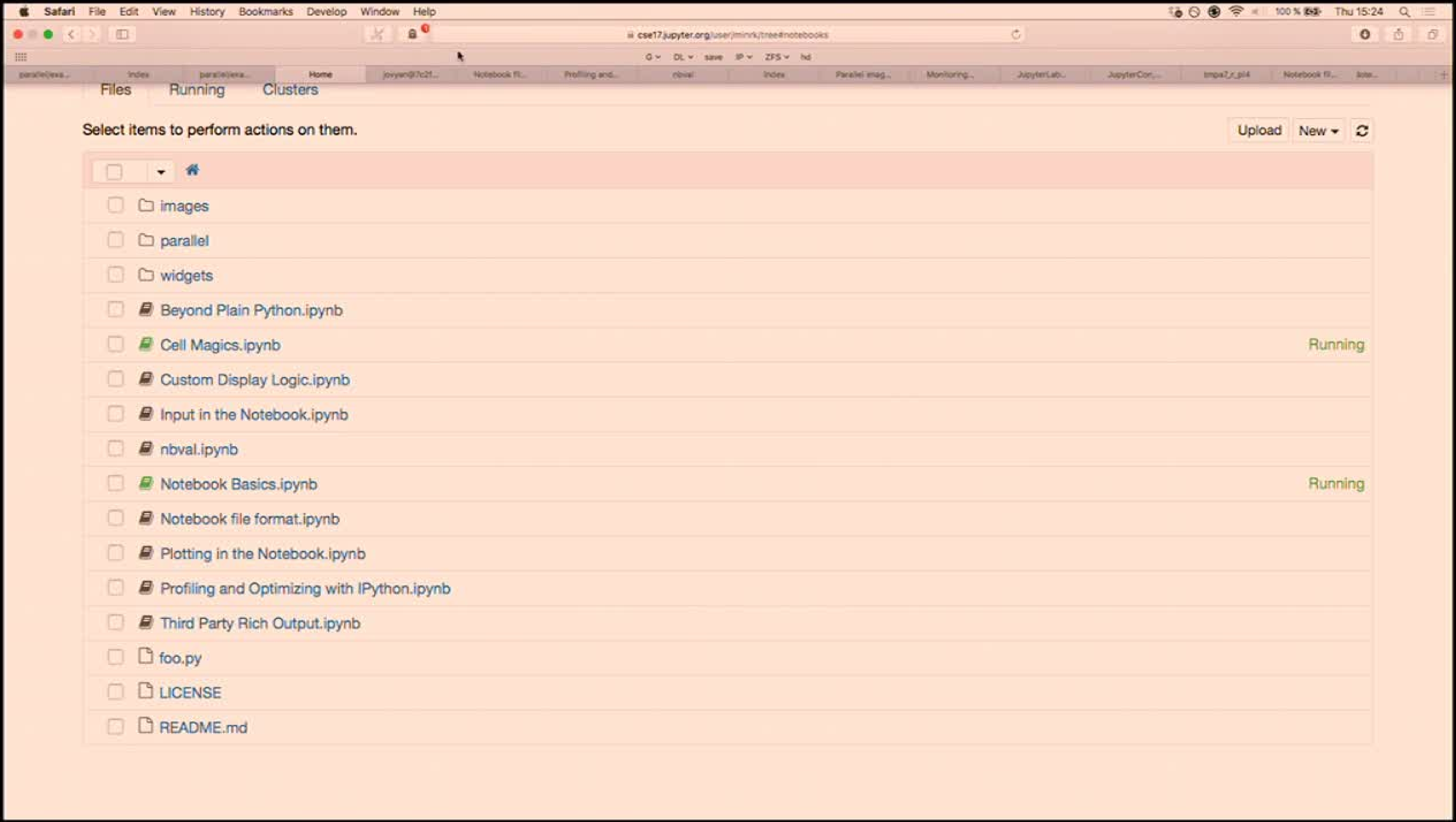
You can download images with this notebook, or get a zip here, or find any images on your computer.

In [ ]:

```
import sys
import requests
from zipfile import ZipFile, BadZipFile
from ipywidgets import IntProgress
from IPython.display import display

images = os.path.join('..', 'images')
images_url = "https://s3.amazonaws.com/ipython-parallel-data/images.zip"


def download_images():
    r = requests.get(images_url, stream=True)
    content_length = r.headers.get('content-length')
    print("Downloading images")
    sys.stdout.flush()
    p = IntProgress(max=content_length)
    display(p)
    with open(images_zip, 'wb') as f:
        for chunk in r.iter_content(chunk_size=8192):
```


jupyter
Control Panel
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
images
widgets
Beyond Plain Python.ipynb
Cell Magics.ipynb
Running
Custom Display Logic.ipynb
Input in the Notebook.ipynb
Notebook Basics.ipynb
Running
Third Party Rich Output.ipynb
foo.py
LICENSE
README.md

```
 create mode 100644 parallel/examples/gutenberg/bryant-stories.txt
 create mode 100644 parallel/examples/gutenberg/burgess-busterbrown.txt
 create mode 100644 parallel/examples/gutenberg/carroll-alice.txt
 create mode 100644 parallel/examples/gutenberg/chesterton-ball.txt
 create mode 100644 parallel/examples/gutenberg/chesterton-brown.txt
 create mode 100644 parallel/examples/gutenberg/chesterton-thursday.txt
 create mode 100644 parallel/examples/gutenberg/edgeworth-parents.txt
 create mode 100644 parallel/examples/gutenberg/melville-moby_dick.txt
 create mode 100644 parallel/examples/gutenberg/milton-paradise.txt
 create mode 100644 parallel/examples/gutenberg/shakespeare-caesar.txt
 create mode 100644 parallel/examples/gutenberg/shakespeare-hamlet.txt
 create mode 100644 parallel/examples/gutenberg/shakespeare-macbeth.txt
 create mode 100644 parallel/examples/gutenberg/whitman-leaves.txt
 create mode 100644 parallel/examples/memmap.ipynb
 create mode 100644 parallel/examples/wikipedia/Wikipedia.ipynb
 create mode 100644 parallel/examples/wikipedia/eventful_dict.py
 create mode 100644 parallel/examples/wikipedia/eventful_graph.py
 create mode 100644 parallel/examples/wikipedia/widget_forcedirectedgraph.js
 create mode 100644 parallel/examples/wikipedia/widget_forcedirectedgraph.py
 create mode 100644 "parallel/exercises/Monte Carlo \317\200.ipynb"
 create mode 100644 parallel/exercises/Remote Iteration.ipynb
 create mode 100644 parallel/figs/allconnections.png
 create mode 100644 parallel/figs/darts.png
 create mode 100644 parallel/figs/latency.png
 create mode 100644 parallel/figs/latency2.png
 create mode 100644 parallel/figs/map.png
 create mode 100644 parallel/figs/throughput1.png
 create mode 100644 parallel/figs/throughput2.png
 create mode 100644 parallel/figs/wideView.png
 create mode 100644 parallel/hints.py
 create mode 100644 parallel/images_common.py
 create mode 100644 parallel/soln/matmul.py
 create mode 100644 parallel/soln/mcpi.py
 create mode 100644 parallel/soln/nestedloop.py
 create mode 100644 parallel/soln/ngrams.py
 create mode 100644 parallel/soln/remote_iter.py
 create mode 100644 parallel/soln/remote_iter_hint.py
 create mode 100644 parallel/tutorial/All Together.ipynb
 create mode 100644 parallel/tutorial/Load-Balancing.ipynb
 create mode 100644 parallel/tutorial/Multiplexing.ipynb
 create mode 100644 parallel/tutorial/Parallel Magics.ipynb
 create mode 100644 parallel/tutorial/Remote Execution.ipynb
 create mode 100644 parallel/tutorial/myscript.py
jovyan@7c2f72f3fd18:~/work/ipython-cse17$
```


Safari File Edit View History Bookmarks Develop Window Help
Thu 15:24
cse17.jupyter.org/user/minrk/tree#notebooks
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
images
parallel
widgets
Beyond Plain Python.ipynb
Cell Magics.ipynb
Running
Custom Display Logic.ipynb
Input in the Notebook.ipynb
nbval.ipynb
Notebook Basics.ipynb
Running
Notebook file format.ipynb
Plotting in the Notebook.ipynb
Profiling and Optimizing with IPython.ipynb
Third Party Rich Output.ipynb
foo.py
LICENSE
README.md

```
 create mode 100644 parallel/figs/wideview.png
 create mode 100644 parallel/hints.py
 create mode 100644 parallel/images_common.py
 create mode 100644 parallel/soln/matmul.py
 create mode 100644 parallel/soln/mcpi.py
 create mode 100644 parallel/soln/nestedloop.py
 create mode 100644 parallel/soln/ngrams.py
 create mode 100644 parallel/soln/remote_iter.py
 create mode 100644 parallel/soln/remote_iter_hint.py
 create mode 100644 parallel/tutorial/All Together.ipynb
 create mode 100644 parallel/tutorial/Load-Balancing.ipynb
 create mode 100644 parallel/tutorial/Multiplexing.ipynb
 create mode 100644 parallel/tutorial/Parallel Magics.ipynb
 create mode 100644 parallel/tutorial/Remote Execution.ipynb
 create mode 100644 parallel/tutorial/myscript.py
jovyan@7c2f72f3fd18:~/work/ipython-cse17$ ipcluster start
bash: ipcluster: command not found
jovyan@7c2f72f3fd18:~/work/ipython-cse17$ pip install ipyparallel
Collecting ipyparallel
  Downloading ipyparallel-6.0.2-py2.py3-none-any.whl (190kB)
    100% |████████████████████████████████| 194kB 2.1MB/s
Requirement already satisfied (use --upgrade to upgrade): decorator in /opt/conda/lib/python3.5/site-packages (from ipyparallel)
Requirement already satisfied (use --upgrade to upgrade): ipykernel in /opt/conda/lib/python3.5/site-packages (from ipyparallel)
Requirement already satisfied (use --upgrade to upgrade): ipython>=4 in /opt/conda/lib/python3.5/site-packages (from ipyparallel)
Requirement already satisfied (use --upgrade to upgrade): python-dateutil>=2.1 in /opt/conda/lib/python3.5/site-packages (from ipyp
arallel)
Requirement already satisfied (use --upgrade to upgrade): tornado>=4 in /opt/conda/lib/python3.5/site-packages (from ipyparallel)
Requirement already satisfied (use --upgrade to upgrade): ipython-genutils in /opt/conda/lib/python3.5/site-packages (from ipyparal
lel)
Requirement already satisfied (use --upgrade to upgrade): pyzmq>=13 in /opt/conda/lib/python3.5/site-packages (from ipyparallel)
Requirement already satisfied (use --upgrade to upgrade): jupyter-client in /opt/conda/lib/python3.5/site-packages (from ipyparalle
l)
Requirement already satisfied (use --upgrade to upgrade): six>=1.5 in /opt/conda/lib/python3.5/site-packages (from python-dateutil>
=2.1->ipyparallel)
Installing collected packages: ipyparallel
Successfully installed ipyparallel-6.0.2
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
jovyan@7c2f72f3fd18:~/work/ipython-cse17$ ipcluster start -n 4
2017-03-02 20:24:47.006 [IPClusterStart] Starting ipcluster with [daemon=False]
2017-03-02 20:24:47.007 [IPClusterStart] Creating pid file: /home/jovyan/.ipython/profile_default/pid/ipcluster.pid
2017-03-02 20:24:47.007 [IPClusterStart] Starting Controller with LocalControllerLauncher
2017-03-02 20:24:48.011 [IPClusterStart] Starting 4 Engines with LocalEngineSetLauncher
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help Python 3

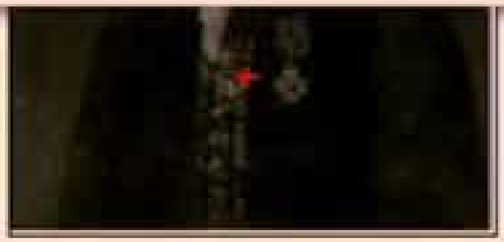
Markdown CellToolbar

```
pictures = []
for directory, subdirs, files in os.walk(pictures_dir):
    for fname in files:
        if fname.lower().endswith(('.jpg', '.png')):
            pictures.append(os.path.join(directory, fname))

pictures[:5]
```

Out[9]:
```
['../images/portrait/Portrait_d%27homme.png',
 '../images/portrait/Portrait_AB.jpg',
 '../images/portrait/Ledoux_portrait.jpg',
 '../images/portrait/PolPlancon_Portrait.jpg',
 '../images/portrait/Richard_Portrait.jpg']
```

Let's test our function locally, to see what it does.

In [10]:
```
for p in pictures[:3]:
    img, corners = find_corners(p)
    plot_corners(img, corners)
```



jupyter Parallel image processing (unsaved changes)

Control Panel Logout

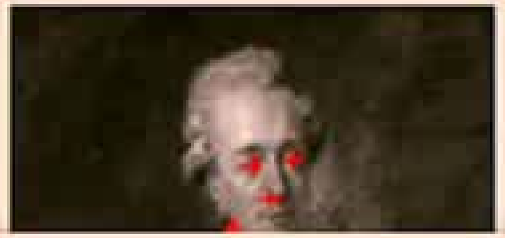
File Edit View Insert Cell Kernel Help

Python 3

Markdown CellToolbar

Let's test our function locally, to see what it does.

In [10]:
```
for p in pictures[:3]:
    img, corners = find_corners(p)
    plot_corners(img, corners)
```



Now in parallel

First, we connect our parallel Client

In [ ]:
```
rc = parallel.Client()
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help Python 3

Markdown CellToolbar

Now in parallel

First, we connect our parallel Client

In [ ]:
```
rc = parallel.Client()
all_engines = rc[:]
view = rc.load_balanced_view()
```

Then we initialize the namespace on all of the engines with imports

In [ ]:
```
%px import os; os.chdir("{os.getcwd()}")
```

In [ ]:
```
%%px
%matplotlib inline
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
for directory, subdirs, files in os.walk(pictures_dir):
    for fname in files:
        if fname.lower().endswith(('.jpg', '.png')):
            pictures.append(os.path.join(directory, fname))

pictures[:5]
```

Out[9]:

```
['../images/portrait/Portrait_d%27homme.png',
 '../images/portrait/Portrait_AB.jpg',
 '../images/portrait/Ledoux_portrait.jpg',
 '../images/portrait/PolPlancon_Portrait.jpg',
 '../images/portrait/Richard_Portrait.jpg']
```

Let's test our function locally, to see what it does.

In [10]:

```
for p in pictures[:3]:
    img, corners = find_corners(p)
    plot_corners(img, corners)
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
    plot_corners(img, corners, show=False)
    fig = plt.gcf()
    pngdata = print_figure(fig)
    plt.close(fig)
    return pngdata
```

In [9]:
```
import os
pictures_dir = os.path.join('..', 'images', 'portrait')

pictures = []
for directory, subdirs, files in os.walk(pictures_dir):
    for fname in files:
        if fname.lower().endswith(('.jpg', '.png')):
            pictures.append(os.path.join(directory, fname))

pictures[:5]
```

Out[9]:
```
['../images/portrait/Portrait_d%27homme.png',
 '../images/portrait/Portrait_AB.jpg',
 '../images/portrait/Ledoux_portrait.jpg',
 '../images/portrait/PolPlancon_Portrait.jpg',
 '../images/portrait/Richard_Portrait.jpg']
```

Let's test our function locally, to see what it does.

In [10]:
```
for p in pictures[:3]:
    img, corners = find_corners(p)
    plot_corners(img, corners)
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

Let's test our function locally, to see what it does.

In [10]:
```
for p in pictures[:3]:
    img, corners = find_corners(p)
    plot_corners(img, corners)
```

Now in parallel

First, we connect our parallel Client

In [ ]:
```
rc = parallel.Client()
```

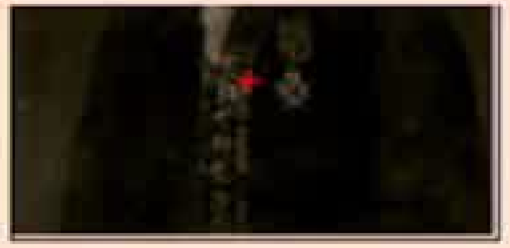
jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar



Now in parallel

First, we connect our parallel Client

In [11]:
```
rc = parallel.Client()
all_engines = rc[:]
view = rc.load_balanced_view()
```

Then we initialize the namespace on all of the engines with imports

In [12]:
```
%px import os; os.chdir("{os.getcwd()}")
```

In [ ]:
```
%%px
%matplotlib inline
import matplotlib.pyplot as plt

from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

Now in parallel

First, we connect our parallel Client

In [11]:
```
rc = parallel.Client()
all_engines = rc[:]
view = rc.load_balanced_view()
```

Then we initialize the namespace on all of the engines with imports

In [12]:
```
%px import os; os.chdir("{os.getcwd()}")
```

In [ ]:
```
%%px
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

Now in parallel

First, we connect our parallel Client

In [11]:
```
rc = parallel.Client()
all_engines = rc[:]
view = rc.load_balanced_view()
```

Then we initialize the namespace on all of the engines with imports

In [*]:
```
%px import os; os.chdir("{os.getcwd()}")
```

In [ ]:
```
%%px
%matplotlib inline
import matplotlib.pyplot as plt

from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

In [14]:

```
%%px
%matplotlib inline
import matplotlib.pyplot as plt

from skimage.io import imread
from skimage.feature import corner_harris, corner_peaks
```

and make sure some functions are defined everywhere (this is only necessary for the `contours_in_url` case)

In [15]:

```
all_engines.push(dict(
    plot_corners=plot_corners,
    find_corners=find_corners,
))
```

Out[15]:

```
<AsyncResult: _push>
```

Now we can iterate through all of our pictures, and detect and display any corners we find

In [ ]:

```
from IPython.display import display, Image

amr = view.map_async(get_corners_image, pictures[:20], ordered=False)
for pngdata in amr:
    display(Image(data=pngdata))
```

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
from skimage.feature import corner_harris, corner_peaks
```

and make sure some functions are defined everywhere (this is only necessary for the `contours_in_url` case)

In [15]:
```
all_engines.push(dict(
    plot_corners=plot_corners,
    find_corners=find_corners,
))
```

Out[15]: `<AsyncResult: _push>`

Now we can iterate through all of our pictures, and detect and display any corners we find

In [*]:
```
from IPython.display import display, Image

amr = view.map_async(get_corners_image, pictures[:20], ordered=False)
for pngdata in amr:
    display(Image(data=pngdata))
```

In [ ]:

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

In [ ]:

jupyter Parallel image processing (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

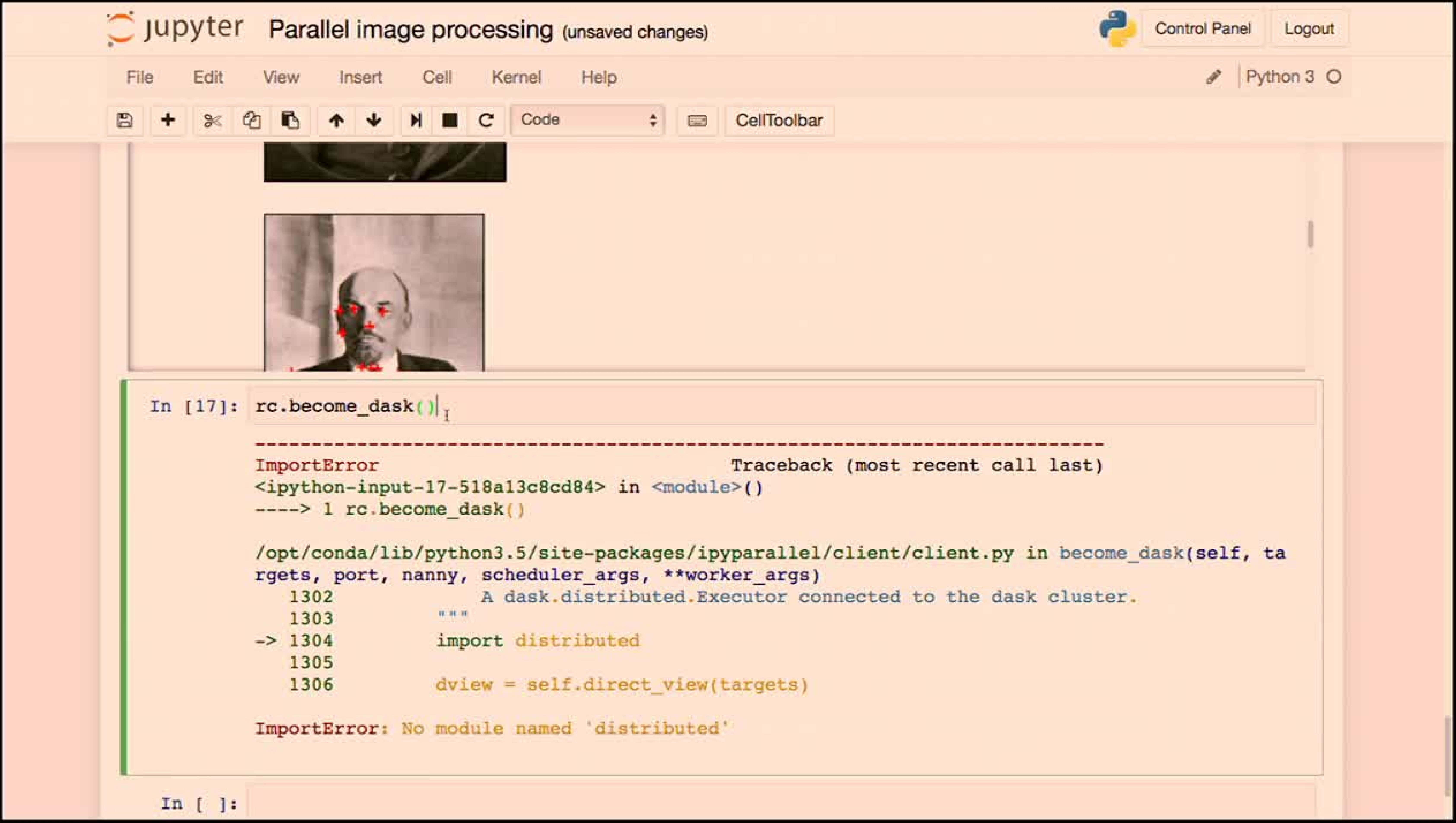
Python 3

Code CellToolbar

In [17]:
```
rc.become_dask()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-17-518a13c8cd84> in <module>()
----> 1 rc.become_dask()

/opt/conda/lib/python3.5/site-packages/ipyparallel/client/client.py in become_dask(self, ta
rgets, port, nanny, scheduler_args, **worker_args)
   1302             A dask.distributed.Executor connected to the dask cluster.
   1303         """
-> 1304         import distributed
   1305 
   1306         dview = self.direct_view(targets)

ImportError: No module named 'distributed'
```

In [ ]:

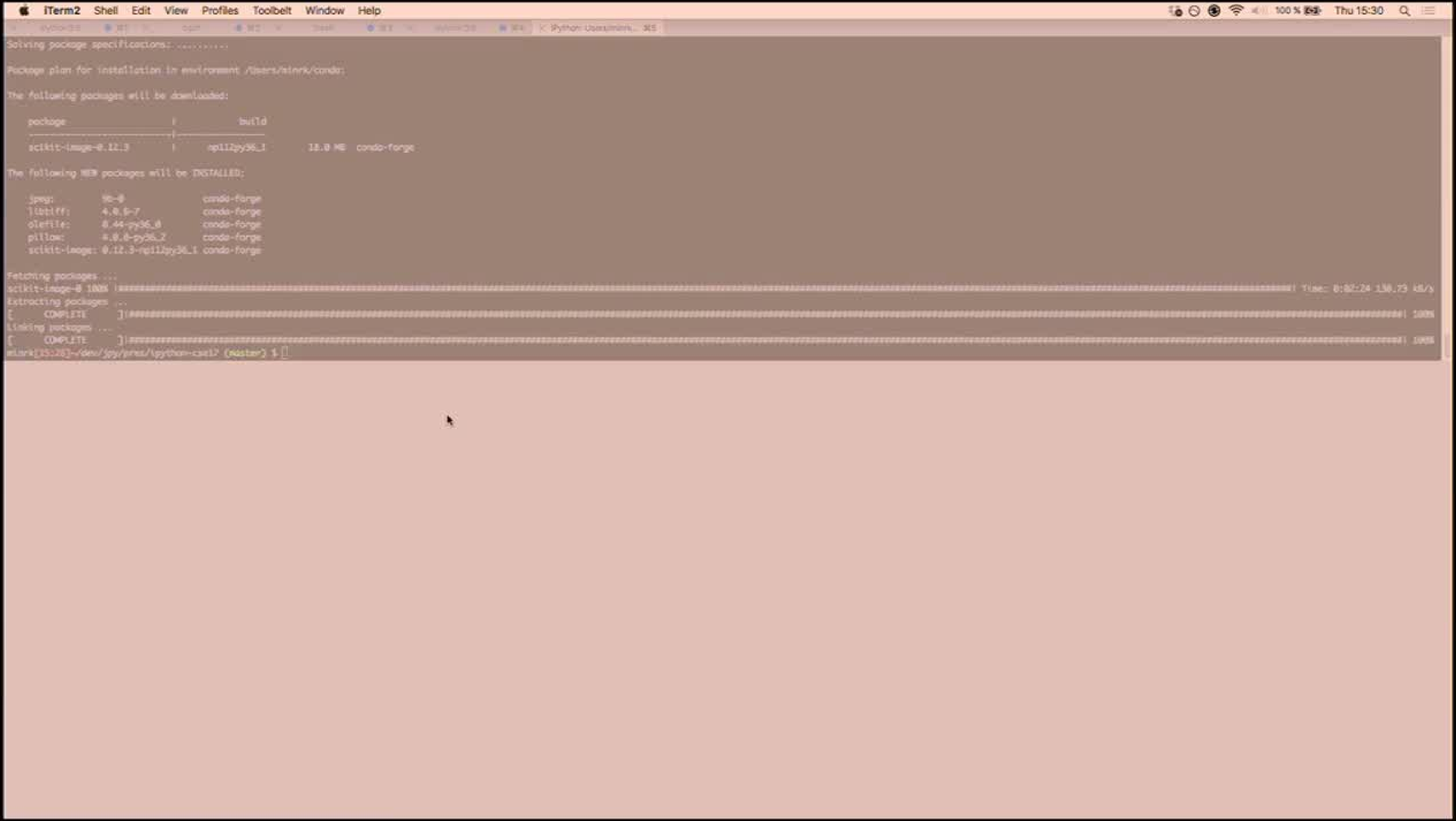


jupyter
Parallel image processing (unsaved changes)
Control Panel
Logout
File
Edit
View
Insert
Cell
Kernel
Help
Python 3
Code
CellToolbar
In [17]: rc.become_dask()

ImportError Traceback (most recent call last)
<ipython-input-17-518a13c8cd84> in <module>()
----> 1 rc.become_dask()
/opt/conda/lib/python3.5/site-packages/ipyparallel/client/client.py in become_dask(self, targets, port, nanny, scheduler_args, **worker_args)
1302 A dask.distributed.Executor connected to the dask cluster.
1303 """
-> 1304 import distributed
1305
1306 dview = self.direct_view(targets)
ImportError: No module named 'distributed'
In [ ]:

iTerm2 Shell Edit View Profiles Toolbelt Window Help Thu 15:30

```
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0 100% |################################################################| Time: 0:02:24 130.73 kB/s
Extracting packages ...
[      COMPLETE      ]|###################################################################| 100%
Linking packages ...
[      COMPLETE      ]|###################################################################| 100%
minrk[15:28]~/dev/jpy/pres/ipython-cal7 (master) $
```


iTerm2 Shell Edit View Profiles Toolbelt Window Help
100 % Thu 15:30
Solving package specifications: ....
Package plan for installation in environment /Users/minrk/conda:
The following packages will be downloaded:
package | build
scikit-image-0.12.3 | np112py36_1 18.0 MB conda-forge
The following NEW packages will be INSTALLED:
jpeg: 9b-0 conda-forge
libtiff: 4.0.6-7 conda-forge
olefile: 0.44-py36_0 conda-forge
pillow: 4.0.0-py36_2 conda-forge
scikit-image: 0.12.3-np112py36_1 conda-forge
Fetching packages ...
scikit-image-0 100% | Time: 0:02:24 130.73 kB/s
Extracting packages ...
[COMPLETE]| 100%
Linking packages ...
[COMPLETE]| 100%
(master) $

In [17]:
```
rc.become_dask()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
<ipython-input-17-518a13c8cd84> in <module>()
----> 1 rc.become_dask()

/opt/conda/lib/python3.5/site-packages/ipyparallel/client/client.py in become_dask(self, ta
rgets, port, nanny, scheduler_args, **worker_args)
   1302             A dask.distributed.Executor connected to the dask cluster.
   1303         """
-> 1304         import distributed
   1305 
   1306         dview = self.direct_view(targets)

ImportError: No module named 'distributed'
```

In [ ]:

1. Running a notebook manually will likely change the output

In [12]:
```
!jupyter nbconvert --to pdf "nbval.ipynb"
```

```
[NbConvertApp] Converting notebook nbval.ipynb to pdf
[NbConvertApp] Support files will be in nbval_files/
[NbConvertApp] Making directory nbval_files
[NbConvertApp] Writing 30279 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.t
ex']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely becau
se there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 43187 bytes to nbval.pdf
```

In [13]:
```
!open nbval.pdf
```

In [ ]:

Safari File Edit View History Bookmarks Develop Window Help

cse17.jupyter.org/user/minrk/terminals/1

```
    sqlite:           3.?.2-0           --> 3.13.0-0
    xz:               5.0.5-1           --> 5.2.2-0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6. 100% |############################################################| Time: 0:00:00 377.54 kB/s
libffi-3.2.1-1 100% |############################################################| Time: 0:00:00   8.55 MB/s
sqlite-3.13.0- 100% |############################################################| Time: 0:00:00  24.37 MB/s
xz-5.2.2-0.tar 100% |############################################################| Time: 0:00:00  11.76 MB/s
libtiff-4.0.6- 100% |############################################################| Time: 0:00:00  28.87 MB/s
python-3.5.3-0 100% |############################################################| Time: 0:00:00  35.46 MB/s
click-6.7-py35 100% |############################################################| Time: 0:00:00   7.63 MB/s
cloudpickle-0. 100% |############################################################| Time: 0:00:00   5.32 MB/s
heapdict-1.0.0 100% |############################################################| Time: 0:00:00   5.84 MB/s
idna-2.2-py35_ 100% |############################################################| Time: 0:00:00   9.19 MB/s
locket-0.2.0-p 100% |############################################################| Time: 0:00:00   6.17 MB/s
msgpack-python 100% |############################################################| Time: 0:00:00   1.73 MB/s
psutil-5.1.3-p 100% |############################################################| Time: 0:00:00  11.29 MB/s
pyasn1-0.1.9-p 100% |############################################################| Time: 0:00:00   7.26 MB/s
pycparser-2.17 100% |############################################################| Time: 0:00:00  13.08 MB/s
requests-2.12. 100% |############################################################| Time: 0:00:00  24.22 MB/s
ruamel_yaml-0. 100% |############################################################| Time: 0:00:00  17.20 MB/s
sortedcontaine 100% |############################################################| Time: 0:00:00   1.27 MB/s
tblib-1.3.0-py 100% |############################################################| Time: 0:00:00  10.01 MB/s
toolz-0.8.2-py 100% |############################################################| Time: 0:00:00   8.38 MB/s
cffi-1.9.1-py3 100% |############################################################| Time: 0:00:00  10.28 MB/s
chest-0.2.3-py 100% |############################################################| Time: 0:00:00   8.98 MB/s
partd-0.3.7-py 100% |############################################################| Time: 0:00:00   7.51 MB/s
sortedcollecti 100% |############################################################| Time: 0:00:00   3.99 MB/s
zict-0.1.1-py3 100% |############################################################| Time: 0:00:00   5.39 MB/s
cryptography-1 100% |############################################################| Time: 0:00:00  24.62 MB/s
pandas-0.18.0- 100% |############################################################| Time: 0:00:00  20.75 MB/s
dask-0.13.0-py 100% |############################################################| Time: 0:00:00  17.65 MB/s
pyopenssl-16.2 100% |############################################################| Time: 0:00:00   7.94 MB/s
conda-4.3.13-p 100% |############################################################| Time: 0:00:00  18.79 MB/s
distributed-1. 100% |############################################################| Time: 0:00:00   2.59 MB/s
Extracting packages ...
[      COMPLETE      ]|###############################################################################| 100%
Unlinking packages ...
[      COMPLETE      ]|###############################################################################| 100%
Linking packages ...
[      COMPLETE      ]|###############################################################################| 100%
jovyan@7c2f72f3fd18:~/work/ipython-cse17$
```

Interactive (parallel) Python

Installation and dependencies

You will need ipyparallel >= 5.x, and pyzmq ≥ 13. To use the demo notebooks, you will also need tornado ≥ 4. I will also make use of numpy and matplotlib. If you have Canopy or Anaconda, you already have all of these.

Quick one-line install for IPython and its dependencies:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

Optional dependencies: I will use NetworkX for one demo, and `scikit-image` for another, but they are not critical. Both packages are in in Anaconda.

For the image-related demos, all you need are some images on your computer. The notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworthy, we have bundled some images here

jupyter Parallel Magics (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help Python 3

Markdown CellToolbar

Using Parallel Magics

IPython has a few magics for working with your engines.

This assumes you have started an IPython cluster, either with the notebook interface, or the `ipcluster/controller/engine` commands.

In [1]:
```
import ipyparallel as parallel
rc = parallel.Client()
rc.block = True
dv = rc[:]
rc.ids
```

Out[1]: `[0, 1, 2, 3]`

In [2]:
```
dv.apply(lambda x: x * 2, 5)
```

Out[2]: `[10, 10, 10, 10]`

Creating a Client registers the parallel magics `%px`, `%%px`, `%pxresult`, `pxconfig`, and `%autopx`.
These magics are initially associated with a DirectView always associated with all currently registered engines.

Now we can execute code remotely with `%px`:

In [24]:
```
%px a=5
```

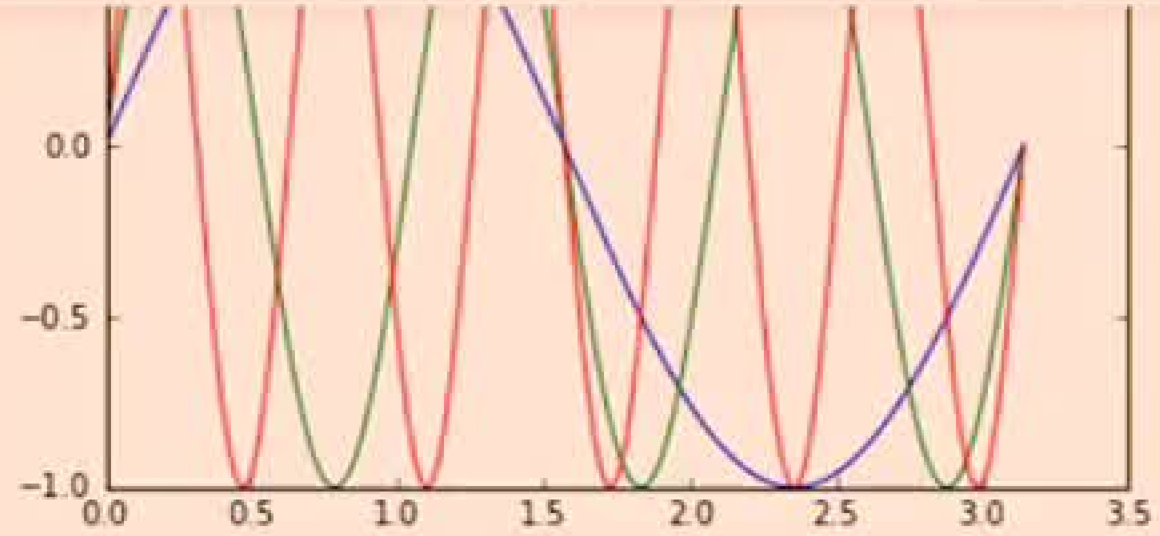
jupyter Parallel Magics (autosaved)

Control Panel Logout

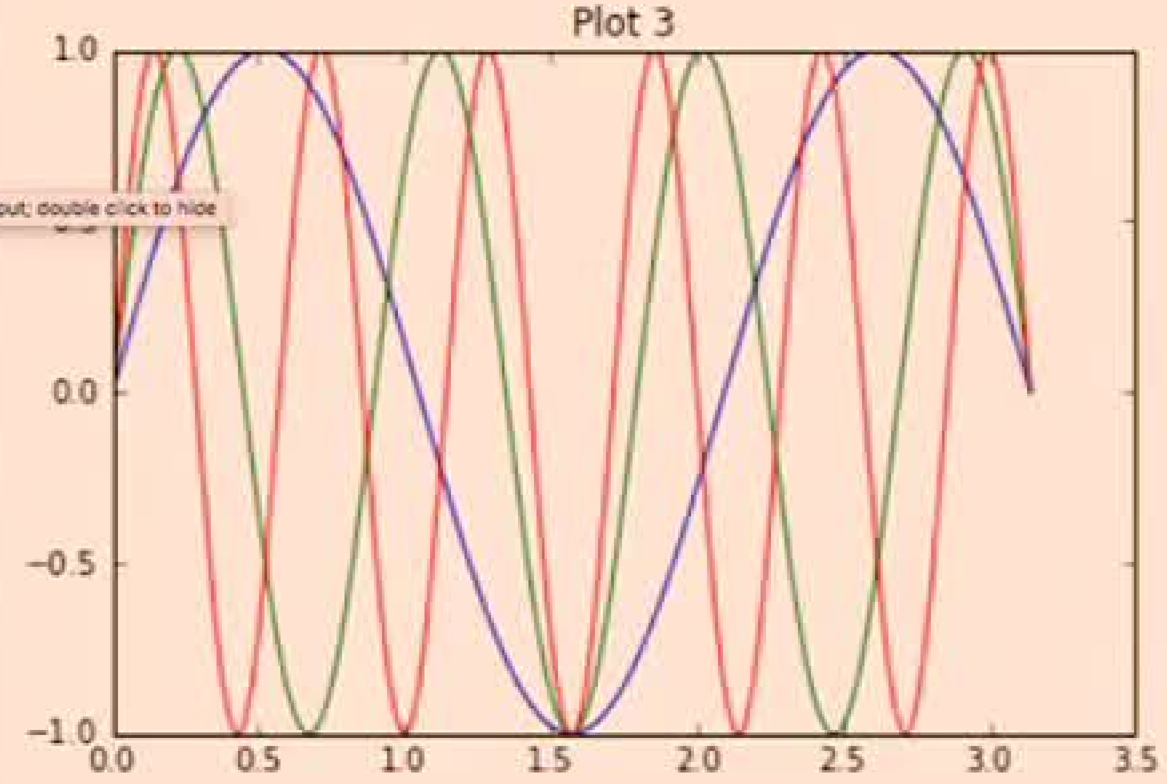
File Edit View Insert Cell Kernel Help

Python 3

Markdown CellToolbar




```
[output:3]
```



click to scroll output; double click to hide

Jupyter Parallel Magics (unsaved changes)

File Edit View Insert Cell Kernel Help

```
[stdout:2] Couldn't find program: 'ruby'
[stdout:3] Couldn't find program: 'ruby'
```

In [20]:
```
dv.scatter('rank', dv.targets, flatten=True)
```

In [21]:
```
%%px
%%timeit
from numpy.random import random
from numpy.linalg import norm
N = 100 * (rank + 1)
A = random((N,N))
norm(A, 2)
```

```
[stdout:0] 100 loops, best of 3: 4.1 ms per loop
[stdout:1] 100 loops, best of 3: 7.87 ms per loop
[stdout:2]
The slowest run took 4.35 times longer than the fastest. This could mean that an intermedia
te result is being cached.
10 loops, best of 3: 38 ms per loop
[stdout:3] 1 loop, best of 3: 199 ms per loop
```

Debugging Engines

Since the IPython engine is precisely the same object used for the notebook and qtconsole, we can connect other fronteds directly to the engine.

The first step is to bind the engine's sockets, so its connection pattern looks like a regular kernel

jupyter Parallel Magics (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [24]:
```
%px %qtconsole
```

In [ ]:

jupyter Parallel Magics (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [24]:
```
%px %qtconsole
```

In [ ]:

```
If the raised exception doesn't match the stored exception, we get a failure

---------------------------------------------------------------------------

4
```

```
RuntimeError                              Traceback (most recent call last)

<ipython-input-3-32dcc1c70a4e> in <module>()
      1 # NBVAL_RAISES_EXCEPTION
      2 print("If the raised exception doesn't match the stored exception, we get a failure"
----> 3 raise RuntimeError("Foo")


RuntimeError: Foo
```

```
In [2]: # NBVAL_IGNORE_OUTPUT
```

python File Edit View Kernel Window Help   100 %   Thu 15:39

Jupyter QtConsole

```
ipdb> btr
*** NameError: name 'btr' is not
defined

ipdb> bt
> <ipython-
input-16-05c9758a9c21>(1)<module>(
)
----> 1 1/0


ipdb>
```

If the raised exception doesn't match the stored exception, we get a failure

---

python File Edit View Kernel Window Help   100 % Thu 15:39

Jupyter QtConsole

```
ipdb> btr
*** NameError: name 'btr' is not
defined

ipdb> bt
> <ipython-
input-16-05c9758a9c21>(1)<module>(
)
----> 1 1/0


ipdb>
```

If the raised exception doesn't match the stored exception, we get a failure

---

jupyter Parallel Magics (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help

Python 3

Code CellToolbar

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [25]:
```
%px %qtconsole
```

In [ ]:

jupyter Parallel Magics (unsaved changes)

Control Panel Logout

File Edit View Insert Cell Kernel Help Python 3

Code CellToolbar

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [25]:
```
%px %qtconsole
```

In [ ]:

```
*** NameError: name 'btr' is not
defined

ipdb> bt
> <ipython-
input-16-05c9758a9c21>(1)<module>(
)
----> 1 1/0


ipdb>
```

If the raised exception doesn't match the stored exception, we get a failure

---

iTerm2 Shell Edit View Profiles Toolbelt Window Help 100 % Thu 15:40

python3.6 ⌘1 bash ⌘2 bash ⌘3 python3.6 ⌘4 IPython: Users/minrk... ⌘5

```
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0 100% |################################| Time: 0:02:24 130.73 kB/s
Extracting packages ...
[      COMPLETE      ]|################################| 100%
Linking packages ...
[      COMPLETE      ]|################################| 100%
minrk[15:26]~/dev/jpy/pres/ipython-cse17 (master) $
```

```
)
----> 1 1/0


ipdb>
```

If the raised exception doesn't match the stored exception, we get a failure

---

4

Thu 15:40

Jupyter QtConsole

```
r: name 'btr' is not

9758a9c21>(1)<module>(
```

```
et a failure

-------------------
```

jupyter Parallel Magics (autosaved)

File Edit View Insert Cell Kernel Help

Code

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> i
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> i
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by ze
```

Now we can connect a qtconsole to the eng

```
In [25]: %px %qtconsole
```

```
In [ ]:
```

Jupyter Parallel Magics (autosaved)

Control Panel Logout

File Edit View Insert Cell Kernel Help Notebook saved Python 3

Code CellToolbar

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [25]:
```
%px %qtconsole
```

In [ ]:

iTerm2 Shell Edit View Profiles Toolbelt Window Help

```
Solving package specifications: ..........

Package plan for installation in environment /Users/minrk/conda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    scikit-image-0.12.3        |      np112py36_1        18.0 MB  conda-forge

The following NEW packages will be INSTALLED:

    jpeg:         9b-0                 conda-forge
    libtiff:      4.0.6-7              conda-forge
    olefile:      0.44-py36_0          conda-forge
    pillow:       4.0.0-py36_2         conda-forge
    scikit-image: 0.12.3-np112py36_1 conda-forge

Fetching packages ...
scikit-image-0 100% |################################################################| Time: 0:02:24 130.73 kB/s
Extracting packages ...
[      COMPLETE      ]|################################################################| 100%
Linking packages ...
[      COMPLETE      ]|################################################################| 100%
minrk[15:26]~/dev/jpy/pres/ipython-cse17 (master) $
```

```
)
----> 1 1/0

ipdb>
```

If the raised exception doesn't match the stored exception, we get a failure

4

```
----> 8 bar(1)
<ipython-input-20-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-20-64d85adf5a05> in foo(a, b)
      1
      2 def foo(a, b):
----> 3     return a/(1-b)
      4
      5 def bar(b):
ZeroDivisionError: division by zero
```

Now we can connect a qtconsole to the engine(s)

In [25]:
```
%px %qtconsole
```

In [ ]:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

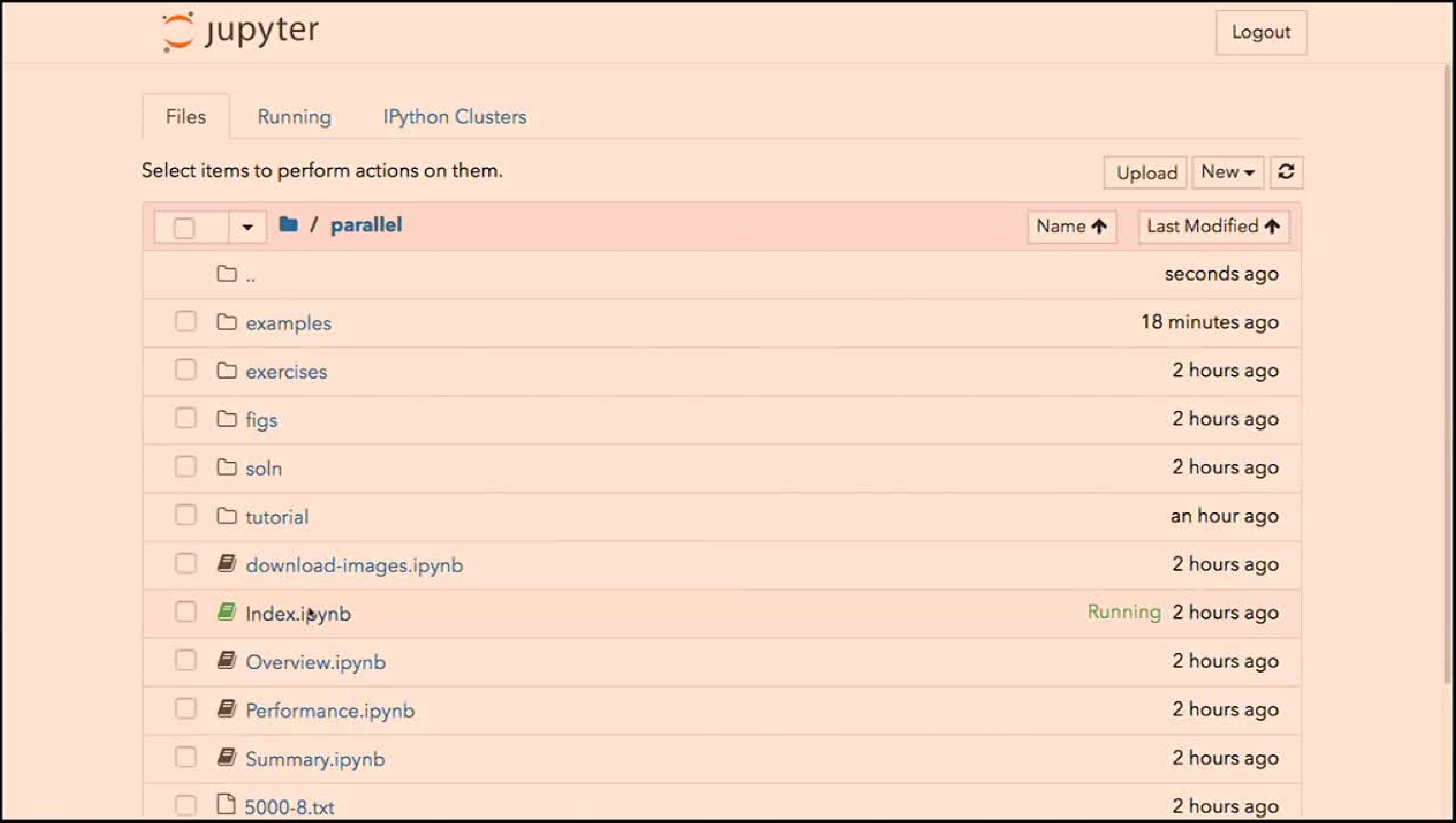
For those who prefer pip or otherwise manual package installation, the following packages will be used:

ipython ipyparallel numpy matplotlib networkx scikit-image requests beautifulsoup mpi4py

Optional dependencies: I will use NetworkX for one demo, and **`scikit-image`** for another, but they are not critical. Both packages are in in Anaconda.

For the image-related demos, all you need are some images on your computer. The notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline


jupyter
Logout
Files
Running
IPython Clusters
Select items to perform actions on them.
Upload
New
/ parallel
Name
Last Modified
..
seconds ago
examples
18 minutes ago
exercises
2 hours ago
figs
2 hours ago
soln
2 hours ago
tutorial
an hour ago
download-images.ipynb
2 hours ago
Index.ipynb
Running
2 hours ago
Overview.ipynb
2 hours ago
Performance.ipynb
2 hours ago
Summary.ipynb
2 hours ago
5000-8.txt
2 hours ago

Kernel ready Saving every 120s Trusted Python 3

File Edit View Insert Cell Kernel Widgets Help

Interactive (parallel) Python

Installation and dependencies

You will need ipyparallel >= 5.x, and pyzmq ≥ 13. To use the demo notebooks, you will also need tornado ≥ 4. I will also make use of numpy and matplotlib. If you have Canopy or Anaconda, you already have all of these.

Quick one-line install for IPython and its dependencies:

```
pip install ipyparallel
```

Or get everything for the tutorial with conda:

```
conda install anaconda mpi4py
```

Loading [MathJax]/extensions/Safe.js

notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline

Kernel ready | Saving every 120s | Trusted | Python 3

File Edit View Insert Cell Kernel Widgets Help

Using Parallel Magics ¶

IPython has a few magics for working with your engines.

This assumes you have started an IPython cluster, either with the notebook interface, or the `ipcluster/controller/engine` commands.

In [23]:

```
import ipyparallel as parallel
rc = parallel.Client()
rc.block = True
dv = rc[:]
rc.ids
```

Out[23]:

```
[0, 1, 2, 3]
```

Loading [MathJax]/extensions/Safe.js

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Now we can raise an exception on the engines

In [43]:
```
%%px

def foo(a, b):
    return a/(1-b)

def bar(b):
    return foo(2, b)

bar(1)
```

```
[0:execute]:
---------------------------------------------------------------------------
------------ZeroDivisionError                              Traceback
(most recent call last)<ipython-input-31-64d85adf5a05> in <modu
le>()
      6     return foo(2, b)
      7
----> 8 bar(1)
<ipython-input-31-64d85adf5a05> in bar(b)
      4
      5 def bar(b):
----> 6     return foo(2, b)
      7
      8 bar(1)
<ipython-input-31-64d85adf5a05> in foo(a, b)
```

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
from ipyparallel import bind_kernel
bind_kernel()
```

Now we can raise an exception on the engines

In [*]:
```
%%px

def foo(a, b):
    return a/(1-b)

def bar(b):
    return foo(2, b)

bar(1)
```

Now we can connect a qtconsole to the engine(s)

In [44]:
```
%px %qtconsole
```

```
Waiting for connection file: ~/.ipython/profile_default/securit
y/ipcontroller-client.json
```

In [45]:

```python
%%px

def foo(a, b):
    return a/(1-b)

def bar(b):
    return foo(2, b)

bar(1)
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recen
t call last)
<ipython-input-45-8a87cd0fcdb9> in <module>()
----> 1 get_ipython().run_cell_magic('px', '', '\ndef foo(a, b)
:\n    return a/(1-b)\n\ndef bar(b):\n    return foo(2, b)\n\nb
ar(1)')

/Users/minrk/conda/lib/python3.6/site-packages/IPython/core/int
eractiveshell.py in run_cell_magic(self, magic_name, line, cell
)
   2113             magic_arg_s = self.var_expand(line,
stack_depth)
   2114             with self.builtin_trap:
```

notebooks will try to fetch images from Wikimedia Commons, but since the networks can be untrustworty, we have bundled some images here.

Outline

- Motivating Example
- Overview
- Tutorial
    - Remote Execution
    - Multiplexing
    - Load-Balancing
    - Both!
    - Parallel Magics
- Examples
- Exercises

```
    sqlite:              3.9.2-0              --> 3.13.0-0
    xz:                  5.0.5-1              --> 5.2.2-0

Proceed ([y]/n)? y

Fetching packages ...
conda-env-2.6. 100% |###################################################################################| Time: 0:00:00 377.54 kB/s
libffi-3.2.1-1 100% |###################################################################################| Time: 0:00:00   8.55 MB/s
sqlite-3.13.0- 100% |###################################################################################| Time: 0:00:00  24.37 MB/s
xz-5.2.2-0.tar 100% |###################################################################################| Time: 0:00:00  11.76 MB/s
libtiff-4.0.6- 100% |###################################################################################| Time: 0:00:00  28.87 MB/s
python-3.5.3-0 100% |###################################################################################| Time: 0:00:00  35.46 MB/s
click-6.7-py35 100% |###################################################################################| Time: 0:00:00   7.63 MB/s
cloudpickle-0. 100% |###################################################################################| Time: 0:00:00   5.32 MB/s
heapdict-1.0.0 100% |###################################################################################| Time: 0:00:00   5.84 MB/s
idna-2.2-py35_ 100% |###################################################################################| Time: 0:00:00   9.19 MB/s
locket-0.2.0-p 100% |###################################################################################| Time: 0:00:00   6.17 MB/s
msgpack-python 100% |###################################################################################| Time: 0:00:00   1.73 MB/s
psutil-5.1.3-p 100% |###################################################################################| Time: 0:00:00  11.29 MB/s
pyasn1-0.1.9-p 100% |###################################################################################| Time: 0:00:00   7.26 MB/s
pycparser-2.17 100% |###################################################################################| Time: 0:00:00  13.08 MB/s
requests-2.12. 100% |###################################################################################| Time: 0:00:00  24.22 MB/s
ruamel_yaml-0. 100% |###################################################################################| Time: 0:00:00  17.20 MB/s
sortedcontaine 100% |###################################################################################| Time: 0:00:00   1.27 MB/s
tblib-1.3.0-py 100% |###################################################################################| Time: 0:00:00  10.01 MB/s
toolz-0.8.2-py 100% |###################################################################################| Time: 0:00:00   8.38 MB/s
cffi-1.9.1-py3 100% |###################################################################################| Time: 0:00:00  10.28 MB/s
chest-0.2.3-py 100% |###################################################################################| Time: 0:00:00   8.98 MB/s
partd-0.3.7-py 100% |###################################################################################| Time: 0:00:00   7.51 MB/s
sortedcollecti 100% |###################################################################################| Time: 0:00:00   3.99 MB/s
zict-0.1.1-py3 100% |###################################################################################| Time: 0:00:00   5.39 MB/s
cryptography-1 100% |###################################################################################| Time: 0:00:00  24.62 MB/s
pandas-0.18.0- 100% |###################################################################################| Time: 0:00:00  20.75 MB/s
dask-0.13.0-py 100% |###################################################################################| Time: 0:00:00  17.65 MB/s
pyopenssl-16.2 100% |###################################################################################| Time: 0:00:00   7.94 MB/s
conda-4.3.13-p 100% |###################################################################################| Time: 0:00:00  18.79 MB/s
distributed-1. 100% |###################################################################################| Time: 0:00:00   2.59 MB/s
Extracting packages ...
[      COMPLETE      ]|#####################################################################################################| 100%
Unlinking packages ...
[      COMPLETE      ]|#####################################################################################################| 100%
Linking packages ...
[      COMPLETE      ]|#####################################################################################################| 100%
jovyan@7c2f72f3fd18:~/work/ipython-cse17$
```

Safari File Edit View History Bookmarks Develop Window Help

100 % Thu 15:41

jupyter.org

G DL save IP ZFS hd

Magics | Notebook fi... | Profiling and... | nbval | Index | Image proce... | Monitoring... | JupyterLab... | JupyterCon... | lmpa7_r_pi4 | Notebook fi... | Jupyter Note... | Jupyter Note... | Console com... | Favorites

G DL save IP ZFS hd

Frequently Visited

G

Inbox (4) - benjaminrk@g...


jupyter.org
Install
About
Resources
Documentation
NBViewer
Widgets
Blog
Donate
jupytercon
Brought to you by NumFOCUS Foundation and O'Reilly Media Inc.
Announcing JupyterCon
August 22 - 25, 2017
New York, NY
Learn more about JupyterCon