



# Introduction to



# Why Python?

## Highly expressive:

```
friends = ['john', 'pat', 'gary', 'michael']
for i, name in enumerate(friends):
    print("iteration {iteration} is {name}".format(iteration=i, name=name))
```

## Object oriented

## Ease:

Easy to learn, change from IDL or Matlab.

## Very active community

## Extendability:

libraries for numerics, data analysis, plotting, financing, ...

## Interoperability:

Import IDL and Matlab code.

“But all my routines are written in IDL.”

### update U V data for matplotlib streamplot

▲ 3 ▼ ★ 1  
After plotting streamlines using 'matplotlib.streamplot' I need to change the U V data and update plot. For imshow and quiver there are the functions 'set\_data' and 'set\_UVC', respectively. There does not seem to be any similar function for streamlines. Is there any way to still update the same functionality?

python matplotlib scipy

share edit delete flag

asked Dec 24 '12 at 10:35

 lomsn  
16 ● 2

3 I suspect the answer is no, because if you change the vectors, it would need to re-compute the streamlines. The objects returned by `streamline` are a line and patch collections, which know nothing about the streamlines. To get this functionality would require writing a new class to wrap everything up and find a sensible way to re-use the existing objects. – tacaswell Dec 24 '12 at 17:31

1 A dirty workaround would be setting the visibility of the arrows and lines to 0 and then plotting the new streamlines. Will try if that is fast enough, since speed is an issue. – lomsn Dec 25 '12 at 0:06

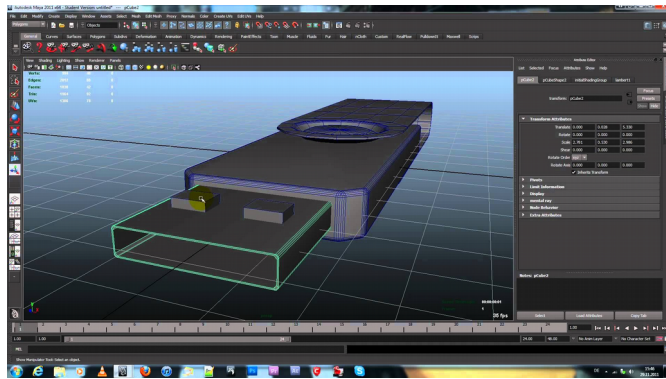
1 Works for the lines, but not for the arrows. – lomsn Dec 25 '12 at 0:21

An improvement over your current workaround, if you only have the streamplot on your axes object, is call `ax.cla()`, and then call `ax.streamplot(U_new, V_new)`. – dmcdougall Apr 2 '13 at 1:22

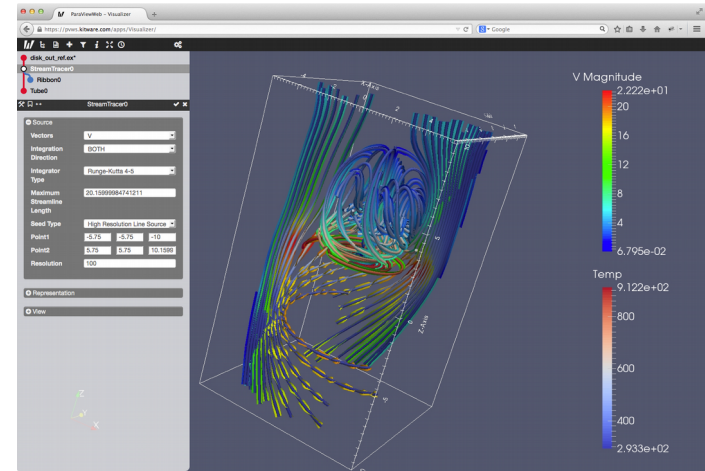
[add a comment](#)

# Why Python?

“Python is everywhere, it is all around us, even now in this very room.”



3ds Max  
Maya  
Blender  
Cinema 4D  
...

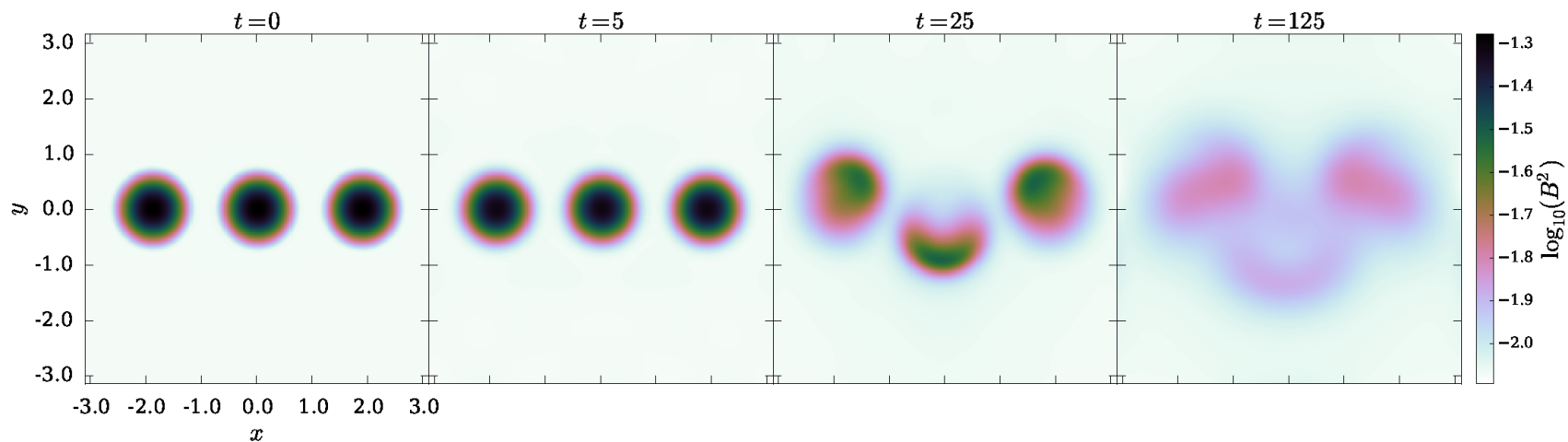
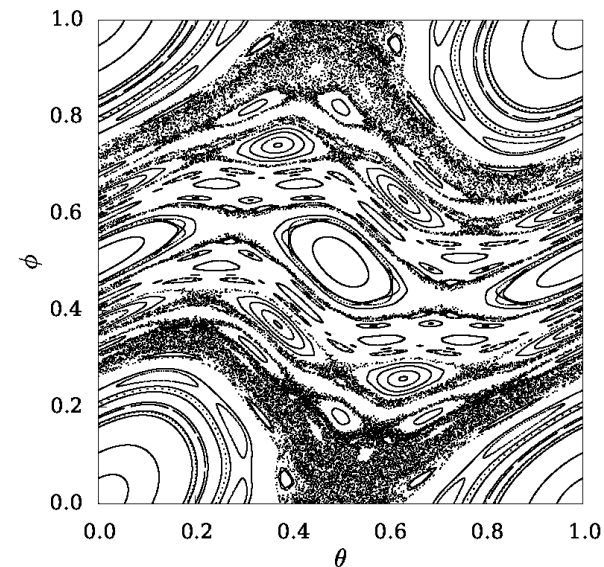
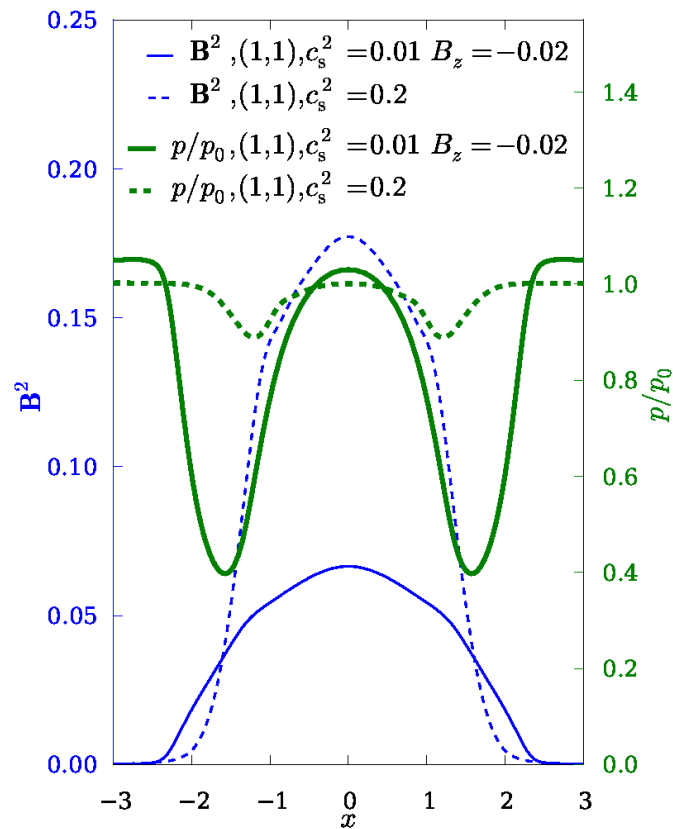
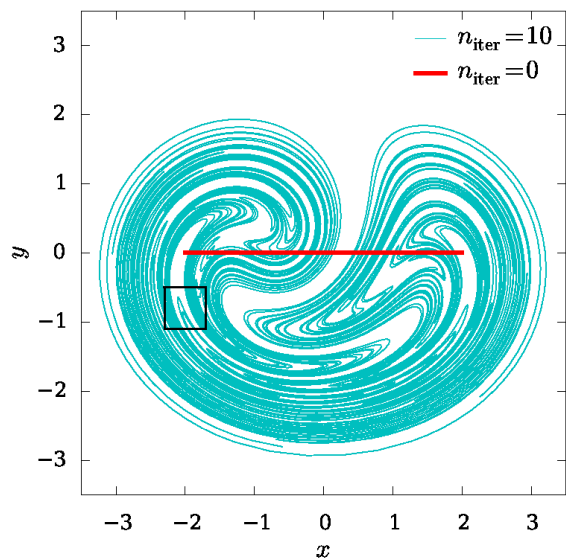


Paraview, Visit, Vapor, ...



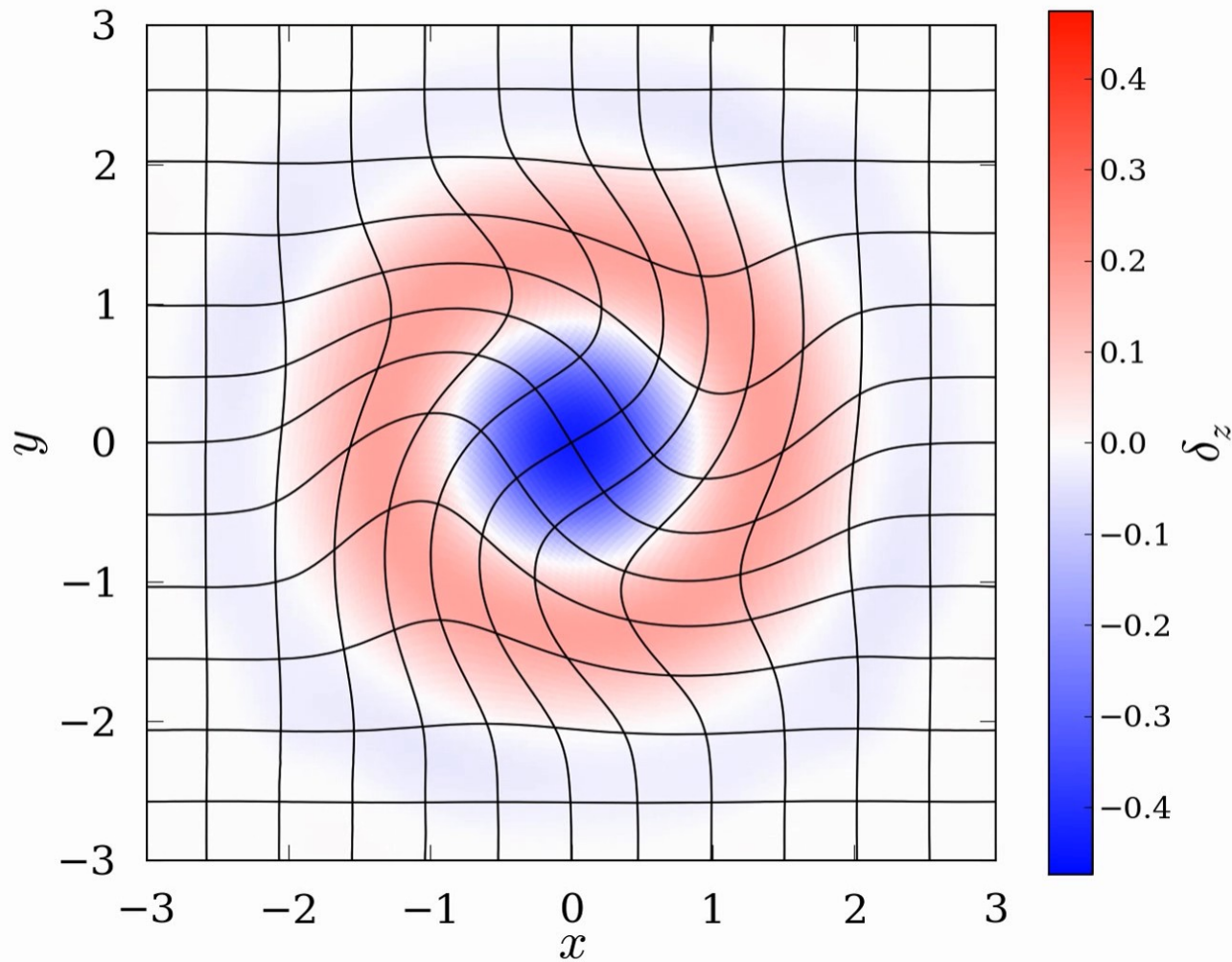
Civilization IV, Battlefield 2,  
World of Tanks, ...

# Why Python?





# Why Python?



# History

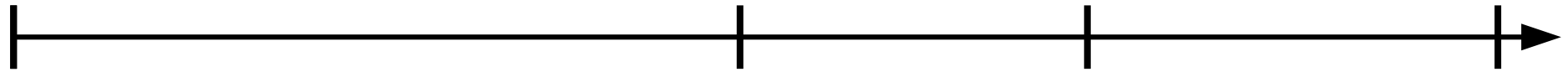


1991, Guido van Rossum

Python 2.0  
2000

Python 3.0  
2008

Python 3.52  
2016



Python  
Course

Namesake:



# Ways of Using Python

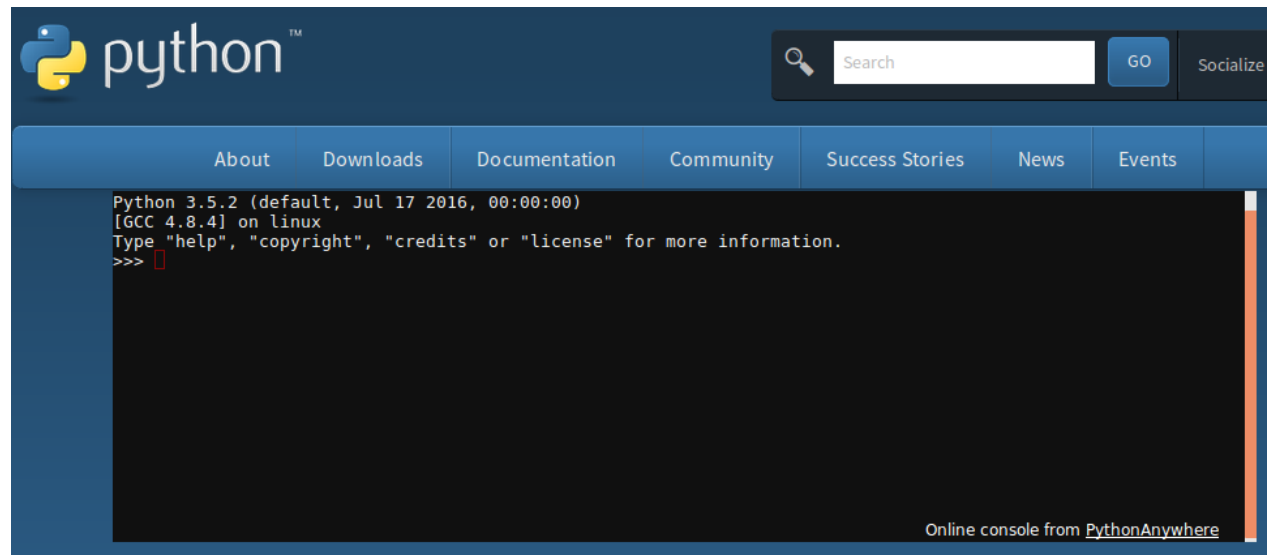
python/ipython

```
iomsn@a523:~/dundee/simon/teaching/python16$ ipython
Python 2.7.12 (default, Jul 1 2016, 15:12:24)
Type "copyright", "credits" or "license" for more information.

IPython 2.4.1 -- An enhanced Interactive Python.
?          -> Introduction and overview of IPython's features.
%quickref  -> Quick reference.
help       -> Python's own help system.
object?    -> Details about 'object', use 'object??' for extra details.
paraview version 5.0.1

In [1]:
```

<https://www.python.org/>



The screenshot shows the Python.org website with a dark blue header and navigation menu. The main content area features an online console for Python 3.5.2. The console output includes the Python version, GCC version, and a prompt for help. The website also has a search bar and a 'Socialize' button in the top right corner.

```
python python™
Search GO Socialize
About Downloads Documentation Community Success Stories News Events
Python 3.5.2 (default, Jul 17 2016, 00:00:00)
[GCC 4.8.4] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Online console from [PythonAnywhere](#)

# Ways of Using Python



The screenshot displays the Spyder Python IDE interface. The main window is titled "Editor - C:\Users\Steve\xy\fresnel\fresnel.py" and contains the following Python code:

```
272 T = (s_data['T'] + p_data['T']) / 2.
273 return {'R': R, 'T': T}
274
275 def position_resolved(layer, dist, fresnel_data):
276     """
277     Starting with output of fresnel_main(), calculate the Poynting vector
278     and absorbed energy density a distance "dist" into layer number "layer"
279     """
280     vw = fresnel_data['vw_list'][layer]
281     kz = fresnel_data['kz_list'][layer]
282     th = fresnel_data['th_list'][layer]
283     n = fresnel_data['n_list'][layer]
284     n_0 = fresnel_data['n_list'][0]
285     th_0 = fresnel_data['th_0']
286     pol = fresnel_data['pol']
287
288     #amplitude of forward-moving wave is Ef, backwards is Eb
289     Ef = vw[0] * exp(1j * kz * dist)
290     Eb = vw[1] * exp(-1j * kz * dist)
291
292     #Poynting vector
293     if(pol=='s'):
294         poyn = ((n*cos(th)*conj(Ef+Eb)*(Ef-Eb)).real) / (n_0*cos(th_0)).real
295     elif(pol=='p'):
296         poyn = (((n*conj(cos(th))*(Ef+Eb)*conj(Ef-Eb)).real)
297                / (n_0*conj(cos(th_0))).real)
298
299     #absorbed energy density
300     if(pol=='s'):
301         absor = (n*cos(th)*kz*abs(Ef+Eb)**2).imag / (n_0*cos(th_0)).real
302     elif(pol=='p'):
303         absor = (n*conj(cos(th))*
304                (kz*abs(Ef-Eb)**2-conj(kz)*abs(Ef+Eb)**2)
305                ).imag / (n_0*conj(cos(th_0))).real
306     return({'poyn':poyn, 'absor':absor})
307
308 def find_in_structure(d_list,dist):
309     """
310     d_list is list of thicknesses of layers, all of which are finite.
311
312     dist is the distance from the front of the whole multilayer structure
313     (i.e., from the start of layer 0.)
314
```

The right-hand side of the IDE shows the "Object inspector" for the function `fresnel_main`. It includes a description of the function and its parameters:

- fresnel\_main(pol, n\_list, d\_list, th\_0, lam\_vac)**  
Function of fresnel module
- Main fresnel calc. Given parameters of a stack, calculates everything you could ever want to know about how light propagates in it. (If performance is an issue, you can delete some of the calculations without affecting the rest.)
- pol is light polarization, "s" or "p".
- n\_list is the list of refractive indices, in the order that the light would pass through them. The 0'th element of the list should be the semi-infinite medium from which the light enters, the last element should be the semi-infinite medium to which the light exits (if any exits).
- th\_0 is the angle of incidence 0 for normal, pi/2 for glancing. Remember, for a dissipative incoming medium (n\_list[0] is not real), th\_0 should be complex so that n\_0 sin(th\_0) is real (intensity is constant as a function of lateral position).
- d\_list is the list of layer thicknesses (front to back). Should correspond one-to-one with elements of n\_list. First and last elements should be "inf".
- lam\_vac is vacuum wavelength of the light.

The "Console" window shows the following IPython 1 session:

```
In [8]: pv_sim.testtt()
ISC = 4.103 mA/cm2
EQE for 400-800nm = (4.103 mA/cm2) / (25.923 mA/cm2) = 15.8%
Reflection into air = 16.2 mA/cm2 = 62.5%
Absorption in mirror = 0.96 mA/cm2
Thin-layer thicknesses in nm = [ 150.    70.    20.    20.    0.34]
Absorption in thin layers = [ 1.18  0.51  4.64  1.91  0.    0.52]
(for, respectively, [ITO,PEDOT,SubPC,C60,TPBi,graphene])
C60 IQE = (1.49 mA/cm2) / (1.91 mA/cm2) = 77.8%
SubPC IQE = (2.61 mA/cm2) / (4.64 mA/cm2) = 56.3%
Out[8]: 4.1029296077801174

In [9]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52
Out[9]: 8.76

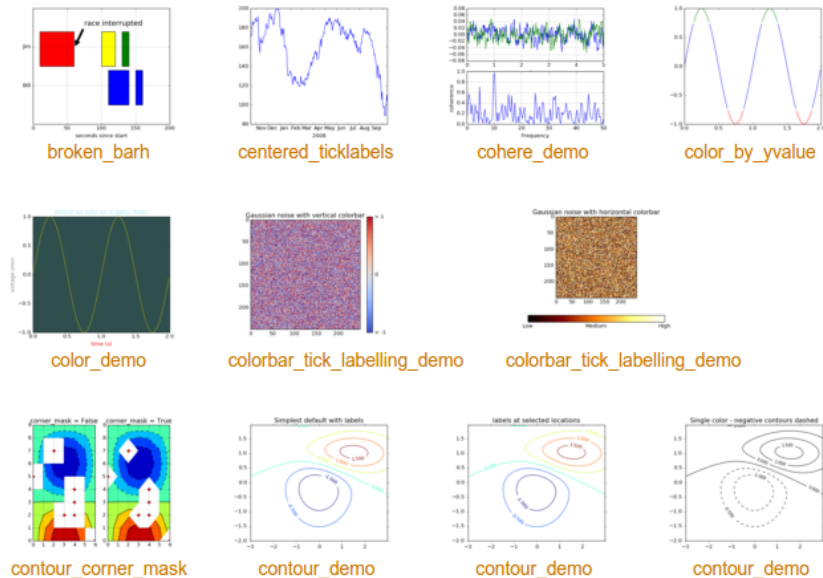
In [10]: 1.18 + 0.51 + 4.64 + 1.91 + 0.52 + 16.2 + 0.96
Out[10]: 25.92

In [11]:
```

The bottom status bar indicates: Permissions: RW, End-of-lines: CRLF, Encoding: UTF-8, Line: 289, Column: 28.

# Outlook

## Matplotlib Gallery



## Scipy Functionalities

### Tutorial

Tutorials with worked examples and background information for most SciPy submodules

- [SciPy Tutorial](#)
  - [Introduction](#)
  - [Basic functions](#)
  - [Special functions \(`scipy.special`\)](#)
  - [Integration \(`scipy.integrate`\)](#)
  - [Optimization \(`scipy.optimize`\)](#)
  - [Interpolation \(`scipy.interpolate`\)](#)
  - [Fourier Transforms \(`scipy.fftpack`\)](#)
  - [Signal Processing \(`scipy.signal`\)](#)
  - [Linear Algebra \(`scipy.linalg`\)](#)
  - [Sparse Eigenvalue Problems with ARPACK](#)
  - [Compressed Sparse Graph Routines \(`scipy.sparse.csgraph`\)](#)
  - [Spatial data structures and algorithms \(`scipy.spatial`\)](#)
  - [Statistics \(`scipy.stats`\)](#)
  - [Multidimensional image processing \(`scipy.ndimage`\)](#)
  - [File IO \(`scipy.io`\)](#)
  - [Weave \(`scipy.weave`\)](#)

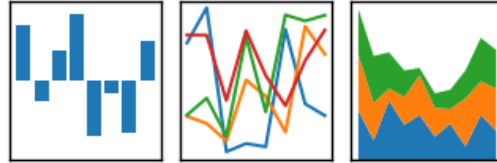


# Outlook

## Python Data Analysis Library

pandas

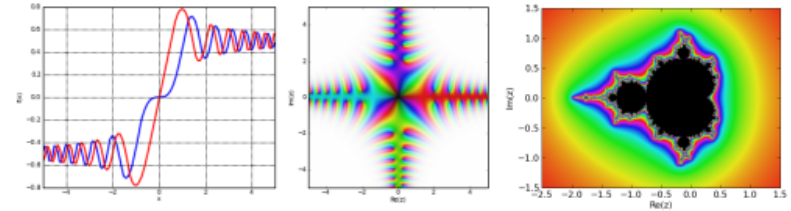
$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



<http://pandas.pydata.org>

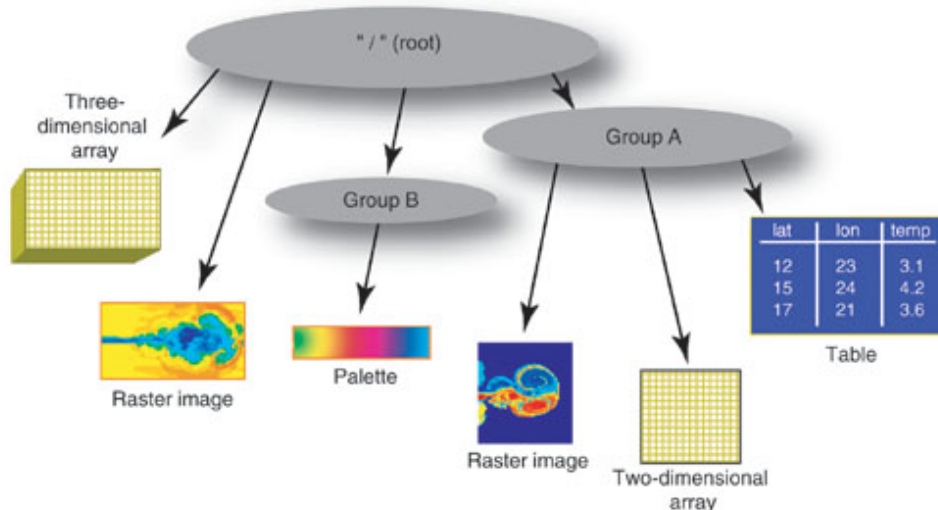
## mpmath

floating-point arithmetic with arbitrary precision

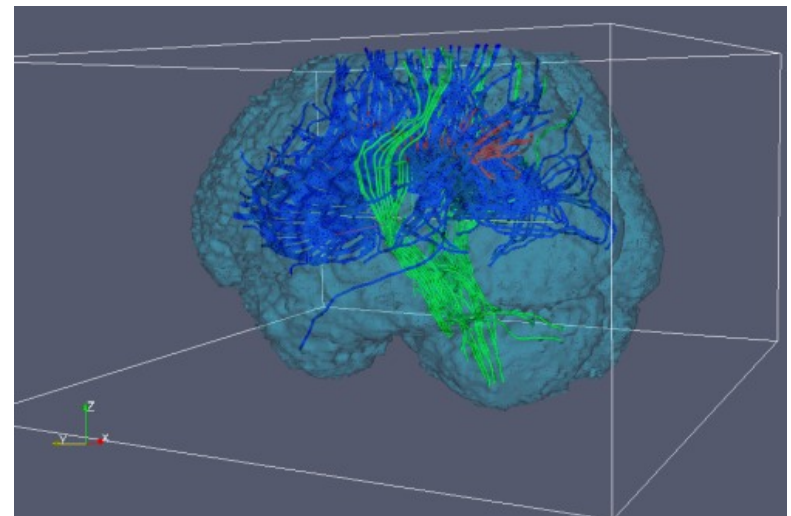


<http://mpmath.org>

## Hierarchical Data Format



## vtk Data Format





# Outlook

SunPy

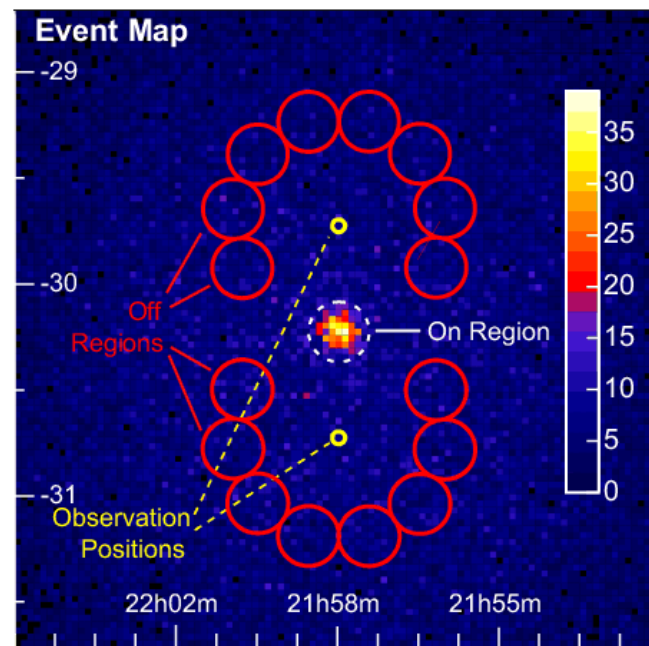
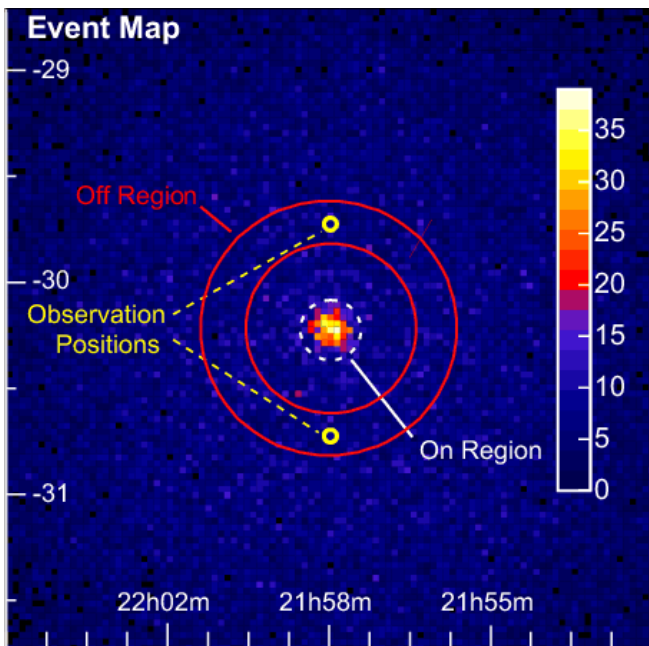


<http://sunpy.org>

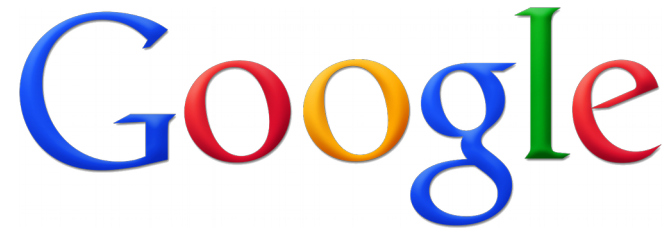
astropy



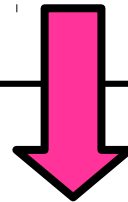
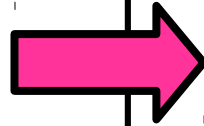
<http://www.astropy.org>



# Help!



```
# Python Documentation  
help(plt.plot)  
plt.plot?  
source(plt.plot)  
plt.plot??
```



## update U V data for matplotlib streamplot



3



1

After plotting streamlines using 'matplotlib.streamplot' I need to change the U V data and update the plot. For imshow and quiver there are the functions 'set\_data' and 'set\_UVC', respectively. There does not seem to be any similar function for streamlines. Is there any way to still update/get similar functionality?

[python](#) [matplotlib](#) [scipy](#)

[share](#) [edit](#) [delete](#) [flag](#)

asked Dec 24 '12 at 10:35



[lomsn](#)

16 ● 2

3 I suspect the answer is no, because if you change the vectors, it would need to re-compute the streamlines. The objects returned by `streamline` are a line and patch collections, which know nothing about the streamlines. To get this functionality would require writing a new class to wrap everything up and finding a sensible way to re-use the existing objects. – [tacaswell](#) Dec 24 '12 at 17:31

1 A dirty workaround would be setting the visibility of the arrows and lines to 0 and then plotting the new streamlines. Will try if that is fast enough, since speed is an issue. – [lomsn](#) Dec 25 '12 at 0:06

# Practice!

