

Python 进阶 (for Solar Group)

Anaconda

A **conda package** is a compressed tarball file that contains system-level libraries, Python or other modules, executable programs and other components. **Conda** keeps track of the dependencies between packages and platforms.

(<https://conda.io/docs/user-guide/concepts.html>)

- 凡使用 `pip install` 安装的包均为编译安装, `conda install` 安装的则为编译好的二进制包.
- conda 包可以打包其他语言, pip 包容易出现系统依赖问题
- conda 包的依赖管理更完善
- conda 环境下仍可使用 pip 进行安装, 但所使用的 pip 命令和系统中的 pip 互不影响, 安装的包的位置也在 conda 本身的目录下.
- 每个环境互不影响, 各环境下的包虽然必须分别安装(也可以 clone 一个现有环境), 但如果有不同环境中有重复的包 conda 将默认使用 *hard link*.

(更多讨论参考 <http://jakevdp.github.io/blog/2016/08/25/conda-myths-and-misconceptions/>)

```
Terminal
lydia@zhang:~$ ls -i -l ~/miniconda3/pkgs/numpy-base-1.14.3-py36h0ea5e3f_1
/lib/python3.6/site-packages/numpy-1.14.3-py3.6.egg-info
920704 dependency_links.txt
920706 PKG-INFO
920707 SOURCES.txt
920705 top_level.txt
lydia@zhang:~$
lydia@zhang:~$ ls -i -l ~/miniconda3/lib/python3.6/site-packages/numpy-1.1
4.3-py3.6.egg-info
920704 dependency_links.txt
920706 PKG-INFO
920707 SOURCES.txt
920705 top_level.txt
lydia@zhang:~$
```

- python 位置 (未使用 conda 时)

使用

```
$ python -m site
```

可以看到有哪些目录.

三个部分:

- root

```
/usr/lib/...  
/usr/bin/...
```

- user

```
/home/<user>/.local/lib/...  
/home/<user>/.local/bin/...
```

- anaconda/miniconda

```
/home/<user>/miniconda3/lib/...  
/home/<user>/miniconda3/bin/...
```

- conda 目录结构 (安装之后自动修改了软连接, 修改环境变量后系统优先使用这个位置)

```
/home/<user>/miniconda3/  
├── pkgs/  
├── bin/  
│   ├── python -> python3.6  
│   ├── python3 -> python3.6  
│   ├── python3.6  
│   ├── jupyter  
│   ├── h5dump  
│   └── ...  
├── include/  
├── lib/  
├── share/  
├── envs  
│   ├── env1/  
│   │   ├── bin/  
│   │   ├── include/  
│   │   ├── lib/  
│   │   ├── share/  
│   │   └── ...  
│   ├── env2/  
│   └── ...  
└── ...
```

IPython & Jupyter

参考文档 <http://jupyter-notebook.readthedocs.io/en/stable/>

- 配置文件路径

一般用户配置文件在 `~/.jupyter`

```
!jupyter --paths
```

```
config:
  /home/lydia/.jupyter
  /home/lydia/miniconda3/etc/jupyter
  /usr/local/etc/jupyter
  /etc/jupyter
data:
  /home/lydia/.local/share/jupyter
  /home/lydia/miniconda3/share/jupyter
  /usr/local/share/jupyter
  /usr/share/jupyter
runtime:
  /run/user/1000/jupyter
```

- 列出所有 magic functions:

```
%lsmagic
```

- 运行脚本:

```
%run script.py
%run script.ipynb
```

- 在单个 Cell 中调用其他语言(子进程):

```
%%sh
cd ~/miniconda3/
pwd
```

```
/home/lydia/miniconda3
```

可以看到实际并未改变当前目录(因为是子进程):

```
pwd
```

```
'/home/lydia/codes/zly/python/python-intro'
```

```
%script idl
cd, current=d
print, d
```

```
/home/lydia/codes/zly/python/python-intro
```

```
IDL Version 8.2.1 (linux x86_64 m64). (c) 2012, Exelis Visual Information Solutions, Inc.
```

- 逐行使用 bash 命令 (注意不加 '!' 的区别):

```
!idl -e field3dgui
```

```
!ls ./
```

得到的结果可以存入变量:

```
current, = !pwd # returns a list
current
```

```
'/home/lydia/python-intro'
```

下面这两句不能写在一个 Cell 中:

```
cd -q ~/miniconda3/
```

```
pwd
```

```
'/home/lydia/miniconda3'
```

用 `{var}` (推荐) 或 `$var` (不推荐) 调用当前 ipython 环境的变量:

```
cd {current}

/home/lydia/codes/zly/python/python-intro
```

```
cd "$current"

/home/lydia/codes/zly/python/python-intro
```

```
cd $current

/home/lydia/codes/zly/python/python-intro
```

见 <http://mmcdan.github.io/posts/interacting-with-the-shell-via-jupyter-notebook/>

- 列出当前的变量信息

```
a = 1
b = 'x'

%whos int str
```

Variable	Type	Data/Info
a	int	1
b	str	x

语法细节

python2 & python3

```
%%python2
print 1 / 2
print 1 // 2
```

0
0

```
%%python3
print(1 / 2)
print(1 // 2)
```

0.5
0

详见 [Python 2 -> 3 的转换](#)

list, tuple

- 对 list 或 tuple, 当只有一个元素时, 注意加逗号:

```
l = [1]
type(l)
```

list

```
l = [1,]
type(l)
```

list

```
l = (1)
type(l)
```

int

```
l = (1,)
type(l)
```

tuple

- 赋值, 星号(*)表达式用来展开 list/tuple, 双星号(**)用来展开 dict(常用于函数参数表):

```
a, b, *c = [0, 1, 2, 3]
a, b, c
```

(0, 1, [2, 3])

```
a, *_ = [0, 1, 2, 3] # 或 a, = ...
a
```

0

dict

- dict 改变某个 key 但 value 不变的方法

```
d = {'a': 1, 'b': 2}
d['c'] = d.pop('b') # 利用 `pop` 返回值, 更改 key 值
d
```

{'a': 1, 'c': 2}

- dict 求并集

```
d1 = {'a': 1, 'b': 2}
d2 = {'a': -1, 'c': 3, 'd': 4}
```

```
dict(d1, **d2) # 重复的 key 取后一个的值
```

{'a': -1, 'b': 2, 'c': 3, 'd': 4}

```
dict(list(d1.items()) + list(d2.items())) # 注意其中的 list
```

{'a': -1, 'b': 2, 'c': 3, 'd': 4}

```
d1.update(d2)
```

d1

{'a': -1, 'b': 2, 'c': 3, 'd': 4}

set

- set 基本操作

```
s = {1, 2}
```

```
s.add(3) # one element only
s
```

{1, 2, 3}

```
s.update([4, 5]) # iterable: list/tuple/set/...
s
```

{1, 2, 3, 4, 5}

```
s.remove(1)
s
```

{2, 3, 4, 5}

```
s.discard(0) # No error if not exist
s
```

```
{2, 3, 4, 5}
```

- set 求交集, 并集, 差集, 对称差集, 子集

```
{1, 2}.intersection({2, 3})
```

```
{2}
```

```
{1, 2} & {2, 3} # 或用 and
```

```
{2}
```

```
{1, 2}.union({2, 3})
```

```
{1, 2, 3}
```

```
{1, 2} | {2, 3} # 或用 or
```

```
{1, 2, 3}
```

```
{1, 2}.difference({2, 3})
```

```
{1}
```

```
{1, 2} - {2, 3} # 在前不在后
```

```
{1}
```

```
{1, 2}.symmetric_difference({2, 3})
```

```
{1, 3}
```

```
{1, 2} ^ {2, 3} # 不同时在两边
```

```
{1, 3}
```

- 属于, 包含于(子集)

```
2 in {1, 2, 3} # list/tuple/set
```

```
True
```

```
4 not in {1, 2, 3} # list/tuple/set
```

```
True
```

```
{1, 2}.issubset({1, 2, 3})
```

```
True
```

```
{1, 2} <= {1, 2, 3}
```

```
True
```

```
{1, 2, 3}.issuperset({1, 2})
```

```
True
```

```
{1, 2, 3} >= {1, 2}
```

```
True
```

一些特性的演示

插件 <https://github.com/lgpage/nbtutor>

网页版 <http://pythontutor.com/>

```
%reload_ext nbtutor
```

```
%%nbtutor -r -f  
l1= (0)  
l2 = (0,)
```

网页版演示 <http://t.cn/Rg9J3bf>

```
%%nbtutor -r -f -i  
a = 3  
b = 4  
t = (1, 2, [a, b]) # list in tuple  
a = 'x' # change element in list  
t[2][0] = 'xx' # change element in tuple
```

```
print(t)
```

网页版演示 <http://t.cn/Rg9idUn>

包含判断的单行循环

```
l = [1, 2, 3, 4, 5]  
[i for i in l if i > 2]  
  
[3, 4, 5]
```

例如用 `globals()` 来得到字符串形式的变量名(python 的变量没有提供这个功能的关键字, 因为"变量名"存储在namespaces中)

```
def namestr(obj, namespace):  
    return [name for name in namespace if namespace[name] is obj][0]  
a = 1  
b = 2  
for var in (a, b):  
    print(f'{namestr(var, globals())} = {var}')
```

```
a = 1  
b = 2
```

单行 if (三元关系运算)

```
a = 1  
b = 'y' if a > 0 else 'n'  
b  
  
'y'
```

循环, 迭代器, 生成器

Python 的 for 循环不使用索引变量, 而使用遍历**迭代器**(iterator)的方式处理**可迭代对象**(range, set 等)

```
for i in range(5):  
    ...
```

遍历的方式是使用 `iter` 和 `next` 函数:

```
a = iter(range(5)) # 生成一个 range_iterator 对象
```

```
next(a)
```

```
0
```

```
list(range(5)) # 转换为 list
```

```
[0, 1, 2, 3, 4]
```

enumerate 函数适合需要同时使用索引和内容时使用:

```
for i, item in enumerate(l): # enumerate() 是一个迭代器
    i = ...
    item = ... # l[i] = ...
```

生成器:

```
a = (i**2 for i in range(5)) # 产生一个生成器, 可转化成 list, tuple
a
```

```
<generator object <genexpr> at 0x14da87e652b0>
```

```
next(a) # 生成器也是迭代器
```

```
4
```

```
a = [i for i in range(3)] # 产生一个 list
a
```

```
[0, 1, 2]
```

```
# 转化成 tuple
tuple(i for i in range(3))
```

```
(0, 1, 2)
```

函数定义

- 一般的函数定义:

```
def func(a, b, *args, **kwargs):
    # func1(x, y, arg1, arg2, ..., key1=val1, key2=val2, ...)
    ...
    return ... # 没有 return 或为空 则返回 None
```

- 单行函数 lambda

```
def f(x, y):
    return x + y
f(1, 2)
```

```
3
```

用 lambda 语法来定义:

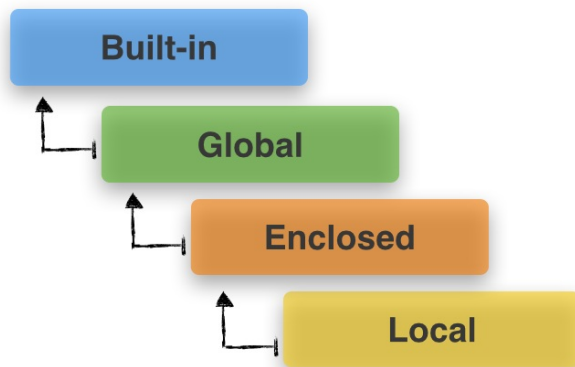
```
f = lambda x, y: x + y
f(1, 2)
```

```
3
```

```
(lambda x, y: x + y)(1, 2) # 不一定需要定义函数名称
```

```
3
```

命名空间和变量作用域



- Built-in Functions <https://docs.python.org/3/library/functions.html>
- Built-in Constants <https://docs.python.org/3/library/constants.html>

`globals()` 以 dict 形式返回当前所有全局变量, 包括导入的包名, ipython/jupyter 的 `In`, `Out` 等.
python 以 `name -> obj` 的形式存储这些变量.

```
a = 1
b = 2
```

```
globals()['a'] is a
```

True

```
i = 1
def f():
    i = 2
    print(i, 'local')
    return
f()
print(i, 'global')
```

```
2 local
1 global
```

```
i = 1
for i in range(4):
    pass
i # 值被改变了 ("leaking")
```

3

```
i = 1
[i for i in range(3)]
i # 用这种方式就不会发生"leaking"
```

1

有关函数中的变量作用方式细节, 参考这里提到的例子: [stackoverflow](https://stackoverflow.com)

- Any time you see **`**varname = **`**, you're **creating a new name** binding within the function's scope. Whatever value `varname` was bound to before is lost within this scope.
- Any time you see **`**varname.foo()**`**, you're **calling a method** on `varname`. The method may alter `varname` (e.g. `list.append`).
`varname` (or, rather, the object that `varname` names) may exist in more than one scope, and since it's the same object, **any changes will be visible in all scopes.**

NumPy

numpy 的数据类型 dtype

https://www.tutorialspoint.com/numpy/numpy_data_types.htm

```
>>> np.dtype(np.int64) is np.dtype('i8') is np.dtype('<i8')
True
>>> np.dtype(np.int64) is np.dtype('int') is np.dtype('int64')
True
>>> np.dtype(np.int64) is np.dtype('>i8')
False

>>> np.dtype(np.float64) is np.dtype('f8') is np.dtype('<f8')
True
>>> np.dtype(np.float64) is np.dtype('float') is np.dtype('float64')
True
```

见 [Fortran 二进制文件读写](#) 中的应用.

np.arange 参数为浮点数时要注意的细节

```
np.arange(0., 1.8, 0.3) # 默认不包含 `stop` 这个点
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5])
```

```
np.arange(0., 2.1, 0.3) # 这里却包括了, 为什么?
array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1])
```

```
'%.16f' % 2.1 # 因为浮点数计算方式产生的精度问题
'2.1000000000000001'
```

看一个求和的例子中同样的问题 (用 list 和 numpy.ndarray 效果一样):

```
print('%.17f' % 0.1)
# sum([0.1, ..., 0.1])

print('%.17f' % sum([0.1] * 10))
# 因为同样问题, 累积了误差

import math
print('%.17f' % math.fsum([0.1] * 10))

0.10000000000000001
0.9999999999999999
1.0000000000000000
```

```
sum([0.1] * 10) == 1
False
```

```
math.fsum([0.1] * 10) == 1
True
```

```
math.fsum([0.1] * 10) == 1.0
True
```

参考 <https://docs.python.org/3/tutorial/floatingpoint.html>

建议是: 使用整数; 对浮点数的情况, stop 值得设得略小一些(比如减去步长的一般)以防止改点被包含, 或使用 `np.linspace` 代替.

矩阵

`np.matrix`(矩阵) 是 `np.ndarray`(数组) 的一个子类, 继承了父级的方法, 也定义了新的方法(例如矩阵运算).

```
issubclass(np.matrix, np.ndarray)
```

True

```
a = np.matrix([[0, 1, 2], [3, 4, 5]]); a
```

```
matrix([[0, 1, 2],
        [3, 4, 5]])
```

```
a = np.matrix("0 1 2; 3 4 5"); a
```

```
matrix([[0, 1, 2],
        [3, 4, 5]])
```

```
type(a)
```

numpy.matrixlib.defmatrix.matrix

`np.asmatrix` (或简写作 `np.mat`) 相当于 `np.matrix(data, copy=False)`

```
x = np.array([[1, 2], [3, 4]])
m = np.matrix(x)
m[0, 0] = 5
x
```

```
array([[1, 2],
       [3, 4]])
```

```
x = np.array([[1, 2], [3, 4]])
m = np.mat(x)
m[0, 0] = 5
x
```

```
array([[5, 2],
       [3, 4]])
```

```
x.__array_interface__['data'][0] == m.__array_interface__['data'][0]
```

True

注意: `np.matrix` 既是一个类名也是一个函数名, 而 `np.mat`, `np.asmatrix` 只是函数名.

数组和矩阵运算

数组运算

```
a = np.arange(12).reshape(3, -1); a
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
a.max()
```

11

```
a * a # 注意这是对对应位置的元素间的运算
```

```
array([[ 0,  1,  4,  9],
       [16, 25, 36, 49],
       [64, 81, 100, 121]])
```

```
a ** 2 # 注意这是对对应位置的元素间的运算
```

```
array([[ 0,  1,  4,  9],
       [16, 25, 36, 49],
       [64, 81, 100, 121]])
```

使用数组进行矩阵运算

scipy.linalg 与 numpy.linalg 的比较: <https://scipy.github.io/devdocs/tutorial/linalg.html>

- scipy.linalg contains **all** the functions in numpy.linalg plus some other more **advanced** ones not contained in numpy.linalg
- Despite its convenience, the use of the numpy.matrix class is **discouraged**.
- scipy.linalg operations can be applied **equally** to numpy.matrix or to 2D numpy.ndarray objects.

```
a = np.diagflat([1., 2., 3.]); a
```

```
array([[1., 0., 0.],
       [0., 2., 0.],
       [0., 0., 3.]])
```

```
b = np.array([[-1., -1., -1.], [0., 0., 0.]]); b
```

```
array([[-1., -1., -1.],
       [ 0.,  0.,  0.]])
```

```
from scipy.linalg import inv
inv(a) # 求逆
```

```
array([[ 1.          ,  0.          , -0.          ],
       [ 0.          ,  0.5         , -0.          ],
       [ 0.          ,  0.          ,  0.33333333]])
```

```
a.dot(b.T) # 矩阵内积
```

```
array([[-1.,  0.],
       [-2.,  0.],
       [-3.,  0.]])
```

直接使用矩阵类型进行运算

使用 numpy 的矩阵类型 numpy.matrix

```
a = np.mat(np.diagflat([1., 2., 3.])); a
```

```
matrix([[1., 0., 0.],
        [0., 2., 0.],
        [0., 0., 3.]])
```

```
b = np.mat(np.array([[-1., -1., -1.], [0., 0., 0.]])); b
```

```
matrix([[-1., -1., -1.],
        [ 0.,  0.,  0.]])
```

```
a * b.T # 矩阵内积, 注意与数组的运算符定义不同
```

```
matrix([[-1.,  0.],
        [-2.,  0.],
        [-3.,  0.]])
```

```
a @ b.T # 矩阵内积的新写法, python 3.5+
```

```
matrix([[-1.,  0.],
        [-2.,  0.],
        [-3.,  0.]])
```

```
a.I # 求逆
```

```
matrix([[ 1.          ,  0.          , -0.          ],
       [ 0.          ,  0.5         , -0.          ],
       [ 0.          ,  0.          ,  0.33333333]])
```

用作坐标格点的数组的几种形式

• `np.meshgrid`

```
>>> x = np.linspace(0, 2, 3); x
array([ 0.,  1.,  2.])
>>> y = np.linspace(3, 6, 4); y
array([3., 4., 5., 6.])
# X, Y 形状相同: N x M 矩阵, shape=(N, M)
>>> X, Y = np.meshgrid(x, y)
>>> X
# X[j, :] 表示同一层 y[j] 对应的不同 x 值
array([[0., 1., 2.],
       [0., 1., 2.],
       [0., 1., 2.],
       [0., 1., 2.]])
>>> Y
# Y[:, i] 表示同一层 x[i] 对应的不同 y 值
array([[3., 3., 3.],
       [4., 4., 4.],
       [5., 5., 5.],
       [6., 6., 6.]])
```

可用来计算坐标的函数(例如磁场的表达式等等, 见 [examples/streamplot.py](#))

```
U = (np.cos(k1 * X) * np.exp(-k1 * Y)
      - np.cos(k2 * X) * np.exp(-k2 * Y))
V = (- np.sin(k1 * X) * np.exp(-k1 * Y)
      + np.sin(k2 * X) * np.exp(-k2 * Y))
```

```
x = np.linspace(0, 2, 3)
y = np.linspace(3, 6, 4)
X, Y = np.meshgrid(x, y, indexing='ij')
Y.shape

(3, 4)
```

• `np.mgrid`

`j` 标记默认是虚数单位, 但在这里表示前面的数字是格点数.

格点数使用变量 n 时, 可写成: $n*1j + 1j$, 1不能省(否则定义为变量)

类似的 `np.ogrid` 可参见 [numpy 相关文档](#)

```
>>> x = np.linspace(0, 2, 3)
>>> y = np.linspace(3, 6, 4)

# 下面三行等价
>>> X, Y = np.mgrid[0:2:3j, 3:6:4j]
>>> X, Y = np.array([i.astype(np.float64) for i in np.mgrid[0:3, 3:7]])
>>> X, Y = np.array(np.meshgrid(x, y, indexing='ij'))
# np.mgrid[0:2:3j, 3:6:4j] 得到的数据类型为 'float64'
# np.mgrid[0:3, 3:6] 得到的数据类型为 'int64'
# np.meshgrid(x, y, indexing='ij') 得到一个 list, 数据格式同x, y

# X, Y 形状相同: M x N 矩阵, shape=(M, N)
>>> X
# X[i, :] 表示 x[i]
array([[0., 0., 0., 0.],
       [1., 1., 1., 1.],
       [2., 2., 2., 2.]])
>>> Y
# Y[:, j] 表示 y[j]
array([[3., 4., 5., 6.],
       [3., 4., 5., 6.],
       [3., 4., 5., 6.]])
```

mask

```
a = np.arange(9).reshape(3, -1); a
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
a > 4 # mask
```

```
array([[False, False, False],
       [False, False,  True],
       [ True,  True,  True]])
```

```
np.where(a > 4) # 得到 tuple(col_indexs, row_indexs)
```

```
(array([1, 2, 2, 2]), array([2, 0, 1, 2]))
```

```
list(zip(*np.where(a > 4))) # 也可以用 np.array(np.where(a > 4)).T
```

```
[(1, 2), (2, 0), (2, 1), (2, 2)]
```

```
a[a > 4] = -1; a # same as a[np.where(a > 4)]
```

```
array([[ 0,  1,  2],
       [ 3,  4, -1],
       [-1, -1, -1]])
```

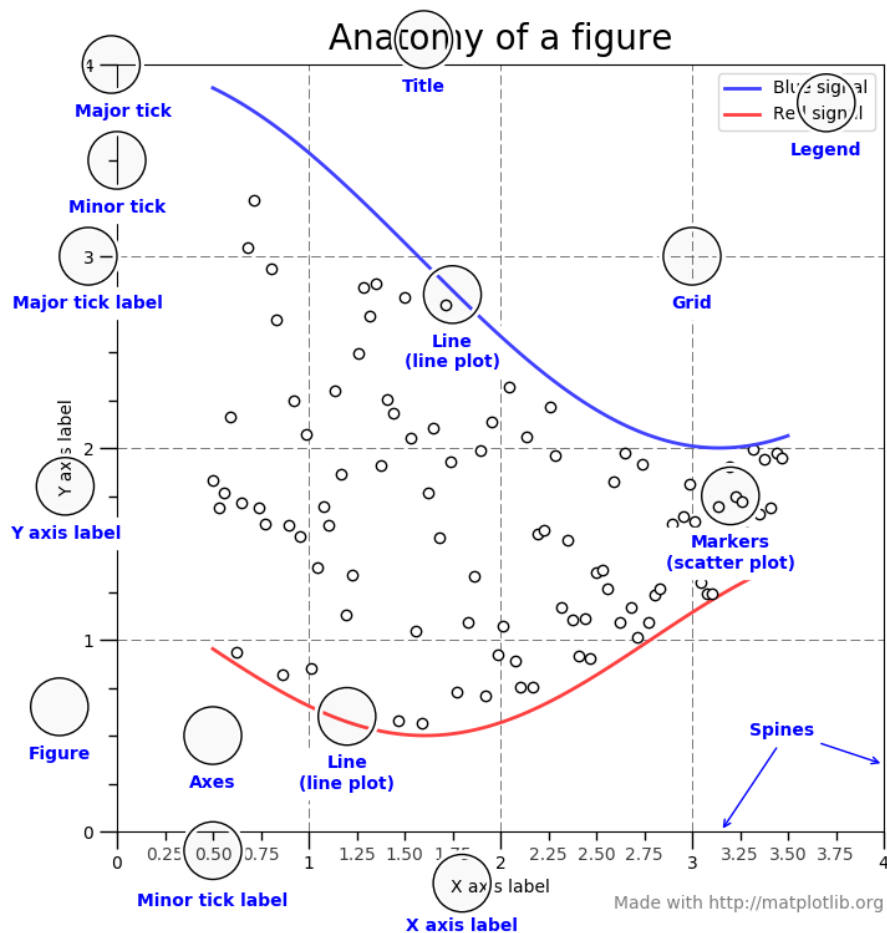
其他对数组元素的处理

处理含有 NaN 的数据:

e.g. 求最值可使用 `np.nanmin(...)` 和 `np.nanmax(...)`

Matplotlib

- 官方教程 <https://matplotlib.org/tutorials/introductory/usage.html>



```
fig = plt.gcf() # current figure
ax = plt.gca() # current `Axes`
im = plt.gci() # current artist (图像内容), 可由诸如 plt.imshow(...) 得到
```

- 得到当前 backend:

```
>>> import matplotlib.pyplot as plt
>>> plt.get_backend()
```

或

```
>>> import matplotlib as mpl
>>> mpl.get_backend()
```

- 选择 backend (在导入其他 matplotlib 模块之前):

```
>>> import matplotlib
>>> matplotlib.use('TkAgg') # 'Qt5Agg', 'TkAgg', 'Agg', ...
```

- 用户配置文档 matplotlibrc
通常的位置: `~/.config/matplotlib/matplotlibrc` (Linux)

- 用 `zorder` 来控制图层:

这个例子中也展示了加入一个动态的竖线的方法, 这条线在保存后的图像中不会显示.

```
%matplotlib
%run -e 'examples/test_plot.py'
```

- 使用 [这个工具](#) 动态标注光标位置的值:

```
%matplotlib
%run -e 'examples/test_imshow.py'
```

- 点击图像后在命令行显示光标处信息的例子

命令行脚本: [examples/test_imshow_click.py](#) (jupyter 中无法使用)

- 布局问题 https://matplotlib.org/tutorials/intermediate/constrainedlayout_guide.html
- 绘制平行投影的3D图 https://matplotlib.org/mpl_toolkits/mplot3d/tutorial.html
- 更复杂的 3D 可转存可视化数据用其他软件或使用 Mayavi (<http://docs.enthought.com/mayavi/mayavi>)

SunPy 示例

示例文件

- 读取 fits, submap, 作图 [example_plothmi](#)
- 投影 [example_projection](#)
- 下载数据 [fido](#), [jsoc](#)

注意: 这些例子只是提供一个使用参考, 不一定是最优的.

示例中的 fits 读取方法

更多 FITS 的读取方法参见 [Python 简易教程#FITS-文件读取](#)
另见 [Astropy 的命令行工具](#)

读取HMIMap 的例子:

```
# 导入 `sunpy.map` 的时间会略有点长
import sunpy.map
smmap = sunpy.map.Map('data/hmi.B_720s.20150827_052400_TAI.field.fits')
type(smmap)
```

`sunpy.map.sources.sdo.HMIMap`

这是 GenericMap 的一个子类:

```
issubclass(sunpy.map.sources.sdo.HMIMap, sunpy.map.mapbase.GenericMap)
True
```

True

```
smmap.meta # headers
```

```
type(smmap.data) # 索引从 0 开始
```

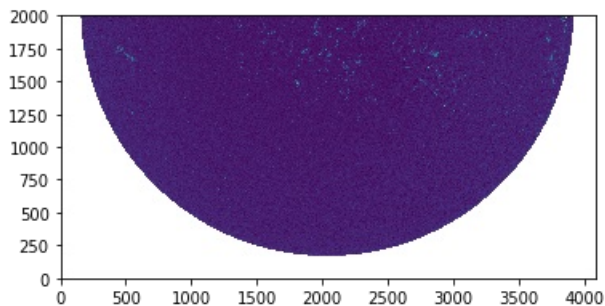
`numpy.ndarray`

```
smmap.data.shape # (dimy, dimx)
```

`(4096, 4096)`

`smmap.data[i, j]` 中的 `i` 为 Y 轴, 单位为 pixel:


```
import matplotlib.pyplot as plt
plt.imshow(smap.data[0:2000, :], origin='lower');
plt.show()
```



关于 Scaled Image Data

这一部分将更仔细地检查 Astropy 对示例中的 FITS 文件的处理。

参见 <http://docs.astropy.org/en/stable/io/fits/usage/image.html#scaled-data>

Images are scaled only when either of the BSCALE/BZERO keywords are present in the header and either of their values is not the default value (BSCALE=1, BZERO=0).

由于 `astropy.io.fits` 的特性, 读取 'Scaled Image Data' 时, 调用 `.data` 将自动对数据进行缩放:

```
physical value = BSCALE * (storage value) + BZERO
```

`astropy.io.fits.open` 中使用 `do_not_scale_image_data=True` 可禁用这一步骤。

见 [astropy.io.fits FAQ](#)

数据类型:

BITPIX	Numpy Data Type
8	numpy.uint8 (UNsigned integer)
16	numpy.int16
32	numpy.int32
-32	numpy.float32
-64	numpy.float64

例:

可以使用 Astropy 的命令行工具在 shell 中查看信息:

```
fname = 'data/hmi.B_720s.20150827_052400_TAI.field.fits'
!fitsinfo {fname}
```

```
Filename: data/hmi.B_720s.20150827_052400_TAI.field.fits
No.      Name      Ver   Type      Cards  Dimensions  Format
  0  PRIMARY          1 PrimaryHDU      6      ()
  1              1 CompImageHDU  155    (4096, 4096)  int32
```

可以看到该文件存储的数据类型是整形。

用命令行工具查看关键字 'BITPIX':

```
!fitsheader -e 1 --keyword BITPIX {fname}

# HDU 1 in data/hmi.B_720s.20150827_052400_TAI.field.fits:
BITPIX = 32 / data type of original image
```

使用 `astropy.io.fits`:

```
from astropy.io import fits
hdulist = fits.open(fname)
hdu = hdulist[1]
```

```
hdu.header['BITPIX']
```

32

'BSCALE' 不为零则为 'scaled':

```
hdu.header['BSCALE']
```

0.01

```
hdu.header['BZERO']
```

0.0

```
hdulist.verify('silentfix+warn') # 调用 `.data` 时如果出错, 需要先进行这一步修复
```

```
hdu.data.dtype # 已自动变换为 'physical value', 同时 `.header` 将被修改
dtype('float64')
```

```
hdu.data[1000, 1000]
```

142.83

```
hdu.data[0,0] # -2147483648 被转为 nan
```

nan

```
hdu.header['BITPIX'] # 值被改变了. 同时 'BSCALE', 'BZERO' 被删除
```

-64

手动 scale:

'BSCALE' 虽然之前已被删除, 但值仍存储在 `._orig_bscale` 中. `._scale` 将使用这个值.

```
hdu.scale('int32') # 在此之后 `.data` 将变换回 'storage value'
```

```
hdu._orig_bscale
```

0.01

```
hdu._orig_bzero
```

0

```
hdu.data[1000, 1000]
```

14283

```
hdu.data[0, 0] # nan 被转为 -2147483648
```

-2147483648

```
hdu.header['BITPIX'] # 值再次被改变了
```

32

使用 SunPy 处理时, 'BITPIX', 'BSCALE', 'BZERO' 的值被保留, `._data` 仍进行了 scale 的过程.

针对不同文件, 如果 SunPy 的程序内部使用 `astropy.io.fits` 处理文件时调用了 `._data` 再返回 header, 即 `astropy.io.fits` 已经如上所述改变了 header, 那么 SunPy 所返回的 header 中 'BITPIX', 'BSCALE', 'BZERO' 可能已经被改变.

下面的例子中得到的 header 是最初的值:

```
import sunpy.map
smmap = sunpy.map.Map('data/hmi.B_720s.20150827_052400_TAI.field.fits')
```

```
smmap.meta['BITPIX'] # 得到的是未被改变的 header
```

32

```
smmap.meta['BSCALE']
```

0.01

```
smmap.meta['BZERO']
```

0.0

```
smmap.data[1000, 1000] # 这一步并不改变 `.meta` 的值
```

142.83

```
smmap.meta['BITPIX'] # 不变
```

32

```
smmap.meta['BSCALE'] # 不变
```

0.01

```
smmap.meta['BZERO'] # 不变
```

0.0

坐标和单位

SunPy map http://docs.sunpy.org/en/stable/code_ref/map.html#using-map-objects

A number of the **properties** of this class are returned as **two-value named tuples** that can either be

- indexed by position: [\[0\]](#), [\[1\]](#)
- be accessed by the names:
 - pixel axes: [.x](#), [.y](#)
x and y refer to the FITS axes (x for columns y for rows)
 - spatial axes: [.axis1](#), [.axis2](#)
axis1 corresponds to the coordinate axis for x and axis2 corresponds to y.

```
smmap.dimensions
```

```
PixelPair(x=<Quantity 4096. pix>, y=<Quantity 4096. pix>)
```

```
smmap.dimensions.x # 得到一个 `astropy.units.quantity.Quantity`, 包含 value 和 unit
```

4096 \AA

```
smmap.dimensions.x.unit
```

\AA

```
smmap.dimensions.x.value
```

4096.0

astropy.units 的使用方法:

```
import astropy.units as u
```

```
u.AA
```

\AA

```
4096 * u.pix # same as `u.pixel`
```

4096 u.pix

```
(4096, 4096) * u.pix # 注意这里是 `astropy.units` 的运算, 和 list 运算规则不同
```

$[4096, \sim 4096] \text{ u.pix}$

```
smap.dimensions.x.unit is u.pix # 说明 x 是单位是 u.pix
```

True

```
type(smap.center)
```

```
/home/lydia/miniconda3/lib/python3.6/site-packages/sunpy/map/mapbase.py:669: Warning: Missing metadata for heliographic longitude: assuming longitude of 0 degrees
lon=self.heliographic_longitude,
```

astropy.coordinates.sky_coordinate.SkyCoord

```
list((0, 0) * u.pix)
```

[<Quantity 0. pix>, <Quantity 0. pix>]

```
smap.pixel_to_world(*(4096, 4096) * u.pix)
```

```
<SkyCoord (Helioprojective: obstime=2015-08-27 05:22:21.800000, rsun=696000000.0 m, observer=
<HeliographicStonyhurst Coordinate (obstime=2015-08-27 05:22:21.800000): (lon, lat, radius)
in (deg, deg, m)
(0., 7.089004, 1.51196988e+11)>): (Tx, Ty) in arcsec
(-1040.36199634, -1030.86554911)>
```

查看 SkyCoord 的帮助:

```
import astropy.coordinates
help(astropy.coordinates.sky_coordinate.SkyCoord)
```

```
smap.center.Tx
```

-7.65399 arcsec

```
smap.pixel_to_world(*(2000, 2000) * u.pix)
```

```
<SkyCoord (Helioprojective: obstime=2015-08-27 05:22:21.800000, rsun=696000000.0 m, observer=
<HeliographicStonyhurst Coordinate (obstime=2015-08-27 05:22:21.800000): (lon, lat, radius)
in (deg, deg, m)
(0., 7.089004, 1.51196988e+11)>): (Tx, Ty) in arcsec
(16.55031204, 26.53575639)>
```

```
smap.pixel_to_world(*(2000, 2000) * u.pix).transform_to('heliographic_stonyhurst')
```

```
<SkyCoord (HeliographicStonyhurst: obstime=2015-08-27 05:22:21.800000): (lon, lat, radius) in
(deg, deg, km)
(1.00568591, 8.68202918, 696000.0000022)>
```

查看坐标信息:

```
smap.coordinate_frame
```

```
/home/lydia/miniconda3/lib/python3.6/site-packages/sunpy/map/mapbase.py:669: Warning: Missing metadata for heliographic longitude: assuming longitude of 0 degrees
lon=self.heliographic_longitude,
```

```
<Helioprojective Frame (obstime=2015-08-27 05:22:21.800000, rsun=696000000.0 m, observer=
<HeliographicStonyhurst Coordinate (obstime=2015-08-27 05:22:21.800000): (lon, lat, radius) in
(deg, deg, m)
(0., 7.089004, 1.51196988e+11)>)>
```

出现这个 Warning 时:

```
smap.meta['hgln_obs'] = 0.; smap.meta['hgln_obs'] = 0.; smap.meta['hgln_obs'] = 0.  
smap.coordinate_frame
```

```
<Helioprojective Frame (obstime=2015-08-27 05:22:21.800000, rsun=696000000.0 m, observer=<Helio  
graphicStonyhurst Coordinate (obstime=2015-08-27 05:22:21.800000): (lon, lat, radius) in  
(deg, deg, m)  
(0., 7.089004, 1.51196988e+11)>>>
```

语法对比

比较 `plothmi.py` 与对应 IDL 程序的语法:

下面的语句中, 为了简便和方便对比, 使用了自定义的一些函数. 导入方法:

```
from usr_sunpy import read_sdo, plot_map, plot_vmap, image_to_helio
```

读取数据

```
; IDL  
read_sdo,<fname>,index,data  
index2map,index,data,mapa
```

```
# Python  
# 为简化过程而自定义的函数, 读入时打印文件名和数组大小  
mapa = read_sdo(<fname>)
```

旋转

```
; IDL  
; 2 表示180°  
mapbx.data = rotate (mapbx,2)  
mapby.data = rotate (mapbx,2)  
mapbz.data = rotate (mapbx,2)
```

```
# Python  
# 利用 'crota2' 的值确定角度  
mapbx = mapbx.rotate(order=1)  
mapby = mapby.rotate(order=1)  
mapbz = mapbz.rotate(order=1)
```

截取

```
; IDL  
sub_map,mapbx,smapbx,xrange=[500,800.0],yrange=[-500,-100.0]  
sub_map,mapby,smapby,ref_map=smapbx  
sub_map,mapbz,smapbz,ref_map=smapbx
```

```
# Python  
# 左下到右上  
bl = SkyCoord(500*u.arcsec, -500*u.arcsec, frame=mapbz.coordinate_frame)  
tr = SkyCoord(800*u.arcsec, -100.*u.arcsec, frame=mapbz.coordinate_frame)  
smapbx = mapbx.submap(bl, tr)  
smapby = mapby.submap(bl, tr)  
smapbz = mapbz.submap(bl, tr)
```

绘图

```
; IDL
plot_map, smapbz, dmax=2000, dmin=-2000, color=0, charsize=1.8, ...
plot_vmap, /over, smapbx, smapby, mapbz=smapbz, limit=180, scale=0.012, iskip=15, jskip=15, ...

# Python
# 主要调用的是 matplotlib
fig = plt.figure()
ax = fig.add_subplot(111, projection=smapbz) # 使用 smapbz 中定义的坐标系
plot_map(ax, smapbz)
plot_vmap(ax, smapbx, smapby, smapbz, iskip=iskip, jskip=jskip, cmin=100., vmax=500., cmap='binary', ...)
```

坐标变换

```
; IDL
; magnetic_modeling_codes/.../05projection_modified_version.pro

# Python
# `image_to_helio` (变换为 Helioprojective (Cartesian) system)
# 调用自定义的旋转矩阵函数 `proj_matrix`, 参考文献与原IDL程序相同
hx, hy = image_to_helio(smapbz) # numpy 数组
smapbx_h, smapby_h, smapbz_h = image_to_helio(smapbx, smapby, smapbz) # sunpy `Map`

plot_map(ax, smapbz_h, coords=(hx, hy))
plot_vmap(ax, smapbx_h, smapby_h, smapbz_h, coords=(hx, hy), ...)
```

插值, 拟合

选择不限于这里列举的包, 以后也可能会有更合适的方法出现.

相关安装见 [安装和配置](#)

插值

- numpy

```
f = interp1d(x, y, kind='cubic', fill_value='extrapolate')
x2 = np.linspace(...)
y2 = f(x2)
```

见 [examples/test_plot.py](#)

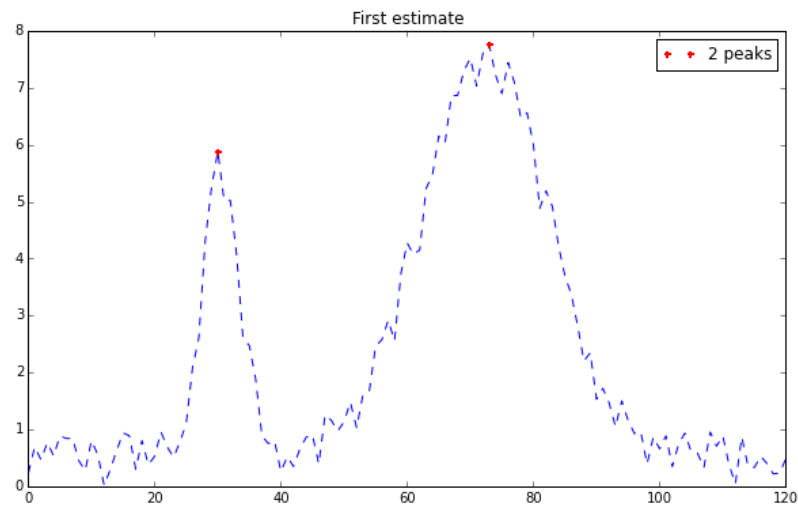
- scipy <https://docs.scipy.org/doc/scipy/reference/tutorial/interpolate.html>

求极值

- PeakUtils <http://peakutils.readthedocs.io/en/latest/>

官方例图 (http://peakutils.readthedocs.io/en/latest/tutorial_a.html):

```
import peakutils
indexes = peakutils.indexes(y, thres=0.5, min_dist=30)
```



Enhancing the resolution by interpolation:

```
peaks_x = peakutils.interpolate(x, y, ind=indexes)
```

拟合

- LMFIT (Non-Linear Least-Squares Minimization and Curve-Fitting)

下载 <https://github.com/lmfit/lmfit-py>

文档 <http://lmfit.github.io/lmfit-py/>

有现成模型, 也可以自定义模型.

可以方便地叠加不同拟合模型(比如多个高斯, 线性加高斯, 等等).

例子见 [examples/lmfits](#) (来自 [LMFIT网站示例](#))

使用内置模型:

```
from lmfit.models import GaussianModel # 内置的模型

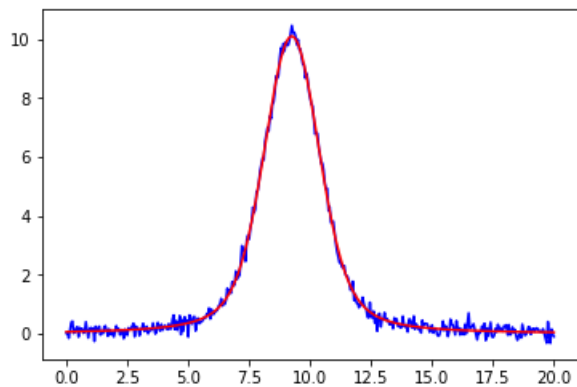
data = np.loadtxt('test_peak.dat')
x = data[:, 0]
y = data[:, 1]

mod = GaussianModel() # 实例化一个模型
pars = mod.guess(y, x=x) # 给初值
out = mod.fit(y, pars, x=x) # 拟合结果

print(out.fit_report()) # 拟合结果的详细信息
plt.plot(x, out.init_fit, 'k--')
plt.plot(x, out.best_fit, 'r-')
```

```
%matplotlib inline
%run examples/lmfits/fit_peak.py
```

```
[[Model]]
  Model(voigt)
[[Fit Statistics]]
  # fitting method      = leastsq
  # function evals      = 21
  # data points         = 401
  # variables           = 4
  chi-square            = 10.9301767
  reduced chi-square    = 0.02753193
  Akaike info crit     = -1436.57602
  Bayesian info crit   = -1420.60017
[[Variables]]
  sigma:      0.89518909 +/- 0.01415450 (1.58%) (init = 0.8775)
  center:     9.24374847 +/- 0.00441903 (0.05%) (init = 9.25)
  amplitude:  34.1914737 +/- 0.17946860 (0.52%) (init = 65.43358)
  gamma:      0.52540198 +/- 0.01857955 (3.54%) (init = 0.7)
  fwhm:       3.22385343 +/- 0.05097475 (1.58%) == '3.6013100*sigma'
  height:     10.0872204 +/- 0.03482130 (0.35%) == 'amplitude*wofz((1j*gamma)/(sigma*sqrt(
2))).real/(sigma*sqrt(2*pi))'
[[Correlations]] (unreported correlations are < 0.100)
  C(sigma, gamma)      = -0.928
  C(amplitude, gamma)  =  0.821
  C(sigma, amplitude)  = -0.651
```



多个模型叠加:

```
from lmfit.models import ExponentialModel, GaussianModel

data = np.loadtxt('NIST_Gauss2.dat')
x = data[:, 0]
y = data[:, 1]

exp_mod = ExponentialModel(prefix='exp_')
gauss1 = GaussianModel(prefix='g1_')
gauss2 = GaussianModel(prefix='g2_')

pars1 = exp_mod.guess(y, x=x) # 分段设参数的方法见下面的脚本文件
pars2 = gauss1.guess(y, x=x)
pars3 = gauss2.guess(y, x=x)

pars = pars1 + pars2 + pars3 # 叠加参数
mod = gauss1 + gauss2 + exp_mod # 叠加模型

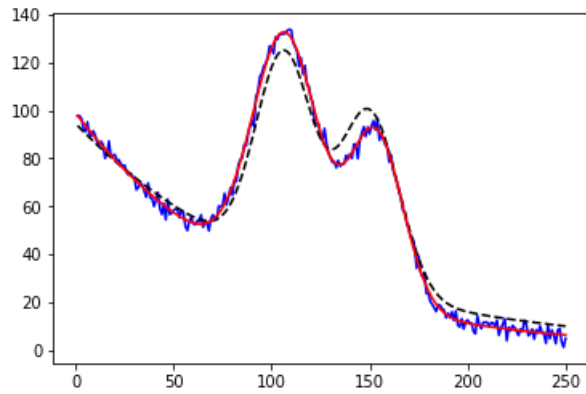
out = mod.fit(y, pars, x=x)

print(out.fit_report())
plt.plot(x, y, 'b')
plt.plot(x, out.init_fit, 'k--')
plt.plot(x, out.best_fit, 'r-')
```



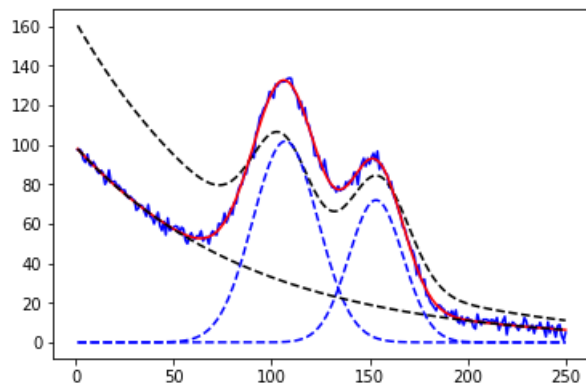
```
%%capture output
%matplotlib inline
%run examples/lmfits/fit_peaks.py
```

```
output.outputs[0]
```



```
%%capture output
%matplotlib inline
%run examples/lmfits/fit_peaks_components.py
```

```
output.outputs[0]
```



自定义模型的方法见 [LMFIT网站示例](#)

其他

- Sherpa(Modeling and Fitting) <http://cxc.cfa.harvard.edu/contrib/sherpa/>
- 2-D Fitting in Sherpa <https://python4astronomers.github.io/fitting/image.html>
- *etc.*

copy 注意事项

"shallow copy" vs "deep copy"

- 对于 list 的情况

演示插件 <https://github.com/lpage/nbtutor>

```
%%nbtutor -r -f -i
from copy import deepcopy
a = [1, 2, 3]
b = a
c = deepcopy(a)
d = a[:] # 列表的切片是深拷贝
b[0] = -1
c[1] = -2
d[2] = -3
```

```
print(id(a), id(b), id(c), id(d))
```

23284866388232 23284866388232 23284855519880 23284856061064

网页版演示 <http://t.cn/RgK1OHf>

- 对于 dict 的情况

```
%%nbtutor -r -f -i
from copy import deepcopy
a = {1:1, 2:2, 3:3}
b = a
c = a.copy() # 深拷贝
d = deepcopy(a)
b[1] = -1
c[2] = -2
d[3] = -3
```

```
print(id(a), id(b), id(c), id(d))
```

22492371132064 22492371132064 22492397510376 22492397507280

网页版演示 <http://t.cn/RgKrHOx>

- 对于 set 的情况

```
%%nbtutor -r -f -i
from copy import deepcopy
a = {1, 2, 3}
b = a
c = a.copy() # 深拷贝
d = deepcopy(a)
b.add(4)
c.add(5)
d.add(6)
```

```
print(id(a), id(b), id(c), id(d))
```

22492476895528 22492476895528 22492476895304 22492374233384

网页版演示 <http://t.cn/RgKd3E7>

- 对于 numpy 的情况

```
%%nbtutor -r -f -i
import numpy as np
a = np.arange(1, 4)
b = a[:] # 浅拷贝!
c = a.copy() # 深拷贝, 同 deepcopy(a)
d = a[[0, 2]] # 花式切片是深拷贝
b[0] = -1
c[1] = -2
d[1] = -3
```

```
# id 虽各不同, 但 __array_interface__['data'][0] 可能相同
for i in (a, b, c, d):
    print(id(i), i.__array_interface__['data'][0])
```

```
22928831926400 93879086511872
22928831928960 93879086511872
22928829201984 93879072448320
22928829199344 93879078700736
```

网页版演示 <http://t.cn/Rg9hST6>

参见之前的 [np.matrix](#), [np.asmatrix](#)

其他格式文件读写

用 pandas 读 csv

```
import pandas as pd
data = pd.read_csv("data/data.csv")
data
```

	col1	col2	col3
0	row1	1	10.0
1	row2	2	0.2
2	row3	3	300.0

```
data.columns
```

```
Index(['col1', 'col2', 'col3'], dtype='object')
```

```
data.col1
```

```
0    row1
1    row2
2    row3
Name: col1, dtype: object
```

```
data['col1']
```

```
0    row1
1    row2
2    row3
Name: col1, dtype: object
```

只取一部分:

```
data = pd.read_csv("data/data.csv", nrows=2, usecols=(0,2))
data
```

	col1	col3
0	row1	10.0
1	row2	0.2

对于超大文件, 参见: <https://www.dataquest.io/blog/pandas-big-data/>

Fortran 二进制文件读写

Unformatted (sequential)

使用 `scipy.io.FortranFile`

参考: https://docs.scipy.org/doc/scipy-0.15.1/reference/generated/scipy.io.FortranFile.read_record.html

```
# python (write)
from scipy.io import FortranFile
var1 = 1.1 # '<f8'
var2 = 0 # '<i8'
arr = np.arange(6.).reshape((2, 3)) # '(2, 3)<f8'
print(var1, var2)
print(arr)
print(f'arr[0, 1] = {arr[0, 1]}')

# 多行存入
with FortranFile('data/data_fortran1-1.dat', 'w') as f:
    f.write_record(var1)
    f.write_record(var2)
    f.write_record(arr)
# 单行存入
with FortranFile('data/data_fortran1-2.dat', 'w') as f:
    f.write_record(var1, var2, arr)
```

```
1.1 0
[[0. 1. 2.]
 [3. 4. 5.]]
arr[0, 1] = 1.0
```

```
# python (read)
from scipy.io import FortranFile
# 多行存入的情况
with FortranFile('data/data_fortran1-1.dat', 'r') as f:
    var1 = float(f.read_record('<f8'))
    var2 = int(f.read_record('<i8')) # 注意精度(上面python存入的是默认'int64')
    arr = f.read_record('(2, 3)<f8')
print('data_fortran1-1.dat')
print(var1, var2)
print(arr)
# 单行存入的情况
with FortranFile('data/data_fortran1-2.dat', 'r') as f:
    # data = f.read_record('<f8, <i8, (2, 3)<f8') # 不指定名称
    data = f.read_record([('var1', '<f8'), ('var2', '<i8'),
                          ('arr', '(2, 3)<f8')]) # 指定名称: (name, dtype)
print('\ndata_fortran1-2.dat')
print(f'keys = {data.dtype.names}') # 如读取时未指定名称则为 ['f0', 'f1', ...]
print(float(data['var1']), int(data['var2']))
print(data['arr'])
```

```
data_fortran1-1.dat
1.1 0
[[0. 1. 2.]
 [3. 4. 5.]]
```

```
data_fortran1-2.dat
keys = ('var1', 'var2', 'arr')
1.1 0
[[[0. 1. 2.]
 [3. 4. 5.]]]
```

```
%%bash
cd examples/
ifort -o test_read_seq test_read_seq.f90
./test_read_seq
```

```
data_fortran1-1.dat
1.1 0.0
0.0 1.0 2.0
3.0 4.0 5.0
arr(2, 1) = 1.0
```

```
data_fortran1-2.dat
1.1 0.0
0.0 1.0 2.0
3.0 4.0 5.0
arr(2, 1) = 1.0
```

见 Structured arrays: <https://docs.scipy.org/doc/numpy/user/basics.rec.html>

Unformatted (stream)

struct 参考 <https://docs.python.org/3/library/struct.html>

```
# python (write)
import struct
var1 = 1.1 # '<f8'
var2 = 0 # '<i8'
arr = np.arange(6.).reshape((2, 3)) # '(2, 3)<f8'
print(var1, var2)
print(arr)
print(f'arr[0, 1] = {arr[0, 1]}')
with open('data/data_fortran2.dat', 'wb') as f:
    f.write(struct.pack('d', var1))
    f.write(struct.pack('i', var2)) # 注意 `struct` 的 'i' 是 '<i4'
    arr.tofile(f)
```

```
1.1 0
[[0. 1. 2.]
 [3. 4. 5.]]
arr[0, 1] = 1.0
```

```
# python (read)
# 使用 `np.fromfile`, 读入的是 `ndarray`
with open('data/data_fortran2.dat', 'rb') as f:
    var1 = float(np.fromfile(f, dtype='<f8', count=1))
    var2 = int(np.fromfile(f, dtype='<i4', count=1))
    arr = np.fromfile(f, dtype='<f8').reshape(2, 3)
print(var1, var2)
print(arr)
```

```
1.1 0
[[0. 1. 2.]
 [3. 4. 5.]]
```

```
%%bash
cd examples/
ifort -o test_read_stream test_read_stream.f90
./test_read_stream
```

```
1.1 0.0
0.0 1.0 2.0
3.0 4.0 5.0
arr(2, 1) = 1.0
```

HDF5 文件读写, NCDF, CDF 文件读取

读写HDF5文件, 读NCDF文件

HDF5 格式介绍 <https://portal.hdfgroup.org/display/HDF5/Introduction+to+HDF5>

相关包: h5py 或 pandas + pytables (<http://www.pytables.org/usersguide/datatypes.html>)

安装见 [安装和配置](#)

• 查看

```
$ h5dump -H <filename>.hdf5
$ ncdump -h <filename>.ncdf
```

- 用 h5py 读写:

```
import h5py
arr = np.arange(10000).reshape(50,-1)
with h5py.File('test.hdf5', 'w') as f:
    f['data'] = arr
    # 或
    # f.create_dataset('data', data=arr)
    # 其中 compression='gzip', compression_opts=[0-9], 可分条目用不同压缩
    f['lb'] = list(range(100))
```

读纯 HDF5/NCDF 文件:

```
f = h5py.File('test.hdf5', 'r')
keys = list(f.keys()) # ['data', 'lb']
items = list(f.items())
a = f['data'].value
```

读 pandas 存的文件:

```
a = f['data']['block0_values'].value
f.close()
# f['<key>'] 得到的是 Dataset 或 Group 的引用
# 用 f['<key>'].value 或切片 f['<key>'][:] 得到原来的格式
# f['<key>'] 可替代为 f.get('<key>')
# 关闭之后 f['data'] 的引用失效, 因此存出数据要加上.value或[:]保存数据, 或使用 deepcopy
```

打印读取的信息:

```
from pprint import pprint
pprint([(f'{k:>20}', f'{str(f[k].shape):>20}', f'{str(f[k].dtype):>20}') for k in f.keys()])
pprint([(f'{k:>20}', f'{str(f[k]['block0_values'].shape):>20}', f'{str(f[k]['block0_values'].dtype):>20}') for k in f.keys()])
```

- 用pandas读写:

```
import pandas as pd
```

写:

```
arr2 = pd.DataFrame(arr)
# complevel=[0-9], complib={'zlib', 'lzo', 'bzip2', 'blosc'}
with pd.HDFStore('test2.hdf5', 'w') as f:
    f['data'] = arr2
    f['lb'] = pd.DataFrame(list(range(100)))
```

读pandas写的文件:

```
f = pd.HDFStore('test2.hdf5', 'r')
keys = f.keys() # ['/data', '/lb'], 存在子目录
items = list(f.items())
a = f.data.values # a = f.<key> 得到的是 DataFrame 的引用, f.<key>.values 得到原来的格式
f.close()
```

- 读纯 HDF5/NCDF 文件:
 比如 h5py 写的文件, 目录结构和 pandas 写的不同, 要用 .root
 其他软件生成的文件可能不能读

```
f = pd.HDFStore('test.hdf5', 'r')
keys = list(f.root._v_children.keys()) # ['data', 'lb']
items = list(f.root._v_children.items()) # ['data', 'lb'], 有些文件没有.items
a = f.root.data.read()
# f.root 得到 tables.group.RootGroup, f.root.<key> 得到tables.group.CArray
# f.get_node('<key>') 得到 tables.group.CArray
# f.root.<key> 可替代为 f['<key>'] (有些文件不行)
f.close()
```

打印(以免数据有 None, 用 np.shape):

```
pprint([(f'{k}'                                     .rjust(20),
        f'{np.shape(f.get_node(k))}' .rjust(20),
        f'{np.dtype(f.get_node(k))}' .rjust(20))
        for k in list(f.root._v_children.keys())])
```

Bug: 无法读出 H5T_STRING

大小比较

用于比较的文件名说明:

Filename	Method
xxx_np.npy	np.save
xxx_np.npz	np.savez_compressed
xxx_compimg.fits	FITS ComplImageHDU (compressed)
xxx_prim.fits	FITS PrimaryHDU
xxx_h5py_0.hdf5	h5py no compression
xxx_h5py_gzip.hdf5	h5py compression='gzip', compression_opts=9
xxx_h5py_lzf.hdf5	h5py compression='lzf'
xxx_pd_0.hdf5	pandas no compression
xxx_pd_blosc.hdf5	pandas complib='blosc', complevel=9
xxx_pd_bzip2.hdf5	pandas complib='bzip2', complevel=9
xxx_pd_lzo.hdf5	pandas complib='lzo', complevel=9
xxx_pd_zlib.hdf5	pandas complib='zlib', complevel=9

- 对于相关性强的数组

```
%run examples/test_hdf5.py
print()
!du -h data/test_*.np*
print()
!du -h data/test_h5py_*.hdf5
print()
!du -h data/test_pd_*.hdf5
```

array: 15.3 M

7.7M data/test_np.npy

1.3M data/test_np.npz

16M data/test_h5py_0.hdf5

2.9M data/test_h5py_gzip.hdf5

7.7M data/test_h5py_lzf.hdf5

16M data/test_pd_0.hdf5

272K data/test_pd_blosc.hdf5

248K data/test_pd_bzip2.hdf5

256K data/test_pd_lzo.hdf5

172K data/test_pd_zlib.hdf5

- 随机数数组

```
%run examples/test_hdf5_2.py
print()
!du -h data/test_*.np*
print()
!du -h data/test_h5py_*.hdf5
print()
!du -h data/test_pd_*.hdf5
```

array: 15.3 M

7.7M data/test_np.npy

7.2M data/test_np.npz

16M data/test_h5py_0.hdf5

8.0M data/test_h5py_gzip.hdf5

9.5M data/test_h5py_lzf.hdf5

16M data/test_pd_0.hdf5

7.7M data/test_pd_blosc.hdf5

7.2M data/test_pd_bzip2.hdf5

7.7M data/test_pd_lzo.hdf5

7.1M data/test_pd_zlib.hdf5

- 从 FITS 文件读取再转存


```

fname = 'data/hmi.B_720s.20150827_052400_TAI.field.fits'
%run examples/test_fits_hdf5.py {fname}
print()
!du -h {fname}
print()
!du -h data/fits_*.np*
print()
!du -h data/fits_*.fits
print()
!du -h data/fits_h5py_*.hdf5
print()
!du -h data/fits_pd_*.hdf5

```

array: 64.0 M

21M data/hmi.B_720s.20150827_052400_TAI.field.fits

65M data/fits_np.npy

27M data/fits_np.npz

21M data/fits_compimg.fits

65M data/fits_prim.fits

65M data/fits_h5py_0.hdf5

27M data/fits_h5py_gzip.hdf5

40M data/fits_h5py_lzf.hdf5

65M data/fits_pd_0.hdf5

23M data/fits_pd_blosc.hdf5

21M data/fits_pd_bzip2.hdf5

23M data/fits_pd_lzo.hdf5

20M data/fits_pd_zlib.hdf5

```

fname = 'data/hmi.B_720s.20150827_052400_TAI.disambig.fits'
%run examples/test_fits_hdf5.py {fname}
print()
!du -h {fname}
print()
!du -h data/fits_*.np*
print()
!du -h data/fits_*.fits
print()
!du -h data/fits_h5py_*.hdf5
print()
!du -h data/fits_pd_*.hdf5

```

array: 32.0 M

5.4M data/hmi.B_720s.20150827_052400_TAI.disambig.fits

33M data/fits_np.npy

5.2M data/fits_np.npz

5.4M data/fits_compimg.fits

33M data/fits_prim.fits

33M data/fits_h5py_0.hdf5

5.3M data/fits_h5py_gzip.hdf5

9.7M data/fits_h5py_lzf.hdf5

33M data/fits_pd_0.hdf5

9.5M data/fits_pd_blosc.hdf5

4.1M data/fits_pd_bzip2.hdf5

7.6M data/fits_pd_lzo.hdf5

4.8M data/fits_pd_zlib.hdf5

读取 CDF 文件

相关包 `cdflib`, 安装见 [安装和配置#其他](#)

```
import cdflib
f = cdflib.CDF("<filename>.cdf")
f.cdf_info() # 总体信息
f.varinq('<key>') # 变量信息
f.varget('<key>') # 值
f.close()
```

NumPy to VTK

参考 <https://pyscience.wordpress.com/2014/09/06/numpy-to-vtk-converting-your-numpy-arrays-to-vtk-arrays-and-files/>
以及 [安装和配置](#)

e.g.

运行 [examples/field2dvtk.py](#),

打开Paraview, 选择 state 文件 'field2d.pvsm', 再选择数据文件 'field2d.vtr'

Matlab mat 文件读写

(未测试)

```
import scipy.io
scipy.io.loadmat("<filename>", mdict=None, appendmat=True, **kwargs)
# 如果 appendmat=True, 则文件名不需要'.mat'后缀)
scipy.io.savemat("<filename>", mdict, appendmat=True, format='5',
                 long_field_names=False, do_compression=False, oned_as='row')
```

脚本相关

脚本结构

文件头:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

脚本结构:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
假设这个文件的名称为 mymodule.py
使用 `import mymodule` 导入
这些文字会出现在 `help(mymodule)` 或 `pydoc mymodule` 中
"""
c = 1
def func1(x, y, *args, **kwargs):
    """
    这个注释必须缩进
    args: a tuple - (arg1, arg2, ...)
    kwargs: a dict - {key1 : val1, key2 : val2}
    """
    global c
    return x + y + c
def func2(x):
    a = func1(x, x)
    return a
def _func3(x): # `from xxx import *` 将不会导入这个函数
    pass
    return # return 不写时返回 None
def main():
    # your code here
if __name__ == "__main__":
    main()
```

import 这个文档时, `__name__` 为模块名, 因此 `main()` 不执行

`help()` 也可以查看模块内的函数文档

```
import usr_sunpy
help(usr_sunpy.plot_map)
```

try ... except ... 语法

```
try:
    ...
except ValueError as e:
    print('ValueError:', e)
except ZeroDivisionError as e:
    print('ZeroDivisionError:', e)
else:
    print('no error!')
finally:
    print('finally...')
```

见 [Paraview 的例子](#)

Paraview 脚本

官方文档 <https://www.paraview.org/ParaView/Doc/Nightly/www/py-doc/index.html>

- 记录语句:
 1. Tools -> Start Trace
 2. 进行各种操作
 3. Tools -> Stop Trace 之后得到一个包含刚才操作命令的脚本, 参考其中的语句就可以写自己的脚本了.

- 需要导入的模块:

```
from paraview.simple import *
```

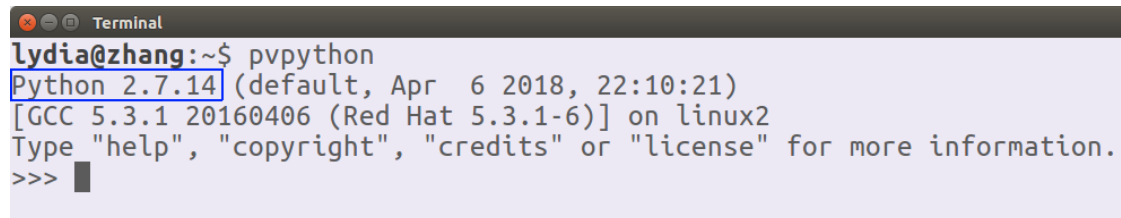
- 使用 pypython(默认渲染) pvbatch(默认不渲染) 来执行脚本:

```
pvbatch myscript.py
```

也可以使用文件头 `#!/usr/bin/env pvbatch`, 加执行权限后直接执行:

```
./myscript.py
```

注意: 打包的 paraview 所使用的 python 是 paraview 自带的 python2.7



```
Terminal
lydia@zhang:~$ pypython
Python 2.7.14 (default, Apr  6 2018, 22:10:21)
[GCC 5.3.1 20160406 (Red Hat 5.3.1-6)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- 常用语句:

```
view = GetActiveView()
scene = GetAnimationScene()
```

`view.ViewTime` 修改时 `scene.AnimationTime` 不变 `scene.AnimationTime` 修改时 `view.ViewTime` 也同时被改变

注意: filter 必须是显示状态 (`Show(<filter>)`) 才能修改时间

```
# e.g.
!code -n examples/stat1.py
```

- 字体选择参考路径:
 - `/usr/share/fonts/truetype/dejavu/DejaVuSans.ttf`
 - `/home//miniconda3/lib/python3.6/site-packages/matplotlib/mpl-data/fonts/ttf/DejaVuSans.ttf`

- python 调用问题(paraview 5.5.0+ 没有这个问题):

paraview 某个版本之后就默认使用 python 2.7.11+.

运行之前先修改环境变量 (`PYTHONPATH` 和 `PYTHONSTARTUP`),

运行结束结束后恢复这两个变量即可.

这个过程可以使用脚本(而不用alias):

```
#!/bin/bash
unset PYTHONSTARTUP # 如果有的话
export PYTHONPATH=/<your_path>/ParaView/lib/python2.7/site-packages
/<your_path>/ParaView/bin/paraview $@
```

用这个脚本替代原有 `paraview`, `pypython` 和 `pvbatch` 命令.

Python 2 -> 3 的转换

比较推荐的安全的检查方式:

- 备份的原文件且自动生成转换过的文件

```
$ 2to3 -w filename
```

- 手动比较, 在需要的地方对照修改 filename.bak

主要查看 print, range, map 等函数.

添加

```
from __future__ import division, print_function
```

在其他 import 前.

```
$ sudo apt-get install meld
$ meld filename filename.bak
```

- 确认无误后, 将手动修改的文件复制回去

```
$ mv -v filename.bak filename
```

python 内查看版本的方法, 可用于 if 语句等 (更推荐用 try...except... 语法):

```
import sys
print(sys.version_info >= (3, 0))
print(sys.version_info.major >= 3)
```

执行 shell 命令

<https://docs.python.org/3/library/subprocess.html>

```
import subprocess

fname = 'installpy.ipynb'

# 默认是以列表形式传递参数
subprocess.run(["ls", "-l", fname]) # doesn't capture output

# 输出结果需要 `decode('ascii')` 转码
print(subprocess.run(["ls", "-l", fname], stdout=subprocess.PIPE)
      .stdout.decode('ascii'))
print(subprocess.check_output(["ls", "-l", fname]).decode('ascii'))

-rw-rw-r-- 1 lydia lydia 142508 Jun 24 16:21 installpy.ipynb

-rw-rw-r-- 1 lydia lydia 142508 Jun 24 16:21 installpy.ipynb
```

利用 bash 的特长来做一些字符串操作:

```
# 使用整行命令需要 `shell=True`, 虽然并不是推荐的做法, 但偶尔使用无妨 (见官方文档)
# `r` 用来取消转义
# `echo` 使用 `-n` 来去掉输出的 '\n'
fname_head = subprocess.check_output(r"echo -n %s | sed 's/\.ipynb/\/'"
                                     % fname, shell=True).decode('ascii')

# 或使用 strip 去掉 '\n'
fname_head = subprocess.check_output(r"echo %s | sed 's/\.ipynb/\/'"
                                     % fname, shell=True).strip().decode('ascii')
print(fname_head)
```

installpy

字符串编码见 [廖雪峰的网站](#)

- 得到命令执行的路径

```
import os
os.getcwd()

'/home/lydia/codes/zly/python/python-intro'
```

- 得到脚本文件所在的路径(仅脚本中使用)

```
os.path.split(os.path.abspath(__file__))[0]
```

或

```
os.path.dirname(os.path.abspath(__file__))
```

其中 `os.path.split` 和 `os.path.dirname` 的用法:

```
os.path.split('your/path/filename')  
('your/path', 'filename')
```

```
os.path.dirname('your/path/filename')  
'your/path'
```

`os.path.dirname` 类似于 `bash` 的 `dirname`

```
!dirname 'your/path/filename'  
your/path
```

脚本选项

推荐 `argparse`, 在过去的其他方法的基础上有所增强 <https://docs.python.org/3/library/argparse.html>

脚本内:

```
import argparse  
parser = argparse.ArgumentParser(description='Process some integers.')  
parser.add_argument('integers', metavar='N', type=int, nargs='+',  
                    help='an integer for the accumulator')  
parser.add_argument('--sum', dest='accumulate', action='store_const',  
                    const=sum, default=max,  
                    help='sum the integers (default: find the max)')  
args = parser.parse_args()  
print(args.accumulate(args.integers))
```

Shell:

```
$ python prog.py -h  
usage: prog.py [-h] [--sum] N [N ...]  
Process some integers.  
positional arguments:  
  N            an integer for the accumulator  
optional arguments:  
  -h, --help  show this help message and exit  
  --sum       sum the integers (default: find the max)
```

处理 warnings

```

# Suppress warnings of NaNs:
with np.errstate(invalid='ignore'):
    ...
    ...
# Suppress other warnings, e.g.
import warnings
with warnings.catch_warnings():
    warnings.simplefilter("ignore", category=UserWarning)
    ...
    ...

```

urllib, requests

示例和文件来自 <http://python4astronomers.github.com>

注意 python3 不再使用 `urllib2`

```

# 方法1
import urllib
url = 'http://python4astronomers.github.com/_downloads/myidlfile.sav'
urllib.request.urlretrieve(url, 'myidlfile.sav')
# 方法2
with open('myidlfile.sav', 'wb') as f:
    f.write(urllib.request.urlopen(url).read())
# 方法3
import requests
with open('myidlfile.sav', 'wb') as f:
    f.write(requests.get(url).content)
# 分块下载
import requests
r = requests.get(<url>)
with open(filename, 'wb') as f:
    for chunk in r.iter_content(chunk_size=128):
        f.write(chunk)

```