Programming with Python 5

Python for non-programmers

Babar Ali

Topics

- More on Python libraries
- More on writing programs.
- Plotting in python.

MORE ON PYTHON LIBRARIES

Libraries

- A library is a collection of *function*s that adds more *function*ality to python.
- Example: numpy extends numerical capabilities of basic python.
- You have to look at the Application Programming Interface (API) to figure out how to call the function.
  - What arguments or parameters are expected to be passed to the function.

APIs

- You need to know the API to use the function.
- In the example from session 4 (right):
- The API says:
  - call the function as myMathTool(...)
  - Two mandatory numbers are expected.
  - An optional mathematical operation is expected.

```
>>> # A function to perform .
>>> def myMathTool(a,b,operation="+"):
...     if operation=="+":
...         result = a+b
...     elif operation=="-":
...         result = a-b
...     elif operation=="*":
...         result = a*b
...     elif operation=="/":
...         if b!=0.:
...             result = a-b
...         else:
...             print "I will not divide by 0.  You should know better"
...             print "returned value is 0"
...             result = 0
...     else:
...         print "Operation not understood "+operation
...         print "returned value is 0"
...         result = 0
...     return result
```

Libraries and APIs

- To learn to use the library, you need to learn its API.
- The help on library should be concerned with telling you about the API.
  - Consequently, you should look for the API description.
- Libraries ***try*** to be consistent in parameter names, conventions, etc.
  - Same goes for your collection of functions.

Finding APIs

- Usually, online documentation on the library shows the full API.

numpy.median(*a, axis=None, out=None, overwrite_input=False***)** [source]

Compute the median along the specified axis.

Returns the median of the array elements.

Parameters : **a** *: array_like*
Input array or object that can be converted to an array.

axis *: int, optional*
Axis along which the medians are computed. The default (axis=None) is to compute the median along a flattened version of the array.

out *: ndarray, optional*
Alternative output array in which to place the result. It must have the same shape and buffer length as the expected output, but the type (of the output) will be cast if necessary.

overwrite_input *: bool, optional*
If True, then allow use of memory of input array (a) for calculations. The input array will be modified by the call to median. This will save memory when you do not need to preserve the contents of the input array. Treat the input as undefined, but it will probably be fully or partially sorted. Default is False. Note that, if *overwrite_input* is True and the input is not already an ndarray, an error will be raised.

Returns : **median** *: ndarray*
A new array holding the result (unless *out* is specified, in which case that array is returned instead). If the input contains integers, or floats of smaller precision than 64, then the output data-type is float64. Otherwise, the output data-type is the same as that of the input.

See also:
mean, percentile

This is an example of a good API description.

MORE ON PROGRAMMING

Objects & Methods

- Objects are fundamental programming constructs.
  - variables are usually objects.
  - data containers (e.g. numpy's ndarray) are objects.
  - They hold data and allow access to functionality.
- Methods are functions attached to an object.

- *Both of these statements are top of the tip of the iceberg. I.e. there is a TON more to objects and methods, which we skip here.*

Finding & Accessing Methods

- Lets define a numpy array

```
z = np.zeros(10, dtype=np.int)
```

- z is now a numpy zeros object.
- It has a host of attached methods.

Finding & Accessing Methods

```
>>> dir(z)
['T', '__abs__', '__add__', '__and__', '__array__', '__array_finalize__', '__array_interface__', '__array_prepare__',
      '__array_priority__', '__array_struct__', '__array_wrap__', '__class__', '__contains__', '__copy__', '__deepcopy__',
      '__delattr__', '__delitem__', '__delslice__', '__div__', '__divmod__', '__doc__', '__eq__', '__float__', '__floordiv__',
      '__format__', '__ge__', '__getattribute__', '__getitem__', '__getslice__', '__gt__', '__hash__', '__hex__', '__iadd__',
      '__iand__', '__idiv__', '__ifloordiv__', '__ilshift__', '__imod__', '__imul__', '__index__', '__init__', '__int__',
      '__invert__', '__ior__', '__ipow__', '__irshift__', '__isub__', '__iter__', '__itruediv__', '__ixor__', '__le__', '__len__',
      '__long__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__nonzero__', '__oct__',
      '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdiv__', '__rdivmod__', '__reduce__', '__reduce_ex__',
      '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__rpow__', '__rrshift__', '__rshift__',
      '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__setitem__', '__setslice__', '__setstate__', '__sizeof__',
      '__str__', '__sub__', '__subclasshook__', '__truediv__', '__xor__', 'all', 'any', 'argmax', 'argmin', 'argpartition',
      'argsort', 'astype', 'base', 'byteswap', 'choose', 'clip', 'compress', 'conj', 'conjugate', 'copy', 'ctypes', 'cumprod',
      'cumsum', 'data', 'diagonal', 'dot', 'dtype', 'dump', 'dumps', 'fill', 'flags', 'flat', 'flatten', 'getfield', 'imag', 'item',
      'itemset', 'itemsize', 'max', 'mean', 'min', 'nbytes', 'ndim', 'newbyteorder', 'nonzero', 'partition', 'prod', 'ptp', 'put',
      'ravel', 'real', 'repeat', 'reshape', 'resize', 'round', 'searchsorted', 'setfield', 'setflags', 'shape', 'size', 'sort', 'squeeze',
      'std', 'strides', 'sum', 'swapaxes', 'take', 'tofile', 'tolist', 'tostring', 'trace', 'transpose', 'var', 'view']
>>>
```

Each of these is a function that can be applied to the object z

Finding & Accessing Methods

- How to apply the method
- z.method(parameters ....)

- Example,
- z.max()

- help(z.max())

Line Continuation

- A single programming line may be split over many lines:
  - to make it easier to read (no horizontal scrolling required)
  - to group like items in a large set
  - style preference
  - …

Line Continuation

- This is a valid python command

```
colNames = ['intnum','cat','comments','code',\
        'umag','umerr','bmag','bmerr','vmag','vmerr','rmag','rmerr','icmag','icmerr',\
        'L1','sdssrmag', 'sdssrmage', 'L2', 'sdssimag', 'sdssimage', \
        'jmag','jmerr','hmag','hmerr','kmag','kmerr',\
        'w1','w1err','w2','w2err','w3','w3err','w4','w4err',\
        'irac1','irac1err','irac2','irac2err','irac3','irac3err','irac4','irac4err',\
        'L3','mips1','mips1err']
```

Line Continuation

- This is a valid python command

The \ tells python the rest of the command is continued on the next line.

```
colNames = ['intnum','cat','comments','code',\
        'umag','umerr','bmag','bmerr','vmag','vmerr','rmag','rmerr','icmag','icmerr',\
        'L1','sdssrmag', 'sdssrmage', 'L2', 'sdssimag', 'sdssimage', \
        'jmag','jmerr','hmag','hmerr','kmag','kmerr',\
        'w1','w1err','w2','w2err','w3','w3err','w4','w4err',\
        'irac1','irac1err','irac2','irac2err','irac3','irac3err','irac4','irac4err',\
        'L3','mips1','mips1err']
```

Executing Python Scripts in Scripts

- execfile("/full/path/scriptname.py")

The ***execfile*** command will automatically execute all instructions in a python script.

PLOTTING IN PYTHON

Plotting

- Python has a number of plotting libraries.
  https://wiki.python.org/moin/NumericAndScientific/Plotting

- Recommendation:
  - Matplotlib and variants
    http://matplotlib.org/

Plotting

- DEMO