Python - Quickstart tutorial for Numpy

- 2017/11/17 23:05
- https://docs.scipy.org/doc/numpy-dev/user/quickstart.html
- numpy
  - the basics
    - NumPy’s main object is the **homogeneous multidimensional array**. It is a table of elements (usually numbers), all of the same type, indexed by **a tuple of positive integers**. In NumPy dimensions are called *axes*. The number of axes is *rank*.
- NumPy’s array class is called `ndarray`.
  - the Standard Python Library class `array.array`, which only handles one-dimensional arrays and offers less functionality
  - important attributes of an `ndarray` object are

- **ndarray.ndim**
  - the number of axes (dimensions) of the array. rank
- **ndarray.shape**
  - the dimensions of the array. `(n,m)`.
- **ndarray.size**
  - the total number of elements of the array.
- **ndarray.dtype**
  - an object describing the type of the elements in the array.
  - One can create or specify dtype’s using standard Python types. Additionally NumPy provides types of its own. **numpy.int32, numpy.int16, and numpy.float64** are some examples.
- **ndarray.itemsize**
  - the size in bytes of each element of the array.
  - It is equivalent to `ndarray.dtype.itemsize`.
- **ndarray.data**
  - the buffer containing the actual elements of the array.

  - Creation
    - create an array from a regular Python list or tuple using the `array` function
      - `1D: a = np.array([2,3,4])`
      - `2D:b = np.array([(1.5,2,3), (4,5,6)])`
      - `c = np.array( [ [1,2], [3,4] ], dtype=complex )`
        - The type of the array can also be explicitly specified at creation time:
    - unknown content but known size
      - NumPy offers several functions to create arrays with initial placeholder content. These minimize

the necessity of growing arrays, an expensive operation.

- `np.zeros( (3,4) )`
- `np.ones( (2,3,4), dtype=np.int16 )`
- `np.empty( (2,3) )`
- By default, the dtype of the created array is `float64`.

- To create sequences of numbers
  - analogous to `range in python`
  - **numpy.arange**([*start*,]*stop*, [*step*,]*dtype=None*)
    - When `arange` is used with floating point arguments, it is generally not possible to predict the number of elements obtained, due to the finite floating point precision
  - so it is usually better to use the function `linspace`
  - `np.linspace( 0, 2, 9 )`
  - **numpy.linspace**(*start*, *stop*, *num=50*, *endpoint=True*, *retstep=False*, *dtype=None*)
    - Return evenly spaced numbers over a specified interval.
  - **numpy.logspace**(*start*, *stop*, *num=50*, *endpoint=True*, *base=10.0*, *dtype=None*)
    - Return numbers spaced evenly on a log scale.
- Printing Arrays
  - print
  - 1D array is printed as row array

- Basic Operations
  - Arithmetic operators on arrays apply *elementwise*. A new array is created and filled with the result.
    - +, -, /, % ...
    - multiply
      - the product operator `*` operates elementwise in NumPy arrays
      - The matrix product can be performed using the `dot` function or method:
  - Some operations, **such as `+=` and `*=`, act in place to modify an existing array** rather than create a new one.
  - When operating with arrays of different types, the type of the resulting array corresponds to the more general or precise one (a behavior known as upcasting).
  - unary operations
    - a.sum(), min(). max() ...
    - **By default, these operations apply to the array as though it were a list of numbers**, regardless of its shape.
    - However, **by specifying the `axis` parameter you can apply an operation along the specified axis of an array:**
      - axis = 0 along the 1st/row
      - axis = 1 along the 2nd/column
- Universal Functions
  - such as sin, cos, and exp
    - Within NumPy, these functions **operate elementwise on an array**, producing an array as output.
- Indexing, Slicing and Iterating
  - **One-dimensional** arrays can be indexed, sliced and iterated over, much like lists and other Python sequences.
    - `a[2:5]`
    - `a[:6:2] = -1000`
      - *# equivalent to a[0:6:2] = -1000; from start to position 6, exclusive, set every 2nd element to -1000*
    - `a[ : :-1]`
      - *# reversed a*
    - `for i in a:`

            - iteration
    - **Multidimensional** arrays can have one index per axis. These indices are given in a tuple separated by commas:
        - almost the same as matlab
            - b[2,3]
            - b[0:5, 1]
            - b[: ,1]
        - When fewer indices are provided than the number of axes, the missing indices are considered complete slices:
            - b[-1]   *# the last row. Equivalent to b[-1,:]*
        - The expression within brackets in b[i] is treated as an **i followed by as many instances of : as needed to represent the remaining axes.** NumPy also allows you to write this using dots as b[i,...].
            - x[1,2,...] is equivalent to x[1,2,:,:,:],
        - **Iterating** over multidimensional arrays is done with respect to the first axis:

```
>>> for row in b:
...     print(row)
...
[0 1 2 3]
[10 11 12 13]
[20 21 22 23]
[30 31 32 33]
[40 41 42 43]
```

            - if one wants to perform an operation on each element in the array, use flat
                - for element in b.flat:

- Shape Manipulation
    - Note that the following three commands all return a modified array, but do not change the original array:
        - a.ravel() *# returns the array, flattened*
            - "C-style", that is, the rightmost index "changes the fastest", so the element after a[0,0] is a[0,1].
        - a.reshape(6,2) *# returns the array with a modified shape*
        - a.T *# returns the array, transposed*
    - The reshape function returns its argument with a modified shape, whereas the ndarray.resize method modifies the array itself:

- If a dimension is given as -1 in a reshaping operation, the other dimensions are automatically calculated:
    - `a.reshape(3,-1)`
- Stacking together different arrays
    - For arrays of with more than two dimensions, hstack stacks along their second axes, vstack stacks along their first axes, and concatenate allows for an optional arguments giving the number of the axis along which the concatenation should happen.
        - `np.vstack((a,b))`
        - `np.hstack((a,b))`
        - `np.r_[1:4,0,4]`
            - `array([1, 2, 3, 0, 4])`
- Copies and Views
    - No Copy at All
        - Simple assignments make no copy of array objects or of their data.
            - ```
              >>> a = np.arange(12)
              >>> b = a            # no new object is created
              >>> b is a           # a and b are two names for the same ndarray object
              ```
                - *a & b are just names, referring to the same object*
        - Python passes mutable objects as references, so function calls make no copy.
            - call by reference
    - View or Shallow Copy
        - Different array objects can share the same data. The `view` method creates a new array object that looks at the same data.
            - ```
              >>> c = a.view()
              >>> c is a
              False
              >>> c.base is a          # c is a view of the data owned by a
              True
              >>> c.shape = 2,6        # a's shape doesn't change
              ```

```
>>> a.shape
(3, 4)
>>> c[0,4] = 1234                   # a's data changes
```

- Slicing an array returns a view of it:
  - `s[:] = 10            # s[:] is a view of s. Note the difference between s=10 and s[:]=10`
  - view can change the value but not the shape
- Deep Copy
  - The `copy` method makes a complete copy of the array and its data.
    - `>>> d = a.copy()                       # a new array object with new data is created`
      - `# d doesn't share anything with a`

- Functions and Methods Overview

  - Array Creation
  - arange, array, copy, empty, empty_like, eye, fromfile, fromfunction, identity, linspace, logspace, mgrid, ogrid, ones, ones_like, r, zeros, zeros_like
  - Conversions
  - ndarray.astype, atleast_1d, atleast_2d, atleast_3d, mat
  - Manipulations
  - array_split, column_stack, concatenate, diagonal, dsplit, dstack, hsplit, hstack, ndarray.item, newaxis, ravel, repeat, reshape, resize, squeeze, swapaxes, take, transpose, vsplit, vstack
  - Questions
  - all, any, nonzero, where
  - Ordering
  - argmax, argmin, argsort, max, min, ptp, searchsorted, sort
  - Operations
  - choose, compress, cumprod, cumsum, inner, ndarray.fill, imag, prod, put, putmask, real, sum
  - Basic Statistics
  - cov, mean, std, var
  - Basic Linear Algebra
  - cross, dot, outer, linalg.svd, vdot

- Broadcasting rules
  - 在dim = 1 的维度上，复制扩展到与最大的matrix 一样大小
  - a "1" dim will be repeatedly prepended to the shapes of the smaller arrays until all the arrays have the same number of dimensions.
  - The value of the array element is assumed to be the same along that dimension for the "broadcast" array.
- Fancy indexing and index tricks
  - Indexing with Arrays of Indices

- ```
  >>> a = np.arange(12)**2                       # the first 12 square numbers
  >>> i = np.array( [ 1,1,3,8,5 ] )              # an array of indices
  >>> a[i]                                       # the elements of a at the positions i
  array([ 1,  1,  9, 64, 25])
  >>>
  >>> j = np.array( [ [ 3, 4], [ 9, 7 ] ] )      # a bidimensional array of indices
  >>> a[j]                                       # the same shape as j
  array([[ 9, 16],
         [81, 49]])
  ```
    - 用array做index，维度按照index array来
- When the indexed array `a` is multidimensional, a single array of indices refers to the first dimension of `a. the other dim are :s`
- Naturally, we can put `i` and `j` in a sequence (say a list) and then do the indexing with the list.
    - ```
      >>> l = [i,j]
      >>> a[l]                                   # equivalent to a[i,j]
      array([[ 2,  5],
             [ 7, 11]])
      ```
- However, we can not do this by putting `i` and `j` into an array, because this array will be interpreted as indexing the first dimension of a.
    - list [i,j] and tuple([i,j]) 可以看作多维index，但array([i,j])不能
- You can also use indexing with arrays as a target to assign to:
    - `a[[0,0,2]]=[1,2,3]`

Indexing with Boolean Arrays

- When we index arrays with arrays of (integer) **indices we are providing the list of indices to pick**. With boolean indices the approach is different; **we explicitly choose which items in the array we want and which ones we don't**.
- use boolean arrays that have *the same shape* as the original array:
    - `b = a > 4         # b is a boolean with a's shape`
        - *b is a boolean array, with True where a[i] > 4*
    - `a[b]                                        # 1d array with the selected elements`

- `c = a[b]`
- `a[b] = 0                                         # All elements of 'a' higher than 4 become 0`

- The ix_() function
    - The ix_ function can be used to combine different vectors so as to obtain the result for each n-uplet.
- Linear Algebra
    - See linalg.py in numpy folder for more
- Tricks and Tips
    - "Automatic" Reshaping
        - To change the dimensions of an array, you can omit one of the sizes which will then be deduced automatically:
        - `a.shape = 2,-1,3  # -1 means "whatever is needed"`
    - Vector Stacking
        - In NumPy this works via the functions `column_stack`, `dstack`, `hstack` and `vstack`, depending on the dimension in which the stacking is to be done.