# 1 Shape Information and Transformation

## 1.1 shape

shape returns the size of all dimensions or an array or matrix as a tuple. shape can be called as a function or an attribute. shape can also be used to reshape an array by entering a tuple of sizes. Additionally, the new shape can contain -1 which indicates to expand along this dimension to satisfy the constraint that the number of elements cannot change.

```
>>> x = randn(4,3)
>>> x.shape
(4L, 3L)
>>> shape(x)
(4L, 3L)
>>> M,N = shape(x)
>>> x.shape = 3,4
>>> x.shape
(3L, 4L)
>>> x.shape = 6,-1
>>> x.shape
(6L, 2L)
```

### 1.1.1 reshape

reshape transforms an array with one set of dimensions and to one with a different set, preserving the number of elements. Arrays with dimensions M by N can be reshaped into an array with dimensions K by L as long as M N = K L. The most useful call to reshape switches an array into a vector or vice versa. ¿¿¿ x = array([[1,2],[3,4]]) ¿¿¿ y = reshape(x,(4,1)) ¿¿¿ y array([[1], [2], [3], [4]]) ¿¿¿ z=reshape(y,(1,4)) ¿¿¿ z array([[1, 2, 3, 4]]) ¿¿¿ w = reshape(z,(2,2)) array([[1, 2], [3, 4]]) The crucial implementation detail of reshape is that arrays are stored using row-major notation. Elements in arrays are counted first across rows and then then down columns. reshape will place elements of the old array into the same position in the new array and so after calling reshape, x (1) = y (1), x (2) = y (2), and so on.

### 1.1.2 size

size returns the total number of elements in an array or matrix. size can be used as a function or an attribute.

```
>>> x = randn(4,3)
>>> size(x)
12
>>> x.size
12
```

### 1.1.3 ndim

ndim returns the size of all dimensions or an array or matrix as a tuple. ndim can be used as a function or an attribute .

```
>>> x = randn(4,3)
>>> ndim(x)
2
>>> x.ndim
2
```

### 1.1.4 tile

tile, along with reshape, are two of the most useful non-mathematical functions. tile replicates an array according to a specified size vector.

MORE

### 1.1.5 ravel

ravel returns a flattened view (1-dimensional) of an array or matrix. ravel does not copy the underlying data (when possible), and so it is very fast. ¿¿¿ x = array([[1,2],[3,4]]) ¿¿¿ x array([[ 1, 2], [ 3, 4]]) ¿¿¿ x.ravel() array([1, 2, 3, 4]) ¿¿¿ x.T.ravel() array([1, 3, 2, 4])

### 1.1.6 flatten

flatten works much like ravel, only that is copies the array when producing the flattened version.

### 1.1.7 split, vsplit, hsplit

vsplit and hsplit split arrays and matrices vertically and horizontally, respectively. Both can be used to split an array into n equal parts or into arbitrary segments, depending on the second argument. If scalar, the array is split into n equal sized parts. If a 1 dimensional array, the array is split using the elements of the array as break points. For example, if the array was [2,5,8], the array would be split into 4 pieces using [:2] , [2:5], [5:8] and [8:]. Both vsplit and hsplit are special cases of split, which can split along an arbitrary axis.

```
>>> x = reshape(arange(20),(4,5))
>>> y = vsplit(x,2)
>>> len(y)
2
>>> y[0]
```

```
array([[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9]])
>>> y = hsplit(x,[1,3])
>>> len(y)
3
>>> y[0]
array([[ 0],
[ 5],
[10],
[15]])
>>> y[1]
array([[ 1, 2],
[ 6, 7],
[11, 12],
[16, 17]])
```

### 1.1.8 delete

delete removes values froman array, and is similar to splitting an array, and then concatenating the values which are not deleted. The form of delete is delete(x,rc, axis) where rc are the row or column indices to delete, and axis is the axis to use (0 or 1 for a 2-dimensional array). If axis is omitted, delete operated on the flattened array.

```
>>> x = reshape(arange(20),(4,5))
>>> delete(x,1,0) # Same as x[[0,2,3]]
array([[ 0, 1, 2, 3, 4],
[10, 11, 12, 13, 14],
[15, 16, 17, 18, 19]])
>>> delete(x,[2,3],1) # Same as x[:,[0,1,4]]
array([[ 0, 1, 4],
[ 5, 6, 9],
[10, 11, 14],
[15, 16, 19]])
>>> delete(x,[2,3]) # Same as hstack((x.flat[:2],x.flat[4:]))
array([ 0, 1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
19])
```

### 1.1.9 squeeze

squeeze removes singleton dimensions from an array, and can be called as a function or a method.

```
>>> x = ones((5,1,5,1))
>>> shape(x)
(5L, 1L, 5L, 1L)
>>> y = x.squeeze()
>>> shape(y)
(5L, 5L)
>>> y = squeeze(x)
```

### 1.1.10 fliplr, flipud

fliplr and flipud flip arrays in a left-to-right and up-to-down directions, respectively. flipud reverses the elements in a 1-dimensional array, and flipud(x) is identical to x[::1]. fliplr cannot be used with 1-dimensional arrays.

```
>>> x = reshape(arange(4),(2,2))
>>> x
array([[0, 1],
[2, 3]])
>>> fliplr(x)
array([[1, 0],
[3, 2]])
>>> flipud(x)
array([[2, 3],
[0, 1]])
```

### 1.1.11 diag

The behavior of diag differs depending depending on the formof the input. If the input is a square array, it will return a column vector containing the elements of the diagonal. If the input is an vector, it will return an array containing the elements of the vector along its diagonal. Consider the following example:

```
>>> x = array([[1,2],[3,4]])
>>> x
array([[1, 2],
[3, 4]])
>>> y = diag(x)
>>> y
array([1, 4])
>>> z = diag(y)
>>> z
array([[1, 0],
[0, 4]])
```

#### 1.1.12 triu, tril

triu and tril produce upper and lower triangular arrays, respectively.

```
>>> x = array([[1,2],[3,4]])
>>> triu(x)
array([[1, 2],
[0, 4]])
>>> tril(x)
array([[1, 0],
[3, 4]])
```