Numpy

Lingua franca for data exchange

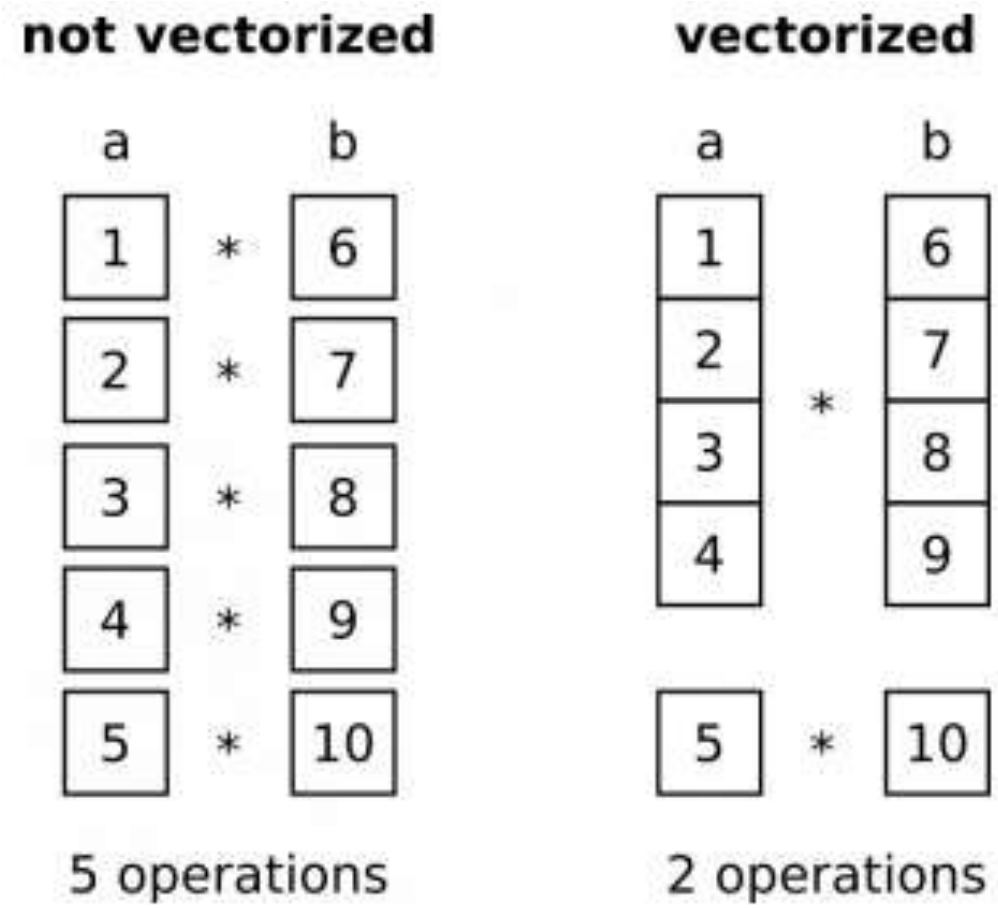
Muhammad Shahid Arshad

Numpy

- Numerical Python
- Developed in 2005 by Travis Oliphant
- Lingua franca for data exchange
- ndarray – a n-dimensional array
- Fast operations on entire arrays
- Reading/writing array data
- Linear algebra operations

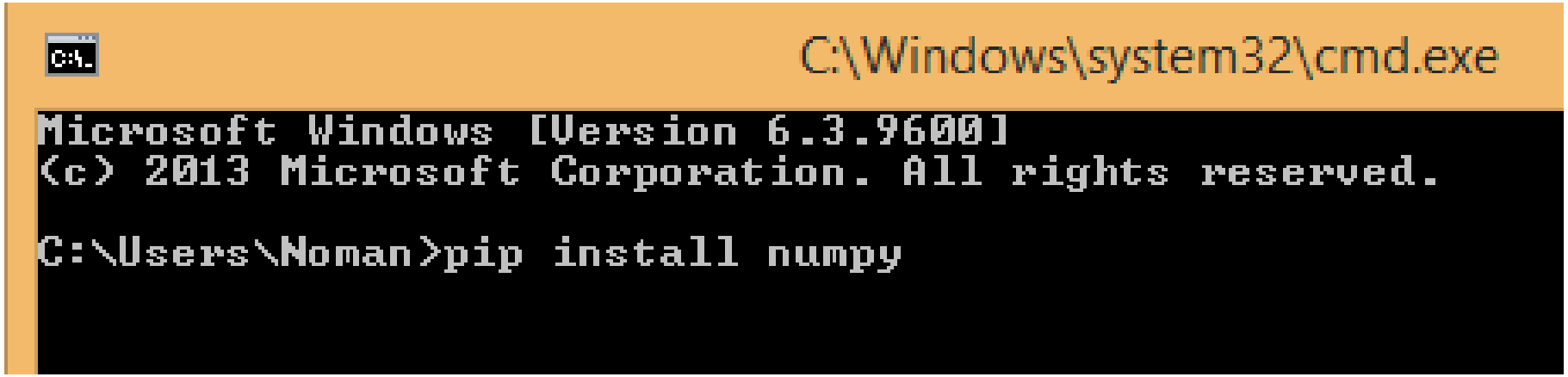
Travis Oliphant

Vectorized operations



Installation

- Prepackaged with Anaconda
- Using pip
  - pip install numpy



Numpy Vs Python list

```
In [7]: import numpy as np

In [8]: my_arr = np.arange(1000000)

In [9]: my_list = list(range(1000000))
```

```
In [10]: %time for _ in range(10): my_arr2 = my_arr * 2
CPU times: user 20 ms, sys: 50 ms, total: 70 ms
Wall time: 72.4 ms

In [11]: %time for _ in range(10): my_list2 = [x * 2 for x in my_list]
CPU times: user 760 ms, sys: 290 ms, total: 1.05 s
Wall time: 1.05 s
```

nd-array

- A fast, flexible container for large datasets in Python
- Homogeneous data i.e. all of the elements must be the same type

Creating ndarray

- np.array(): convert input data to an ndarray
- np.zeros(): produces arrays of 0s
- np.ones(): produces arrays of 1s
- np.empty(): create new arrays by allocating new memory, but do not populate with any values
- np.arange(): like the built-in range but returns an ndarray instead of a list

Examples

```
np.zeros(9)
```

0 0 0 0 0 0 0 0 0

```
np.ones(9)
```

1 1 1 1 1 1 1 1 1

```
np.zeros((3, 3))
```

0 0 0
0 0 0
0 0 0

```
np.ones((3, 3))
```

1 1 1
1 1 1
1 1 1

Summary of array creation functions

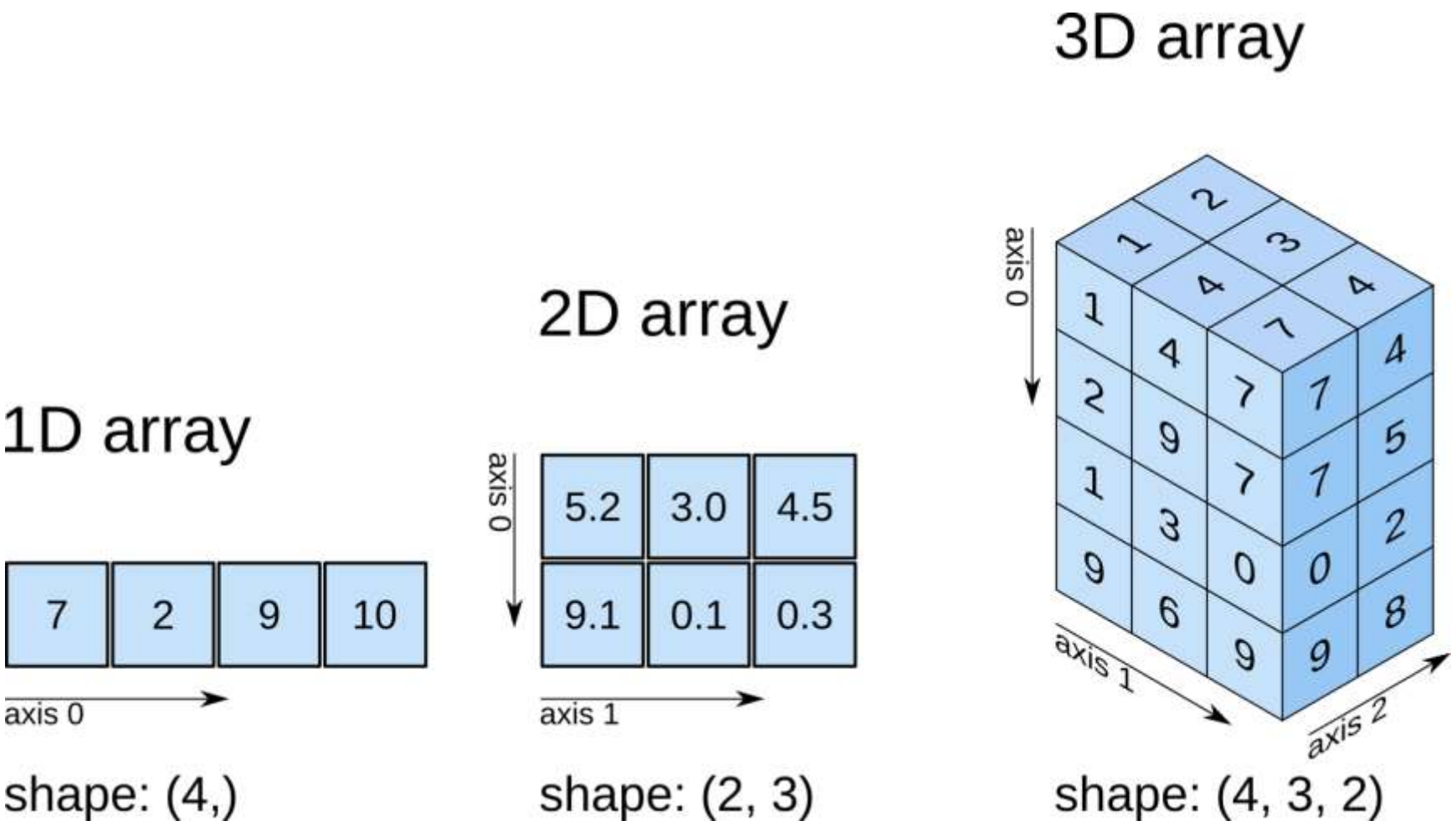
Table 4-1. Array creation functions

Function	Description
`array`	Convert input data (list, tuple, array, or other sequence type) to an ndarray either by inferring a dtype or explicitly specifying a dtype; copies the input data by default
`asarray`	Convert input to ndarray, but do not copy if the input is already an ndarray
`arange`	Like the built-in `range` but returns an ndarray instead of a list
`ones`, `ones_like`	Produce an array of all 1s with the given shape and dtype; `ones_like` takes another array and produces a ones array of the same shape and dtype
`zeros`, `zeros_like`	Like `ones` and `ones_like` but producing arrays of 0s instead
`empty`, `empty_like`	Create new arrays by allocating new memory, but do not populate with any values like `ones` and `zeros`
`full`, `full_like`	Produce an array of the given shape and dtype with all values set to the indicated "fill value" `full_like` takes another array and produces a filled array of the same shape and dtype
`eye, identity`	Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)

Attributes of numpy array

- shape: a tuple indicating the size of each dimension
- dtype: an object describing the data type of the array
- ndim: the number of dimensions of the array

shape attribute



Numpy data types

Type	Type code	Description
`int8, uint8`	`i1, u1`	Signed and unsigned 8-bit (1 byte) integer types
`int16, uint16`	`i2, u2`	Signed and unsigned 16-bit integer types
`int32, uint32`	`i4, u4`	Signed and unsigned 32-bit integer types
`int64, uint64`	`i8, u8`	Signed and unsigned 64-bit integer types
`float16`	`f2`	Half-precision floating point
`float32`	`f4 or f`	Standard single-precision floating point; compatible with C float
`float64`	`f8 or d`	Standard double-precision floating point; compatible with C double and Python `float` object
`float128`	`f16 or g`	Extended-precision floating point
`complex64, complex128, complex256`	`c8, c16, c32`	Complex numbers represented by two 32, 64, or 128 floats, respectively
`bool`	`?`	Boolean type storing `True` and `False` values
`object`	`O`	Python object type; a value can be any Python object
`string_`	`S`	Fixed-length ASCII string type (1 byte per character); for example, to create a string dtype with length 10, use `'S10'`
`unicode_`	`U`	Fixed-length Unicode type (number of bytes platform specific); same specification semantics as `string_` (e.g., `'U10'`)

Vectorization

- Express batch operations on data without writing any for loops
- Any arithmetic operations between equal-size arrays applies the operation element-wise
- Arithmetic operations with scalars propagate the scalar argument to each element in the array
- Comparisons between arrays of the same size yield boolean arrays
- Operations between differently sized arrays is called broadcasting

Operations on two matrices

```
In [51]: arr = np.array([[1., 2., 3.], [4., 5., 6.]])

In [52]: arr
Out[52]:
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])

In [53]: arr * arr
Out[53]:
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])

In [54]: arr - arr
Out[54]:
array([[ 0.,  0.,  0.],
       [ 0.,  0.,  0.]])
```

Operations between matrix and scalar

```
In [55]: 1 / arr
Out[55]:
array([[ 1.    ,  0.5   ,  0.3333],
       [ 0.25  ,  0.2   ,  0.1667]])

In [56]: arr ** 0.5
Out[56]:
array([[ 1.    ,  1.4142,  1.7321],
       [ 2.    ,  2.2361,  2.4495]])
```

Comparison between matrices

```
In [4]: a

Out[4]: array([-0.96206195, -0.98798431, -0.16438853,  1.20690665, -0.56897293,
               -0.01326404, -0.17575675,  0.76762435,  0.56095689, -0.85087959])
```

```
In [10]: a>0

Out[10]: array([False, False, False,  True, False, False, False,  True,  True,
                False])
```

```
In [11]: a[a>0]

Out[11]: array([1.20690665, 0.76762435, 0.56095689])
```

Array indexing

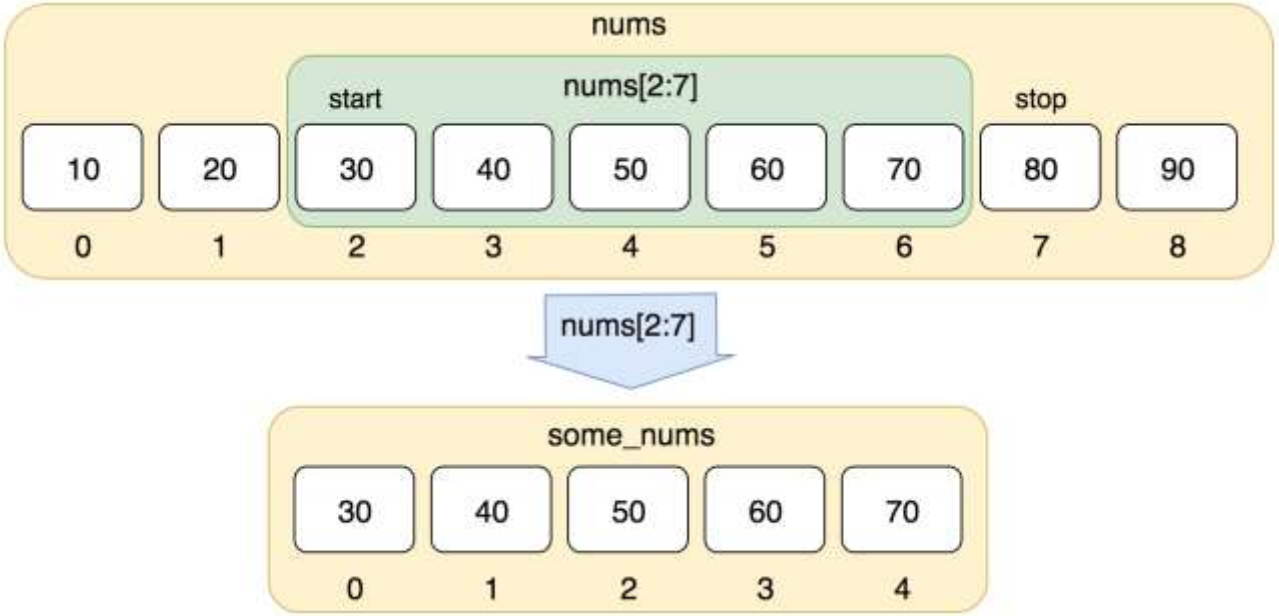
```
In [2]: x=np.array([1,7,9,8,5])

In [3]: x[4]
Out[3]: 5
```

Fancy indexing

```
In [120]: arr[[4, 3, 0, 6]]
```

Slicing

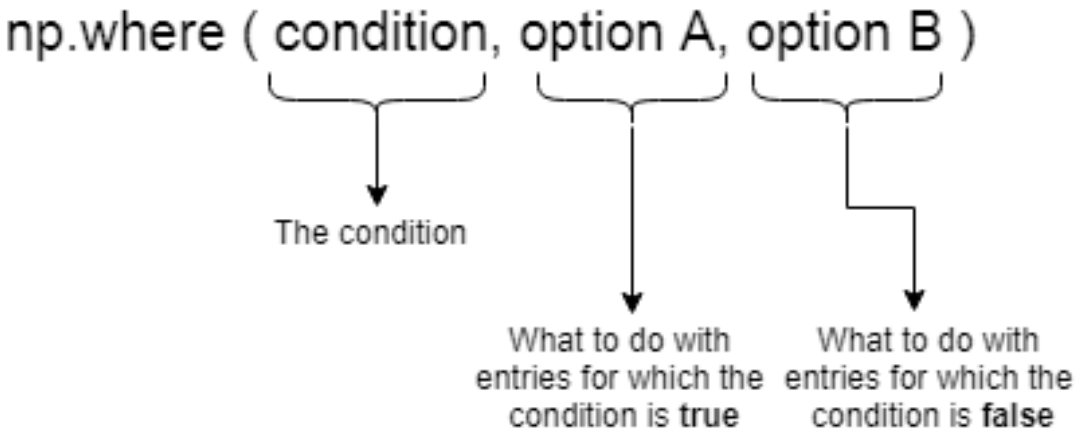
- Array slices are views on the original array
- Format
  - start: end: step



Element-wise array functions

Function	Description
`abs, fabs`	Compute the absolute value element-wise for integer, floating-point, or complex values
`sqrt`	Compute the square root of each element (equivalent to `arr ** 0.5`)
`square`	Compute the square of each element (equivalent to `arr ** 2`)
`exp`	Compute the exponent e^x of each element
`log, log10, log2, log1p`	Natural logarithm (base *e*), log base 10, log base 2, and log(1 + x), respectively
`sign`	Compute the sign of each element: 1 (positive), 0 (zero), or –1 (negative)
`ceil`	Compute the ceiling of each element (i.e., the smallest integer greater than or equal to that number)
`floor`	Compute the floor of each element (i.e., the largest integer less than or equal to each element)
`rint`	Round elements to the nearest integer, preserving the `dtype`
`modf`	Return fractional and integral parts of array as a separate array
`isnan`	Return boolean array indicating whether each value is `NaN` (Not a Number)
`isfinite, isinf`	Return boolean array indicating whether each element is finite (non-`inf`, non-`NaN`) or infinite, respectively
`cos, cosh, sin, sinh, tan, tanh`	Regular and hyperbolic trigonometric functions
`arccos, arccosh, arcsin, arcsinh, arctan, arctanh`	Inverse trigonometric functions
`logical_not`	Compute truth value of `not x` element-wise (equivalent to `~arr`).

np.where()




```
In [3]: salary=np.array([0,-1,100000,50000])

In [4]: np.where(salary<=0,25000,salary)
Out[4]: array([ 25000,  25000, 100000,  50000])
```

Transpose of a matrix

```
In [128]: arr.T
Out[128]:
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

```
In [134]: arr.transpose((1, 0, 2))
Out[134]:
array([[[ 0,  1,  2,  3],
        [ 8,  9, 10, 11]],
       [[ 4,  5,  6,  7],
        [12, 13, 14, 15]]])
```

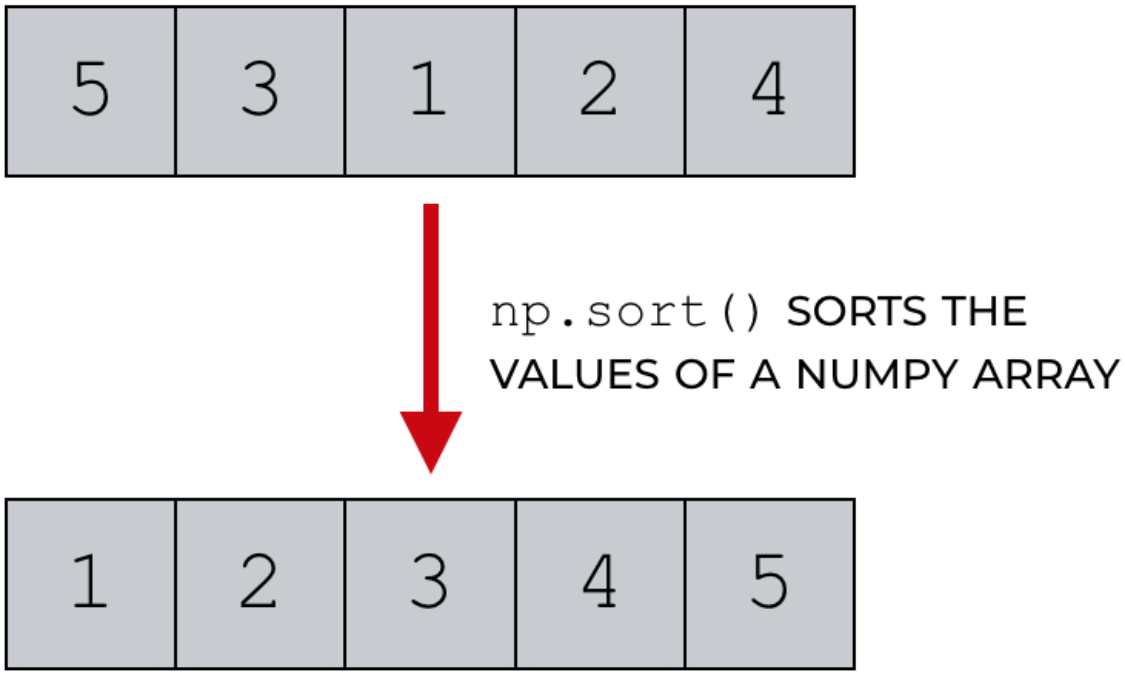
Mathematical and Statistical Methods

- mean(): returns the mean value computed over the array
- cumsum(): returns the cumulative sum of array elements
- cumprod(): returns the cumulative product of array elements

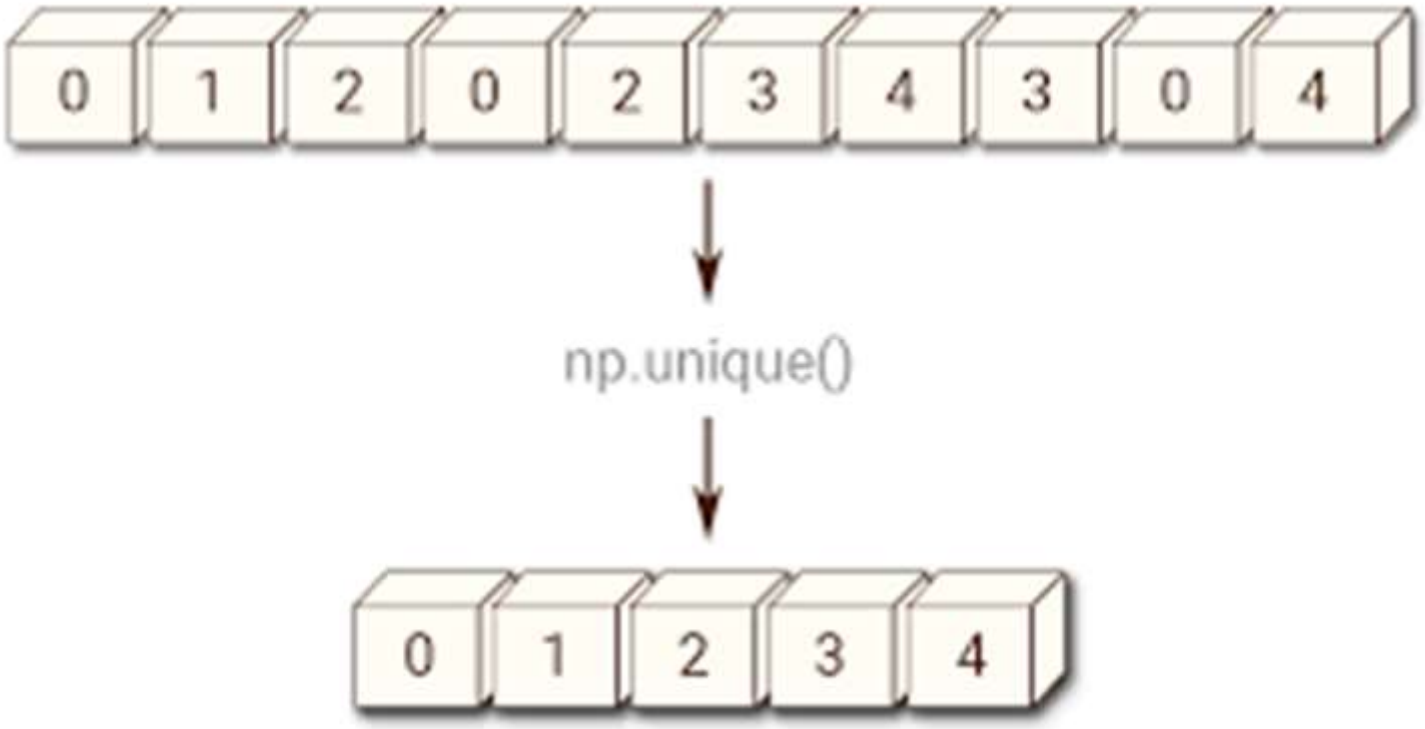
Methods for boolean arrays

- sum(): counting True values in a boolean array
- any(): tests whether one or more values in an array is True
- all(): checks if every value is True

Sorting



Unique



File Input Output

- np.save(*filename.npy*, *array*)
- np.load(*filename.npy*)
- np.savez(*filename.npz*,array1,array2)
- When loading an .npz file, you get back a dict-like object

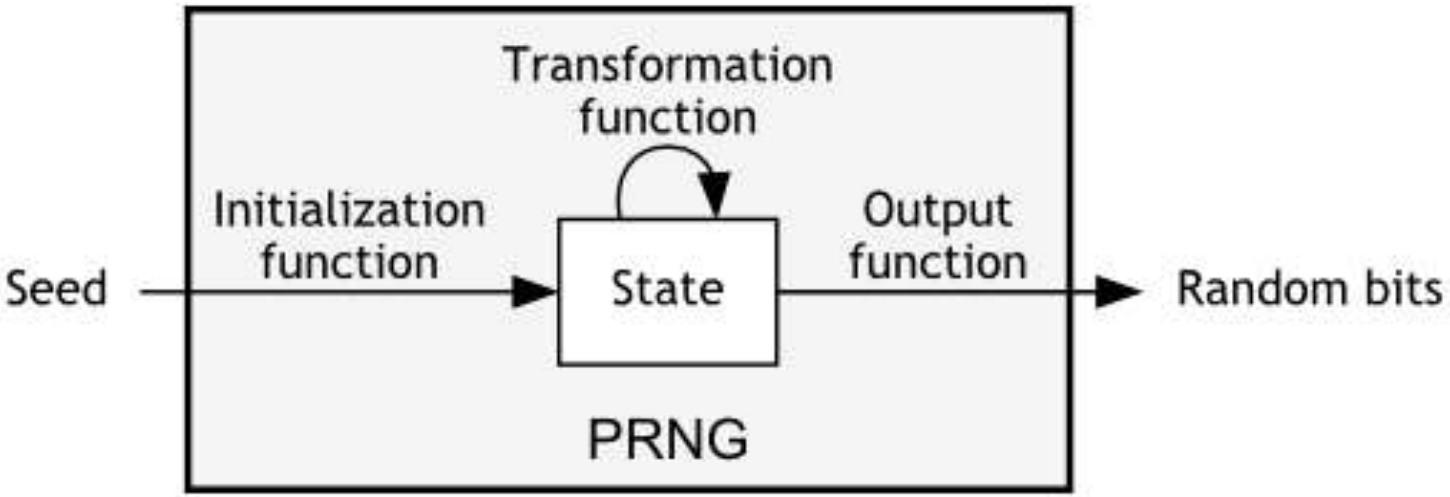
Linear algebra functions

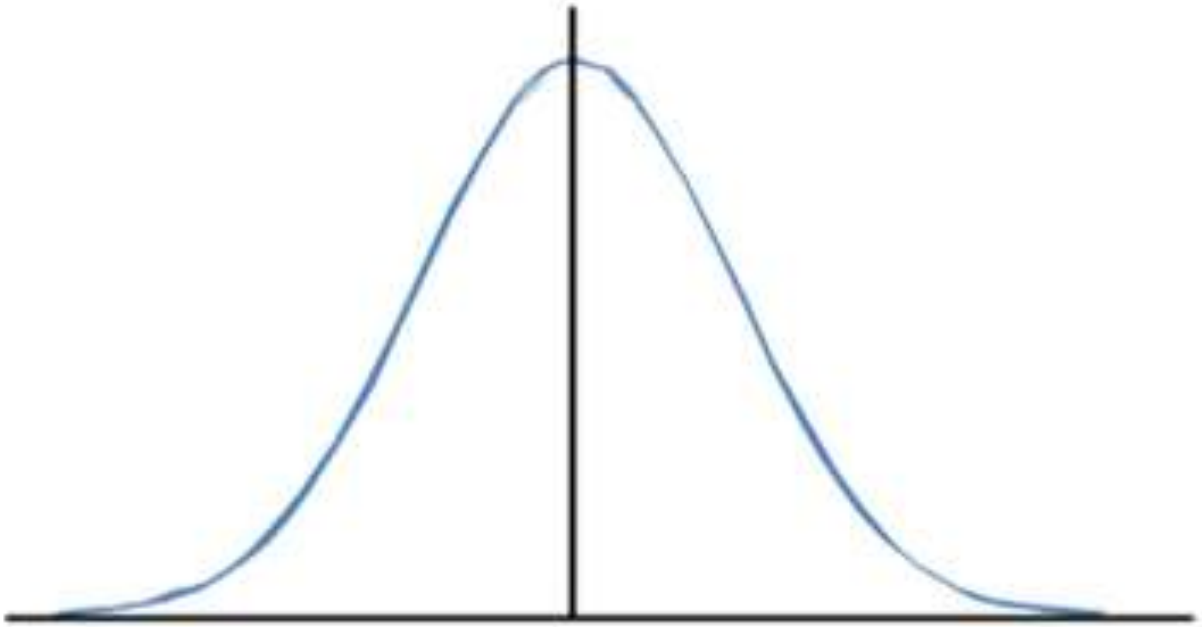
Function	Description
`diag`	Return the diagonal (or off-diagonal) elements of a square matrix as a 1D array, or convert a 1D array into a square matrix with zeros on the off-diagonal
`dot`	Matrix multiplication
`trace`	Compute the sum of the diagonal elements
`det`	Compute the matrix determinant
`eig`	Compute the eigenvalues and eigenvectors of a square matrix
`inv`	Compute the inverse of a square matrix
`pinv`	Compute the Moore-Penrose pseudo-inverse of a matrix
`qr`	Compute the QR decomposition
`svd`	Compute the singular value decomposition (SVD)
`solve`	Solve the linear system Ax = b for x, where A is a square matrix
`lstsq`	Compute the least-squares solution to Ax = b

Pseudo-random number generation

- normal()
- seed()
- gamma()
- uniform()

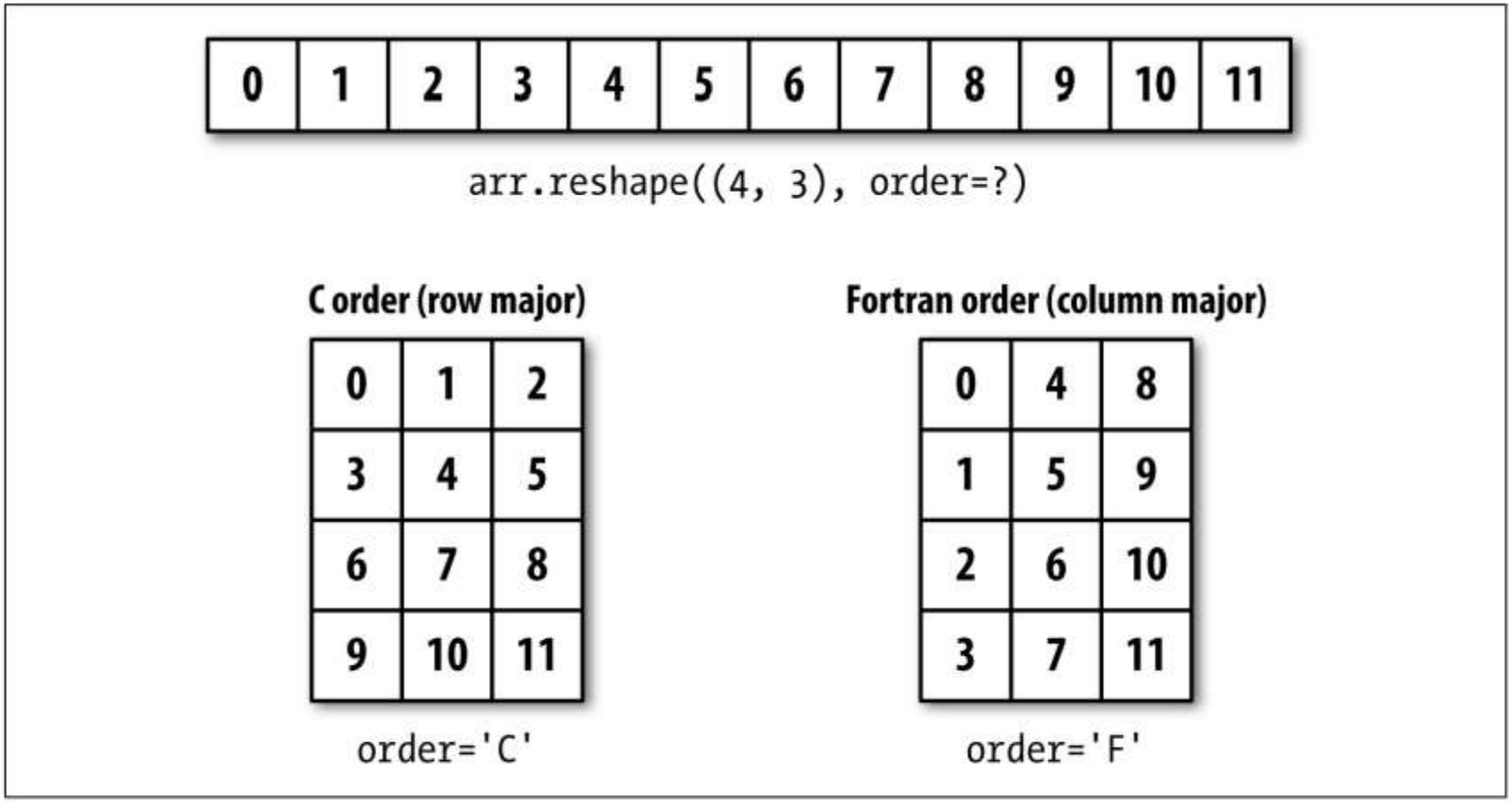
Pseudo random number generator





Normal distribution curve.

Reshape



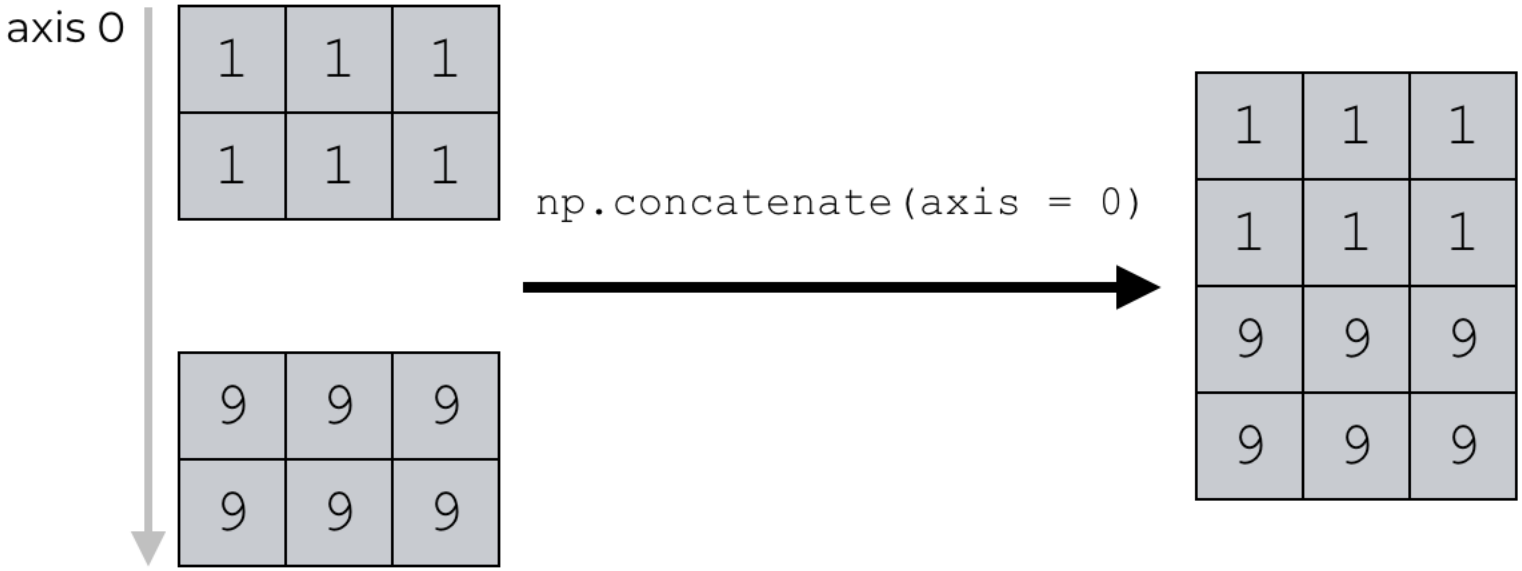
Reshaping in C (row major) or Fortran (column major) order

Converting to 1D array

- flatten()
- ravel()

concatenate

Setting `axis=0` concatenates along the row axis



Convenience functions

- vstack: stack arrays row-wise (along axis 0)
- hstack: stack arrays column-wise (along axis 1)

Stacking helpers

- np.r_[arr1, arr2]
- np.c_[np.r_[arr1, arr2], arr]

Splitting an array