**NumPy** for numerical computing on arrays

NumPy is a Python package for numerical computing via its **ndarray**, an *n*-dimensional array of values of the same type. Get access to it via `import numpy as np`.

Create an ndarray

- `np.array(x)`: array from a list or tuple `x`
- `np.zeros(shape)`: array of zeros with dimension(s) given by `shape` tuple
- `np.full(shape, fill_value)`: array of length `shape` filled with `fill_value`
- `A.shape`: gives tuple of dimension(s) of `A`; `A.reshape(shape)` changes dimension(s) to `shape`; if `shape` includes `-1`, the required value is inferred
- `np.arange([start[, stop[, step]])`: gives step-spaced values over `[start, stop]`
- `np.linspace(start, stop, num=50)`: gives `num` evenly-spaced values over `[start, stop]`
- `c = a.copy()` makes a copy of `ndarray` `a` (note: `v = a` gives a view—changes to `v` affect `a`)

  e.g. `np.array(['a', 'b']), np.zeros(3), np.full(3, 2.71), np.arange(0, 10, 2), np.linspace(0, 10)
  np.arange(6), A.shape, A.reshape((-1, 2))`

  e.g. `a = np.arange(3); copy = a.copy(); copy[0] = 10; view = a; view[1] = 11;`  
  `print(f'a={a}, copy={copy}, view={view}'))`  
  `# show with a = [0, 1, 2] at pythontutor.com`

**Array types (dtype)**

- `int`: integers (..., −1, 0, 1, 2, ...), e.g. `np.array([1, 2, 3])`
- `float`: real numbers (the whole number line), e.g. `A = np.array([1, np.sqrt(2), np.e, np.pi]); A`
- `bool`: True or False (which become 1 and 0 when used in arithmetic), e.g. `1 < A, np.sum(1 < A)`
- `str`: strings, e.g. `np.array(['apple', 'banana', 'cherry'])`

**A few functions**

- `np.amin(), np.argmin(); np.amax(), np.argmax(); np.sum()`: array minimum, index; maximum, index; sum; e.g. `np.amax(A), np.argmax(A)`
- `np.mean(), np.median(), np.std(a, ddof=1)`: mean, median, standard deviation
- `np.sort(), np.argsort(), np.flip()`: sort, indices to sort (a[np.argsort(a)] is sorted), flip; e.g. `B = np.array([13, 10, 12, 11]); np.sort(B), np.argsort(B), B[np.argsort(B)], np.flip(np.sort(B))`
- `np.isin()`: is in, e.g. `np.isin(np.array([1, 2, 3]), np.array([2, 7]))`
Operators (which act element-wise)

- arithmetic: + - * / ** (and, for integer division, // is quotient, % is remainder)
e.g. The sample standard deviation of $x_1, x_2, \cdots, x_n$ is

$$s_x = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2};$$

```python
x = np.arange(5) + 1
n = len(x); np.sqrt(np.sum((x - np.mean(x))**2) / (n - 1)) # no loop; inspect parts
```

- relation: > >= < <= == != (last two are equals and is not equal to)

- logic:

<table>
<thead>
<tr>
<th>&amp; (and)</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(or)</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>~ (xor)</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>~ (not)</th>
<th>T</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
</tr>
</tbody>
</table>

- assignment: = (which is not ==)

Indexing

- A[i] (ith value), A[-i] ((n - i)th), and A[low:high] work as with sequences

- For an array a of int, x[a] is those values \{x[i]\} for each index i in a; e.g.

```python
x = np.arange(10, 20); a = np.array([0, 9]); x[a], x[np.array([0, -1])]
```

- For an array a of bool,

  - np.nonzero(a) is an array of indices for which a[i] is TRUE; e.g. indices = np.nonzero(1 < A)

    Now use the indices, e.g. A[indices]

  - x[a] is those elements of x corresponding to True values in a \(\text{so } \text{“np.nonzero()} \text{” was unnecessary in previous example}\), e.g. A[1 < A], x[(x % 2) == 0]

    The idiom is that A[condition on A] gives values of A satisfying the condition.

  - np.all(a) tells whether all values in a are True; np.any(a) tells whether any are True, e.g. np.all(1 < A), np.any(1 < A)

Loop through values or indices, as with sequences

```python
for value in x:
    print(f' value={value}')
for i in np.arange(len(x)):
    print(f' i={i}, x[{i}]={x[i]}')
```

File input/output

np.loadtxt(fname, dtype, comments='#') reads dtype from file fname, ignoring comments lines

np.savetxt(fname, X) saves X to fname, e.g. np.savetxt('A.txt', A), C = np.loadtxt('A.txt', float)

To learn more, see [NumPy quickstart] [NumPy mathematical functions] [Numpy copies and views]