COMP1730/COMP6730
Programming for Scientists

NumPy special

Lecture outline

* Recap of Sequences
* Numpy arrays

Sequences

* A *sequence* contains zero or more values.
* Each value in a sequence has a *position*, or *index*, ranging from 0 to $n - 1$.
* The *indexing operator* can be applied to all sequence types, and returns the value at a specified position in the sequence.
  - Indexing is done by writing the index in square brackets after the sequence value, like so:

    `sequence[pos]`

Sequence data types

* python has three built-in sequence types:
  - strings (`str`) contain only text;
  - lists (`list`) can contain a mix of value types;
  - tuples (`tuple`) are like lists, but immutable.
* Sequence types provided by other modules:
  - NumPy arrays (`numpy.ndarray`).

NumPy arrays

NumPy Arrays

* (Assuming `import numpy as np`.)
* `np.ndarray` is sequence type, and can also represent *n*-dimensional arrays.
  - `len(A)` is the size of the first dimension.
  - Indexing an *n*-d array returns an $(n-1)$-d array.
  - `A.shape` is a sequence of the size in each dimension.
* All values in an array must be of the same type.
  - Typically numbers (integers, floating point or complex) or Booleans, but can be any type.

NumPy and SciPy

* The NumPy and SciPy libraries are not part of the python standard library, but often considered essential for scientific / engineering applications.
* The NumPy and SciPy libraries provide
  - an *n*-dimensional array data type (`ndarray`);
  - fast math operations on arrays/matrices;
  - linear algebra, Fourier transform, random number generation, signal processing, optimisation, and statistics functions;
  - plotting (via `matplotlib`).
* Documentation: `numpy.org` and `scipy.org`.

The NumPy `ndarray` type

* `ndarray` is a sequence type.
* All values in an array must be of the same type.
* Typically numbers (integers, floating point or complex) or Booleans, but can be any type.

```
>>> import numpy as np
>>> x = np.array([1.0, 2, 3])
>>> x
array([ 1., 2., 3.])
>>> type(x)
<class 'numpy.ndarray'>
>>> x.dtype
dtype('float64')
```

Indexing & length

array:	3.0	1.5	0.0	-1.5	-3.0
index:	0	1	2	3	4
	-5	-4	-3	-2	-1

* In python, all sequences are indexed from 0.
* The index must be an integer.
* python also allows indexing from the sequence end using negative indices, starting with -1.
* The length of a sequence is the number of elements, *not* the index of the last element.

* `len(`*`sequence`*`)` returns sequence length.
* Sequence elements are accessed by writing the index in square brackets, `[]`.

```
>>> x = np.array([3, 1.5, 0, -1.5, -3])
>>> x[1]
1.5
>> x[-1]
-3.0
>>> len(x)
5
>>> x[5]
IndexError: index 5 is out of bounds
            for axis 0 with size 5
```

More operations on NumPy arrays

Creating 1-dimensional arrays

```
>>> np.zeros(5)
array([ 0., 0., 0., 0., 0.])
>>> np.ones(3) * 5
array([ 5., 5., 5.])
>>> np.linspace(3, -3, 5)
array([3. , 1.5, 0. , -1.5, -3. ])
>>> import numpy.random as rnd
>>> rnd.normal(0, 2, 10)
array([0.11224282, -1.84772958, ...
```

Element-wise operators

* Arithmetic (`+`,`-`,`*`,`/`,`**`,`//`,`%`), comparison (`==`, `!=`,`<`,`>`,`<=`,`>=`) and logical (`&`, `|`) operators can be applied to
  - an `ndarray` and a single value: the operation is done between each element of the array and the value; or
  - two `ndarray`s of the same size: the operation is done between pairs of elements in equal positions.

```
>>> x = np.array([-2., -1., 0., 1., 2.])
>>> -(x ** 2) + 2
array([-2., 1., 2., 1., -2.])
>>> y = 2 ** x
>>> y
array([ 0.25, 0.5, 1., 2., 4.])
>>> x + y
array([ -1.75, -0.5, 1., 3., 6.])
>>> x + array([1., -1.])
ValueError: operands could not be
broadcast with shapes (5,) (2,)
```

* NumPy provides most math functions (e.g., `cos`, `exp`, `log`, `sqrt`, etc) that also work element-wise on arrays.

```
>>> x = np.linspace(-np.pi, np.pi, 9)
>>> np.cos(x)
array([-1.00e+00, -7.07e-01,  6.12e-17,
        7.07e-01,  1.00e+00,  7.07e-01,
        6.12e-17, -7.07e-01, -1.00e+00])
>>> np.sqrt(x)
RuntimeWarning:  invalid value ...
array([ nan, nan, 0., 1., 1.41421356])
```

Functions of arrays

```
>>> x = np.linspace(-1, 3, 5)
>>> np.min(x ** 2)
0.0
>>> np.max(x)
3.0
>>> np.sum(x)
5.0
>>> np.mean(x)
1.0
>>> np.std(x)
1.4142135623730951
```

Generalised indexing

* Most python sequence types support *slicing* – accessing a subsequence by indexing a range of positions:

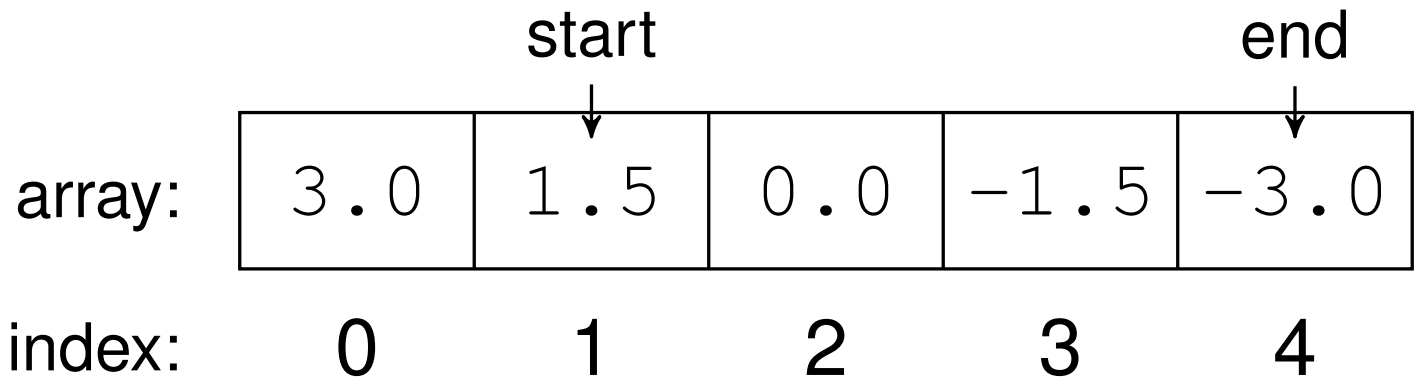
  `sequence[start:end]`

* Unique to NumPy array:
  - Indexing with an *array of integers* selects elements from the positions in the index array.
  - Indexing with an *array of Booleans* selects elements from the positions where the index array contains `True`.

Slicing

* The slice range is “half-open”: start index is included, end index is one after last included element.

```
>>> x = np.array([3, 1.5, 0, -1.5, -3])
>>> x[1:4]
array([ 1.5,  0, -1.5])
```



Indexing with arrays

```
>>> x = np.array([3, 1.5, 0, -1.5, -3])
>>> i = np.array([0, 1, 4])
>>> x[i]
array([ 3., 1.5., -3.])
>>> j = (x == np.floor(x))
>>> j
array([True, False, True, False, True])
>>> x[j]
array([ 3., 0., -3.])
```

Generalised indexing

* If `L` is an array of `bool` of the same size as `A`, `A[L]` returns an array with the elemnts of `A` where `L` is `True` (does not preserve shape).
* If `I` is an array of integers, `A[I]` returns an array with the elemnts of `A` at indices `I` (does not preserve shape).
* If `A` is a 2-d array,
  - `A[i,j]` is element at `i`, `j` (like `A[i][j]`).
  - `A[i,:]` is row `i` (same as `A[i]`).
  - `A[:,j]` is column `j`.
  - `:` can be `start:end`.

Copying and reshaping

* Most indexing/slicing operations on arrays do *not* copy, but return a "view" into the array.
* `np.copy(A)` copies array `A`.
* `np.reshape(A, shape)` returns a copy of the elements in `A` arranged into `shape` (size must match).
* `np.concatenate((A, B), axis = i)` returns a new array with `A` and `B` concatenated along dimension `i` (sizes must be equal in all other dimensions).