Introduction to Programming Language (ITP101)

Python Packages: NumPy

Mulualem Teku

GCIT, Bhutan

Nov 2019

Brainstorm

- Lists? Arrays? Lists vs Arrays?
- What is a **module** in Python? How do you use one?
- Consider the following code snippet. Output? Efficiency?

```
x = range(1000000)
y = range(1000000)
L = []

for i in range(len(x)):
        L.append(x[i] + y[i])
print (L)
```

Brainstorm

- Lists? Arrays? Lists vs Arrays?
- What is a **module** in Python? How do you use one?
- Consider the following code snippet. Output? Efficiency?

```
x = range(10000000)
y = range(10000000)
L = []

for i in range(len(x)):
        L.append(x[i] + y[i])
print (L)
```

Recap: Modules

- A module is a file containing Python definitions and statements.
- Can contain executable statements and function definitions.
- Be imported to be used using the import statement:

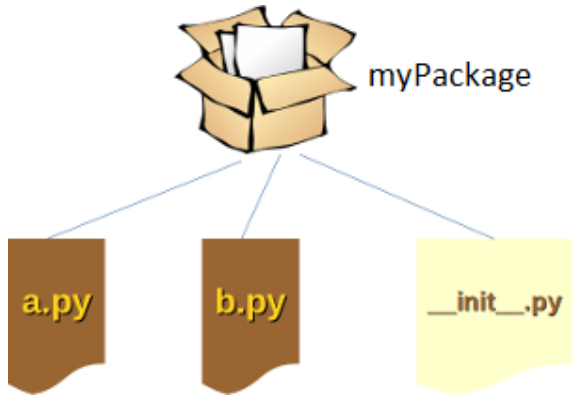
```
import <module_name> [as <someVariable>]

from <module_name> import <moduleMembers>   (or * for all)
```

- The dir() function shows names a module defines.
- The sys.path shows list of module search directories.

Packages

- Simply a collection of modules.
- Package is a directory housing Python files and a special file named __init__.py. The special file helps Python treat directories containing the files as packages.



- It is a way of structuring Python's module namespace by using "dotted module names". E.g. a module named `X.Y` designates a submodule named Y inside a package named X.

Example: sound package

```
sound/                          Top-level package
      __init__.py               Initialize the sound package
      formats/                  Subpackage for file format conversions
              __init__.py
              wavread.py
              wavwrite.py
              aiffread.py
              aiffwrite.py
              auread.py
              auwrite.py
              ...
      effects/                  Subpackage for sound effects
              __init__.py
              echo.py
              surround.py
              reverse.py
              ...
      filters/                  Subpackage for filters
              __init__.py
              equalizer.py
              vocoder.py
              karaoke.py
              ...
```

(source: Python.org)

- `NumPy` (Numerical Python) is an open-source numeric Python extension.
- Based on two earlier Python array modules: `Numeric` and `Numarray` (deprecated now).
  - `Numeric` was designed for high-performance, numeric computing.
  - `NumArrray` was a rewrite of `Numeric`.
- NumPy merges these two for enhanced numeric/scientific computing.

NumPy - Numerical Python (2)

- Provides the array object for efficient manipulation of large, multidimensional arrays and matrices.
- Lists vs arrays? The `array` module vs NumPy's array?
- Provides large library of mathematical functions on the arrays.
- Not installed by default. You can get it from www.numpy.org.

Why the Extension?

1. Manipulation of millions of numbers using the standard objects (e.g. lists, tuples) is *inefficient*.
   - Speed
   - Space
2. The need for complex operations on these objects.
   - Numerical analysis
   - Fourier transforms
   - Image processing
   - Interpolations
   - Linear algebra
   - Signal processing
   - Stat and random numbers
   - etc...

Why the Extension?

1. Manipulation of millions of numbers using the standard objects (e.g. lists, tuples) is *inefficient*.
   - Speed
   - Space
2. The need for complex operations on these objects.
   - Numerical analysis
   - Image processing
   - Linear algebra
   - Stat and random numbers
   - Fourier transforms
   - Interpolations
   - Signal processing

   etc...

Why the Extension? (2)

3. But there are already tools such as Matlab, FORTRAN, etc... right?
   - Python is a free, modern and powerful alternative.
   - `SciPy`, which is based on `NumPy`, adds even more Matlab-like functionalities.
   - With `Matplotlib`, plotting functionality can be added to Python.

- Usually, NOT installed by default.

- Once installed successfully, `NumPy` can be invoked just like any other module.

```
>>>import numpy

>>>import numpy as np

>>>from numpy import <specific_items>

>>>from numpy import *
```

Array Objects

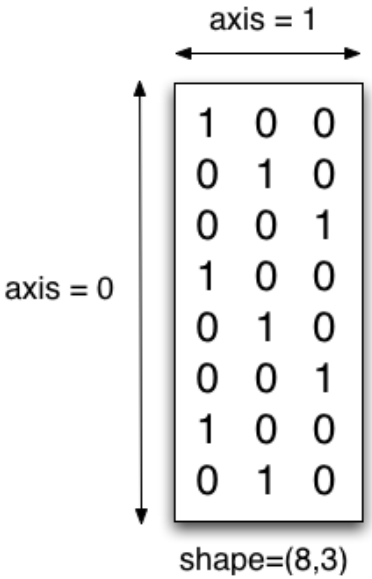
- The central object in `NumPy` is the homogeneous and multidimensional array object.
- Holds large block of *similar* elements (unlike most container objects).

Key attributes of `array`

`Size`	Total number of elements.
`Rank`	Number of dimensions/axes of the array.
`Shape`	A tuple of integers indicating the size of the array in each dimension.
`Typecode`	Describes the type of the elements in the array (int, float,complex, etc)
`Itemsize`	The size in bytes of each element of the array.

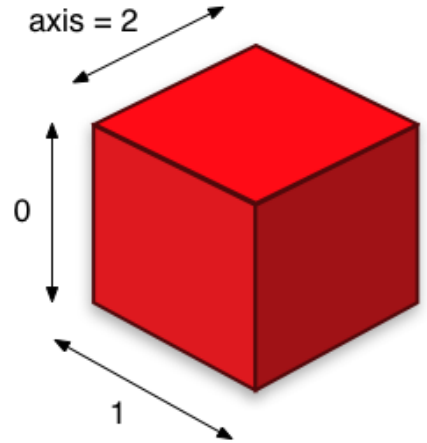
Array Objects (2)

Anatomy of an array



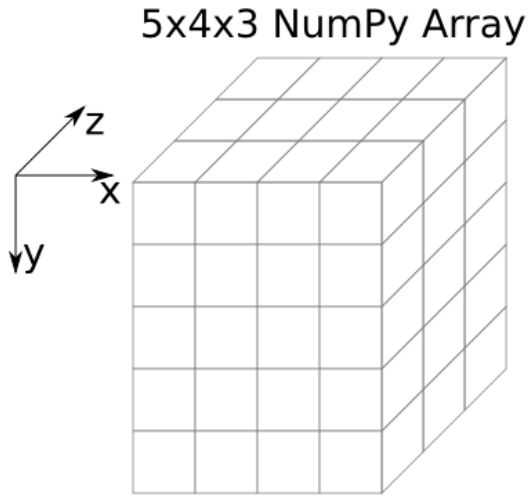
The **axes** of an array describe the order of indexing into the array, e.g., axis=0 refers to the first index coordinate, axis=1 the second, etc.

The **shape** of an array is a tuple indicating the number of elements along each axis. An existing array **a** has an attribute **a.shape** which contains this tuple.



- all elements must be of the same dtype (datatype)

- arrays constructed from list of mixed dtype will be upcast to the "greatest" common type



Constructing Arrays

- Using the `array()` built-in.

```
array(values [, typecode])
```

<values> must be a single list of elements, rather than multiple arguments.

Example

```
>>>import numpy as np
>>>a = np.array(1,2,3,4)                    # ??
>>>a = np.array([1,2,3,4])
>>>print (a)
>>>a.size
>>>a.ndim                                  # rank/dimension
>>>a.shape
```

Constructing Arrays (2)

Example

```
b = np.array([(5 ,6), (7, 8)], float)       # float type
>>>print (b)
>>>b.ndim
>>>b.itemsize
>>>b.dtype                                  # Typecode
```

Constructing Arrays (2)

Example

```
b = np.array([(5 ,6), (7, 8)], float)       # float type
>>>print (b)
>>>b.ndim
>>>b.itemsize
>>>b.dtype                                  # Typecode
```

- Using the `arange()` built-in - similar to the `range()` function.

```
arange(start, end, offset)
```

```
>>>from numpy import arange

>>>arange(1,10, 2)        >>>arange(5,20, 0.5)        >>>arange(50,10, -3)
```

Constructing Arrays (3)

- Using the constructors for common arrays:
  - `zeros()`
  - `ones()`
  - `empty()`
  - `identity()`

Example

```
>>>zeros((3,3))                    # 3x3 array of zeros

>>>ones((4,4,4), int)              # 4x4x4 array of ones

>>>empty((2,2))                    # random content

>>>identity(5)                     # 5x5 identity matrix
```

Basic Operations

1. Indexing - access individual elements via indexes.
2. Element-wise operations, broadcasting
3. Slicing (1D, $\geq$2D)

1. Indexing

Example

```
>>>a = arange(1,11)
>>>print (a[0])
>>>print (a[-1])
>>>print (a[11])
```

1. Indexing

Example

```
>>>a = arange(1,11)
>>>print (a[0])
>>>print (a[-1])
>>>print (a[11])
```

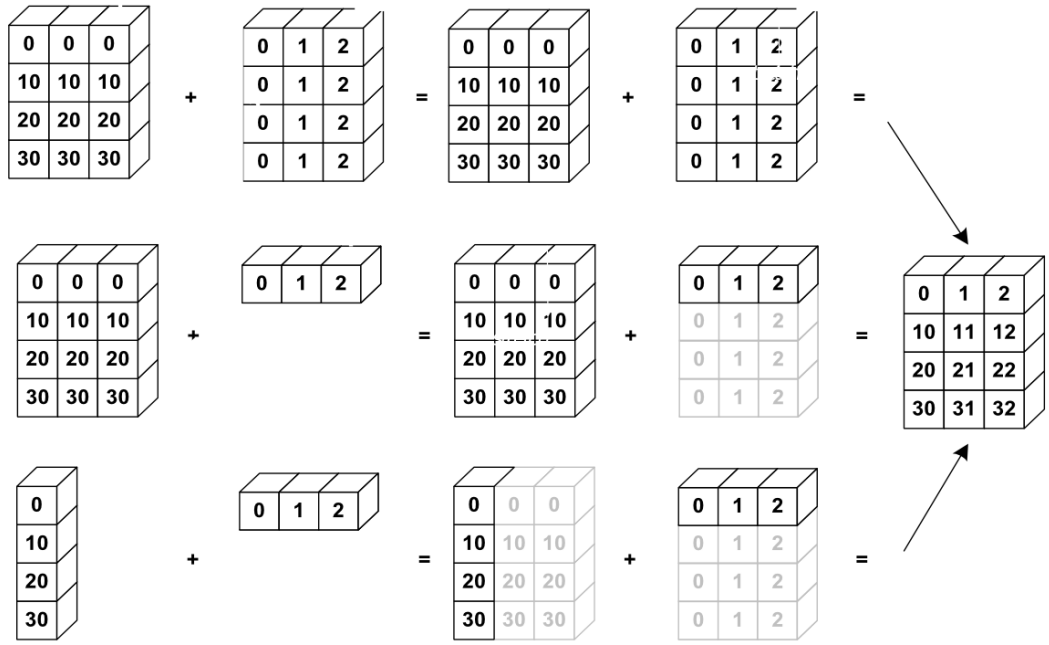
2. Element-wise Math Operations

Example

```
>>>b = array([1,2,3,4])
>>>print (b + 3)              # a.k.a broadcasting
>>>print (b * b)              # also /, -, %
>>>print (-b)
>>>print ( cos(b) )
```

Broadcasting

- `NumPy`'s attempt to match shapes of arrays.
- Smaller-ranked array is replicated/extended (broadcast) across the larger.



3 Slicing

Example

```
>>>x = arange(20,1,-2)
>>>x[1:5]
>>>x[:10:3] = 0
>>>for i in x:
...     print (i**2)
```

1D Slicing

Example

```
>>>a = array([[1,2,3], [4,5,6],[7,8,9]])
>>>print (a[0][0])
>>>print (a[0,0])          # same as above
>>>a[0:3,1]                # ??
>>>a[:,2]                  # ??
>>>a[1:, 1:]               # ??
```

≥ 2D Slicing

3 Slicing

Example

```
>>>x = arange(20,1,-2)
>>>x[1:5]
>>>x[:10:3] = 0
>>>for i in x:
...     print (i**2)
```

1D Slicing

Example

```
>>>a = array([[1,2,3], [4,5,6],[7,8,9]])
>>>print (a[0][0])
>>>print (a[0,0])            # same as above
>>>a[0:3,1]                  # ??
>>>a[:,2]                    # ??
>>>a[1:, 1:]                 # ??
```

$\geq$ 2D Slicing

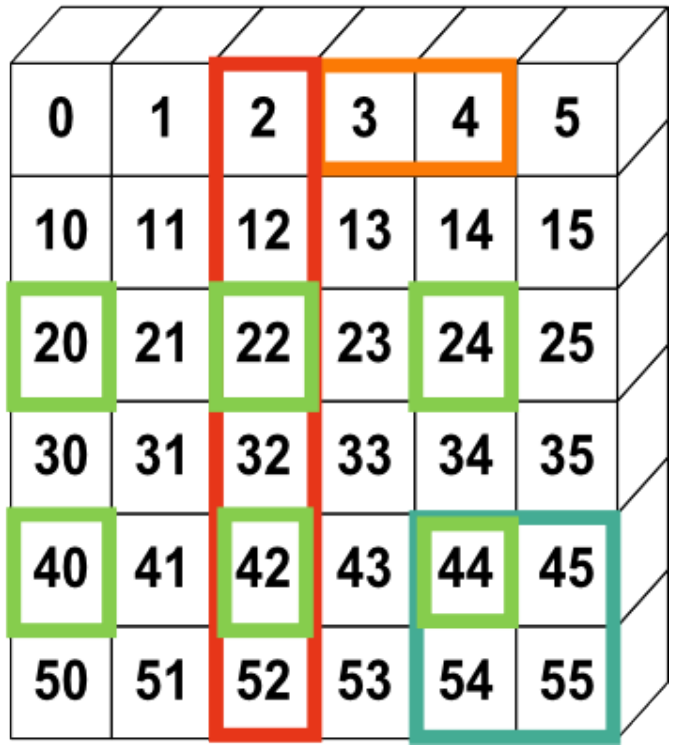
Slicing (2)

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Useful Resources

Check these out to get on track real quick:

- NumPy's quickstart tutorial at the official site here.

- Python's/IPython's `help()` facility

```
>>>help(numpy)                    # on the numpy module

>>>help(numpy.array)              # on the array object
```