NP01_Introduction_to_Numpy

December 10, 2022

1 Introduction to NumPy

[1]:
```
import numpy
numpy.__version__
```

[1]:
```
'1.23.5'
```

[2]:
```
import numpy as np
```

[3]:
```
np?
```

```
Type:        module
String form: <module 'numpy' from 'C:
 ↪\\Users\\onkar\\AppData\\Local\\Packages\\PythonSoftwareFoundation.Python.3.
 ↪11_qbz5n2kfra8p0\\LocalCache\\local-packages\\Python311\\site-packages\\numpy\\__init__.
 ↪py'>
File:        c:\users\onkar\appdata\local\packages\pythonsoftwarefoundation.
 ↪python.3.
 ↪11_qbz5n2kfra8p0\localcache\local-packages\python311\site-packages\numpy\__init__.
 ↪py
Docstring:
NumPy
=====

Provides
  1. An array object of arbitrary homogeneous items
  2. Fast mathematical operations over arrays
  3. Linear Algebra, Fourier Transforms, Random Number Generation

How to use the documentation
----------------------------
Documentation is available in two forms: docstrings provided
with the code, and a loose standing reference guide, available from
`the NumPy homepage <https://numpy.org>`_.

We recommend exploring the docstrings using
`IPython <https://ipython.org>`_, an advanced Python shell with
TAB-completion and introspection capabilities.  See below for further
```

```
instructions.

The docstring examples assume that `numpy` has been imported as `np`::

  >>> import numpy as np

Code snippets are indicated by three greater-than signs::

  >>> x = 42
  >>> x = x + 1

Use the built-in ``help`` function to view a function's docstring::

  >>> help(np.sort)
  … # doctest: +SKIP

For some objects, ``np.info(obj)`` may provide additional help.  This is
particularly true if you see the line "Help on ufunc object:" at the top
of the help() page.  Ufuncs are implemented in C, not Python, for speed.
The native Python help() does not know how to view their help, but our
np.info() function does.

To search for documents containing a keyword, do::

  >>> np.lookfor('keyword')
  … # doctest: +SKIP

General-purpose documents like a glossary and help on the basic concepts
of numpy are available under the ``doc`` sub-module::

  >>> from numpy import doc
  >>> help(doc)
  … # doctest: +SKIP

Available subpackages
---------------------
lib
    Basic functions used by several sub-packages.
random
    Core Random Tools
linalg
    Core Linear Algebra Tools
fft
    Core FFT routines
polynomial
    Polynomial tools
testing
    NumPy testing tools
```

```
distutils
    Enhancements to distutils with support for
    Fortran compilers support and more.

Utilities
---------
test
    Run numpy unittests
show_config
    Show numpy build configuration
dual
    Overwrite certain functions with high-performance SciPy tools.
    Note: `numpy.dual` is deprecated.  Use the functions from NumPy or Scipy
    directly instead of importing them from `numpy.dual`.
matlib
    Make everything matrices.
__version__
    NumPy version string

Viewing documentation using IPython
-----------------------------------
Start IPython with the NumPy profile (``ipython -p numpy``), which will
import `numpy` under the alias `np`.  Then, use the ``cpaste`` command to
paste examples into the shell.  To see which functions are available in
`numpy`, type ``np.<TAB>`` (where ``<TAB>`` refers to the TAB key), or use
``np.*cos*?<ENTER>`` (where ``<ENTER>`` refers to the ENTER key) to narrow
down the list.  To view the docstring for a function, use
``np.cos?<ENTER>`` (to view the docstring) and ``np.cos??<ENTER>`` (to view
the source code).

Copies vs. in-place operation
-----------------------------
Most of the functions in `numpy` return a copy of the array argument
(e.g., `np.sort`).  In-place versions of these functions are often
available as array methods, i.e. ``x = np.array([1,2,3]); x.sort()``.
Exceptions to this rule are documented.
```

1.1 Fixed Type Arrays in Python

[4]:
```
import array
L = list(range(10))
A = array.array('i', L) # 'i' indicates that contents are integers
A
```

[4]:
```
array('i', [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

1.2 Creating Arrays from Python Lists

[5]:
```
np.array([1,2,3,4,5])
```

[5]:
```
array([1, 2, 3, 4, 5])
```

[6]:
```
np.array([1,2,3.14,4,5]) # NumPy will upcast if types do not match
```

[6]:
```
array([1.  , 2.  , 3.14, 4.  , 5.  ])
```

[7]:
```
np.array([1,2,3.14,4,5],dtype='float32') # Data type can be explicitly set␣
↪using keyword 'dtype'
```

[7]:
```
array([1.  , 2.  , 3.14, 4.  , 5.  ], dtype=float32)
```

[8]:
```
np.array([range(i,i+3) for i in range(2,7,2)]) # nested lists are converted to␣
↪multidimensional arrays
```

[8]:
```
array([[2, 3, 4],
       [4, 5, 6],
       [6, 7, 8]])
```

1.3 Creating Arrays from Scratch

[9]:
```
np.zeros(10,dtype="int") # length 10 int array with zeroes
```

[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[10]:
```
np.ones((3,5),dtype="float") # 3*5 floating-point array with ones
```

[10]:
```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

[11]:
```
np.full((3,5),3.14) # 3*5 floating-point array with 3.14
```

[11]:
```
array([[3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14],
       [3.14, 3.14, 3.14, 3.14, 3.14]])
```

[12]:
```
np.arange(0,20,2) # starting at 0, ending at 20, sstepping size 2
```

[12]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

[13]:
```
np.linspace(0,1,5) # 5 values evenly spaced between 0 and 1
```

[13]:
```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

[14]:
```
np.random.random((3,3)) # 3*3 array (uniformly distributed) with random values␣
↪between 0 and 1
```

[14]:
```
array([[0.9174918 , 0.4896383 , 0.95786394],
       [0.2186783 , 0.84429385, 0.10723626],
       [0.96688974, 0.2739895 , 0.31981254]])
```

[15]:
```
np.random.normal((3,3)) # 3*3 array (normally distributed) with random values␣
↪between 0 and 1
```

[15]:
```
array([3.40408317, 3.69649029])
```

[16]:
```
np.random.randint(0,10,(3,3)) # 3*3 array with random int in interval [0,10)
```

[16]:
```
array([[9, 3, 3],
       [3, 3, 5],
       [0, 5, 6]])
```

[17]:
```
np.eye(3) # 3*3 identity matrix
```

[17]:
```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

[18]:
```
np.empty(3) # uninitialized array of 3 int with value from previous allocation␣
↪in that memory location
```

[18]:
```
array([1., 1., 1.])
```

1.4 NumPy Standard Data Types

Data Type	Description
bool_	Boolean (True or False) stored as a byte
int_	Default integer type (same as C long; normally either int64 or int32)
intc	Identical to C int (normally int32 or int64)
intp	Integer used for indexing (same as C ssize_t; normally either int32 or int64)
int8	Integer (–127 to 128)
int16	Integer (–32768 to 32767)
int32	Integer (–2147483648 to 2147483647)
int64	Integer (–9223372036854775808 to 9223372036854775807)
uint8	Unsigned integer (0 to 255)
uint16	Unsigned integer (0 to 65535)
uint32	Unsigned integer (0 to 4294967295)
uint64	Unsigned integer (0 to 18446744073709551615)
float_	Shorthand for float64

Data Type	Description
float16	Half-precision foat: sign bit, 5 bits exponent, 10 bits mantissa
float32	Single-precision foat: sign bit, 8 bits exponent, 23 bits mantissa
float64	Double-precision foat: sign bit, 11 bits exponent, 52 bits mantissa
complex_	Shorthand for complex128
complex64	Complex number, represented by two 32-bit foats
complex128	Complex number, represented by two 64-bit foats

[19]:
```
np.zeros(10,dtype="int16") # length 10 int16 array with zeroes
```

[19]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int16)
```

[20]:
```
np.zeros(10,dtype=np.int16) # np.int16 is associated NumPy object
```

[20]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int16)
```