

Python 快速上手： 模組介紹 numpy

2022/09/28

Modules

import module

We don't have to start from the very beginning every time.

- Build-in module
- Used-defined module
- Using the specified version module
- Using the module from github (e.g. developer version)

The different way to import module (1/2)

Take the build-in os module as example, we are going to invoke the `os.path.abspath` function to get the absolute path.

```
>>> import os # import whole os module
```

```
>>> os.path.abspath('..')
```

```
>>> from os import * # import whole os module
```

```
>>> os.path.abspath('..')
```

The different way to import module (2/2)

Take the build-in os module as example, we are going to invoke the `os.path.abspath` function to get the absolute path.

```
>>> from os import path # import whole path submodule
```

```
>>> path.abspath('..')
```

```
>>> from os.path import abspath # import abspath function
```

```
>>> abspath('..')
```

Numpy

Numpy 資料型態

- Numpy 是很常被拿來處理多維資料的 Python library，透過基本資料型態 array，可以用來存放多維資料。
- e.g. 希望將 list 中的值全部加 2

```
>>> A = [1, 2, 3]
```

```
>>> new_A = [ x + 2 for x in A]
```

```
>>> print(new_A)
```

```
[3, 4, 5]
```

```
>>> import numpy as np
```

```
>>> A = [1, 2, 3]
```

```
>>> array_A = np.array(A)
```

```
>>> array_A = array_A + 2
```

```
>>> print(array_A)
```

```
array([3, 4, 5])
```

矩陣基本屬性 – 變數資料型態

- 我們可以透過 `type` 來檢查資料型態是否屬於 `numpy array` 。

```
type(a) # <class 'numpy.ndarray'>
```

- 有時候在 `function` 裡面的時候需要先檢查操作對象是否為 `ndarray` ，
可以透過 `isinstance` 這個 `build-in function` 。

```
isinstance(a, np.ndarray) # True
```

```
isinstance([[1, 2, 3], [4, 5, 6]], np.ndarray) # False
```


矩陣基本屬性 – 矩陣值資料型態

- `numpy.ndarray` 建立時可指定型態

```
>>> a = np.arange(10, dtype='int64')
```

- 矩陣裡的每一個值我們稱為 [scalars](#)，我們可以透過 `dtype` 檢查，通常在處理圖片或是神經網路的權重時會有影響。

```
>>> a.dtype # dtype('int64')
```

矩陣基本屬性 – 矩陣維度資訊

- 做矩陣運算之前，因為維度的特性，我們必須要先確認矩陣的一些屬性。
- e.g. 矩陣乘法要第一個矩陣的 column 個數 = 第二個矩陣的 row 個數才成立

$$(3, 2) \cdot (2, 1) = (3, 1)$$

矩陣基本屬性 – 矩陣維度資訊

- 透過 shape 我們可以得知每一維度的個數，進一步推算 row 與 column 個數

```
a.shape # (2, 3) => 2 rows, 3 columns
```

- 如果單純要知道矩陣的維度可以透過 ndim

```
a.ndim # 2 => 2 dimensions
```

矩陣類型

在矩陣操作的過程中有時候會用到特殊的矩陣：

- 空矩陣，可透過 `np.zeros` 或是 `np.ones` 宣告一個 0 或 1 的矩陣
- 對角矩陣，可透過 `np.eye` 來宣告一個對角線為 1 其餘為 0 的矩陣

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

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

```
>>> np.eye(3)
```

axis 的概念

axis 指運算的方向。在二維矩陣裡有兩個方向：

沿列 (axis=0)，或是**沿行(axis=1)**的方向。

axis=0 ↓ [[0, 1, 2, 3, 4],
 [5, 6, 7, 8, 9]]

記得，類似矩陣的東西都是「先列後行」

axis 的概念

axis 指運算的方向。在二維矩陣裡有兩個方向：

沿列 (**axis=0**)，或是沿行(**axis=1**)的方向。

[[0, 1, 2, 3, 4],
[5, 6, 7, 8, 9]]



axis=1

記得，類似矩陣的東西都是「先列後行」

矩陣索引 - slicing

- ndarray 跟 list 取出片段的方式一樣，可以透過 index 檢視部分矩陣資訊

```
a = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
```

array([[1, 2, 3, 4],
[5, 6, 7, 8],
[9, 10, 11, 12]])

a[:2, :]

a[:, 1:3]

矩陣索引 - integer indexing

- 前面提到的 slicing 通常用於觀察整個 row 或是 column 的連續片段，但是有時候我們只是想看某些 scalars，可以透過每個維度的 index 來觀察。

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]])
```

```
array([[1, 2],  
       [3, 4],  
       [5, 6]])
```

```
>>> a[[0, 1, 2], [0, 1, 0]]
```

```
array([1, 4, 5])
```


矩陣索引 – Boolean indexing

- integer indexing 必須給定確切的 index，但是索引時最常見的情境是給定條件下，找出符合的 scalars。
- ndarray 根據給定的條件會回傳一個 Boolean matrix，我們就可以根據 boolean matrix 當作 mask 取出符合條件的 scalars。


```
>>> a = np.array([[1, 2], [3, 4], [5, 6]])
```

```
>>> b = a>2
```



```
>>> print(b)
```

```
>>> print(a[b])
```


矩陣索引

 NumPy


User Guide API reference Development Learn

Search the docs ...

What is NumPy?
Installation 
NumPy quickstart
NumPy: the absolute basics for beginners
NumPy fundamentals ^
 Array creation
 Indexing on ndarrays
 I/O with NumPy
 Data types
 Broadcasting
 Byte-swapping
 Structured arrays
 Writing custom array containers
 Subclassing ndarray
 Universal functions (`ufunc`) basics
 Copies and views
Miscellaneous
NumPy for MATLAB users
Building from source

Indexing on `ndarrays`

 See also
Indexing routines

`ndarrays` can be indexed using the standard Python `x[obj]` syntax, where `x` is the array and `obj` the selection. There are different kinds of indexing available depending on `obj`: basic indexing, advanced indexing and field access.

Most of the following examples show the use of indexing when referencing data in an array. The examples work just as well when assigning to an array. See [Assigning values to indexed arrays](#) for specific examples and explanations on how assignments work.

Note that in Python, `x[(exp1, exp2, ..., expN)]` is equivalent to `x[exp1, exp2, ..., expN]`; the latter is just syntactic sugar for the former.

Basic indexing

Single element indexing

Single element indexing works exactly like that for other standard Python sequences. It is 0-based, and accepts negative indices for indexing from the end of the array.

```
>>> x = np.arange(10)
>>> x[2]
```

On this page

Basic indexing

- Single element indexing
- Slicing and striding
- Dimensional indexing tools

Advanced indexingField accessFlat Iterator indexingAssigning values to indexed arraysDealing with variable numbers of indices within programsDetailed notes

矩陣變形 – np.expand_dims

- 為了符合某些 function 的 input 需求，或是在處理圖片的過程中，我們有時候需要在不改變原始值的情況下改變維度。
- e.g. 灰階圖像 (grayscale) 是二維資料要轉換成與 RGB 圖像同維度的格式

`shape(200, 200) -> shape(200, 200, 1)`

```
>>> a = np.array([[1, 2, 3, 4], [5, 6, 7, 8]]) # shape = (2, 4)
```

```
>>> b = np.expand_dims(a, 2) # shape = (2, 4, 1)
```

Numpy operation

矩陣四則運算 – Arithmetic operation

- 矩陣的四則運算透過 Numpy 內部機制，實做 element-wise。

```
a = np.array([[1, 2], [3, 4]], dtype=np.float64)
```

```
b = np.array([[5, 6], [7, 8]], dtype=np.float64)
```

```
print(a+b)
```

```
print(a-b)
```

```
print(a*b)
```

```
print(a/b)
```

基本線性代數

- 轉置矩陣： $m \times n$ 矩陣在向量空間上轉置為 $n \times m$
- 逆矩陣： $n \times n$ 矩陣 A 存在一個 $n \times n$ 矩陣 B ，使得 $AB = BA = I$

```
>>> a = np.array([[0, 1], [2, 3]])
```

```
>>> print(a.T)
```

```
>>> inverse = np.linalg.inv(a)
```

```
>>> print(inverse)
```

```
>>> print(np.dot(a, inverse))
```

矩陣堆疊 - stack

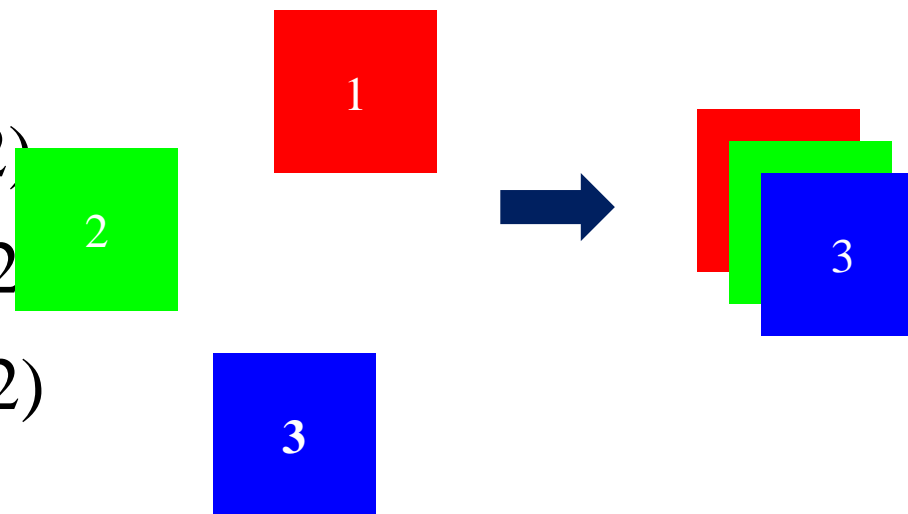
- 在實做深度學習網路或是圖片的時候，我們有時候需要對多個 array 做合併 `np.stack` 可以根據為度將多個 array 合併。

```
r = np.array([[0, 1], [2, 3]]) # shape = (2, 2)
```

```
g = np.array([[4, 5], [6, 7]]) # shape = (2, 2)
```

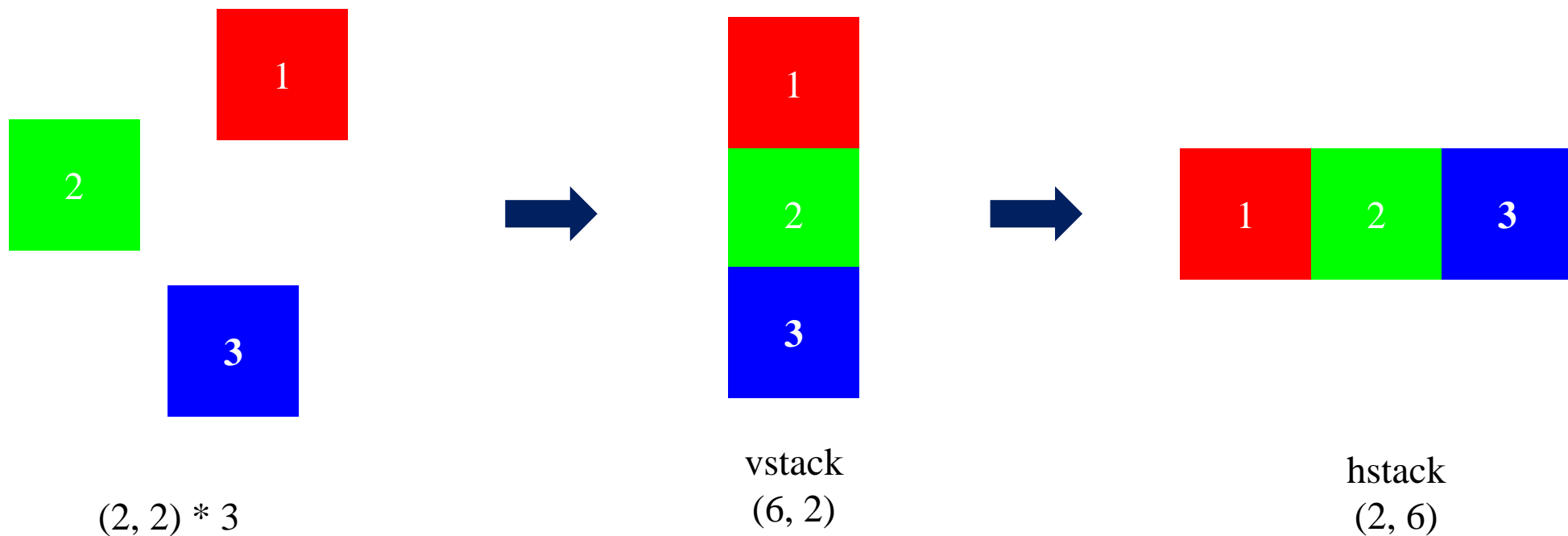
```
b = np.array([[8, 9], [1, 3]]) # shape = (2, 2)
```

```
np.stack([r, g, b], axis = 0)
```



矩陣堆疊 – vstack & hstack

- 有時候我們只是希望把矩陣水平或垂直的合併
 - `np.vstack((x, y))` : vertical , 垂直合併
 - `np.hstack((x, y))` : horizontal , 水平合併



More Numpy

Numpy Tutorials

Numpy 100

The image shows two screenshots. The top screenshot is of the NumPy website's 'NumPy Tutorials' page. It features the NumPy logo, a search bar, and a sidebar with a list of tutorial categories: 'NumPy Features' (expanded), 'Linear algebra on n-dimensional arrays', 'Saving and sharing your NumPy arrays', 'Masked Arrays', 'NumPy Applications', and 'Contributing'. The main content area is titled 'NumPy Features' and describes a collection of notebooks. It lists three topics: 'Linear algebra on n-dimensional arrays', 'Saving and sharing your NumPy arrays', and 'Masked Arrays'. Navigation links for 'Previous NumPy tutorials' and 'Next Linear algebra on n-dimensional arrays' are present. The bottom screenshot is of a Jupyter Notebook interface. The left sidebar shows a file explorer with a list of files, including '100_Numpy_exercises.ipynb' which is selected. The main area displays the '100 numpy exercises' notebook, which includes an introduction, instructions on how to use the exercises, and a list of three tasks: 1. Import the numpy package under the name np, 2. Print the numpy version and the configuration, and 3. Create a null vector of size 10.

NumPy

NumPy Tutorials

Search the docs ...

NumPy Features

- Linear algebra on n-dimensional arrays
- Saving and sharing your NumPy arrays
- Masked Arrays

NumPy Applications

Contributing

Theme by the Executable Book Project

NumPy Features

A collection of notebooks pertaining to built-in NumPy functionality.

- Linear algebra on n-dimensional arrays
- Saving and sharing your NumPy arrays
- Masked Arrays

Previous NumPy tutorials

Next Linear algebra on n-dimensional arrays

By the NumPy community

© Copyright 2020, the NumPy community.

File Edit View Run Kernel Tabs Settings Help

Filter files by name

Name	Last Modified
source	a month ago
100_Numpy_exercises_with_hints_w...	a month ago
100_Numpy_exercises_with_hints.md	a month ago
100_Numpy_exercises_with_solutio...	a month ago
100_Numpy_exercises.ipynb	2 minutes ago
100_Numpy_exercises.md	a month ago
100_Numpy_random.ipynb	a month ago
generators.py	a month ago
initialise.py	a month ago
LICENSE.txt	a month ago
README.md	a month ago
requirements.txt	a month ago

Launcher

100_Numpy_exercises.ipynb

Python 3 (ipykernel)

100 numpy exercises

This is a collection of exercises that have been collected in the numpy mailing list, on stack overflow and in the numpy documentation. The goal of this collection is to offer a quick reference for both old and new users but also to provide a set of exercises for those who teach.

If you find an error or think you've a better way to solve some of them, feel free to open an issue at <https://github.com/rougier/numpy-100>.

File automatically generated. See the documentation to update questions/answers/hints programmatically.

Run the `initialize.py` module, then for each question you can query the answer or an hint with `hint(n)` or `answer(n)` for `n` question number.

```
[ ]: %run initialise.py
```

1. Import the numpy package under the name `np` (☆☆☆)
2. Print the numpy version and the configuration (☆☆☆)
3. Create a null vector of size 10 (☆☆☆)