

## 16

## Manipulating NumPy Arrays

## NumPy 数组规整

## 重塑数组的维数、形状



哪里有物质，哪里就有几何学。

*Where there is matter, there is geometry.*

—— 约翰内斯·开普勒 (Johannes Kepler) | 德国天文学家、数学家 | 1571 ~ 1630



- ▶ `numpy.append()` 用于将值添加到数组的末尾，生成一个新的数组，并不会修改原始数组
- ▶ `numpy.arange()` 创建一个具有指定范围、间隔和数据类型的等间隔数组
- ▶ `numpy.block()` 用于按照指定的块结构组合多个数组，生成一个新的数组
- ▶ `numpy.column_stack()` 按列堆叠多个数组，生成一个新的二维数组
- ▶ `numpy.concatenate()` 沿指定轴连接多个数组，生成一个新的数组
- ▶ `numpy.delete()` 用于删除数组中指定位置的元素，生成一个新的数组，并不会修改原始数组
- ▶ `numpy.flatten()` 用于将多维数组转换为一维数组。与 `numpy.ravel()` 不同的是，`numpy.flatten()` 返回数组的副本，而不是原始数组的视图
- ▶ `numpy.flip()` 用于沿指定轴翻转数组的元素顺序
- ▶ `numpy.fliplr()` 沿着水平方向（左右方向）翻转数组的元素顺序
- ▶ `numpy.flipud()` 沿着垂直方向（上下方向）翻转数组的元素顺序
- ▶ `numpy.hsplit()` 用于沿水平方向分割数组为多个子数组
- ▶ `numpy.hstack()` 按水平方向堆叠多个数组，生成一个新的数组
- ▶ `numpy.insert()` 用于在数组的指定位置插入值，生成一个新的数组，并不会修改原始数组
- ▶ `numpy.ravel()` 用于将多维数组转换为一维数组，按照 C 风格的顺序展平数组元素
- ▶ `numpy.repeat()` 将数组中的元素重复指定次数，生成一个新的数组
- ▶ `numpy.reshape()` 用于改变数组的形状，重新排列数组元素，但不改变原始数据本身
- ▶ `numpy.resize()` 用于调整数组的形状，并可以在必要时重复数组的元素来填充新的形状
- ▶ `numpy.rot90()` 默认将数组按指定次数逆时针旋转 90 度
- ▶ `numpy.row_stack()` 按行堆叠多个数组，生成一个新的数组
- ▶ `numpy.shares_memory()` 用于检查两个数组是否共享相同的内存位置
- ▶ `numpy.split()` 用于将数组沿指定轴进行分割成多个子数组
- ▶ `numpy.squeeze()` 用于从数组的形状中去除维度为 1 的维度，使得数组更紧凑
- ▶ `numpy.stack()` 用于沿新的轴将多个数组堆叠在一起，生成一个新的数组
- ▶ `numpy.swapaxes()` 用于交换数组的两个指定轴的位置
- ▶ `numpy.tile()` 用于将数组沿指定方向重复指定次数，生成一个新的数组
- ▶ `numpy.transpose()` 完成矩阵转置，即将数组的行和列进行互换
- ▶ `numpy.vsplit()` 用于沿垂直方向分割数组为多个子数组
- ▶ `numpy.vstack()` 按垂直方向堆叠多个数组，生成一个新的数组

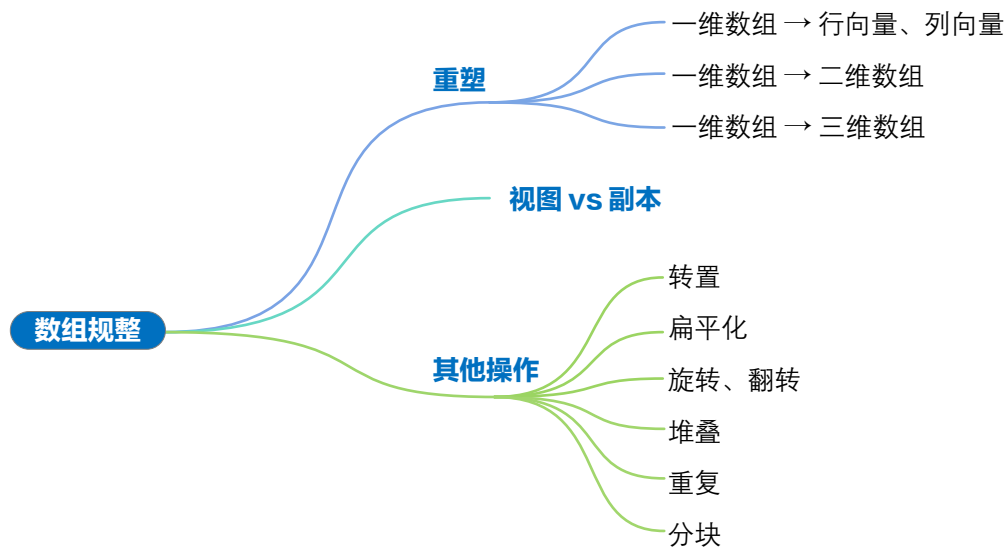
本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)



本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

## 16.1 从 reshape() 函数说起

在 NumPy 中，要改变数组的形状（也称重塑数组），可以使用 `numpy.reshape()` 函数。  
`reshape()` 函数允许你指定一个新的形状，然后返回一个拥有相同数据但具有新形状的数组。

下面我们先了解一下这个话题的核心函数——`numpy.reshape()`。



`numpy.reshape(a, newshape, order='C')`

这个函数的重要输入参数：

- `a` 参数是要被重塑的数组，可以是一个数组对象，也可以是一个 Python 列表、元组等支持迭代的对象。
- `newshape` 参数是新的形状，可以是一个整数元组或列表，也可以是一个整数序列。
- `order` 参数表示重塑数组的元素在内存中存储的顺序，可以是 'C'（按行顺序存储）或 'F'（按列顺序存储），默认值为 'C'。

下面是 `numpy.reshape()` 函数一些常见用法：

a) 改变数组的维度：可以将一个数组从一维改为二维、三维等。例如：

```
import numpy as np
a = np.arange(12)           # 创建一个长度为 12 的一维数组
b = np.reshape(a, (3, 4))   # 改变为 3 行 4 列的二维数组
c = np.reshape(a, (2, 3, 2)) # 改变为 2 个 3 行 2 列的三维数组
```

b) 展开数组：可以将一个多维数组展开为一维数组。例如：

```
import numpy as np
a = np.array([[1, 2], [3, 4]])
b = np.reshape(a, -1) # 将二维数组展开为一维数组
```

c) 改变数组的顺序：可以改变数组在内存中的存储顺序。例如：

```
import numpy as np
a = np.arange(6).reshape((2, 3)) # 创建一个 2 行 3 列的二维数组
b = np.reshape(a, (3, 2), order='F') # 按列顺序存储
```

注意：`numpy.reshape()` 函数并不会改变数组的数据类型和数据本身，只会改变其形状。如果改变后的形状与原数组的元素数量不一致，将会抛出 `ValueError` 异常。

请大家在 JupyterLab 中自行运行如上三段代码。

更多有关 `numpy.reshape()` 函数的用法，请大家参考如下技术文档：

<https://numpy.org/doc/stable/reference/generated/numpy.reshape.html>

下面结合实例详细讲解如何利用 `numpy.reshape()` 完成数组变形。



本节配套的 Jupyter Notebook 文件是 `Bk1_Ch16_01.ipynb` 和 `Bk1_Ch16_02.ipynb`，请大家一边阅读本章一边实践。

## 16.2 一维数组 → 行向量、列向量

一维数组 → 行向量

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

本书前文提过，行向量、列向量都是特殊矩阵。因此，行向量、列向量都是二维数组。也就是说，行向量是一行若干列的数组，形状为  $1 \times D$ 。列向量是若干行一列的数组，形状为  $n \times 1$ 。

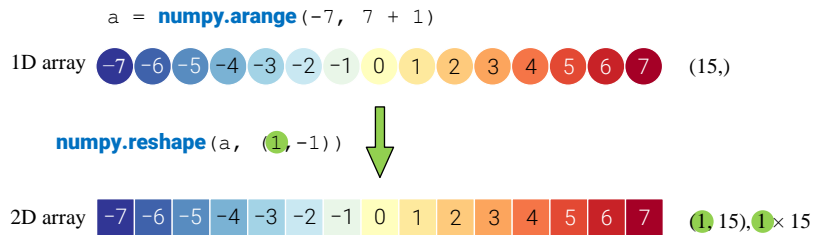


图 1. 将一维数组转换为行向量

如图 1 所示，用 `a = numpy.arange(-7, 7+1)` 生成的是一个一维数组 `a`，这个数组有 15 个元素。由于数组为一维，所以可视化时采用了“圆圈”，而不是方块。利用 `numpy.reshape(a, (1, -1))`，我们将 `a` 转化为形状为 `(1, 15)` 的二维数组，也称行向量，即  $1 \times 15$  矩阵。

⚠ 注意，使用 `-1` 作为形状参数时，`numpy.reshape()` 会根据数组中的数据数量和其它指定的维数来自动计算该维度的大小。

### 一维数组 → 列向量

如图 2 所示，利用 `numpy.reshape(a, (-1, 1))`，我们可以把一维数组 `numpy.arange(-7, 7+1)` 转化为形状为 `(15, 1)` 的二维数组，也称列向量，即  $15 \times 1$  矩阵。

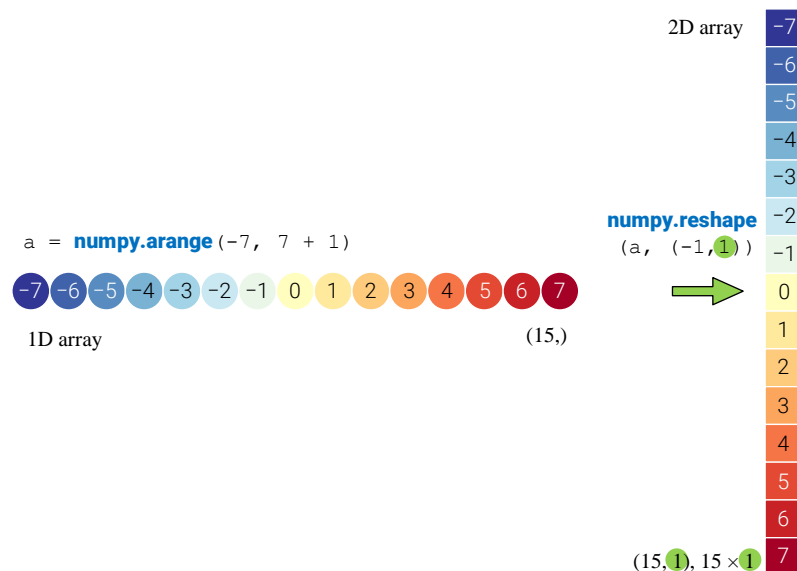


图 2. 将一维数组转换为列向量

## 16.3 一维数组 → 二维数组

用 `a = numpy.arange(-7, 7+1)` 生成的数组有 15 个元素，可以被 3、5 整除，因此一维数组 `a` 可以写成  $3 \times 5$  矩阵。如图 3 所示，我们可以分别按先行后列、先列后行两种形式重塑数组。

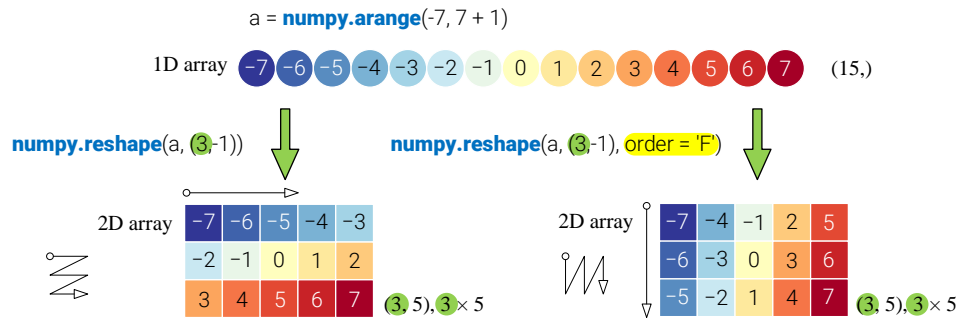


图 3. 将一维数组转换为  $3 \times 5$  矩阵，先行后列，先列后行

图 4 所示为将 `numpy.arange(-7, 7+1)` 一维数组写成  $5 \times 3$  矩阵。图 4 给出了先行后列、先列后行两种顺序。如图 5 所示已经完成转换的  $3 \times 5$  数组，通过 `numpy.reshape()` 可以进一步转化为  $5 \times 3$  数组。此外，请比较 `numpy.reshape()` 和 `numpy.resize()` 用法的异同。

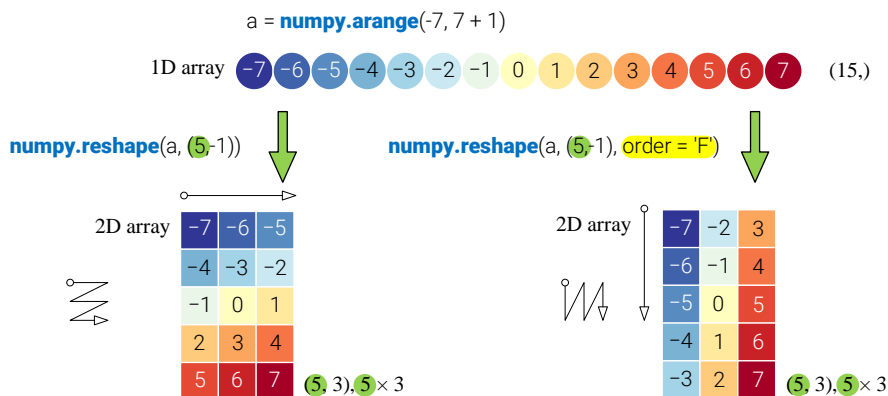


图 4. 将一维数组转换为  $5 \times 3$  矩阵，先行后列，先列后行

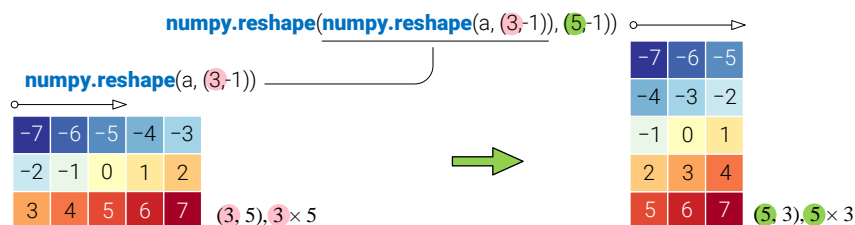
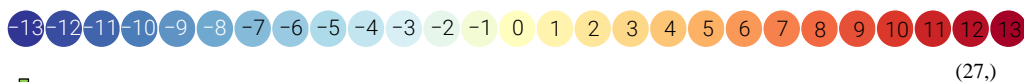


图 5. 将  $3 \times 5$  矩阵转换为  $5 \times 3$  矩阵，先行后列

## 16.4 一维数组 → 三维数组

图 6 所示为将 `numpy.arange(-13, 13+1)` 一维数组转化形状为  $3 \times 3 \times 3$  的三维数组。

```
a = numpy.arange(-13, 13 + 1)
```



↓ `numpy.reshape(a, (3, 3, 3))`

3D array  
(3, 3, 3),  $3 \times 3 \times 3$

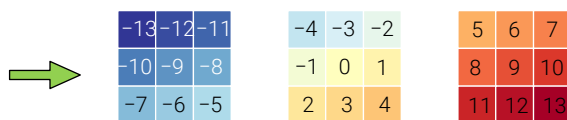


图 6. 将一维数组转换为三维数组

## 16.5 视图 vs 副本

本书前文特别提过，NumPy 中要特别注意视图（view）、副本（copy）的区别。简单来说，视图和副本是 NumPy 中的两种不同的数组对象。

视图是指一个数组的不同视角或者不同形状的表现方式，视图和原始数组共享数据存储区，因此在对视图进行操作时，会影响原始数组的数据。视图可以通过数组的切片、转置、重塑等操作创建。

副本则是指对一个数组的完全复制，副本和原始数组不共享数据存储区，因此对副本进行操作不会影响原始数组。使用 `numpy.reshape()` 也需要注意视图、副本问题。

本节配套的 Jupyter 笔记中，大家可以看到，我们用 `numpy.shares_memory()` 判断两个数组是否指向同一个内存。

如图 7 所示，`numpy.reshape()` 仅仅改变了观察同一数组的视角，也就是改变了 index。

⚠ 注意，不同函数的历史、未来版本可能存在不一致，需要大家自行判断。

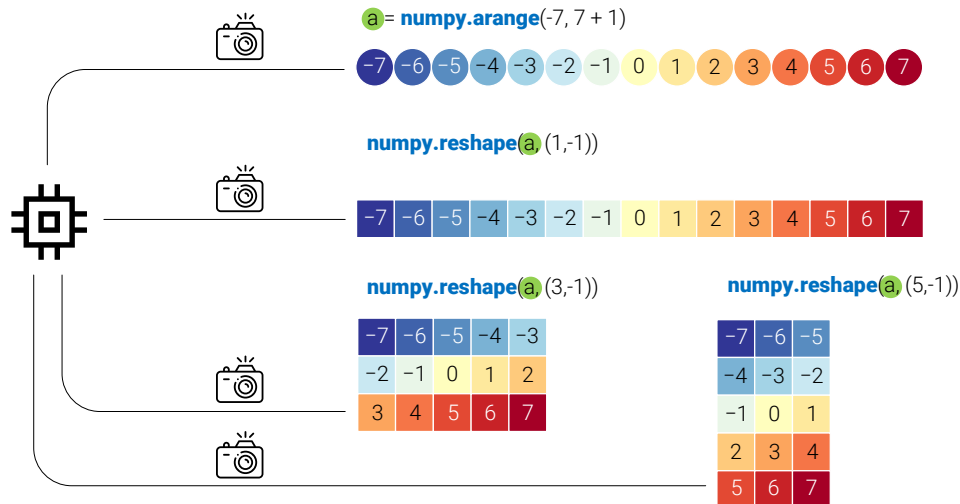


图 7. 视图，还是副本？

## 16.6 转置

如图 8 所示，一个  $n \times D$  矩阵  $A$  转置得到  $D \times n$  矩阵  $B$ ，整个过程相当于矩阵  $A$  绕主对角线镜像。具体来说，矩阵  $A$  位于  $(i, j)$  的元素转置后的位置为  $(j, i)$ ，即行列序号互换。这就是，为什么位于主对角线上的元素转置前后位置不变。矩阵  $A$  的转置 (the transpose of a matrix  $A$ ) 记作  $A^T$  或  $A'$ 。为了和求导记号区分，本书仅采用  $A^T$  记法。

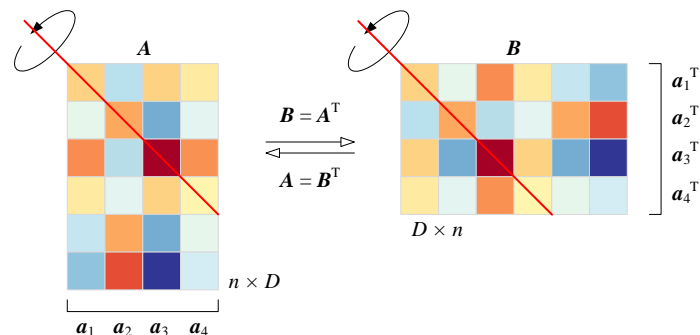


图 8. 矩阵转置，图片来自《矩阵力量》第 4 章

⚠ 需要大家特别注意的是，NumPy 的 `numpy.transpose()` 方法和 `.T` 属性都返回原始数组的转置，两者都返回原始数组的视图，而不是副本。

➡ “鸢尾花书”中《矩阵力量》第 4 章将专门讲解矩阵的转置运算。

图 9 所示为二维数组的转置。行向量转置得到列向量，反之亦然。 $3 \times 5$  矩阵转置得到  $5 \times 3$  矩阵。而一维数组的转置不改变形状。

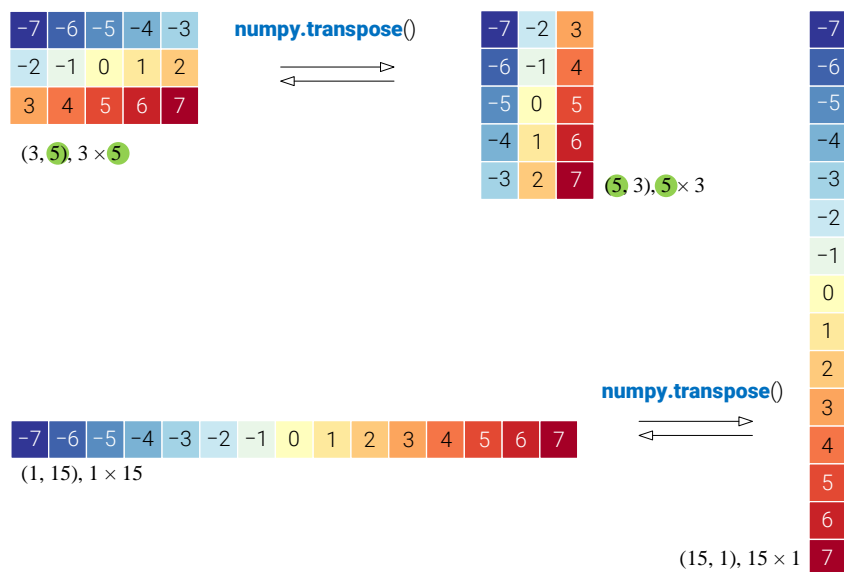


图 9. 二维数组的转置

## 16.7 扁平化

扁平化可以理解图 1、图 2、图 3 等 `numpy.reshape()` 的“逆操作”。完成扁平化的方法有很多，比如 `array.ravel()`、`array.reshape(-1)`、`array.flatten()`。大家也可以使用 `numpy.ravel()`、`numpy.flatten()` 这两个函数。图 10 所示为将二维转化为一维数组。

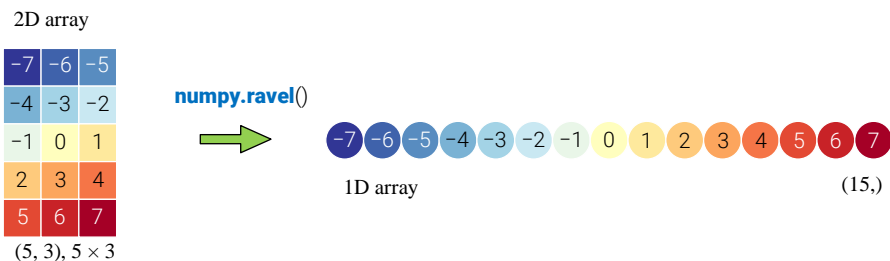


图 10. 二维数组转化为一维数组

请大家格外注意，`ravel()`、`reshape(-1)` 返回的是原始数组的视图，而不是其副本。因此，如果修改新数组中的任何元素，原始数组也会受到影响。如果需要返回一个数组副本，可以使用 `flatten()` 函数。本节配套的 Jupyter 笔记中给出一个详细的例子，请大家自行学习。



## 16.8 旋转、翻转

如图 11 所示，`numpy.rot90()` 的作用是将一个数组逆时针旋转 90 度。默认情况下，这个函数会将数组的前两个维度 `axes=(0, 1)` 进行旋转。此外，还可以利用参数 `k`（正整数）逆时针旋转  $k \times 90$  度。默认， $k = 1$ 。

注意，`numpy.rot90()` 的结果也是返回原始数组的视图，而不是副本。

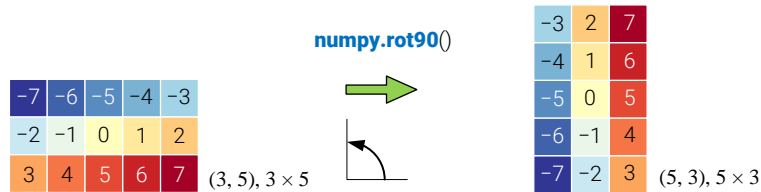


图 11. 3 × 5 矩阵逆时针旋转 90 度

`numpy.flip()` 函数用于翻转数组中的元素，即将数组沿着一个或多个轴翻转。`numpy.flip(A, axis=None)` 中，`A` 是要进行翻转的数组，`axis` 指定要翻转的轴。如图 12 所示，如果不指定 `axis`，则默认将整个数组沿着所有的轴进行翻转。类似的函数还有 `numpy.fliplr()`、`numpy.flipud()`，请大家自行学习。



图 12. 3 × 5 矩阵沿着两个轴翻转

## 16.9 堆叠

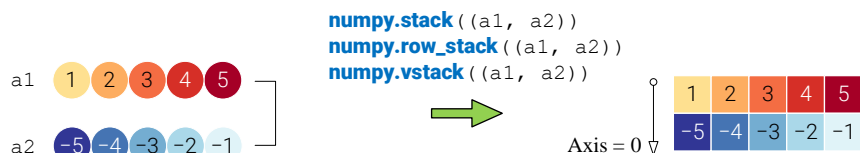
### 沿行堆叠

用 `numpy.arange()` 产生如图 1 所示的两个一维等长数组。图 14 所示为三种办法将两个等长一维数组沿行 `axis = 0` 方向堆叠，结果为二维数组。

`numpy.stack()` 函数将沿着指定轴将多个数组堆叠在一起，返回一个新的数组；默认轴为 `axis = 0`。`numpy.row_stack()` 函数将多个数组沿着行方向进行堆叠，生成一个新的数组。`numpy.vstack()` 将多个数组沿着垂直方向（行方向）进行堆叠，生成一个新的数组。



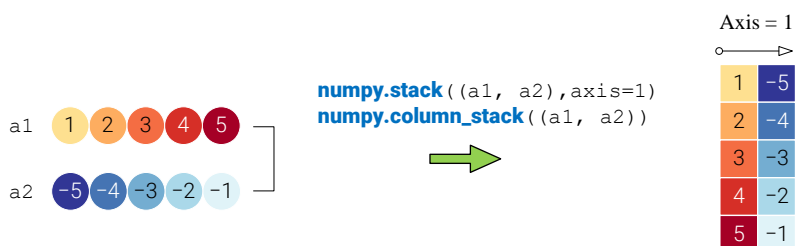
图 13. 两个等长一维数组

图 14. 沿行 `axis = 0` 方向堆叠

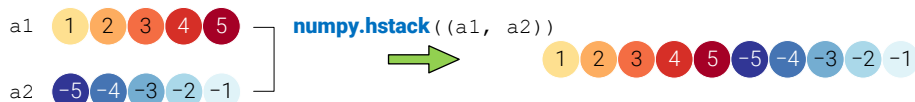
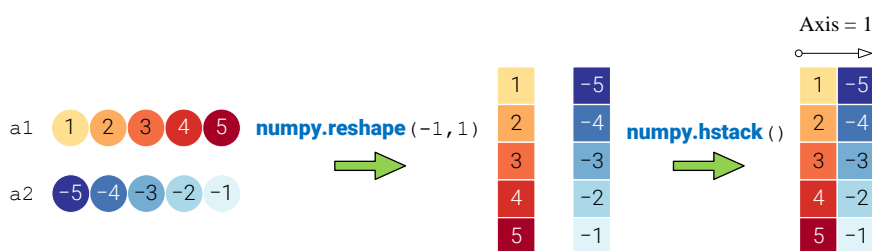
## 沿列堆叠

图 15 所示为沿列 `axis = 1` 方向堆叠两个一维等长数组。图中给出两种办法。

`numpy.column_stack()` 将多个一维数组沿着列方向进行堆叠，生成一个新的二维数组。

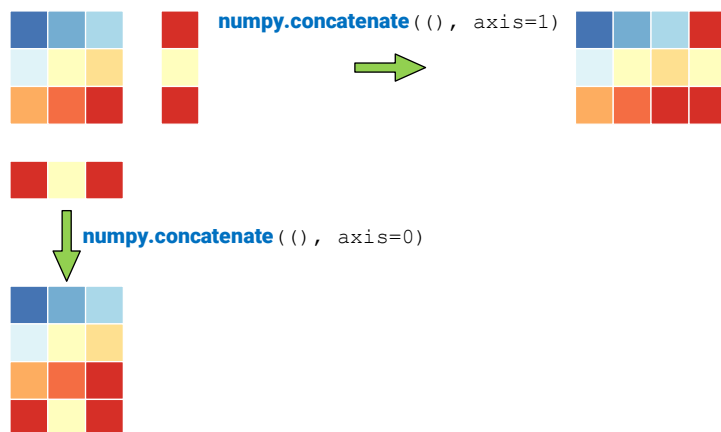
图 15. 沿列 `axis = 1` 方向堆叠

如图 16 所示，用 `numpy.hstack()` 堆叠一维数组的结果还是一个一维数组。`numpy.hstack()` 将多个数组沿着水平方向（列方向）进行堆叠，生成一个新的数组。为了获得图 15 结果，需要先将两个一维数组变形为列向量，然后用 `numpy.hstack()` 函数沿列堆叠，具体如图 17 所示。

图 16. 沿列 `axis = 1` 方向堆叠，用 `numpy.hstack()`图 17. 沿列 `axis = 1` 方向堆叠，两个列向量

## 拼接

我们还可以用 `numpy.concatenate()` 完成数组拼接。如所示，利用 `numpy.concatenate()`，我们可以分别完成沿行、列方向数组拼接。

图 18. 用 `numpy.concatenate()` 拼接

### 堆叠结果为三维数组

此外，利用 `numpy.stack()`，我们还可以将二维数组堆叠为三维数组。图 19 所示为沿三个不同方向堆叠结果的效果图。

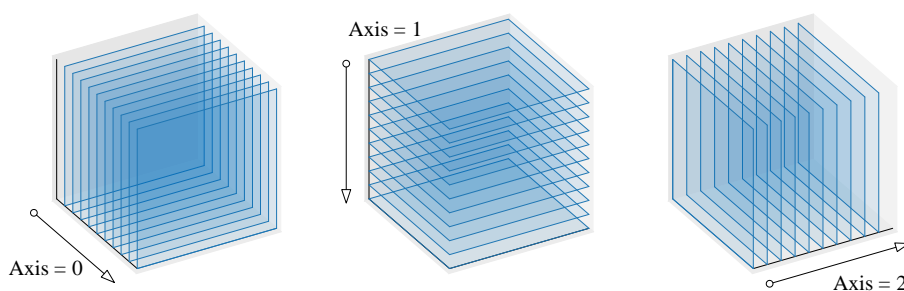


图 19. 沿着三个不同方向堆叠

举个例子，给定图 20 所示两个形状相同的二维数组。它俩按图 19 所示为沿三个不同方向堆叠的结果如图 21 所示。



图 20. 两个形状相同的二维数组

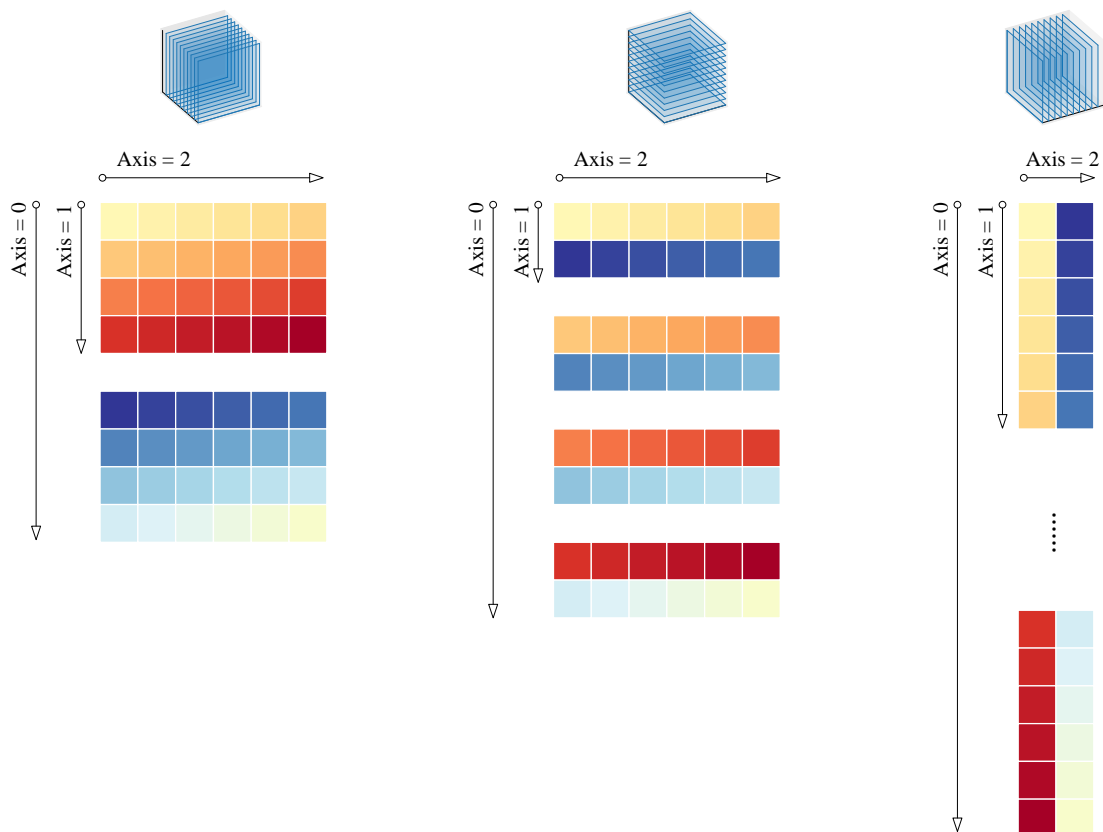
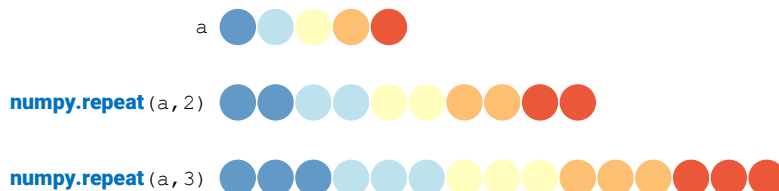


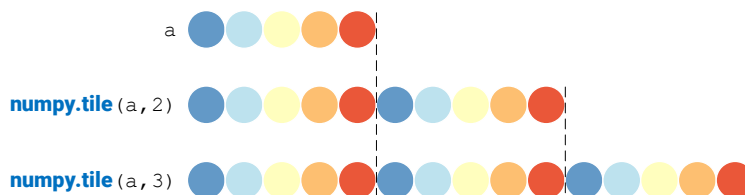
图 21. 得到三个不同的三维数组

## 16.10 重复

`numpy.repeat()` 和 `numpy.tile()` 都可以用来重复数据。`numpy.repeat()` 和 `numpy.tile()` 的区别在于重复的对象不同。`numpy.repeat()` 重复的是分别数组中的每个元素。`numpy.repeat()` 还可以指定具体的轴，以及不同元素重复的次数，请大家参考其技术文档。

`numpy.tile()` 重复的是整个数组，如图 23 所示。本章配套 Jupyter Notebook 还提供其他示例，请大家自行练习。

图 22. 利用 `numpy.repeat()` 重复一维数组

图 23. 利用 `numpy.tile()` 重复一维数组

## 16.11 分块矩阵

### 合成

`numpy.block()` 函数用于将多个数组沿不同的轴组合成一个分块矩阵。它接受一个嵌套列表作为输入，每个列表代表一个块矩阵，然后根据指定的轴将这些块矩阵组合在一起。

在图 24 给出的例子中，我们创建了四个小的矩阵，并使用 `numpy.block()` 函数将它们组合成一个分块矩阵  $M$ 。

分块矩阵经常用来简化某些线性代数运算，鸢尾花书《矩阵力量》将专门介绍分块矩阵。

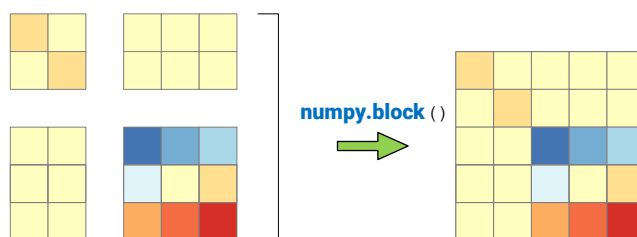


图 24. 四个二维数组组合成一个矩阵



#### 什么是分块矩阵？

分块矩阵是由多个小矩阵组合而成的大矩阵。它将一个大的矩阵划分为若干个小的矩阵，这些小矩阵可以是实数矩阵、向量矩阵或者其他的矩阵形式。通常情况下，分块矩阵可以使用一个方括号将小矩阵组合在一起，然后按照一定的规则排列。分块矩阵可以简化一些复杂的矩阵计算，同时也常常用于表示具有特定结构的矩阵，例如对角矩阵或者上下三角矩阵等。

### 切割

`numpy.split()` 函数可以将一个数组沿指定轴分割为多个子数组。`numpy.split()` 接受三个参数：要分割的数组、分割的索引位置、沿着哪个轴进行分割。图 25 所示为将一个一维数组三等分得到三个子数组。本章配套的 Jupyter Notebook 中，大家可以看到如何设定分割索引位置，请自行练习。

图 26 所示为利用 `numpy.split()` 将二维数组沿不同轴三等分。大家也可以分别尝试使用 `numpy.hsplit()` 和 `numpy.vsplit()` 完成类似操作。本章配套 Jupyter Notebook 中还介绍如何使用 `numpy.append()`、`numpy.insert()`、`numpy.delete()` 完成附加、插入、删除操作，请大家自行学习。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：[jiang.visualize.ml@gmail.com](mailto:jiang.visualize.ml@gmail.com)

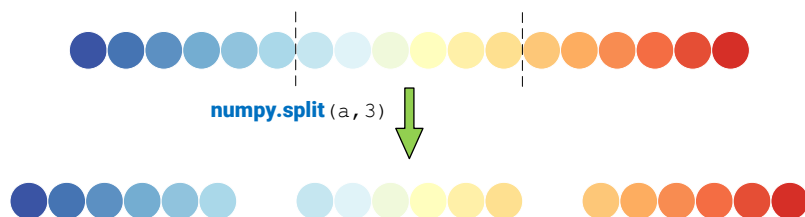


图 25. 将一维数组三等分

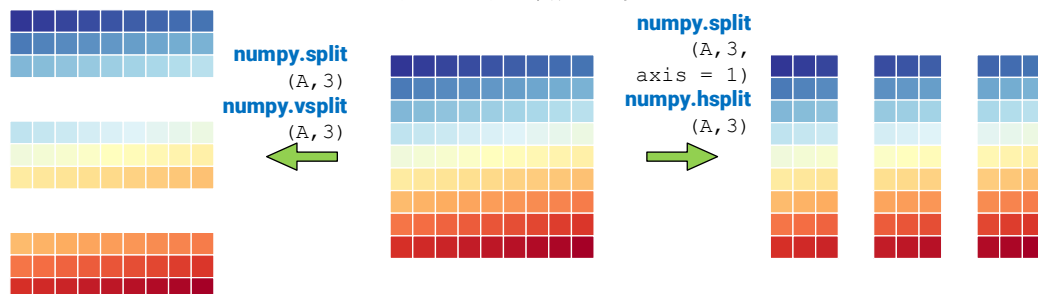


图 26. 将二维数组三等分，沿不同轴



请大家完成以下几道题目。

Q1. 首先生成一个一维数组 `[1, 2, 3, 4, 5, 6]`，然后将其转换为一个形状为 `(2, 3)` 的二维数组，并打印结果。注意，元素按先后顺序存储。最后，想办法判断转换前后的数组是视图，还是副本。

Q2. 将一个二维数组 `[[1, 2], [3, 4], [5, 6]]` 转换为一个形状为 `(6,)` 的一维数组，并打印结果。注意，按先列后行顺序存储。

Q3. 将一个三维数组 `[[[1, 2], [3, 4]], [[5, 6], [7, 8]]]` 转换为一个形状为 `(2, 4)` 的二维数组，并按列顺序存储，最后打印结果。

Q4. 请生成 `[0, 1]` 区间内的连续均匀两个随机数数组，数组形状为 `(10,)`。将它俩分别按行、按列堆叠起来形成二维数组。

Q5. 请生成 `[0, 1]` 区间内的连续均匀一个随机数数组，数组形状为 `(12, 12)`。将它分别按行、按列三等分。

Q6. 请生成 `[0, 1]` 区间内的连续均匀两个随机数数组，数组形状分别为 `(8, 5)`、`(3, 5)`。用几种不同办法将它们拼接成一个数组。

\* 题目很基础，本书不给答案。



本章介绍了很多有关数组的操作。为了帮助大家理解，我们用图 27 来总结其中主要操作。

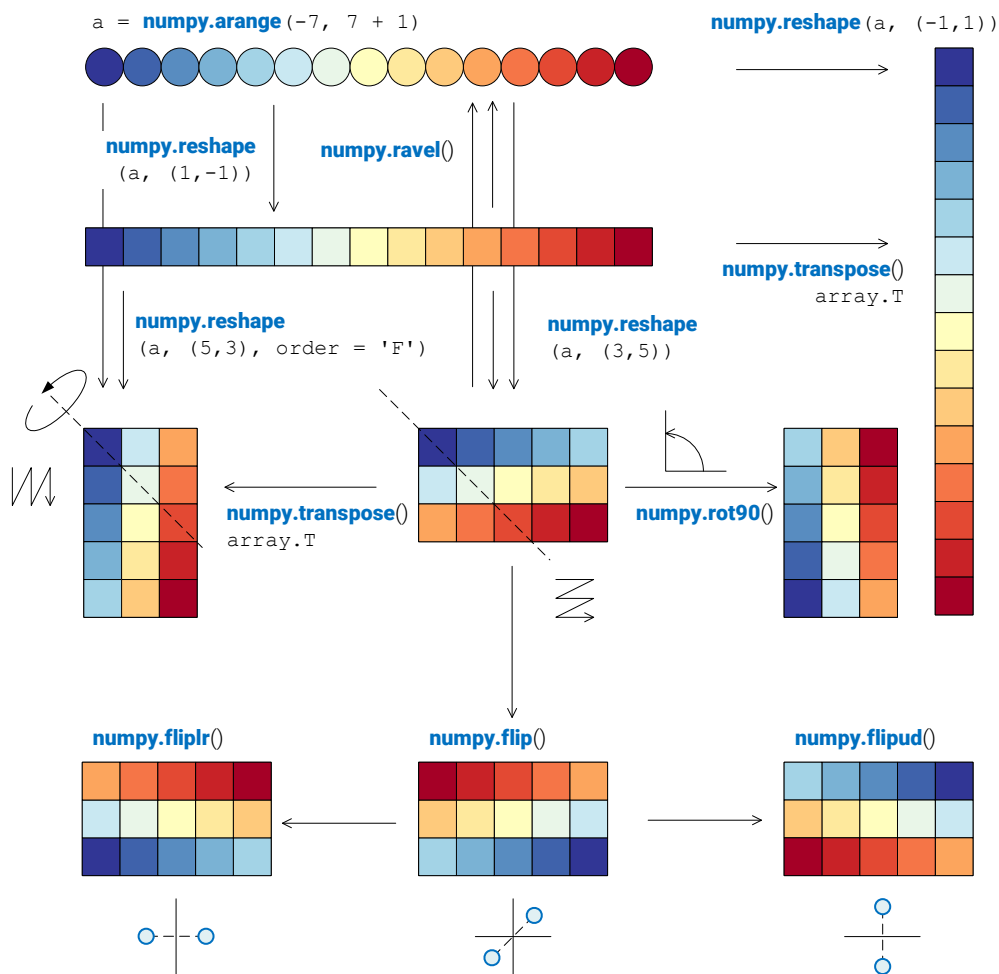


图 27. 总结本章主要操作