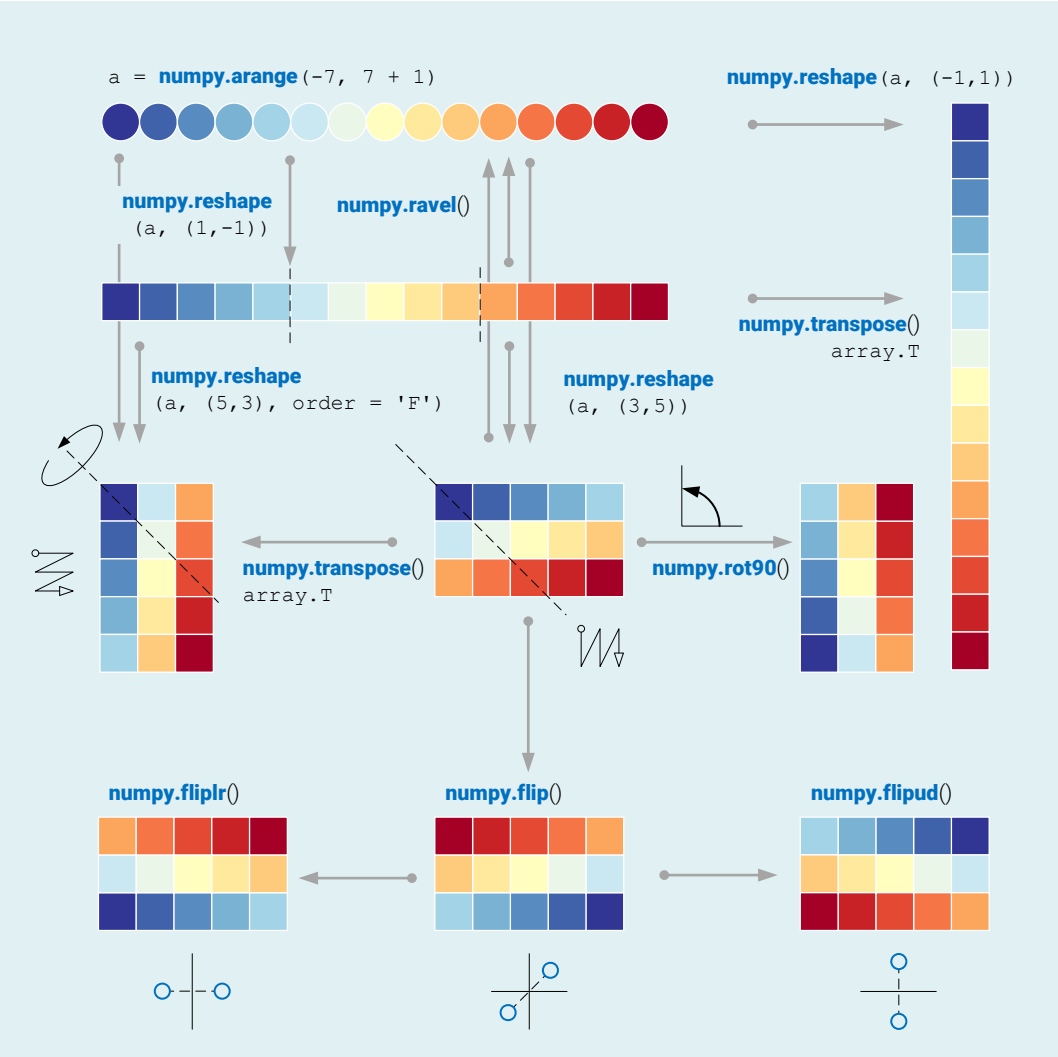
4.4 Reshaping Arrays 数组变形

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



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 BK_2_Topic_4.04_1.ipynb。

一维数组 → 行向量

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 $n \times 1$。

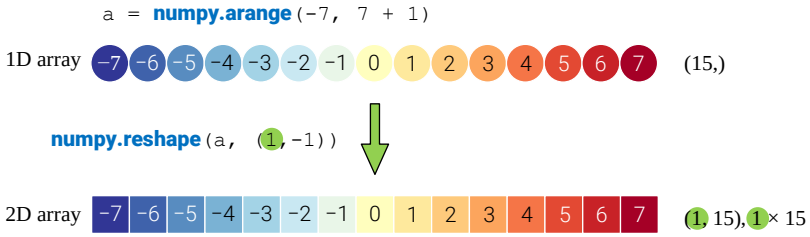


图 1. 将一维数组转换为行向量

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 × 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 × 1 矩阵。

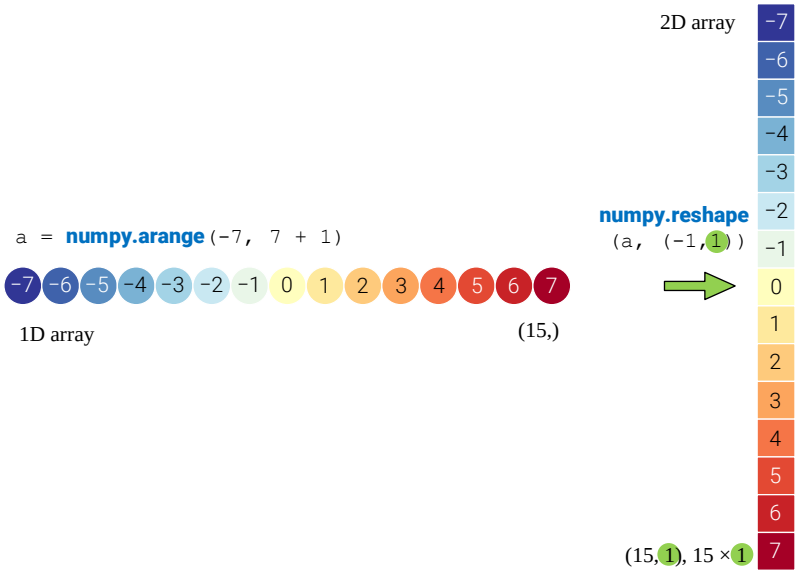


图 2. 将一维数组转换为列向量

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 × 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。

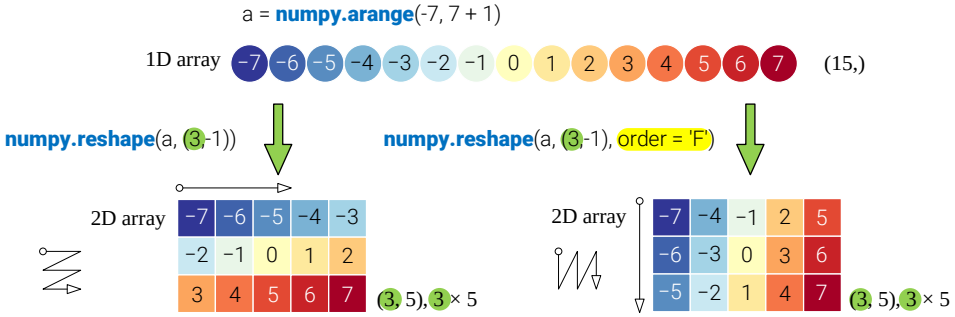


图 3. 将一维数组转换为 3 × 5 矩阵，先行后列，先列后行

图 4 所示为将 **numpy.arange**(-7, 7+1) 一维数组写成 5 × 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 × 5 数组，通过 **numpy.reshape**() 可以进一步转化为 5 × 3 数组。

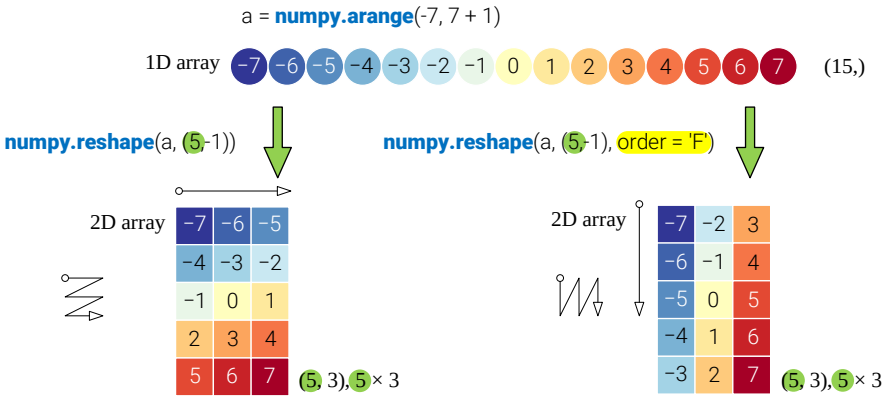


图 4. 将一维数组转换为 5 × 3 矩阵，先行后列，先列后行

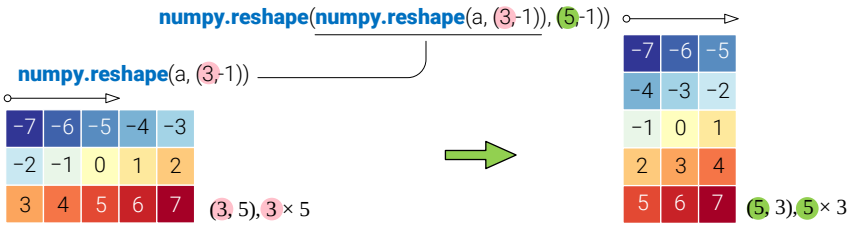


图 5. 将 3 × 5 矩阵转换为 5 × 3 矩阵，先行后列

一维数组 → 三维数组

图 6 所示为将 **numpy.arange**(-13, 13+1) 一维数组转化成形状为 3 × 3 × 3 的三维数组。

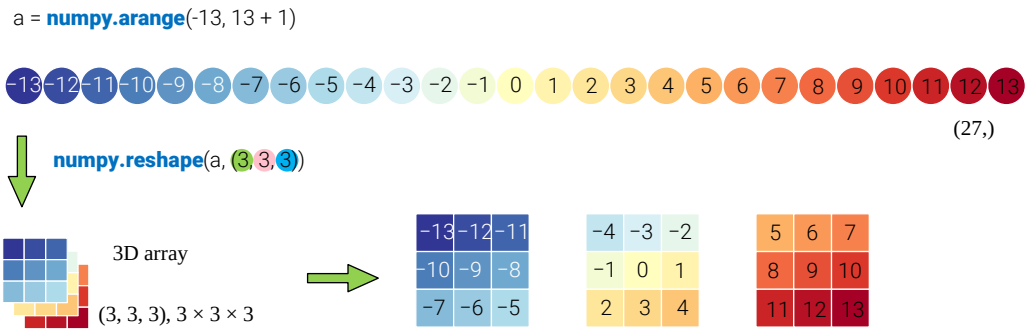


图 6. 将一维数组转换为三维数组

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。

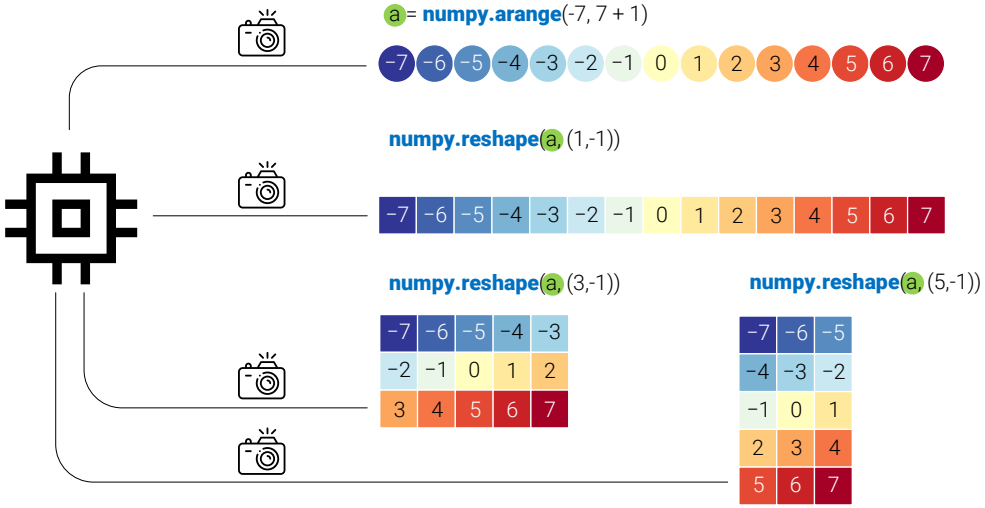


图 7. 视图，还是副本？

转置

如图 8 所示，一个 $n \times D$ 矩阵 $\boldsymbol{A}$ 转置得到 $D \times n$ 矩阵 $\boldsymbol{B}$，整个过程相当于矩阵 $\boldsymbol{A}$ 绕主对角线镜像。具体来说，矩阵 $\boldsymbol{A}$ 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 $\boldsymbol{A}$ 的转置 (the transpose of a matrix $\boldsymbol{A}$) 记作 $\boldsymbol{A}^{\mathrm{T}}$ 或 $\boldsymbol{A}'$。为了和求导记号区分，本书仅采用 $\boldsymbol{A}^{\mathrm{T}}$ 记法。

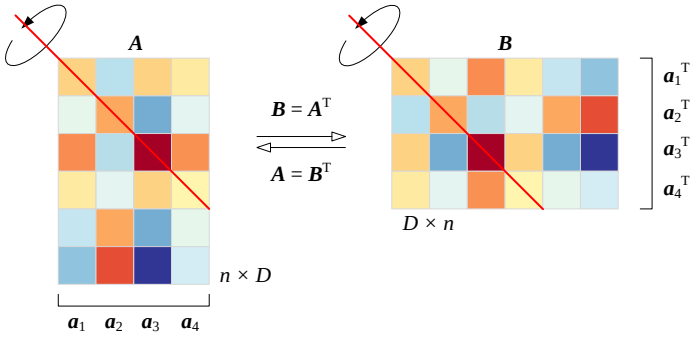


图 8. 矩阵转置，图片来自《矩阵力量》第 4 章

⚠ 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 × 5 矩阵转置得到 5 × 3 矩阵。而一维数组的转置不改变形状。

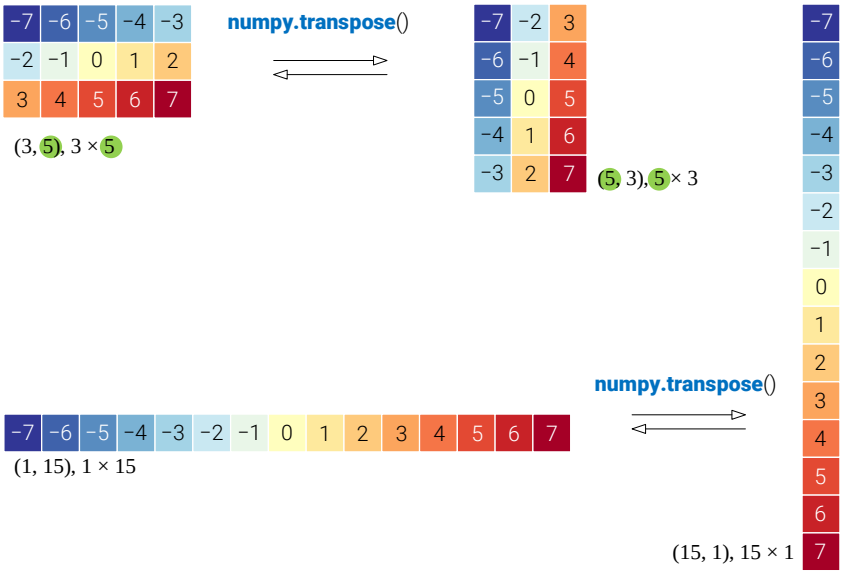


图 9. 二维数组的转置

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。

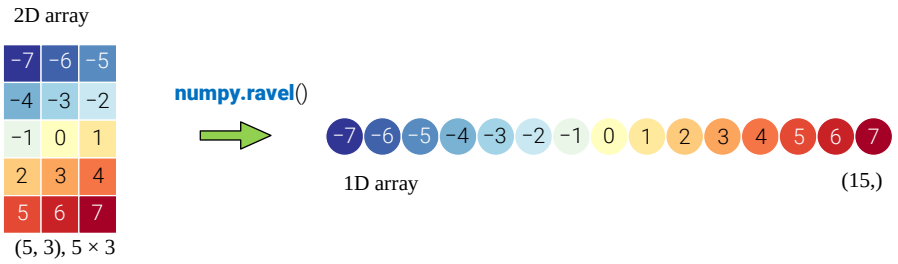


图 10. 二维数组转化为一维数组

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。

其他操作

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。

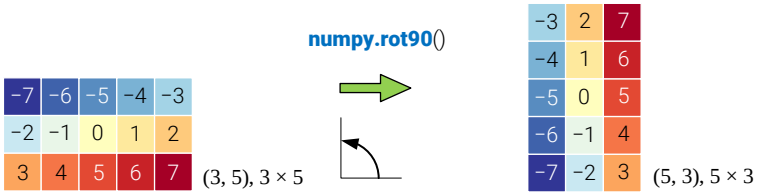


图 11. 3 × 5 矩阵逆时针旋转 90 度

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。

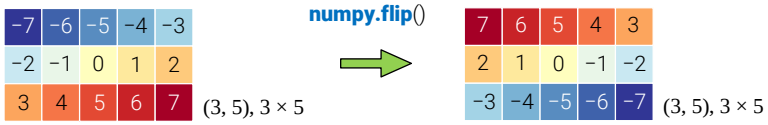


图 12. 3 × 5 矩阵沿着两个轴翻转

下面，是有关使用 **numpy.reshape**() 函数的三道习题，请大家完成。

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，元素按先行后列顺序存储。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。

* 这三道题目很基础，本书不给答案。