2.1 NumPy简介

NumPy是Numerical Python的缩写，它是Python中一个用于科学计算的扩展程序库。NumPy因其包含丰富的数学函数、支持高维数组与矩阵且运行速度快等特点而得到了广泛应用。具体而言，NumPy的主要功能有：

ndarray，一种支持矢量计算和广播能力的多维数组，运行速度快且节省空间；

能快速处理整组数据的数学函数；

整合C/C++/Fortran代码的工具；

线性代数、傅立叶变换、随机数生成等功能；

磁盘数据读写工具；

内存映射文件操作工具。

2.2 NumPy的安装

和许多扩展库一样，numpy的安装可以通过pip进行。如果你已经安装了Python 3.x或以上版本，则安装Python时就已经自带了pip功能。打开命令行窗口，输入pip install numpy命令，即可自动下载并安装numpy。

安装完成后，可以打开一个命令行窗口进入Python终端会话，对之前的安装进行测试。输入import numpy as np命令，尝试导入numpy，如果没有出现任何错误信息提示，说明说明numpy已经成功安装，可以使用。

如果你已经安装了Anaconda，则Anaconda中已经集成了numpy，无需重复安装。

2.3 NumPy数组简介

**2.3.1 NumPy数组的创建、访问与基本属性**

数组NumPy中所提供的最基本，也是最常用的一种数据类型。NumPy数组的维度数量称之为秩（rank），即n维数组的秩为n。对于多维数组而言，构成它的每一个单一维度的数组称之为一个轴（axis）或维度（dimension）。例如，一个二维数组可以看作两个一维数组；这两个一维数组中的每个元素又可以看做一个一维数组。

首先介绍如何创建一个NumPy数组并查看一些关键属性。可以使用np.array()方法创建数组。下面这条语句便创建了一个简单的一维数组：

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

输入这条语句后，再键入数组名a并按回车键，就会出现array([0, 1, 2, 3])字样，表明我们已经创建了数组 [0, 1, 2, 3]。

直接创建二维或更高维度的数组方法同样是使用array()函数，只是在参数的输入上略有区别：

|  |
| --- |
| >>> b = np.array([[1, 2, 3, 4],[11, 12, 13, 14]]) # 创建二维数组  >>> b  array([[ 1, 2, 3, 4],  [11, 12, 13, 14]]) |

访问数组元素的方法与许多其他语言中对数组的访问类似，是通过指定数组元素的索引（下标）来访问某个位置的元素。对于一维数组而言，其访问方式几乎与Python中对列表元素的访问相同，而对于高维数组，则略有区别：

|  |
| --- |
| >>> a = np.array([1, 2, 4, 8, 16])  >>> b = np.array([[1, 2, 3, 4],[11, 12, 13, 14]])  >>> a[1] # 访问一维数组中的元素  2  >>> b[1, 3] # 访问二维数组中的元素  14  >>> b[1] # 获取二维数组中某一行元素  array([11, 12, 13, 14]) |

以如上的对二维数组b的访问为例，若要访问某个具体元素（如第1行第3列的元素），则通过形如a[1, 3]的方式指定其索引；而如果直接访问b[1]，则将取得该二维数组中的整行元素。

NumPy数组对象的属性非常丰富，接下来介绍一些常用的属性及其查看方式。

使用type(a)函数可以查看到，对象a的类型为ndarray：

|  |
| --- |
| >>> type(a)  <class 'numpy.ndarray'> |

而具体到数组中元素的数值类型，可以用a.dtype查看：

|  |
| --- |
| >>> a.dtype  dtype('int32') |

使用arange（m, n）函数可以创建一组包含从m到n-1的整数的数列。如果在输入参数时在整数后额外添加小数点，则数列中元素的数据类型为浮点型，否则默认为整型。

|  |
| --- |
| >>> a = np.arange(2, 10)  >>> a  array([2, 3, 4, 5, 6, 7, 8, 9])  >>> type(a[0])  <class 'numpy.int32'>  >>> b = np.arange(2., 10.) # 创建元素类型为浮点型的数组  >>> b  array([ 2., 3., 4., 5., 6., 7., 8., 9.]) |

在数组中需要注意数据类型的强制转换问题。例如，若试图向int32类型的数组中赋一个浮点型数值，则小数部分会被丢弃：

|  |
| --- |
| >>> a.dtype  dtype('int32')  >>> a[0] = 3.14  >>> a  array([3, 3, 4, 5, 6, 7, 8, 9]) |

由于NumPy的最主要作用之一就是处理矩阵数组，因此了解一个数组的维度就很关键。在NumPy中，我们可以通过shape属性查看数组的维度，shape是一个整数元组，元组中每一个元素代表每个维度的大小。例如，假设对某个数组b查看属性b.shape，得到一元组(3,)，表示b是一个一维数组，包含3个元素；如果得到二元组(2, 3)，表示b是一个二维数组，在两个维度上分别含有2个元素和3个元素，以此类推。

NumPy中的数组对象ndarray有如下较为常用的属性：

|  |  |
| --- | --- |
| ndarray.shape | 数组的维度，或矩阵的行列数。它返回一个元组，元组的长度就是维度的数目 |
| ndarray.size | 数组中元素的总数 |
| ndarray.dtype | ndarray对象的元素类型 |
| ndarray.itemsize | ndarray对象中单个元素大小。单位为字节 |
| ndarray.ndim | 维度数 |

**2.3.2 数组切片**

相信列表的切片操作对于较有经验的Python开发者来说并不陌生。本节将介绍NumPy数组的切片操作，供对Python列表切片较为生疏的读者了解，而对于较为熟悉列表切片的读者，也可作为一次快捷的复习。

数组切片访问的基本语法格式如下：

var[下界:上界:步长]

数组的切片访问方式允许用户通过指定下界和上界获取数组中指定部分，通过指定步长，则可以进一步确定要获取的数组元素的下标间隔值。需要注意的是，通过切片方式获得的数组遵循“左闭右开”规则，即该数组包括下标恰好为下界值的元素，但不包括下标恰好为上界值的元素，或用数学的语言描述，切片数组的范围为[下界， 上界)。

需要注意的是，负数同样可以作为访问数组元素的索引值。数组最后一个元素的索引值为-1，倒数第二个元素为-2，以此类推。通过这种方式可以方便地从后向前访问数组。

|  |
| --- |
| >>> # 负数索引 -5 -4 -3 -2 -1  ... # 对应正数索引： 0 1 2 3 4  ... a = np.array([10, 11, 12, 13, 14])  >>> # [10, 11, 12, 13, 14]  ... a[1:3]  array([11, 12])  >>> # 指定步长的切片访问  ... a[1:4:2]  array([11, 13])  >>> # 也可通过负数索引访问数组  ... a[1:-2]  array([11, 12])  >>> a[-4:3]  array([11, 12]) |

在切片访问数组时，可以省略对下界或上界的指定。若省略下界，则在切片时则认为下界为数组起点；若省略上界，则在切片时认为上界为数组终点；步长显然也可以省略，默认的步长为1，即依次访问界限区域内所有元素。

|  |
| --- |
| >>> # 获取前三个元素  ... a[:3]  array([10, 11, 12])  >>> # 获取后两个元素  ... a[-2:]  array([13, 14])  >>> # 从数组中每隔两个元素取一个元素  ... a[::2]  array([10, 12, 14]) |

对高维数组的切片访问相比于列表切片访问，在格式上略有区别。用户可对每个维度指定具体的切片方式。下面以对一个二维数组的访问为例进行说明。其中，我们使用了np.arange(25).reshape(5, 5)方法，创建了一个5行5列的、从0到24的二维数组。

|  |
| --- |
| >>> a = np.arange(25).reshape(5, 5) # 创建二维数组  >>> a  array([[ 0, 1, 2, 3, 4],  [ 5, 6, 7, 8, 9],  [10, 11, 12, 13, 14],  [15, 16, 17, 18, 19],  [20, 21, 22, 23, 24]])  >>> a[0, 3:5]  array([3, 4])  >>> a[4:, 4:]  array([[24]])  >>> a[:, 2]  array([ 2, 7, 12, 17, 22])  >>> a[2::2, ::2] # 指定步长的切片  array([[10, 12, 14],  [20, 22, 24]]) |

**2.2.3 切片的本质是引用**

切片的本质是对内存中原始数组的引用。对数组切片进行的修改最终都会影响到原始数组。我们来看下面一个例子：

|  |
| --- |
| >>> a = np.array([0, 1, 2, 3, 4])  >>> # 创建一个包含两个a中元素的数组切片  ... b = a[2:4]  >>> b  array([2, 3])  >>> b[0] = 10  >>> # 对b的修改也会修改a  ... a  array([ 0, 1, 10, 3, 4]) |

在上例中可以看到，将a的切片a[2:4]赋给b之后，对数组b进行修改的同时，原始数组a对应位置的元素也发生了同样的变化。

#end