Guru Gobind Singh Public School

Sector : V/B, Bokaro Steel City

Subject : IP Notes/Assignment : 2

Numpy

Indexing and Slicing

NumPy arrays can be indexed, sliced and iterated over.

Indexing

We have learnt about indexing single-dimensional array in section 6.2. For 2-D arrays indexing for both dimensions starts from 0, and each element is referenced through two indexes i and j, where i represents the row number and j represents the column number.

Table Marks of students in different subjects

Ramesh	78	67	56
Vedika	76	75	47
Harun	84	59	60
Prasad	67	72	54

Consider Table showing marks obtained by students in three different subjects. Let us create an array called marks to store marks given in three subjects for four students given in this table. As there are 4 students (i.e. 4 rows) and 3 subjects (i.e. 3 columns), the array will be called marks[4][3]. This array can store 4*3 = 12 elements. Here, marks[i,j] refers to the element at $(i+1)_{th}$ row and $(j+1)_{th}$ column because the index values start at 0. Thus marks[3,1] is the element in 4_{th} row and second column which is 72 (marks of Prasad in English). # accesses the element in the 1_{st} row in # the 3_{rd} column

```
>>> marks[0,2]
56
>>> marks [0,4]
index Out of Bound "Index Error". Index 4 is out of bounds for axis with size 3
```

Slicing

Sometimes we need to extract part of an array. This is done through slicing. We can define which part of the array to be sliced by specifying the start and end index values using [start : end] along with the array name.

Example

```
>>> array8
array([-2, 2, 6, 10, 14, 18, 22])
# excludes the value at the end index
>>> array8[3:5]
array([10, 14])
# reverse the array
>>> array8[ : : -1]
array([22, 18, 14, 10, 6, 2, -2])
```

Now let us see how slicing is done for 2-D arrays.

For this, let us create a 2-D array called array9 having 3 rows and 4 columns.

```
>>> array9 = np.array([[ -7, 0, 10, 20], [ -5, 1, 40, 200], [ -1, 1, 4, 30]])
# access all the elements in the 3rd column
>>> array9[0:3,2]
```

```
array([10, 40, 4])
```
Note that we are specifying rows in the range 0:3 because the end value of the range is excluded.
```
# access elements of 2nd and 3rd row from 1st and 2nd column
>>> array9[1:3,0:2]
array([[-5, 1], [-1, 1]])
```
If row indices are not specified, it means all the rows are to be considered. Likewise, if column indices are not specified, all the columns are to be considered.
Thus, the statement to access all the elements in the 3rd column can also be written as:
```
>>>array9[:,2]
array([10, 40, 4])
```
Operations on Arrays
Once arrays are declared, we can access it's element or perform certain operations the last section, we learnt about accessing elements. This section describes multiple operations that can be applied on arrays.
Arithmetic Operations
Arithmetic operations on NumPy arrays are fast and simple. When we perform a basic arithmetic operation like addition, subtraction, multiplication, division etc. on two arrays, the operation is done on each corresponding pair of elements. For instance, adding two arrays will result in the first element in the first array to be added to the first element in the second array, and so on.
Consider the following element-wise operations on two arrays:
```
>>> array1 = np.array([[3,6],[4,2]])
>>> array2 = np.array([[10,20],[15,12]])

#Element-wise addition of two matrices.
>>> array1 + array2
array([[13, 26],
[19, 14]])
#Subtraction
>>> array1 - array2
array([[ -7, -14],[-11, -10]])
#Multiplication
>>> array1 * array2
array([[ 30, 120],[ 60, 24]])
#Matrix Multiplication
>>> array1 @ array2
array([[120, 132],[ 70, 104]])
#Exponentiation
>>> array1 ** 3
array([[ 27, 216],[ 64, 8]], dtype=int32)
#Division
>>> array2 / array1
array([[3.33333333, 3.33333333],[3.75 , 6. ]])
#Element wise Remainder of Division
#(Modulo)
>>> array2 % array1
array([[1, 2],[3, 0]], dtype=int32)
```
It is important to note that for element-wise operations, size of both arrays must be same. That is,
array1.shape must be equal to array2.shape.

Sorting

Sorting is to arrange the elements of an array in hierarchical order either ascending or descending. By default, numpy does sorting in ascending order.

```
>>> array4 = np.array([1,0,2,-3,6,8,4,7])
>>> array4.sort()
>>> array4
array([-3, 0, 1, 2, 4, 6, 7, 8])
```

In 2-D array, sorting can be done along either of the axes i.e., row-wise or column-wise. By default, sorting is done row-wise (i.e., on axis = 1). It means to arrange elements in each row in ascending order. When axis=0, sorting is done column-wise, which means each column is sorted in ascending order.

```
>>> array4 = np.array([[10,-7,0, 20],[-5,1,200,40],[30,1,-1,4]])
>>> array4
array([[ 10, -7, 0, 20],[ -5, 1, 200, 40],[ 30, 1, -1, 4]])
#default is row-wise sorting
>>> array4.sort()
>>> array4
array([[ -7, 0, 10, 20],[ -5, 1, 40, 200],[ -1, 1, 4, 30]])
>>> array5 = np.array([[10,-7,0, 20],[-5,1,200,40],[30,1,-1,4]])
#axis =0 means column-wise sorting
>>> array5.sort(axis=0)
>>> array5
array([[ -5, -7, -1, 4],[ 10, 1, 0, 20],[ 30, 1, 200, 40]])
```

Concatenating Arrays

Concatenation means joining two or more arrays. Concatenating 1-D arrays means appending the sequences one after another. NumPy.concatenate() function can be used to concatenate two or more 2-D arrays either row-wise or column-wise. All the dimensions of the arrays to be concatenated must match exactly except for the dimension or axis along which they need to be joined. Any mismatch in the dimensions results in an error. By default, the concatenation of the arrays happens along axis=0.

Example

```
>>> array1 = np.array([[10, 20], [-30,40]])
>>> array2 = np.zeros((2, 3), dtype=array1.
dtype)
>>> array1
array([[ 10, 20],[-30, 40]])
>>> array2
array([[0, 0, 0],[0, 0, 0]])
>>> array1.shape
(2, 2)
>>> array2.shape
(2, 3)
```

Reshaping Arrays

We can modify the shape of an array using the reshape() function. Reshaping an array cannot be used to change the total number of elements in the array. Attempting

to change the number of elements in the array using
reshape() results in an error.

Example

```
>>> array3 = np.arange(10,22)
>>> array3
array([10, 11, 12, 13, 14, 15, 16, 17, 18,
19, 20, 21])

>>> array3.reshape(3,4)
array([[10, 11, 12, 13],[14, 15, 16, 17],[18, 19, 20, 21]])
>>> array3.reshape(2,6)
array([[10, 11, 12, 13, 14, 15],[16, 17, 18, 19, 20, 21]])
```

Splitting Arrays

We can split an array into two or more subarrays. numpy.split() splits an array along the specified axis. We can either specify sequence of index values where an array is to be split; or we can specify an integer N, that indicates the number of equal parts in which the array
is to be split, as parameter(s) to the NumPy.split() function. By default, NumPy.split() splits along axis =0. Consider the array given below:

```
>>> array4
array([[ 10, -7, 0, 20],[ -5, 1, 200, 40],[ 30, 1, -1, 4],[ 1, 2, 0, 4],[ 0, 1, 0, 2]])
# [1,3] indicate the row indices on which
# to split the array
>>> first, second, third = numpy split(array4,[1, 3])
# array4 is split on the first row and
# stored on the sub-array first
>>> first
array([[10, -7, 0, 20]])
# array4 is split after the first row and
# upto the third row and stored on the
# sub-array second
>>> second
array([[ -5, 1, 200, 40],[ 30, 1, -1, 4]])
# the remaining rows of array4 are stored
# on the sub-array third
>>> third
array([[1, 2, 0, 4],[0, 1, 0, 2]])

#[1, 2], axis=1 give the columns indices
#along which to split
>>> firstc, secondc, thirdc =numpy split(array4, [1, 2], axis=1)
>>> firstc
array([[10],[-5],[30],[ 1],[ 0]])
>>> secondcarray([[-7],[ 1],[ 1],[ 2],[ 1]])
>>> thirdcarray([[ 0, 20],[200, 40],[ -1, 4],[ 0, 4],[ 0, 2]])
# 2nd parameter 2 implies array is to be
# split in 2 equal parts axis=1 along the
# column axis
>>> firsthalf, secondhalf =np.split(array4,2,axis=1)
>>> firsthalfarray([[10, -7],
```

```
[-5, 1],[30, 1],[ 1, 2],[ 0, 1]])
>>> secondhalfarray([[ 0, 20],[200, 40],[ -1, 4],[ 0, 4],[ 0, 2]])
```

Statistical Operations on Arrays

NumPy provides functions to perform many useful statistical operations on arrays. In this section, we will apply the basic statistical techniques called descriptive statistics that we have learnt in chapter 5.

Let us consider two arrays:

```
>>> arrayA = np.array([1,0,2,-3,6,8,4,7])
>>> arrayB = np.array([[3,6],[4,2]])
```

1. The max() function finds the maximum element from an array.

```
# max element form the whole 1-D array
>>> arrayA.max()
8
# max element form the whole 2-D array
>>> arrayB.max()
6
# if axis=1, it gives column wise maximum
>>> arrayB.max(axis=1)
array([6, 4])
# if axis=0, it gives row wise maximum
>>> arrayB.max(axis=0)
array([4, 6])
```

2. The min() function finds the minimum element from an array.

```
>>> arrayA.min()
-3
>>> arrayB.min()
2
>>> arrayB.min(axis=0)
array([3, 2])
```

3. The sum() function finds the sum of all elements of an array.

```
>>> arrayA.sum()
25
>>> arrayB.sum()
15
#axis is used to specify the dimension
#on which sum is to be made. Here axis = 1
#means the sum of elements on the first row
>>> arrayB.sum(axis=1)
array([9, 6])
```

4. The mean() function finds the average of elements of the array.

```
>>> arrayA.mean()
3.125
>>> arrayB.mean()
3.75
3.550968177835448
```

Assignment :

1. Create an arrays and write NumPy commands for the following:
a) Find the dimensions, shape, size, data type of the items and itemsize of arrays zeros, vowels, ones.
b) Reshape the array ones to have all the 10 elements in a single row.
c) Display the 2nd and 3rd element of the array vowels.
d) Display all elements in the 2nd and 3rd row of the array.
e) Display the elements in the 1st and 2nd column of the array.
f) Display the elements in the 1st column of the 2nd and 3rd row of the array.
g) Reverse the array of vowels.
2. By creating an array marray1 and marray2, write NumPy commands for the following:

a) Divide all elements of array ones by 3.
b) Add the arrays myarray1 and myarray2.
c) Subtract myarray1 from myarray2 and store the result in a new array.
d) Multiply myarray1 and myarray2 elementwise.
e) Do the matrix multiplication of myarray1 and myarray2 and store the result in a new array myarray3.
f) Divide myarray1 by myarray2.
g) Find the cube of all elements of myarray1 and divide the resulting array by 2.
h) Find the square root of all elements of myarray2 and divide the resulting array by 2. The result should be rounded to two places of decimals.

Note : Solve assignment in a separate copy/Class work copy

For Reference refer to assignment I