Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.

A **numpy array** is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.

```
import numpy as np
```

```
# Create a rank 1 array
a = np.array([1, 2, 3])
print (type(a), a.shape, a[0], a[1], a[2])

# Change an element of the array
a[0] = 5
print (a)
```

```
# Create a rank 2 array
b = np.array([[1,2,3],[4,5,6]])
print (b)
```

```
# Create an array of all zeros
a = np.zeros((2,2))
print (a)
```

```
# Create an array of all ones
b = np.ones((1,2))
print (b)
```

```
# Create a constant array
c = np.full((2,2), 7)
print (c)
```

```
# Create a 2x2 identity matrix
d = np.eye(2)
print (d)
```

```
# Create an array filled with random values
e = np.random.random((2,2))
print (e)
```

Array indexing: Numpy offers several ways to index into arrays.

Slicing: Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

```
# Create the following rank 2 array with shape (3, 4)
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows and columns 1 and 2; b is the following array of shape (2, 2):
b = a[:2, 1:3]
print (b)
```

```
print (a[0, 1])
# b[0, 0] is the same piece of data as a[0, 1]
b[0, 0] = 77
print (a[0, 1])
```

Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array:

```
# Rank 1 view of the second row of a
row_r1 = a[1, :]

# Rank 2 view of the second row of a
row_r2 = a[1:2, :]
```

```
# Rank 2 view of the second row of a
row_r3 = a[[1], :]
print (row_r1, row_r1.shape)
print (row_r2, row_r2.shape)
print (row_r3, row_r3.shape)

# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print (col_r1, col_r1.shape)
print
print (col_r2, col_r2.shape)
```

In contrast, **integer array indexing** allows you to construct arbitrary arrays using the data from another array.

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

# An example of integer array indexing.
# The returned array will have shape (3,) and
print (a[[0, 1, 2], [0, 1, 0]])

# The above example of integer array indexing is equivalent to this:
print (np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

One **useful trick with integer array indexing** is selecting or mutating one element from each row of a matrix:

```
# Create a new array from which we will select elements
a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
print (a)

# Create an array of indices
b = np.array([0, 2, 0, 1])

# Select one element from each row of a using the indices in b
print (a[np.arange(4), b])


# Mutate one element from each row of a using the indices in b
a[np.arange(4), b] += 10
print (a)
```

Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

```
import numpy as np

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

# Find the elements of a that are bigger than 2; this returns a numpy array of Booleans of the same shape as a, where each slot of bool_idx t

bool_idx = (a > 2)

print (bool_idx)


# We use boolean array indexing to construct a rank 1 array consisting of the elements of a corresponding to the True values of bool_idx
print (a[bool_idx])
```

Array math: Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module.

```
x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
print (x + y)
print (np.add(x, y))
```

```
# Elementwise difference; both produce the array
print (x - y)
print (np.subtract(x, y))


# Elementwise product; both produce the array
print (x * y)
print (np.multiply(x, y))


# Elementwise division; both produce the array
print (x / y)
print (np.divide(x, y))


# Elementwise square root; produces the array
print (np.sqrt(x))


v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print (v.dot(w))
print (np.dot(v, w))


# Matrix / vector product; both produce the rank 1 array [29 67]
print( x.dot(v))
print (np.dot(x, v))


# Matrix / matrix product; both produce the rank 2 array
print (x.dot(y))
print (np.dot(x, y))


x = np.array([[1,2],[3,4]])

# Compute sum of all elements
print (np.sum(x))

# Compute sum of each column
print (np.sum(x, axis=0))

# Compute sum of each row
print (np.sum(x, axis=1))


#Transpose

print (x)
print (x.T)
```

Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

```
# We will add the vector v to each row of the matrix x,
# storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])

# Create an empty matrix with the same shape as x
y = np.empty_like(x)

# Add the vector v to each row of the matrix x with an explicit loop
for i in range(4):
    y[i, :] = x[i, :] + v

print (y)


vv = np.tile(v, (4, 1))  # Stack 4 copies of v on top of each other
print (vv)
```

```
y = x + vv  # Add x and vv elementwise
print (y)
```

```
import numpy as np

# We will add the vector v to each row of the matrix x, storing the result in the matrix y
x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
v = np.array([1, 0, 1])

# Add v to each row of x using broadcasting
y = x + v
print (y)
```

```
# Compute outer product of vectors
v = np.array([1,2,3])  # v has shape (3,)
w = np.array([4,5])    # w has shape (2,)

# To compute an outer product, we first reshape v to be a column vector of shape (3, 1); we can then broadcast it against w to yield an outpu
print (np.reshape(v, (3, 1)) * w)
```

```
# Add a vector to each row of a matrix
x = np.array([[1,2,3], [4,5,6]])

# x has shape (2, 3) and v has shape (3,) so they broadcast to (2, 3), giving the following matrix:
print (x + v)
```

```
# Add a vector to each column of a matrix
print ((x.T + w).T)
```

Exercises

1. Import the numpy package under the name np and print the numpy version and the configuration
2. Convert a list of numeric value into a one-dimensional NumPy array
3. Create a null vector of size 5
4. Create a null vector of size 12 but the fourth value which is 100
5. Create a vector with values ranging from 29 to 60
6. Create a 3x3 matrix with values ranging from 50 to 58
7. Find indices of non-zero elements from [5,8,0,9,0,0]
8. Create a 5x5 identity matrix
9. Create a 3x3x3 array with random values
10. Create a 9x9 array with random values and find the minimum and maximum values
11. Create a random vector of size 20 and find the mean value
12. Create a 6x6 matrix with values 1,2,3,4 just below the diagonal
13. Consider a (8,9,10) shape array, what is the index (x,y,z) of the 100th element?
14. Consider two random array A and B, check if they are equal
15. How to sort an array of shape (5, 4) by the nth column?
16. Consider an array a = [1,2,3,4,5,6,7,8,9,10,11,12,13,14], how to generate an array b = [[1,2,3,4], [2,3,4,5], [3,4,5,6], ..., [11,12,13,14]]?
17. How to find the most frequent value in an array?