Experiment No.: 1

Title: Write python program to demonstrate array creation, indexing, basic operations and unary and binary operator using Numpy

Objectives: 1. Implement the basic python programs using Numpy

Theory:

Python is agreat general-purpose programming language on its own, but with the help of a few popularlibraries (NumPy, SciPy, Matplotlib) it becomes a powerful environment for scientfic computing.

NumPy:

Numpy (Numerical Python) is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and routines for processing these arrays.NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library for python).

Operations using NumPy:

Using NumPy, a developer can perform the following operations −

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation.

NumPy 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

np.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0**)**

The above constructor takes the following parameters −

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**Ndmin** Specifies minimum dimensions of resultant array

We can initialize numpy arrays from nested Python lists, and access elements using square brackets:

Examples:

```
import numpy as np                                # import numpy and name it np
a = np.array([1, 2, 3])  # Create a rank 1 array
print(type(a))  # Prints "<class 'numpy.ndarray'>"
print(a.shape)  # Prints "(3,)"
print(a[0], a[1], a[2])  # Prints "1 2 3"
a[0] = 5  # Change an element of the array
print(a)  # Prints "[5, 2, 3]"
b = np.array([[1,2,3],[4,5,6]])  # Create a rank 2 array
print(b.shape)  # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])  # Prints "1 2 4"
```

Numpy also provides many functions to create arrays:

```
a = np.zeros((2,2))  # Create an array of all zeros
print(a)  # Prints "[[ 0. 0.]
#  [0. 0.]]"

b = np.ones((1,2)) # Create an array of all ones
print(b)  # Prints "[[ 1. 1.]]"
c = np.full((2,2), 7)  # Create a constant array
print(c)  # Prints "[[ 7. 7.]
                                                  #  [ 7. 7.]]"

d = np.eye(2)  # Create a 2x2 identity matrix
print(d)  # Prints "[[ 1. 0.]
                                                  #  [ 0. 1.]]"
e = np.random.random((2,2))  # Create an array with random values
print(e)  # Might print "[[0.91940167 0.08143941]
                                                  #  [0.68744134 0.87236687]]"
```

Creating an array from sub-classes:

```
np.array(np.mat('1 2; 3 4'))                     # Creates      array([[1, 2],
[3, 4]])

np.array(np.mat('1 2; 3 4'), subok=True)          # Creates  matrix([[1, 2],
     [3, 4]])
```

Array indexing:

Numpy offers several ways to index into arrays:fields access, Slicing and advanced indexing.

Slicing: Slicing is the way to choose a range of values in the array. We use a colon (:) in square brackets.

Syntax: **[Start : Stop : Step]**

```
# slice items between indexes
a = np.arange(10)
print a[2:5]                                      # prints [2   3   4]

# slice items starting from index
a = np.array([[1,2,3],[3,4,5],[4,5,6]])
print a[1:]                                       # prints [[3 4 5]
#[4 5 6]]
```

Slicing can also include ellipsis (…) to make a selection tuple of the same length as the dimension of an array. If ellipsis is used at the row position, it will return an ndarray comprising of items in rows.
a = np.array([[1,2,3],[3,4,5],[4,5,6]]) print a[...,1] # this returns array of items in the second column
print a[1,...] # this returns array of all items from the second row
print a[...,1:] # this returns array of all items from column 1 onwards
IntegerIndexing: selecting any arbitrary item in an array based on its Ndimensional index
x = np.array([[1, 2], [3, 4], [5, 6]]) y = x[[0,1,2], [0,1,0]] # includes elements at (0,0), (1,1) and (2,0) from the first array print y # [1 4 5]
Boolean Indexing: x = np.array([[0, 1, 2],[3, 4, 5],[6, 7, 8],[9, 10, 11]])
print x[x > 5] # prints [6 7 8 9 10 11]

NumPy - Arithmetic Operations

Input arrays for performing arithmetic operations such as add(), subtract(), multiply(), and divide() must be either of the same shape or should conform to array broadcasting rules.

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print 'First array:'
print a
print '\n'

print 'Second array:'
b = np.array([10,10,10])
print b
print '\n'

print 'Add the two arrays:'
print np.add(a,b)
print '\n'

print 'Subtract the two arrays:'
print np.subtract(a,b)
print '\n'

print 'Multiply the two arrays:'
print np.multiply(a,b)
print '\n'

print 'Divide the two arrays:'
print np.divide(a,b)
```

numpy.reciprocal()

This function returns the reciprocal of argument, element-wise. For elements with absolute values larger than 1, the result is always 0 because of the way in which Python handles integer division. For integer 0, an overflow warning is issued.

```
import numpy as np
a = np.array([0.25, 1.33, 1, 0, 100])

print 'Our array is:'
print a
print '\n'

print 'After applying reciprocal function:'
print np.reciprocal(a)
print '\n'

b = np.array([100], dtype = int)
print 'The second array is:'
print b
print '\n'

print 'After applying reciprocal function:'
print np.reciprocal(b)
```

numpy.power()

This function treats elements in the first input array as base and returns it raised to the power of the corresponding element in the second input array.

```
import numpy as np
a = np.array([10,100,1000])

print 'Our array is:'
print a
print '\n'

print 'Applying power function:'
print np.power(a,2)
print '\n'

print 'Second array:'
b = np.array([1,2,3])
print b
print '\n'

print 'Applying power function again:'
print np.power(a,b)
```

numpy.mod()

This function returns the remainder of division of the corresponding elements in the input array. The function **numpy.remainder()** also produces the same result.

```
import numpy as np
a = np.array([10,20,30])
b = np.array([3,5,7])

print 'First array:'
print a
print '\n'

print 'Second array:'
print b
print '\n'

print 'Applying mod() function:'
print np.mod(a,b)
print '\n'

print 'Applying remainder() function:'
print np.remainder(a,b)
```

Bitwise_and

The bitwise AND operation on the corresponding bits of binary representations of integers in input arrays is computed by np.bitwise_and() function.

```
import numpy as np
print 'Binary equivalents of 13 and 17:'
a,b = 13,17
print bin(a), bin(b)
print '\n'

print 'Bitwise AND of 13 and 17:'
print np.bitwise_and(13, 17)
```

The bitwise OR operation on the corresponding bits of binary representations of integers in input arrays is computed by **np.bitwise_or()** function.

numpy.invert()

This function computes the bitwise NOT result on integers in the input array. For signed integers, two's complement is returned.

The **numpy.left_shift()** function shifts the bits in binary representation of an array element to the left by specified positions. Equal number of 0s are appended from the right.

- Do some basic python programming using jupyter notebook
- Write implementation steps