`WORKING WITH NUMPY ARRAYS

EX. NO.:2.a **BASIC NUMPY OPERATIONS**

DATE:

AIM:

To perform basic NumPy operations in python for

(i) creating different types of NumPy arrays and displaying basic information, such as the data type, shape, size, and strides

(ii) creating an array using built-in NumPy functions

(iii) performing file operations with NumPy arrays

ALGORITHM:

Step 1: Start the program.

Step 2: Import the NumPy Library.

Step 3: Define a one-dimensional array, two-dimensional array, and three-dimensional array.

Step 4: Print the memory address, the shape, the data type, and the stride of the array.

Step 5: Then, create an array using built-in NumPy functions.

Step 6: Perform file operations with NumPy arrays.

Step 7: Display the output.

Step 8: Stop the program.

PROGRAM:

(i) Creation of different types of Numpy arrays and displaying basic information

```
# Importing numpy
import numpy as np

# Defining 1D array
my1DArray = np.array([1, 8, 27, 64])
```

```
print(my1DArray)

# Defining and printing 2D array
my2DArray = np.array([[1, 2, 3, 4], [2, 4, 9, 16], [4, 8, 18, 32]])
print(my2DArray)

#Defining and printing 3D array
my3Darray = np.array([[[ 1, 2 , 3 , 4],[ 5 , 6 , 7 ,8]], [[ 1, 2, 3, 4],[ 9, 10, 11,
12]]])
print(my3Darray)

# Print out memory address
print(my2DArray.data)

# Print the shape of array
print(my2DArray.shape)

# Print out the data type of the array
print(my2DArray.dtype)

# Print the stride of the array.
print(my2DArray.strides)
```

(ii) Creation of an array using built-in NumPy functions

```
# Array of ones
ones = np.ones((3,4))
print(ones)

# Array of zeros
zeros = np.zeros((2,3,4),dtype=np.int16)
print(zeros)

# Array with random values
np.random.random((2,2))

# Empty array
emptyArray = np.empty((3,2))
print(emptyArray)

# Full array
fullArray = np.full((2,2),7)
print(fullArray)
```

```
# Array of evenly-spaced values
evenSpacedArray = np.arange(10,25,5)
print(evenSpacedArray)

# Array of evenly-spaced values
evenSpacedArray2 = np.linspace(0,2,9)
print(evenSpacedArray2)
```

(iii) Performing file operations with NumPy arrays

```
import numpy as np

#initialize an array
arr = np.array([[[11, 11, 9, 9], [11, 0, 2, 0]], [[10, 14, 9, 14], [0, 1, 11, 11]]])

# open a binary file in write mode
file = open("arr", "wb")

# save array to the file
np.save(file, arr)

# close the file
file.close

# open the file in read binary mode
file = open("arr", "rb")

#read the file to numpy array
arr1 = np.load(file)
#close the file
print(arr1)
```

OUTPUT:

(i) Creation of different types of Numpy arrays and displaying basic information

```
[ 1  8 27 64]

[[ 1  2  3  4]
 [ 2  4  9 16]
 [ 4  8 18 32]]

[[[ 1  2  3  4]
  [ 5  6  7  8]]
 [[ 1  2  3  4]
  [ 9 10 11 12]]]

<memory at 0x00000247AE2A0A00>

(3, 4)

int32

(16, 4)
```

(ii) Creation of an array using built-in NumPy functions

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

[[[0 0 0 0]
  [0 0 0 0]
  [0 0 0 0]]
```

```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]]

[[0. 0.]

 [0. 0.]
 [0. 0.]]

[[7 7]

 [7 7]]

[10 15 20]

[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
```

(iii) Performing file operations with NumPy arrays

```
[[[11 11  9  9]
  [11  0  2  0]]


 [[10 14  9 14]
  [ 0  1 11 11]]]
```

RESULT:

Thus, the program to implement NumPy operations with arrays using Python has been executed and the output was verified successfully.

EX. NO.:2.b BASIC ARITHMETIC OPERATIONS WITH NUMPY ARRAYS

DATE

AIM:

To implement arithmetic operations with NumPy arrays using python.

ALGORITHM:

Step 1: Start the program.

Step 2: Import the NumPy Library.

Step 3: Initialize the NumPy arrays to two different variables.

Step 4: Perform the arithmetic operations on the two arrays using NumPy.

Step 5: Display the output.

Step 6: Stop the program.

PROGRAM:

```
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))
```

OUTPUT:

```
First array:
[[ 0. 1. 2.]
 [ 3. 4. 5.]
 [ 6. 7. 8.]]

Second array:
[10 10 10]

Add the two arrays:
[[ 10. 11. 12.]
 [ 13. 14. 15.]
 [ 16. 17. 18.]]

Subtract the two arrays:
[[-10. -9. -8.]
 [ -7. -6. -5.]
 [ -4. -3. -2.]]

Multiply the two arrays:
[[ 0. 10. 20.]
```

Ex.No 2 **Working with Numpy arrays**

PROGRAM:

1. **Write a Numpy program to convert an array to a float type.**

Program:

```
import numpy as np
integer_array = np.array([1, 2, 3, 4, 5])
float_array = integer_array.astype(float)
print("Original Array (integers):", integer_array)
print("Converted Array (float):", float_array)
```

Output:

```
Original Array (integers): [1 2 3 4 5]
Converted Array (float): [1. 2. 3. 4. 5.]
```

2. **Write a Numpy program to add a border (filled with 0's) around an existing array.**

Program:

```
import numpy as np
existing_array = np.array([[1, 2, 3],[4, 5, 6]]
padded_array = np.pad(existing_array, 1, mode='constant',
constant_values=0)
print(padded_array)
```

Output:

```
[[0 0 0 0 0]
 [0 1 2 3 0]
[0 4 5 6 0]
 [0 0 0 0 0]]
```

3. **Write a Numpy program to convert list and tuple into arrays.**

Program:

```
import numpy as np
list_data = [1, 2, 3, 4, 5]
array_from_list = np.array(list_data)
tuple_data = (5, 4, 3, 2, 1)
array_from_tuple = np.array(tuple_data)
print("Array from List:", array_from_list)
print("Array from Tuple:", array_from_tuple)
```

Output:

```
Array from List: [1 2 3 4 5]
Array from Tuple: [5 4 3 2 1]
```

4. **Write a Numpy program to convert append values to the end of the array.**

Program:

```
import numpy as np
initial_array = np.array([[1, 2, 3], [4, 5, 6]])
values_to_append = [[7, 8, 9], [10, 11, 12]]
new_array = np.append(initial_array, values_to_append, axis=0)
print("Initial 2D Array:")
print(initial_array)
print("\nArray after appending values along rows:")
print(new_array)
```

Output:

```
Initial 2D Array:
[[1 2 3]
 [4 5 6]]
Array after appending values along rows:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

5. **Write a Numpy program to create an empty and a full array.**

Program:

```
import numpy as np
empty_array = np.empty((3, 3))
fill_value = 7
full_array = np.full((2, 4), fill_value)
print("Empty Array:")
print(empty_array)
print("\nFull Array (filled with", fill_value, "):")
print(full_array)
```

Output:

```
Empty Array:
 [[5.07035229e-310 0.00000000e+000 0.00000000e+000] [0.00000000e+000
0.00000000e+000 0.00000000e+000]
[0.00000000e+000 0.00000000e+000 0.00000000e+000]]
 Full Array (filled with 7 ):
 [[7 7 7 7]
[7 7 7 7]]
```

6. **Write a Numpy program to find real and imaginary parts of an array of complex numbers.**

Program:

```
import numpy as np
```

```
complex_array = np.array([1 + 2j, 3 - 4j, 5 + 6j])
real_parts = np.real(complex_array)
imaginary_parts = np.imag(complex_array)
print("Original Complex Array:", complex_array)
print("Real Parts:", real_parts)
print("Imaginary Parts:", imaginary_parts)
```

Output:

```
Original Complex Array: [1.+2.j 3.-4.j 5.+6.j]
Real Parts: [1. 3. 5.]
Imaginary Parts: [ 2. -4.  6.]
```

7. **Write a Numpy program to convert a Python Dictionary to a Numpy ndarray**

Original dictionary:

{'column0': {'a': 1, 'b': 0.0, 'c': 0.0, 'd': 2.0},
'column1': {'a': 3.0, 'b': 1, 'c': 0.0, 'd': -1.0},
'column2': {'a': 4, 'b': 1, 'c': 5.0, 'd': -1.0},
'column3': {'a': 3.0, 'b': -1.0, 'c': -1.0, 'd': -1.0}}

Program:

```
import numpy as np
from ast import literal_eval
udict = """{"column0":{"a":1,"b":0.0,"c":0.0,"d":2.0},
"column1":{"a":3.0,"b":1,"c":0.0,"d":-1.0},
"column2":{"a":4,"b":1,"c":5.0,"d":-1.0},
"column3":{"a":3.0,"b":-1.0,"c":-1.0,"d":-1.0}
}"""
t = literal_eval(udict)
print("\nOriginal dictionary:")
print(t)
print("Type: ", type(t))
result_nparra = np.array([[v[j] for j in ['a', 'b', 'c', 'd']] for k, v in t.items()])
print("\nndarray:")
print(result_nparra)
print("Type: ", type(result_nparra))
```

Output:

```
Original dictionary:
{'column0': {'a': 1, 'b': 0.0, 'c': 0.0, 'd': 2.0},
'column1': {'a': 3.0, 'b': 1, 'c': 0.0, 'd': -1.0},
'column2': {'a': 4, 'b': 1, 'c': 5.0, 'd': -1.0},
'column3': {'a': 3.0, 'b': -1.0, 'c': -1.0, 'd': -1.0}}
Type:  <class 'dict'>
ndarray:
[[ 1.  0.  0.  2.]
```

```
 [ 3.  1.  0. -1.]
 [ 4.  1.  5. -1.]
 [ 3. -1. -1. -1.]]
Type:  <class 'numpy.ndarray'>
```

8. Write a NumPy program to search the index of a given array in another given array

Program:

```
import numpy as np
np_array = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
test_array = np.array([4, 5, 6])
print("Original NumPy array:")
print(np_array)
print("Searched array:")
print(test_array)
result = np.where((np_array == test_array).all(1))[0]
print("Index of the searched array in the original array:")
print(result)
```

Output:

```
 Original Numpy array:
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Searched array:
[4 5 6]
Index of the searched array in the original array:[1]
```

9. Creating Arrays from Python Lists:

In[1]: import numpy as np
In[2]: np.array([1, 4, 2, 5, 3])
Out|2]: array([1, 4, 2, 5, 3])
In|3|: np.array([3.14, 4, 2, 3])
Out|3]: array([3.14, 4. , 2. , 3.])
#If we want to explicitly set the data type of the resulting array, we can use the dtype keyword
In|4]: np.array([1, 2, 3, 4], dtype='float32')
Out|4|: array([1., 2., 3., 4.], dtype=float32)
In[5]: # nested lists result in multidimensional arrays
np.array([range(i, 1+ 3) for i in [2, 4, 6]])

Out|5]: array([[2, 3, 4],
[4,5,6],
[6,7, 8]])

[30. 40. 50.]

[60. 70. 80.]]

Divide the two arrays:

[[0. 0.1 0.2]

[0.3 0.4 0.5]

[0.6 0.7 0.8]]

RESULT:

Thus, the program to implement NumPy arithmetic operations with arrays using Python has been executed and the output was verified successfully.