ASSIGNMENT 3- RLMCA 369

ASHNA KARIM

INTMCA S9

ROLL NO: 16

Learn NumPy Fundamentals (Python Library for Data Science)

This course's primary purpose is to provide a thorough introduction to NumPy, a powerful Python library created primarily for numerical computing. The major goal of this course is to provide a thorough introduction to NumPy, a powerful Python library designed primarily for numerical computing. While the course's primary focus is NumPy, it also includes supplementary content on Python programming, which provides learners with additional value. . In essence, the course seeks to give students with the skills needed to manipulate and analyze numerical data in Python.

Array Creation in NumPy:

NumPy offers several functions to create arrays with initial placeholder content. These minimize the necessity of growing arrays, an expensive operation. For example: np.zeros,np.empty etc.
The numpy.array() function converts Python lists or tuples into arrays. Specialized functions like numpy.zeros(), numpy.ones(), and numpy.arange() facilitate creating arrays with default values, arrays filled with ones, and arrays with specified ranges.

numpy.empty(shape, dtype = float, order = 'C') :
Return a new array of given shape and type, with random values.

```
# Python Programming illustrating
# numpy.empty method

import numpy as geek

b = geek.empty(2, dtype = int)
print("Matrix b : \n", b)

a = geek.empty([2, 2], dtype = int)
print("\nMatrix a : \n", a)

c = geek.empty([3, 3])
print("\nMatrix c : \n", c)
```

Run on IDE

Output :

```
Matrix b :

[       0 1079574528]


Matrix a :

 [[0 0]

 [0 0]]
```

Matrix a :

[[0. 0. 0.]

[0. 0. 0.]

[0. 0. 0.]]

Reshaping and Indexing:

Reshaping and indexing are powerful operations in NumPy, enabling precise control over array structures. Reshaping involves altering the dimensions of an array, crucial for adapting data to different analyses or combining arrays efficiently. The numpy.reshape() function is commonly used for this purpose.

Indexing, on the other hand, allows for selective access and modification of array elements. It goes beyond basic slicing and includes advanced techniques like boolean indexing and fancy indexing. Mastering these methods provides flexibility in extracting specific data subsets.

NumPy's reshape() enables you to change the shape of an array into another compatible shape. Not all shapes are compatible since all the elements from the original array needs to fit into the new array.

You can use reshape() as either a function or a method. The documentation for the function shows three parameters:

1. a is the original array.
2. newshape is a tuple or an integer with the shape of the new array. When newshape is an integer, the new array will have one dimension.
3. order enables you to determine how the data is configured in the new array.

When using reshape() as a method of np.ndarray, you no longer need to use the first parameter, a, since the object is always passed to the method as its first argument. You'll use reshape() as a method in the rest of this tutorial. However, everything you learn about this method also applies to the function.

Note: A method is a function defined inside a class body. You call reshape() as a function using np.reshape(a, newshape). The equivalent method call is a.reshape(newshape).
In this section of the tutorial, you'll explore NumPy's reshape() through an example. You need to write code to help a school's head of computing, who needs to transform some data.

There are five classes in the same grade level, with ten pupils each. A 2D array stores the anonymized test results for each student. The shape of the array is (5, 10). The five rows represent the classes, and each row has the test score for each student in the class and can re-create a version of this array using random values. You can generate random values in NumPy by using the default_rng() generator and then calling one of its methods. In this example, you call .integers():

```
>>> import numpy as np

>>> rng = np.random.default_rng()

>>> results = rng.integers(0, 100, size=(5, 10))
>>> results
array([[53, 28, 33, 23, 81, 58, 16,  6, 91, 84],
    [ 4, 20, 20, 27, 27, 82, 19, 76, 57, 47],
    [56, 97, 53, 21, 72, 18, 91, 29, 38, 15],
    [85, 20, 30, 67, 65,  1, 52, 95, 87, 70],
    [66, 70,  9, 30, 73,  1, 56, 29, 10, 76]])
```

The .integers() method creates uniformly distributed random integers in the range defined by the first two arguments. The third argument, size, sets the shape of the array. The uniform distribution of integers is convenient for this example, but you can also use other distributions.

The head of computing analyzed the results of each class separately and now wants to convert the array into a single row containing all the results to continue working on the year as a whole.

One option is to convert the array with five rows and ten columns into a new array with one row and fifty columns. You can achieve this using reshape():

```
>>>
>>> year_results = results.reshape((1, 50))
>>> year_results
array([[53, 28, 33, 23, 81, 58, 16,  6, 91, 84,  4, 20, 20, 27, 27, 82,
    19, 76, 57, 47, 56, 97, 53, 21, 72, 18, 91, 29, 38, 15, 85, 20,
    30, 67, 65,  1, 52, 95, 87, 70, 66, 70,  9, 30, 73,  1, 56, 29,
    10, 76]])


>>> year_results.shape
(1, 50)
>>> year_results.ndim
2
```

Default Arrays:

Default arrays are important in NumPy because they provide an easy way to initialize arrays with predefined values. The numpy.zeros() function generates arrays of zeros that can be used as a starting point for a variety of numerical computations. Numpy.ones(), on the other hand, generates arrays of ones, which are frequently used to initialize weights in machine learning models.

Advanced Indexing:

NumPy's advanced indexing goes beyond typical methods, providing a variety of techniques for accessing and manipulating array elements. Boolean indexing and fancy indexing are two popular ways.

Array Math in NumPy:

NumPy's strength is in its array math, which allows for efficient and vectorized numerical operations on arrays. Basic arithmetic operations like addition, subtraction, multiplication, and division can be performed element-by-element, removing the need for explicit loops and significantly increasing computation performance.

Broadcasting :

Broadcasting is a strong NumPy feature that allows for the efficient handling of arrays of various shapes during arithmetic operations. It reduces the need for explicit looping and wasteful data duplication, resulting in more concise and computationally efficient code.