NUMpyASS

September 7, 2024

1. What is a Python library? Why do we use Python libraries? A Python library is a collection of modules and packages that provide pre-written code to perform common tasks. Libraries offer functions, classes, and methods that help avoid reinventing the wheel and make development more efficient. They are used to handle various tasks such as data analysis, visualization, web development, and more.

Why use Python libraries?

Efficiency: They provide ready-to-use solutions for common problems. Productivity: They save time and effort by providing pre-built functions and tools. Reliability: They are often well-tested and maintained by the community or organizations. Specialization: Libraries often focus on specific domains, such as NumPy for numerical operations or pandas for data manipulation. 2. What is the difference between NumPy array and List? Performance: NumPy arrays are more efficient than Python lists for numerical computations due to their contiguous memory allocation and support for vectorized operations. Functionality: NumPy arrays support mathematical operations, broadcasting, and more complex operations directly on the arrays, while lists do not. Data Types: NumPy arrays are homogeneous (all elements have the same data type), while Python lists can be heterogeneous (elements can have different data types).

[1]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
shape = arr.shape
size = arr.size
dimension = arr.ndim

print("Shape:", shape)
print("Size:", size)
print("Dimension:", dimension)
```

```
Shape: (3, 4)
Size: 12
Dimension: 2
```

[2]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
first_row = arr[0]
print("First Row:", first_row)
```

```
First Row: [1 2 3 4]
```

[3]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
element = arr[2, 3]
print("Element at third row and fourth column:", element)
```

```
Element at third row and fourth column: 12
```

[4]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
odd_indexed_elements = arr[::2, ::2]
print("Odd-indexed elements:", odd_indexed_elements)
```

```
Odd-indexed elements: [[ 1  3]
 [ 9 11]]
```

[5]:
```
import numpy as np

random_matrix = np.random.rand(3, 3)
print("Random 3x3 matrix with values between 0 and 1:\n", random_matrix)
```

```
Random 3x3 matrix with values between 0 and 1:
 [[0.70502475 0.15239195 0.90004226]
 [0.56237479 0.07217683 0.24420643]
 [0.52041585 0.13846388 0.66062439]]
```

8. Describe the difference between np.random.rand and np.random.randn? np.random.rand: Generates random numbers from a uniform distribution over [0, 1). It returns values that are evenly distributed between 0 and 1.

np.random.randn: Generates random numbers from a standard normal distribution (mean = 0, variance = 1). It returns values that are normally distributed around the mean.

[6]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
expanded_arr = np.expand_dims(arr, axis=0)
print("Array with increased dimension:\n", expanded_arr)
print("New shape:", expanded_arr.shape)
```

```
Array with increased dimension:
 [[[ 1  2  3  4]
  [ 5  6  7  8]
  [ 9 10 11 12]]]
New shape: (1, 3, 4)
```

[7]:
```
import numpy as np

arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
transposed_arr = arr.T
print("Transposed Array:\n", transposed_arr)
```

```
Transposed Array:
 [[ 1  5  9]
 [ 2  6 10]
 [ 3  7 11]
 [ 4  8 12]]
```

[8]:
```
import numpy as np

A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
B = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])

# Index wise multiplication
index_wise_multiplication = A * B

# Matrix multiplication
matrix_multiplication = np.dot(A, B.T)

# Add both matrices
matrix_addition = A + B

# Subtract matrix B from A
matrix_subtraction = A - B

# Divide Matrix B by A
matrix_division = B / A

print("Index wise multiplication:\n", index_wise_multiplication)
print("Matrix multiplication:\n", matrix_multiplication)
print("Matrix addition:\n", matrix_addition)
print("Matrix subtraction:\n", matrix_subtraction)
print("Matrix division:\n", matrix_division)
```

```
Index wise multiplication:
 [[  1   4   9  16]
 [ 25  36  49  64]
 [ 81 100 121 144]]
Matrix multiplication:
 [[ 30  70 110]
 [ 70 174 278]
 [110 278 446]]
Matrix addition:
 [[ 2  4  6  8]
```

```
 [10 12 14 16]
 [18 20 22 24]]
Matrix subtraction:
 [[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
Matrix division:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

[9]:
```python
import numpy as np

arr = np.array([1, 2, 3, 4], dtype=np.int32)
swapped_arr = arr.byteswap()
print("Array with byte order swapped:\n", swapped_arr)
```

```
Array with byte order swapped:
 [16777216 33554432 50331648 67108864]
```

13. What is the significance of the np.linalg.inv function? The np.linalg.inv function computes the (multiplicative) inverse of a matrix. It is used to solve systems of linear equations, among other applications.

14. What does the np.reshape function do, and how is it used? The np.reshape function changes the shape of an array without changing its data. The new shape must be compatible with the original shape.

[10]:
```python
import numpy as np

arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
reshaped_arr = np.reshape(arr, (3, 3))
print("Reshaped Array:\n", reshaped_arr)
```

```
Reshaped Array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
```

15. What is broadcasting in NumPy? Broadcasting is a technique used in NumPy to perform operations on arrays of different shapes. It automatically expands the smaller array to match the shape of the larger array, allowing for element-wise operations without explicitly replicating data. This makes operations on arrays more efficient and concise.

[ ]:
```python

```