**Basic Operations on NumPy Arrays**

NumPy is a versatile library in Python for numerical computing, offering a wide range of operations for manipulating arrays efficiently. In this blog post, we'll delve into various operations on NumPy arrays, including binary operations, mathematical functions, and string operations.

## 

## **1. Numpy Binary Operations:**

**Definition:** Binary operations involve element-wise operations between two arrays.

**Examples:**

import numpy as np

arr1 = np.array([1, 2, 3])

arr2 = np.array([4, 5, 6])

# Addition

addition = arr1 + arr2

# Subtraction

subtraction = arr1 - arr2

# Element-wise multiplication

multiplication = arr1 \* arr2

# Element-wise division

division = arr1 / arr2

print("Addition:", addition)

print("Subtraction:", subtraction)

print("Multiplication:", multiplication)

print("Division:", division)

**Explanation:**

* In this example, we perform various binary operations (addition, subtraction, multiplication, and division) between two NumPy arrays.
* Each operation is performed element-wise, resulting in arrays of the same shape as the input arrays.

### **Bitwise Shift Operations**

Bitwise shift operations involve moving the bits of a binary number to the left or right by a specified number of positions. These operations are essential for tasks like encoding and decoding data, optimizing memory usage, and implementing various algorithms.

NumPy offers functions for left and right bitwise shifts, allowing users to perform these operations on arrays of integers efficiently.

### **numpy.left\_shift()**

This function shifts the bits of a given integer to the left. It appends zeros to the right of the binary representation of the input integer, effectively multiplying the integer by 2 raised to the power of the shift amount.

Let's illustrate this with an example:

import numpy as np

input\_num = 5

bit\_shift = 2

output\_num = np.left\_shift(input\_num, bit\_shift)

print("After left shifting {} by {}: {}".format(input\_num, bit\_shift, output\_num))

In this example, the left\_shift() function shifts the bits of the input number 5 to the left by 2 positions, resulting in the output 20.

### **\* numpy.right\_shift()**

Conversely, the numpy.right\_shift() function shifts the bits of an integer to the right. It divides the integer by 2 raised to the power of the shift amount, effectively shifting the bits to the right and discarding the least significant bits.

Here's how it works:

import numpy as np

input\_num = 20

bit\_shift = 2

output\_num = np.right\_shift(input\_num, bit\_shift)

print("After right shifting {} by {}: {}".format(input\_num, bit\_shift, output\_num))

In this example, the right\_shift() function shifts the bits of the input number 20 to the right by 2 positions, resulting in the output 5.

## 

## **2. Numpy Mathematical Functions:**

**Definition:** NumPy provides a wide range of mathematical functions for performing operations on arrays.

**Examples:**

import numpy as np

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

# Exponential function

exp\_arr = np.exp(arr)

# Square root function

sqrt\_arr = np.sqrt(arr)

# Trigonometric functions

sin\_arr = np.sin(arr)

cos\_arr = np.cos(arr)

print("Exponential:", exp\_arr)

print("Square Root:", sqrt\_arr)

print("Sine:", sin\_arr)

print("Cosine:", cos\_arr)

**Explanation:**

* In this example, we demonstrate various mathematical functions available in NumPy, such as exponential, square root, sine, and cosine functions.
* These functions are applied element-wise to the input array, producing arrays of the same shape as the input.

**3. Numpy String Operations:**

**Definition:** NumPy also provides string operations for arrays containing string elements.

**Examples:**

import numpy as np

arr = np.array(['apple', 'banana', 'cherry'])

# Concatenation

concatenated = np.char.add(arr, ' pie')

# Uppercase

uppercased = np.char.upper(arr)

# Length of each string

lengths = np.char.str\_len(arr)

print("Concatenated:", concatenated)

print("Uppercased:", uppercased)

print("Lengths:", lengths)

**Explanation:**

* In this example, we showcase string operations available in NumPy, including concatenation, uppercase conversion, and length calculation.
* These operations are applied element-wise to each string in the array, preserving the array's shape.

NumPy's array operations offer a powerful way to manipulate numerical and string data efficiently in Python. By leveraging these operations, you can perform a wide range of tasks, from simple arithmetic operations to complex mathematical computations and string manipulations. Experiment with different examples and explore NumPy's extensive documentation to unlock the full potential of array operations in your Python projects.