Arrays and Matrices in Python: Operations Unleashed!

Hey Python explorers! Today, we're tackling **arrays** (think lists with a twist) and **matrices** (2D grids of numbers), using operators to crunch them. We'll:

- Use **lists** as basic arrays with operators (+, *, etc.).
- Power up with **NumPy** for matrices and advanced ops (dot product, transpose).

Arrays with Lists: The Basics

Python lists act like arrays—ordered, mutable sequences. We'll use operators to manipulate them. No NumPy yet—just pure Python!

```
# Example 1: Arithmetic Operators
numbers = [1, 2, 3, 4]
# Addition (concatenation with another list)
more_numbers = numbers + [5, 6]
print("Addition (+):", more_numbers)
```

```
Addition (+): [1, 2, 3, 4, 5, 6]
```

```
# Multiplication (repetition)
repeated = numbers * 2
print("Multiplication (*):", repeated)
```

```
Multiplication (*): [1, 2, 3, 4, 1, 2, 3, 4]
```

```
# Example 2: Element-wise with Loops
doubled = []
for num in numbers:
    doubled.append(num * 2)  # Multiply each element
print("Element-wise (*):", doubled)
```

```
Element-wise (*): [2, 4, 6, 8]
```

```
# Example 3: Comparison Operators
a = [1, 2, 3]
b = [1, 2, 4]
print("List ==:", a == b)  # False—checks whole list
print("Element >:", [x > y for x, y in zip(a, b)])  # Compare elements
```

```
List ==: False
Element >: [False, False, False]
```

```
# Challenge: Your array ops!
your_list = [int(x) for x in input("Enter 3 numbers (space-separated): ").split()]
added = your_list + [10]
multiplied = [x * 3 for x in your_list]
```

```
Enter 3 numbers (space-separated): 1 2 3
```

```
# Challenge: Your array ops!
your_list = [int(x) for x in input("Enter 3 numbers (space-separated): ").split()]
added = your_list + [10]
multiplied = [x * 3 for x in your_list]
print("Your list + [10]:", added)

```

```
Enter 3 numbers (space-separated): 1 2 3
Your list + [10]: [1, 2, 3, 10]
```

```

print("Your list * 3 (element-wise):", multiplied)
# Try subtraction (x - 1) or > 5 comparison!
```

NumPy: Real Arrays and Matrices

For serious array and matrix work, we use **NumPy**—Python's math powerhouse. Install it with `pip install numpy` if needed. It handles element-wise ops and matrix math like a champ!

```
import numpy as np

# Example 1: Creating Arrays
array1 = np.array([1, 2, 3, 4])
print("NumPy Array:", array1)
```

```
NumPy Array: [1 2 3 4]
```

```
# Arithmetic Operators (Element-wise)
print("Add 2:", array1 + 2)      # [3, 4, 5, 6]
print("Multiply by 3:", array1 * 3)  # [3, 6, 9, 12]
print("Divide by 2:", array1 / 2)    # [0.5, 1. , 1.5, 2. ]
```

```
Add 2: [3 4 5 6]
Multiply by 3: [ 3  6  9 12]
Divide by 2: [0.5 1.  1.5 2. ]
```

```
# Example 2: Comparison Operators
print("Greater than 2:", array1 > 2)  # [False, False,  True,  True]
```

```
print("Equal to 3:", array1 == 3)     # [False, False,  True, False]
```

```
Greater than 2: [False False  True  True]
Equal to 3: [False False  True False]
```

```
# Challenge: Your NumPy array!
your_nums = np.array([int(x) for x in input("Enter 4 numbers (space-separated): ").split
print("Your array:", your_nums)
print("Add 5:", your_nums + 5)
print("Less than 10:", your_nums < 10)
# Try * 4 or == your_nums[0]!
```

```
Enter 4 numbers (space-separated): 1 2 3 4
Your array: [1 2 3 4]
Add 5: [6 7 8 9]
Less than 10: [ True  True  True  True]
```

Matrices with NumPy: 2D Power

A **matrix** is a 2D array—rows and columns. NumPy makes it easy with operators and special matrix ops (like dot product). Let's roll!

```
import numpy as np

# Example 1: Creating a Matrix
matrix = np.array([[1, 2],
                   [3, 4]])
print("Matrix:\n", matrix)
```

```
Matrix:
 [[1 2]
 [3 4]]
```

```
print(matrix[1][0])
```

```
3
```

```
# Arithmetic Operators (Element-wise)
print("Add 1:\n", matrix + 1)
print("Multiply by 2:\n", matrix * 2)
```

```
Add 1:
 [[2 3]
 [4 5]]
Multiply by 2:
 [[2 4]
 [6 8]]
```

```
# Example 2: Matrix Multiplication (Dot Product)
matrix2 = np.array([[5, 6], [7, 8]])
dot_result = np.dot(matrix, matrix2)
print("Dot Product (matrix * matrix2):\n", dot_result)
```

```
Dot Product (matrix * matrix2):
 [[19 22]
 [43 50]]
```

```

# 1*5 + 2*7 = 19, etc.
```

```
# Example 3: Transpose (Flip rows and columns)
print("Transpose:\n", matrix.T)
```

```
Transpose:
 [[1 3]
 [2 4]]
```

```
# Example 4: Comparison
print("Greater than 2:\n", matrix > 2)
```

```
Greater than 2:
 [[False False]
 [ True  True]]
```

```
# Challenge: Your matrix!
rows = int(input("How many rows? "))
cols = int(input("How many cols? "))
print(f"Enter {rows * cols} numbers (space-separated):")
data = [int(x) for x in input().split()]
your_matrix = np.array(data).reshape(rows, cols)
print("Your matrix:\n", your_matrix)
print("Times 3:\n", your_matrix * 3)
print("Dot with itself:\n", np.dot(your_matrix, your_matrix.T if rows != cols else your_
```

Bonus: Arrays/Matrices with Loops and Ifs

Let's mix arrays and matrices with flow control for extra fun!

```
import numpy as np

# Function with array and if
def filter_array(arr):
    return [x for x in arr if x > 0]

```

```
my_array = np.array([-1, 2, -3, 4])
print("Filtered positives:", filter_array(my_array))
```

```
# Nested loop with matrix
matrix = np.array([[1, 2], [3, 4]])
for i in range(matrix.shape[0]):  # Rows
    for j in range(matrix.shape[1]):  # Columns
        if matrix[i, j] % 2 == 0:
            matrix[i, j] *= 2
print("Even numbers doubled:\n", matrix)
```

Wrap-Up: Array and Matrix Masters!

You've conquered:

- **Lists as Arrays**: +, *, and loops.
- **NumPy Arrays**: Element-wise ops (+, -, *, /, >, ==).
- **Matrices**: Dot product, transpose, and more.

Keep experimenting! Multiply matrices, filter arrays, or try `np.zeros()` for an empty matrix. What's your favorite op? Share it!

```
Start coding or generate with AI.
```