**Advanced Guide to NumPy for Computer Science Majors**

## Overview of NumPy

NumPy (Numerical Python) is a fundamental package for scientific computing with Python. It supports large multi-dimensional arrays and matrices and includes a suite of mathematical functions to perform operations on these data structures efficiently.

## Core Capabilities of NumPy

1. **ndarray**: Efficient N-dimensional array structure.
2. **Broadcasting**: Enables arithmetic operations on arrays of different shapes.
3. **Vectorized Computations**: Eliminates the need for explicit loops.
4. **C/C++ Integration**: Seamlessly interoperates with lower-level languages.
5. **High Performance**: Offers optimized performance in numerical computations.

## Installation

pip install numpy

## Importing the Library

import numpy as np

## Creating Arrays

# One-dimensional array  
arr1 = np.array([1, 2, 3])  
  
# Two-dimensional array (matrix)  
arr2 = np.array([[1, 2, 3], [4, 5, 6]])  
  
# Three-dimensional array (tensor)  
arr3 = np.array([[[1, 2], [3, 4]], [[5, 6], [7, 8]]])

## Understanding Array Attributes

arr = np.array([[1, 2], [3, 4]])  
print(arr.ndim) # Number of dimensions  
print(arr.shape) # Dimensions as tuple (rows, columns)  
print(arr.size) # Total number of elements  
print(arr.dtype) # Data type of array elements

## Initializing Arrays

np.zeros((2, 3)) # Matrix filled with zeros  
np.ones((2, 2)) # Matrix filled with ones  
np.eye(3) # Identity matrix  
np.full((2, 2), 7) # Matrix with constant values  
np.arange(0, 10, 2) # Evenly spaced values  
np.linspace(0, 1, 5) # Linearly spaced values between 0 and 1

## Array Indexing and Slicing

arr = np.array([10, 20, 30, 40])  
print(arr[1]) # Access element at index 1  
print(arr[1:3]) # Slice from index 1 to 2  
  
arr2 = np.array([[1, 2, 3], [4, 5, 6]])  
print(arr2[1, 1]) # Access row 1, column 1  
print(arr2[:, 1]) # Access entire second column

## Performing Array Operations

a = np.array([1, 2, 3])  
b = np.array([4, 5, 6])  
  
print(a + b) # Element-wise addition  
print(a \* b) # Element-wise multiplication  
print(a \*\* 2) # Element-wise exponentiation

## Utilizing Universal Functions (ufuncs)

np.sqrt([1, 4, 9]) # Square roots  
np.exp([1, 2, 3]) # Exponentials  
np.log([1, np.e, np.e\*\*2]) # Logarithms  
np.sin(np.pi / 2) # Trigonometric functions

## Aggregation and Statistical Analysis

arr = np.array([1, 2, 3, 4, 5])  
print(np.sum(arr)) # Total sum  
print(np.mean(arr)) # Average  
print(np.std(arr)) # Standard deviation  
print(np.min(arr)) # Minimum  
print(np.max(arr)) # Maximum  
print(np.argmax(arr)) # Index of max value

## Reshaping and Flattening Arrays

arr = np.arange(1, 10)  
print(arr.reshape((3, 3))) # Reshape into 3x3 matrix  
print(arr.flatten()) # Flatten to 1D array

## Concatenating Arrays

a = np.array([[1, 2], [3, 4]])  
b = np.array([[5, 6], [7, 8]])  
  
print(np.vstack((a, b))) # Stack vertically  
print(np.hstack((a, b))) # Stack horizontally

## Splitting Arrays

arr = np.array([1, 2, 3, 4, 5, 6])  
print(np.split(arr, 3)) # Split into 3 equal parts

## Random Number Generation

np.random.rand(3, 2) # Uniform distribution  
np.random.randn(2, 3) # Standard normal distribution  
np.random.randint(1, 10, 5) # Random integers

## Linear Algebra with NumPy

A = np.array([[1, 2], [3, 4]])  
B = np.array([[2, 0], [1, 3]])  
  
print(np.dot(A, B)) # Matrix multiplication  
print(np.linalg.inv(A)) # Inverse of matrix A  
print(np.linalg.det(A)) # Determinant of A  
print(np.linalg.eig(A)) # Eigenvalues and eigenvectors

## File Input/Output

np.save('my\_array.npy', arr) # Save in binary format  
arr\_loaded = np.load('my\_array.npy') # Load binary array  
  
np.savetxt('my\_file.txt', arr) # Save as text  
np.loadtxt('my\_file.txt') # Load text file

## Applied Mini Projects with NumPy

### 1. **Student Performance Analyzer**

* Input: Numerical arrays representing exam scores across multiple subjects.
* Task: Calculate descriptive statistics such as average, minimum, maximum, and standard deviation.
* Output: Statistical summary to understand performance distribution.

### 2. **Image Processing with Arrays**

* Convert images to arrays using external libraries like PIL or matplotlib.
* Apply filters such as grayscale conversion, inversion, and edge detection using array manipulation.

### 3. **Climate Data Evaluation**

* Load time-series temperature data from CSV files.
* Compute rolling averages, identify outliers or spikes.

### 4. **Probabilistic Dice Simulation**

* Simulate rolling two dice 10,000 times.
* Analyze the probability distribution of summed outcomes.

### 5. **Interactive Matrix Calculator**

* Create a terminal-based tool for matrix operations: addition, multiplication, inversion, and determinant evaluation.

This guide equips computer science students with a practical understanding of NumPy’s computational capabilities. With its application across scientific domains, NumPy is indispensable for data analysis, machine learning, and algorithm development.

If you’d like, I can generate a Jupyter notebook or Python script for any of the listed projects.