## NumPy: Comprehensive Notes

NumPy (Numerical Python) is a fundamental package for numerical computation in Python. It provides support for large, multi-dimensional arrays1 and matrices, along with a large library of high-level mathematical functions to operate on2 these arrays. Here's a comprehensive overview covering its core concepts and functionalities:

**I. Introduction to NumPy**

* **What is NumPy?**
  + A powerful Python library for numerical computing.
  + Provides high-performance multi-dimensional array objects (ndarrays).
  + Offers a vast collection of routines for mathematical, logical, shape manipulation, sorting, selection, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random3 number generation, and more.
  + Forms the foundation for many other scientific computing libraries in Python (e.g., SciPy, Pandas, Matplotlib, scikit-learn).
* **Why NumPy?**
  + **Efficiency:** NumPy arrays are implemented in C, making operations significantly faster than standard Python lists, especially for large datasets.
  + **Convenience:** Provides a concise and expressive syntax for array operations, reducing the need for explicit loops.
  + **Functionality:** Offers a rich set of built-in functions for common numerical tasks.
  + **Memory Efficiency:** NumPy arrays typically store data in contiguous blocks of memory, leading to better cache utilization.
  + **Integration:** Seamlessly integrates with other Python scientific computing libraries.
* **Installation:**  
  Bash  
  pip install numpy
* **Importing NumPy:**  
  Python  
  import numpy as np # Standard convention to import as 'np'
* **Checking NumPy Version:**  
  Python  
  import numpy as np  
  print(np.\_\_version\_\_)

**II. The NumPy ndarray (N-dimensional Array)**

* **Core Data Structure:** The fundamental object in NumPy is the ndarray, a homogeneous multi-dimensional array.
  + **Homogeneous:** All elements in an ndarray must be of the same data type (e.g., integer, float, boolean).
  + **Multi-dimensional:** Can represent vectors (1D), matrices (2D), or higher-dimensional tensors.
* **Creating ndarrays:**
  + **From Python Lists:**  
    Python  
    my\_list = [1, 2, 3]  
    my\_array = np.array(my\_list)  
    my\_nested\_list = [[1, 2], [3, 4]]  
    my\_matrix = np.array(my\_nested\_list)
  + **Using NumPy Functions:**
    - np.zeros(shape): Creates an array filled with zeros.  
      Python  
      np.zeros(5) # 1D array of 5 zeros  
      np.zeros((2, 3)) # 2x3 array of zeros
    - np.ones(shape): Creates an array filled with ones.  
      Python  
      np.ones(3)  
      np.ones((4, 2))
    - np.full(shape, fill\_value): Creates an array filled with a specified value.  
      Python  
      np.full((2, 2), 7)
    - np.eye(n): Creates an identity matrix (square matrix with ones on the main diagonal and zeros elsewhere).  
      Python  
      np.eye(3)
    - np.arange(start, stop, step): Creates an array with evenly spaced values within a given range (similar to Python's range but returns an ndarray).  
      Python  
      np.arange(10) # [0 1 2 3 4 5 6 7 8 9]  
      np.arange(2, 10) # [2 3 4 5 6 7 8 9]  
      np.arange(0, 11, 2) # [ 0 2 4 6 8 10]  
      np.arange(10, 0, -1) # [10 9 8 7 6 5 4 3 2 1]
    - np.linspace(start, stop, num): Creates an array with num evenly spaced values over a specified interval (inclusive of both endpoints by default).  
      Python  
      np.linspace(0, 1, 5) # [0. 0.25 0.5 0.75 1. ]  
      np.linspace(0, 1, 5, endpoint=False) # [0. 0.2 0.4 0.6 0.8]
    - np.random.rand(d0, d1, ..., dn): Creates an array of given shape with random floats in the interval [0, 1).  
      Python  
      np.random.rand(3) # 1D array of 3 random numbers  
      np.random.rand(2, 2) # 2x2 array of random numbers
    - np.random.randn(d0, d1, ..., dn): Creates an array of given shape with random floats from a standard normal distribution (mean 0, variance 1).  
      Python  
      np.random.randn(4)  
      np.random.randn(3, 3)
    - np.random.randint(low, high=None, size=None, dtype='l'): Returns random integers from low (inclusive) to high (exclusive). If high is None, values are from [0, low).  
      Python  
      np.random.randint(10) # A single random integer between 0 and 9  
      np.random.randint(2, 10) # A single random integer between 2 and 9  
      np.random.randint(2, 10, size=5) # 1D array of 5 random integers  
      np.random.randint(2, 10, size=(2, 3)) # 2x3 array of random integers
* **Array Attributes:**
  + ndarray.ndim: The number of dimensions (axes) of the array.
  + ndarray.shape: A tuple indicating the size of each dimension. For a matrix with n rows and m columns, shape will be (n, m).
  + ndarray.size: The total number of elements in the array (product of the elements in shape).
  + ndarray.dtype: The data type of the elements in the array (e.g., int64, float64, bool, object).
  + ndarray.itemsize: The size (in bytes) of each element in the array.
  + ndarray.data: The buffer containing the actual elements of the array4 (rarely used directly).

Python  
arr = np.array([[1, 2, 3], [4, 5, 6]])  
print(arr.ndim) # Output: 2  
print(arr.shape) # Output: (2, 3)  
print(arr.size) # Output: 6  
print(arr.dtype) # Output: int64 (or similar)  
print(arr.itemsize) # Output: 8 (for int64)

* **Data Types (dtypes):**
  + NumPy supports a wide range of numerical and other data types.
  + Common dtypes include:
    - int8, int16, int32, int64 (signed integers of different sizes)
    - uint8, uint16, uint32, uint64 (unsigned integers)
    - float16, float32, float64, float128 (floating-point numbers of different precisions)
    - complex64, complex128 (complex numbers)
    - bool (Boolean values: True or False)
    - object (Python objects - use with caution due to potential performance overhead)
    - string\_ or S<number> (fixed-length strings)
    - unicode\_ or U<number> (fixed-length Unicode strings)
  + You can explicitly specify the dtype when creating an array:  
    Python  
    np.array([1, 2, 3], dtype=float)  
    np.zeros((2, 2), dtype=np.int32)
  + You can change the data type of an existing array using the astype() method (creates a new array):  
    Python  
    arr\_int = np.array([1, 2, 3])  
    arr\_float = arr\_int.astype(float)  
    arr\_bool = arr\_int.astype(bool) # Non-zero becomes True, zero becomes False

**III. Array Indexing and Slicing**

* **1D Arrays:** Similar to Python lists.  
  Python  
  arr = np.array([10, 20, 30, 40, 50])  
  print(arr[0]) # Output: 10  
  print(arr[2:4]) # Output: [30 40]  
  print(arr[:3]) # Output: [10 20 30]  
  print(arr[2:]) # Output: [30 40 50]  
  print(arr[-1]) # Output: 50 (last element)  
  print(arr[::-1]) # Output: [50 40 30 20 10] (reversed array)
* **2D Arrays (Matrices):**
  + arr[row\_index, column\_index] or arr[row\_index][column\_index]

Python  
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(matrix[0, 0]) # Output: 1  
print(matrix[1, 2]) # Output: 6  
print(matrix[0]) # Output: [1 2 3] (first row)  
print(matrix[:, 0]) # Output: [1 4 7] (first column)  
print(matrix[0:2, 1:3]) # Output: [[2 3] [5 6]] (sub-matrix)

* **N-dimensional Arrays:** Use a comma-separated tuple of indices or slices for each dimension.  
  Python  
  tensor = np.arange(24).reshape((2, 3, 4)) # 3D array  
  print(tensor[0, 1, 2]) # Access element at first "plane", second row, third column  
  print(tensor[1, :, 0]) # Access the first column of the second "plane"
* **Slicing creates views, not copies:** Modifying a slice will modify the original array. To get a copy, use the .copy() method.  
  Python  
  arr = np.array([1, 2, 3, 4, 5])  
  slice\_arr = arr[1:4]  
  slice\_arr[0] = 99  
  print(slice\_arr) # Output: [99 3 4]  
  print(arr) # Output: [ 1 99 3 4 5] (original array modified)  
    
  arr\_copy = arr[1:4].copy()  
  arr\_copy[0] = -1  
  print(arr\_copy) # Output: [-1 3 4]  
  print(arr) # Output: [ 1 99 3 4 5] (original array unchanged)
* **Boolean Indexing (Masking):** Select elements based on a Boolean array of the same shape.  
  Python  
  arr = np.array([1, 2, 3, 4, 5])  
  mask = arr > 2  
  print(mask) # Output: [False False True True True]  
  print(arr[mask]) # Output: [3 4 5]  
  print(arr[arr % 2 == 0]) # Output: [2 4] (select even numbers)  
    
  matrix = np.array([[1, 2], [3, 4], [5, 6]])  
  mask = matrix > 3  
  print(matrix[mask]) # Output: [4 5 6]
* **Fancy Indexing:** Select elements using an array of indices. This always returns a copy.
  + **1D Array:**  
    Python  
    arr = np.array([10, 20, 30, 40, 50])  
    indices = [1, 3, 0]  
    print(arr[indices]) # Output: [20 40 10]
  + **2D Array:**  
    Python  
    matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
    row\_indices = [0, 2]  
    col\_indices = [1, 2]  
    print(matrix[row\_indices, col\_indices]) # Output: [2 9] (elements at (0, 1) and (2, 2))  
      
    # To select sub-matrices:  
    print(matrix[[0, 1], :]) # Select first two rows  
    print(matrix[:, [0, 2]]) # Select first and third columns

**IV. Array Operations**

* **Element-wise Operations:** Standard arithmetic operators (+, -, \*, /, \*\*, %) operate element-wise on arrays of the same shape.  
  Python  
  a = np.array([1, 2, 3])  
  b = np.array([4, 5, 6])  
  print(a + b) # Output: [5 7 9]  
  print(a \* 2) # Output: [2 4 6]  
  print(b \*\* 2) # Output: [16 25 36]  
  print(a < b) # Output: [ True True True] (element-wise comparison)
* **Universal Functions (ufuncs):** NumPy provides a wide range of ufuncs that perform element-wise operations.
  + **Mathematical Functions:** np.sin(), np.cos(), np.tan(), np.exp(), np.log(), np.sqrt(), np.abs(), np.round(), np.floor(), np.ceil().
  + **Comparison Functions:** np.equal(), np.not\_equal(), np.greater(), np.greater\_equal(), np.less(), np.less\_equal().
  + **Logical Functions:** np.logical\_and(), np.logical\_or(), np.logical\_not(), np.logical\_xor().

Python  
arr = np.array([0, np.pi/2, np.pi])  
print(np.sin(arr)) # Output: [0. 1. 0.]  
print(np.exp([0, 1, 2])) # Output: [1. 2.71828183 7.3890561 ]  
  
a = np.array([True, False, True])  
b = np.array([False, True, True])  
print(np.logical\_and(a, b)) # Output: [False False True]

* **Array Broadcasting:** NumPy's powerful mechanism for performing operations on arrays with different shapes. Broadcasting rules determine how smaller arrays are "stretched" to match the shape of larger arrays.
  + **Rule 1:** If the arrays have a different number of dimensions, the array with fewer dimensions is padded with ones on its leading (left) side.
  + **Rule 2:** If the shape of two arrays does not match in any dimension, the array with shape equal to 1 in that dimension5 is stretched to match the other shape.
  + **Rule 3:** If6 the shape in any dimension disagrees and neither is equal to 1, then broadcasting is not possible, and a ValueError is raised.

Python  
a = np.array([1, 2, 3]) # Shape (3,)  
scalar = 5 # Shape ()  
print(a + scalar) # Output: [6 7 8] (scalar is broadcasted to (3,))  
  
b = np.array([[10], [20], [30]]) # Shape (3, 1)  
c = np.array([1, 2, 3]) # Shape (3,) -> (1, 3) after padding  
print(b + c) # Output: [[11 12 13]  
 # [21 22 23]  
 # [31 32 33]]  
  
d = np.array([1, 2]) # Shape (2,)  
e = np.array([[10, 20, 3

**Sources**

1. <https://www.read.careercredentials.in/blog/tags/python/>

2. <https://github.com/donnemartin/dev-setup>

3. <https://solace.cnrs.fr/slides/23-07-03/Seminar.html>

4. <https://github.com/damipaigu/Python-Learning-Notes>

5. <https://frankmbrown.net/jupyter-notebooks/80fde586-26d6-4275-8a5d-e153f902a9fe/NumPy%20Review>

6. <https://procodebase.com/article/mastering-numpy-array-indexing-and-slicing>

7. <https://github.com/FFizzZZ/Fizz>

8. <https://towardsdatascience.com/broadcasting-in-numpy-58856f926d73>