😊 NumPy Arrays in Python

NumPy (Numerical Python) is a fundamental library in Python used for numerical and scientific computing. It provides support for large, multi-dimensional arrays and matrices, along with a collection of high-level mathematical functions to operate on these data structures efficiently. NumPy is built for performance, allowing for fast operations on large data sets through optimized C-based backend implementations.

Use Cases:

- **Numerical Computations**: Performing mathematical operations such as linear algebra, statistics, and Fourier transforms.
- **Data Analysis and Manipulation**: Used as a foundation for data manipulation in libraries like Pandas.
- **Machine Learning**: Frequently used in preprocessing data and implementing algorithms in machine learning workflows.
- **Scientific Computing**: Supports simulations, modeling, and mathematical operations in disciplines like physics, chemistry, and engineering.
- **Image Processing**: Assists in converting and manipulating pixel data in image arrays.
- **Backend for Other Libraries**: Many popular libraries such as SciPy, Pandas, and scikit-learn are built on top of NumPy.

Benefits:

- **Efficiency**: Operations are faster than native Python lists due to vectorized operations and efficient memory usage.
- **Convenience**: Provides a wide range of mathematical functions and broadcasting capabilities that make code more concise and expressive.
- **Integration**: Easily integrates with other Python libraries and external systems (e.g., C/C++, Fortran).
- **Consistency**: Offers a consistent data structure that simplifies development and debugging.

Overall, NumPy is a core tool in the Python ecosystem for data science, engineering, and computational tasks.

😄 To begin with the Lab

1. Open Jupyter Notebook, and start with installing numpy in the Notebook.

   ! pip install numpy

2. Now we will import **numpy as np.** NumPy has many built-in functions and capabilities. We won't cover them all, but instead we will focus on some of the most important aspects of NumPy: vectors, arrays, matrices, and number generation.
3. Then we created a list with a few numbers and we checked for it, after that, we printed out our list.

```
[6]: import numpy as np

[8]: mylist = [1,2,3]

[10]: type(mylist)

[10]: list

[12]: mylist

[12]: [1, 2, 3]
```

4. NumPy arrays are the main way we will use NumPy throughout the course. NumPy arrays essentially come in two flavors: vectors and matrices. Vectors are strictly 1-dimensional (1D) arrays and matrices are 2D (but you should note a matrix can still have only one row or one column).

Why use Numpy array? Why not just a list?

There are lot's of reasons to use a Numpy array instead of a "standard" python list object. Our main reasons are:

- Memory Efficiency of Numpy Array vs list
- Easily expands to N-dimensional objects
- Speed of calculations of numpy array
- Broadcasting operations and functions with numpy
- All the data science and machine learning libraries we use are built with Numpy

5. The code creates two variables: my_list, which is a standard Python list containing the elements 1, 2, and 3, and my_array, which is a NumPy array created from the same values.
6. By using the type() function on my_list, it checks and confirms that my_list is a built-in Python list.
7. This example highlights the difference between native Python lists and NumPy arrays, which, although similar in appearance, have different underlying structures and capabilities—NumPy arrays being more efficient and powerful for numerical computations.

```
[3]: my_list = [1,2,3]
     my_array = np.array([1,2,3])

[4]: type(my_list)

[4]: list
```

8. We can create an array by directly converting a list or list of lists.

```
[2]: my_list = [1,2,3]
     my_list

[2]: [1, 2, 3]

[3]: np.array(my_list)

[3]: array([1, 2, 3])

[4]: my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
     my_matrix

[4]: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]

[5]: np.array(my_matrix)

[5]: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

9. There are lots of built-in ways to generate arrays. The arange() function returns evenly spaced values within a given interval.

```
[6]: np.arange(0,10)

[6]: array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

[7]: np.arange(0,11,2)

[7]: array([ 0,  2,  4,  6,  8, 10])
```

10. The **zeros and ones** method generates arrays of zeros or ones.

```
[8]: np.zeros(3)

[8]: array([0., 0., 0.])

[9]: np.zeros((5,5))

[9]: array([[0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.],
            [0., 0., 0., 0., 0.]])

[10]: np.ones(3)

[10]: array([1., 1., 1.])

[11]: np.ones((3,3))

[11]: array([[1., 1., 1.],
            [1., 1., 1.],
            [1., 1., 1.]])
```

11. The np.linspace(start, stop, num) function in NumPy generates num evenly spaced values between the start and stop values (inclusive).

```
[12]: np.linspace(0,10,3)

[12]: array([ 0.,  5., 10.])

[13]: np.linspace(0,5,20)

[13]: array([0.        , 0.26315789, 0.52631579, 0.78947368, 1.05263158,
            1.31578947, 1.57894737, 1.84210526, 2.10526316, 2.36842105,
            2.63157895, 2.89473684, 3.15789474, 3.42105263, 3.68421053,
            3.94736842, 4.21052632, 4.47368421, 4.73684211, 5.        ])
```

Note that `.linspace()` *includes* the stop value. To obtain an array of common fractions, increase the number of items:

```
[14]: np.linspace(0,5,21)

[14]: array([0.  , 0.25, 0.5 , 0.75, 1.  , 1.25, 1.5 , 1.75, 2.  , 2.25, 2.5 ,
            2.75, 3.  , 3.25, 3.5 , 3.75, 4.  , 4.25, 4.5 , 4.75, 5.  ])
```

12. The eye() method returns an identity matrix.

```
[15]: np.eye(4)

[15]: array([[1., 0., 0., 0.],
            [0., 1., 0., 0.],
            [0., 0., 1., 0.],
            [0., 0., 0., 1.]])
```

13. Numpy also has lots of ways to create random number arrays.
14. The rand() method or funtion creates an array of the given shape and populates it with random samples from a uniform distribution over [0, 1).

```
[16]: np.random.rand(2)

[16]: array([0.37065108, 0.89813878])

[17]: np.random.rand(5,5)

[17]: array([[0.03932992, 0.80719137, 0.50145497, 0.68816102, 0.1216304 ],
       [0.44966851, 0.92572848, 0.70802042, 0.10461719, 0.53768331],
       [0.12201904, 0.5940684 , 0.89979774, 0.3424078 , 0.77421593],
       [0.53191409, 0.0112285 , 0.3989947 , 0.8946967 , 0.2497392 ],
       [0.5814085 , 0.37563686, 0.15266028, 0.42948309, 0.26434141]])
```

15. The randn() returns a sample (or samples) from the "standard normal" distribution [σ = 1]. Unlike **rand** which is uniform, values closer to zero are more likely to appear.

```
[18]: np.random.randn(2)

[18]: array([-0.36633217, -1.40298731])

[19]: np.random.randn(5,5)

[19]: array([[-0.45241033,  1.07491082,  1.95698188,  0.40660223, -1.50445807],
       [ 0.31434506, -2.16912609, -0.51237235,  0.78663583, -0.61824678],
       [-0.17569928, -2.39139828,  0.30905559,  0.1616695 ,  0.33783857],
       [-0.2206597 , -0.05768918,  0.74882883, -1.01241629, -1.81729966],
       [-0.74891671,  0.88934796,  1.32275912, -0.71605188,  0.0450718 ]])
```

16. The randint() returns random integers from low (inclusive) to high (exclusive).

```
[20]: np.random.randint(1,100)

[20]: 61

[21]: np.random.randint(1,100,10)

[21]: array([39, 50, 72, 18, 27, 59, 15, 97, 11, 14])
```

17. The seed() can be used to set the random state, so that the same "random" results can be reproduced.

```
[22]: np.random.seed(42)
      np.random.rand(4)

[22]: array([0.37454012, 0.95071431, 0.73199394, 0.59865848])

[23]: np.random.seed(42)
      np.random.rand(4)

[23]: array([0.37454012, 0.95071431, 0.73199394, 0.59865848])
```

18. The code creates two different NumPy arrays. The first array, generated using np.arange(25), contains a sequence of integers starting from 0 up to 24, providing a convenient way to create a range of numbers.

19. The second array, produced using np.random.randint(0, 50, 10), contains 10 randomly selected integers between 0 and 49, which is useful for generating test data, simulating randomness, or initializing values for various computational tasks.

```
[24]: arr = np.arange(25)
      ranarr = np.random.randint(0,50,10)
```

•••

```
[25]: arr
```

```
[25]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
             17, 18, 19, 20, 21, 22, 23, 24])
```

```
[26]: ranarr
```

```
[26]: array([38, 18, 22, 10, 10, 23, 35, 39, 23,  2])
```

20. The reshape() returns an array containing the same data with a new shape.

```
[27]: arr.reshape(5,5)
```

```
[27]: array([[ 0,  1,  2,  3,  4],
             [ 5,  6,  7,  8,  9],
             [10, 11, 12, 13, 14],
             [15, 16, 17, 18, 19],
             [20, 21, 22, 23, 24]])
```

21. Max, min, argmax, argmix these are useful methods for finding max or min values. Or to find their index locations using argmin or argmax.

```
[28]: ranarr
```

```
[28]: array([38, 18, 22, 10, 10, 23, 35, 39, 23,  2])
```

```
[29]: ranarr.max()
```

```
[29]: 39
```

```
[30]: ranarr.argmax()
```

```
[30]: 7
```

```
[31]: ranarr.min()
```

```
[31]: 2
```

```
[32]: ranarr.argmin()
```

```
[32]: 9
```

22. Shape() is an attribute that arrays have (not a method).

[33]:
```
# Vector
arr.shape
```

[33]:
```
(25,)
```

[34]:
```
# Notice the two sets of brackets
arr.reshape(1,25)
```

[34]:
```
array([[ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
        16, 17, 18, 19, 20, 21, 22, 23, 24]])
```

[35]:
```
arr.reshape(1,25).shape
```

[35]:
```
(1, 25)
```

[36]:
```
arr.reshape(25,1)
```

[36]:
```
array([[ 0],
       [ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10],
       [11],
       [12],
       [13],
       [14],
       [15],
       [16],
       [17],
       [18],
       [19],
       [20],
       [21],
       [22],
       [23],
       [24]])
```

[37]:
```
arr.reshape(25,1).shape
```

[37]:
```
(25, 1)
```

23. Using dtype you can also grab the data type of the object in the array.

[38]:
```
arr.dtype
```

[38]:
```
dtype('int32')
```

[39]:
```
arr2 = np.array([1.2, 3.4, 5.6])
arr2.dtype
```

[39]:
```
dtype('float64')
```