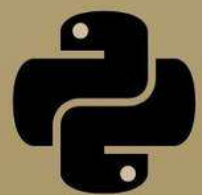


Python series

# NumPy Fundamentals



With Code Examples

# What is NumPy?

NumPy is a powerful library for numerical computing in Python. It provides support for large, multi-dimensional arrays and matrices, along with a vast collection of high-level mathematical functions to operate on these arrays.



```
import numpy as np
```

# Creating NumPy Arrays

NumPy arrays can be created from Python lists or using special functions.



```
# From a Python list
a = np.array([1, 2, 3, 4])
# Output: [1 2 3 4]

# Using special functions
b = np.zeros((2, 3)) # Create an array of zeros
# Output: [[0. 0. 0.]
#          [0. 0. 0.]]
c = np.ones((3, 2)) # Create an array of ones
# Output: [[1. 1.]
#          [1. 1.]
#          [1. 1.]]
d = np.random.rand(2, 2) # Create an array of random values
# Output will be different each time
```

# Array Indexing and Slicing

NumPy arrays can be indexed and sliced like Python lists, but with more flexibility.

```
● ● ●  
  
a = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])  
print(a[1, 2]) # Output: 6  
print(a[:, 1]) # Output: [2 5 8]  
print(a[1:, :2]) # Output: [[4 5]  
                  #         [7 8]]
```

# Array Operations

NumPy provides a wide range of mathematical operations that can be applied to arrays elementwise or across entire arrays.



```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

c = a + b    # Elementwise addition
# Output: [5 7 9]
d = a * b    # Elementwise multiplication
# Output: [4 10 18]
e = np.sum(a) # Sum of all elements in the array
# Output: 6
```

# Broadcasting

NumPy's broadcasting feature allows arithmetic operations between arrays with different shapes.



```
a = np.array([[1, 2, 3], [4, 5, 6]])  
b = np.array([10, 20, 30])  
  
c = a + b # Broadcasting: b is "stretched" to match a's shape  
# Output: [[11 22 33]  
#          [14 25 36]]
```

# Array Reshaping

NumPy arrays can be reshaped to different dimensions without changing their data.



```
a = np.array([1, 2, 3, 4, 5, 6])
b = a.reshape(2, 3) # Reshape to a 2x3 array
# Output: [[1 2 3]
#          [4 5 6]]
c = a.reshape(3, 2) # Reshape to a 3x2 array
# Output: [[1 2]
#          [3 4]
#          [5 6]]
```



# Array Concatenation

NumPy provides functions to concatenate arrays along different axes.



```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

c = np.concatenate((a, b), axis=0) # Concatenate along rows
# Output: [[1 2]
#          [3 4]
#          [5 6]
#          [7 8]]

d = np.concatenate((a, b), axis=1) # Concatenate along columns
# Output: [[1 2 5 6]
#          [3 4 7 8]]
```



# Conditions and Boolean Arrays

NumPy allows you to apply conditions and create boolean arrays for advanced indexing and filtering.



```
a = np.array([1, 2, 3, 4, 5])
condition = a > 2
# Output: [False False True True True]
b = a[condition] # Filter elements greater than 2
# Output: [3 4 5]
```

# Mathematical Functions

NumPy provides a wide range of mathematical functions to perform various operations on arrays.



```
a = np.array([1, 2, 3, 4])
b = np.sin(a) # Compute sine values
# Output: [ 0.84147098  0.90929743  0.14112001 -0.7568025 ]
c = np.exp(a) # Compute exponential values
# Output: [2.71828183e+00 7.38905610e+00 2.00855369e+01 5.45981500e+01]
d = np.sqrt(a) # Compute square root values
# Output: [1.          1.41421356 1.73205081 2.          ]
```

# Loading and Saving Arrays

NumPy provides functions to load and save arrays from/to disk in various formats.



```
# Save an array to a binary file
a = np.array([1, 2, 3, 4])
np.save('data.npy', a)

# Load an array from a binary file
b = np.load('data.npy')
# Output: [1 2 3 4]
```

# NumPy and Data Analysis

NumPy seamlessly integrates with other data analysis libraries like Pandas and Matplotlib, making it an essential tool for scientific computing and data analysis in Python.



```
import pandas as pd
import matplotlib.pyplot as plt

data = pd.read_csv('data.csv')
x = data['x'].values # Convert to NumPy array
y = data['y'].values

plt.plot(x, y)
plt.show()
# Displays a line plot using the NumPy arrays x and y
```

# Array Statistics

NumPy provides functions to compute various statistical properties of arrays.



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

```
mean_value = np.mean(a)
```

```
# Output: 3.0
```

```
median_value = np.median(a)
```

```
# Output: 3.0
```

```
std_dev = np.std(a)
```

```
# Output: 1.4142135623730951
```

# Random Sampling

NumPy's random module allows you to generate random numbers and perform random sampling from arrays.

```
import numpy as np

# Generate 5 random numbers between 0 and 1
random_nums = np.random.rand(5)
# Output: [0.37454012 0.95071431 0.73199394 0.59865848 0.15601864]

# Randomly select 3 elements from an array
a = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
random_elements = np.random.choice(a, size=3, replace=False)
# Output: [5 9 1]
```



# Linear Algebra with NumPy

NumPy provides functions for performing various linear algebra operations on arrays.



```
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])

# Matrix multiplication
c = np.matmul(a, b)
# Output: [[19 22]
#          [43 50]]

# Compute the determinant
det_a = np.linalg.det(a)
# Output: -2.000000000000000004
```



# Array Sorting

NumPy provides functions to sort arrays along one or more axes.



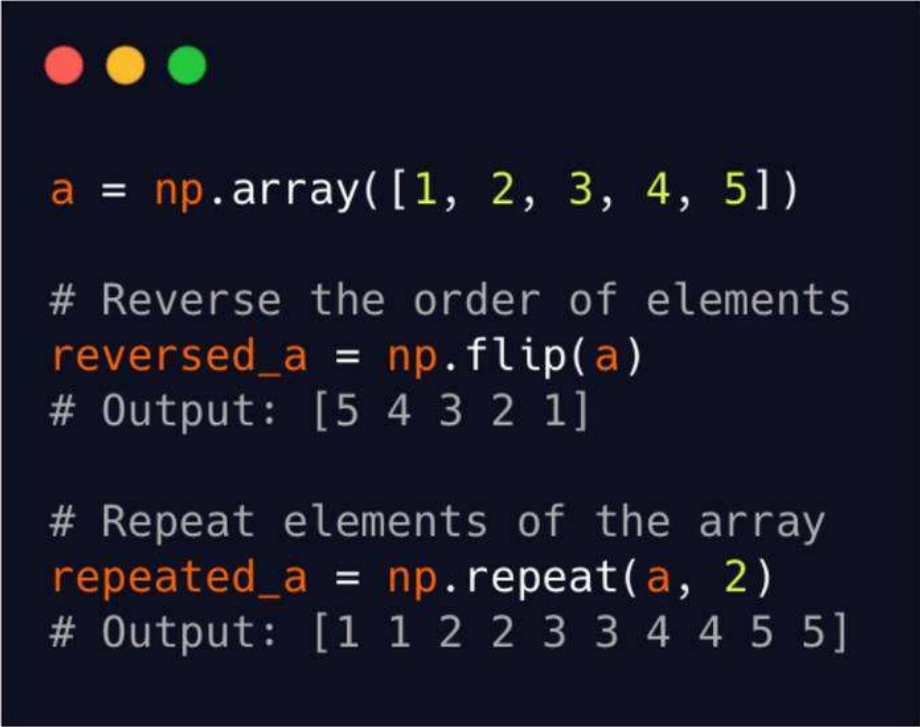
```
a = np.array([5, 2, 9, 1, 7])

# Sort the array
sorted_a = np.sort(a)
# Output: [1 2 5 7 9]

# Sort a 2D array along the columns
b = np.array([[3, 1, 9], [4, 2, 8], [5, 6, 7]])
sorted_b = np.sort(b, axis=1)
# Output: [[1 3 9]
#          [2 4 8]
#          [5 6 7]]
```

# Array Manipulation with NumPy

NumPy provides various functions to manipulate and transform arrays.



```
a = np.array([1, 2, 3, 4, 5])  
  
# Reverse the order of elements  
reversed_a = np.flip(a)  
# Output: [5 4 3 2 1]  
  
# Repeat elements of the array  
repeated_a = np.repeat(a, 2)  
# Output: [1 1 2 2 3 3 4 4 5 5]
```

# Array Comparisons

NumPy allows you to perform element-wise comparisons between arrays, resulting in boolean arrays.



```
a = np.array([1, 2, 3, 4, 5])
b = np.array([3, 4, 5, 6, 7])

# Element-wise comparison
greater_than = a > b
# Output: [False False False False False]

less_than_equal = a <= b
# Output: [ True  True  True  True  True]
```

# **Data scientist &ML Engineer**



**Follow For More Data  
Science Content**