Python Package - Numpy

NumPy, short for Numerical Python, is a fundamental library for numerical computing in Python.

It provides support for large multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently

Key Features of NumPy

1. **N-Dimensional Array Object (ndarray)**:
   - The core feature of NumPy is its powerful N-dimensional array object called ndarray. It allows for efficient storage and manipulation of large datasets.
2. **Element-wise Operations**:
   - NumPy supports element-wise operations, enabling fast mathematical computations on arrays, such as addition, subtraction, multiplication, and division.
3. **Broadcasting**:
   - Broadcasting allows NumPy to perform arithmetic operations on arrays of different shapes, making code more concise and avoiding the need for manual resizing.
4. **Mathematical Functions**:
   - NumPy offers a wide range of mathematical functions, including trigonometric, statistical, and linear algebra functions, to perform complex calculations on arrays.
5. **Slicing and Indexing**:
   - NumPy provides advanced slicing and indexing capabilities to access and modify array elements efficiently.
6. **Integration with Other Libraries**:

- NumPy seamlessly integrates with other scientific computing libraries, such as SciPy, pandas, and matplotlib, making it a cornerstone of the scientific Python ecosystem.

Array Creation and Initialization

```
import numpy as np

# Creating a 1-dimensional array (vector)
vector = np.array([1, 2, 3, 4, 5])
print("Vector:", vector)
        Vector: [1 2 3 4 5]
# Creating a 2-dimensional array (matrix)
matrix = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
print("Matrix:\n", matrix)
                Matrix:
                 [[1 2 3]
                 [4 5 6]
                 [7 8 9]]
# Element-wise operations
sum_vector = vector + 2
print("Vector + 2:", sum_vector)
                Vector + 2: [3 4 5 6 7]
```

```
product_matrix = matrix * 2
print("Matrix * 2:\n", product_matrix)
            Matrix * 2:
             [[ 2  4  6]
             [ 8 10 12]
             [14 16 18]]

# Slicing and indexing
sub_array = matrix[0:2, 1:3]  # Selecting a sub-array
print("Sub-array:\n", sub_array)
            Sub-array:
             [[2 3]
             [5 6]]

# Broadcasting
broadcast_sum = matrix + vector[:3]
print("Broadcast Sum:\n", broadcast_sum)
            Broadcast Sum:
             [[ 2  4  6]
             [ 5  7  9]
             [ 8 10 12]]
```

```
# Mathematical functions
mean_vector = np.mean(vector)
print("Mean of Vector:", mean_vector)
```

```
Mean of Vector: 3.0
```

```
det_matrix = np.linalg.det(matrix)
print("Determinant of Matrix:", det_matrix)
```

```
Determinant of Matrix: 0.0
```

1. **numpy.zeros():** Creates an array filled with zeros.

```
import numpy as np
zeros_array = np.zeros((3, 3))
print(zeros_array)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

2. **numpy.ones():** Creates an array filled with ones.

```
ones_array = np.ones((2, 4))
print(ones_array)
```

```
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

3. **numpy.full():** Creates an array filled with a specified value.

```
full_array = np.full((3, 3), 7)
print(full_array)
```

```
[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

4. **numpy.eye():** Creates an identity matrix.

```
identity_matrix = np.eye(4)
print(identity_matrix)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

5. **numpy.random.rand():** Creates an array of the given shape and populates it with random samples from a uniform distribution over **[0, 1).**

```
random_array = np.random.rand(3, 3)
print(random_array)
```

```
[[0.44271926    0.22662296    0.97060748]
 [0.48818755    0.19711331    0.49495352]
 [0.27694657    0.54148347    0.28210088]]
```

6. **numpy.random.randint():** Creates an array of random integers from a specified range.

```
randint_array = np.random.randint(0, 10, (3, 3))
print(randint_array)
```

```
[[0 5 9]
 [1 5 0]
 [7 1 8]]
```

Array Manipulation

1. **numpy.reshape():** Reshapes an array without changing its data.

```
array = np.array([1, 2, 3, 4, 5, 6])
reshaped_array = array.reshape((2, 3))
print(reshaped_array)
```

```
[[1 2 3]
 [4 5 6]]
```

2. numpy.flatten(): Flattens a multi-dimensional array into a one-dimensional array.

```
multi_array = np.array([[1, 2], [3, 4], [5, 6]])
flattened_array = multi_array.flatten()
print(flattened_array)
```

```
[1 2 3 4 5 6]
```

3. **numpy.transpose():** Transposes the dimensions of an array.

```
matrix = np.array([[1, 2, 3], [4, 5, 6]])
transposed_matrix = np.transpose(matrix)
print(transposed_matrix)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

4. numpy.concatenate(): Concatenates two or more arrays along a specified axis.

```
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
concatenated_array = np.concatenate((array1, array2))
print(concatenated_array)
```

```
    [1 2 3 4 5 6]
```

5. **numpy.stack(arrays, axis=0):** Stacks arrays along a new axis.

   arrays: Sequence of array-like elements to stack. These arrays must have the same shape along all but the specified axis.

   axis: The axis along which the arrays will be stacked. The default value is 0. The axis parameter specifies the position of the new axis in the result.

```
import numpy as np
array1 = np.array([1, 2, 3])
array2 = np.array([4, 5, 6])
# Stack arrays along the first axis (default axis=0)
stacked_array = np.stack((array1, array2))
print("Stacked Array (axis=0):\n", stacked_array)
```

```
Stacked Array (axis=0):
 [[1 2 3]
 [4 5 6]]
```

```
# Stack arrays along the second axis (axis=1)
stacked_array_axis1 = np.stack((array1, array2), axis=1)
print("Stacked Array (axis=1):\n", stacked_array_axis1)
Stacked Array (axis=1):
 [[1 4]
 [2 5]
```

Mathematical and Statistical Functions

1. **numpy.mean():** Computes the mean of array elements.

```
array = np.array([1, 2, 3, 4, 5])
mean_value = np.mean(array)
print(mean_value)
        3.0
```

2. **numpy.std():** Computes the standard deviation of array elements.

```
std_value = np.std(array)
print(std_value)
        1.4142135623730951
```

3. **numpy.sum():** Computes the sum of array elements.

```
sum_value = np.sum(array)
print(sum_value)
        15
```

4. numpy.prod(): Computes the product of array elements.

```
prod_value = np.prod(array)
print(prod_value)
```

5. numpy.min(): Finds the minimum value in an array.

```
min_value = np.min(array)
print(min_value)
```

6. numpy.max(): Finds the maximum value in an array.

```
max_value = np.max(array)
print(max_value)
```

7. numpy.cumsum(): Computes the cumulative sum of array elements.

```
cumsum_array = np.cumsum(array)
print(cumsum_array)
                [1  3  6 10 15]
```

8. numpy.cumprod(): Computes the cumulative product of array elements.

```
cumprod_array = np.cumprod(array)
print(cumprod_array)
        [ 1   2   6  24 120]
```

Logical and Comparison Functions

1. **numpy.all():** Returns True if all elements evaluate to True.

```
bool_array = np.array([True, True, False])
all_true = np.all(bool_array)
print(all_true)     FALSE
```

```
bool_array = np.array([1,3,5, 10,9])
all_true = np.all(bool_array)
print(all_true)        TRUE
```

```
bool_array = np.array([1,3,5, 0,9])
all_true = np.all(bool_array)
print(all_true)       FALSE
```

2. **numpy.any():** Returns True if any element evaluates to True.

```
bool_array = np.array([0,False,4])
any_true = np.any(bool_array)
print(any_true)       TRUE
```

3. **numpy.where():**

   The numpy.where() function in NumPy is a powerful and versatile tool used for conditional selection of elements from arrays. It can be used to filter arrays, apply conditions, and perform element-wise operations based on specified criteria.

```
numpy.where(condition, [x, y, ])
```

- **condition**: An array-like or scalar value that specifies the condition to evaluate.
- **x**: (Optional) An array-like or scalar value to use as the output when the condition is True.
- **y**: (Optional) An array-like or scalar value to use as the output when the condition is False.

Returns

- If x and y are provided, numpy.where() returns an array with elements selected from x or y, depending on the condition. (you can pass either x and y both or None)
- If only the condition is provided, numpy.where() returns a tuple of arrays (indices) where the condition is True.

```
import numpy as np

array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])

indices = np.where(array > 5)

print(indices)
```

```
(array([5, 6, 7, 8]),)
```

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

result = np.where(array > 5, 'T', -1)

print(result)
```

```
['-1' '-1' '-1' '-1' '-1' 'T' 'T' 'T' 'T']
```