NumPy: Array Creation, Manipulation, and Operations

NumPy is a powerful library for numerical computing in Python. It provides support for arrays, matrices, and a variety of mathematical functions to operate on these data structures.

1. Array Creation and Manipulation

In this section, we create arrays and manipulate their elements.

```
import numpy as np
# Create a numpy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print('Original Array:')
print(array)
# Replace all the elements in the third column with 1
array[:, 2] = 1
print('Modified array:')
print(array)
print(array[2:4, 2:])
# Create a numpy array of shape (4, 4) with values from 1 to 16
array = np.arange(1, 17).reshape((4, 4))
print('Original array:')
print(array)
# Replace the diagonal elements with 0
np.fill_diagonal(array, 0)
print('Modified array:')
print(array)
```

Output:

```
Original Array:
[[ 4 3 18 19 12]
[13 19 2 10 18]
[ 7 12 8 15 19]
[11 9 7 19 13]
[17 1 17 4 2]]
Modified array:
[[ 4 3 1 19 12]
[13 19 1 10 18]
[ 7 12 1 15 19]
[11 9 1 19 13]
[17 1 1 4 2]]
[[1 15 19]
[1 19 13]]
Original array:
[[ 1 2 3 4]
[ 5 6 7 8]
[ 9 10 11 12]
[13 14 15 16]]
Modified array:
[[ 0 2 3 4]
```

```
[ 5 0 7 8]
[ 9 10 0 12]
[13 14 15 0]]
```

2. Array Indexing and Slicing

This section demonstrates how to index and slice arrays.

```
# Create a numpy array of shape(6, 6) with values from 1 to 36
array = np.arange(1, 37).reshape((6, 6))
print('Original array:')
print(array)
# Extract the sub-array
sub_array = array[2:5, 1:4]
print('Sub-array:')
print(sub_array)
# Create a numpy array of shape (5, 5) with random integers
array = np.random.randint(1, 21, size=(5, 5))
print('Original Array:')
print(array)
# Extract the elements on the border
border_elements = np.concatenate((array[0, :], array[-1, :], array[1:-1, 0],
array[1:-1, -1]))
print(border_elements)
```

Output:

```
Original array:
[[ 1 2 3 4 5 6]
[ 7 8 9 10 11 12]
[13 14 15 16 17 18]
[19 20 21 22 23 24]
[25 26 27 28 29 30]
[31 32 33 34 35 36]]
Sub-array:
[[14 15 16]
[20 21 22]
[26 27 28]]
Original Array:
[[ 1 11 1 13 16]
[ 7 4 4 14 17]
[ 2 19 10 16 19]
[18 6 16 5 5]
[ 3 15 20 9 12]]
Border elements:
[ 1 11 1 13 16 3 15 20 9 12 7 2 18 17 19 5]
```

3. Array Operations

This section covers element-wise operations on arrays.

```
# Create two NumPy arrays of shape (3, 4) filled with random integers
```

```
array1 = np.random.randint(1, 11, size=(3, 4))
array2 = np.random.randint(1, 11, size=(3, 4))
print('Array 1:')
print(array1)
print('Array 2:')
print(array2)
# Perform element-wise operations
addition = array1 + array2
subtraction = array1 - array2
multiplication = array1 * array2
division = array1 / array2
print('Element-wise addition:')
print(addition)
print('Element-wise subtraction:')
print(subtraction)
print('Element-wise multiplication:')
print(multiplication)
print('Element-wise division:')
print(division)
# Create a NumPy array of shape (4, 4) with values from 1 to 16
array = np.arange(1, 17).reshape((4, 4))
print('Original array:')
print(array)
# Compute the row-wise and column-wise sum
row_sum = np.sum(array, axis=1)
column_sum = np.sum(array, axis=0)
print('Row-wise sum:')
print(row_sum)
print('Column-wise sum:')
print(column_sum)
```

Output:

```
Array 1:
[[ 3 4 6 10]
[ 1 10 1 5]
[ 5 9 4 8]]
Array 2:
[[10 1 10 5]
[ 4 8 5 10]
[10 1 2 8]]
Element-wise addition:
[[13 5 16 15]
[ 5 18 6 15]
[15 10 6 16]]
Element-wise subtraction:
[[-7 3 -4 5]
[-3 2 -4 -5]
[-5 8 2 0]]
Element-wise multiplication:
[[30 4 60 50]
[ 4 80 5 50]
[50 9 8 64]]
```

```
Element-wise division:
[[0.3 4. 0.6 2. ]
[0.25 1.25 0.2 0.5 ]
[0.5 9. 2. 1. ]]
Original array:
[[ 1 2 3 4]
[ 5 6 7 8]
[ 9 10 11 12]
[13 14 15 16]]
Row-wise sum:
[10 26 42 58]
Column-wise sum:
[28 32 36 40]
```

4. Statistical Operations

This section demonstrates how to compute statistical values.

```
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print('Original array:')
print(array)
# Compute the statistical values
mean = np.mean(array)
median = np.median(array)
std_dev = np.std(array)
variance = np.var(array)
print('Mean:', mean)
print('Median:', median)
print('Standard Deviation:', std_dev)
print('Variance:', variance)
# Create a NumPy array of shape (3, 3) with values from 1 to 9
array = np.arange(1, 10).reshape((3, 3))
print('Original array:')
print(array)
# Normalize the array
mean = np.mean(array)
std_dev = np.std(array)
normalized_array = (array - mean) / std_dev
print('Normalized array:')
print(normalized_array)
```

Output:

```
Original array:
[[ 2 4 4 3 1]
[ 3 6 16 10 10]
[13 20 12 18 9]
[ 1 17 12 9 13]
[ 5 8 17 7 16]]
Mean: 9.44
Median: 9.0
```

```
Standard Deviation: 5.671542999925152
Variance: 32.166399999999996
Original array:
[[1 2 3]
[4 5 6]
[7 8 9]]
Normalized array:
[[-1.54919334 -1.161895 -0.77459667]
[-0.38729833 0. 0.38729833]
[ 0.77459667 1.161895 1.54919334]]
```

5. Broadcasting

This section covers broadcasting in NumPy.

```
# Create a NumPy array of shape (3, 3) filled with random integers
array = np.random.randint(1, 11, size=(3, 3))
row_array = np.random.randint(1, 11, size=(3,))
print('Original array:')
print(array)
print('1D array:')
print(row_array)
# Add the 1D array to each row of the 2D array using broadcasting
result = array + row_array
print('Resulting array:')
print(result)
# Create a NumPy array of shape (4, 4) filled with random integers
array = np.random.randint(1, 11, size=(4, 4))
column_array = np.random.randint(1, 11, size=(4,))
print('Original array:')
print(array)
print('1D array:')
print(column_array)
# Subtract the 1D array from each column of the 2D array using broadcasting
result = array - column_array[:, np.newaxis]
print('Resulting array:')
print(result)
```

Output:

```
Original array:
[[10 3 10]
[10 9 2]
[ 6 9 8]]
1D array:
[3 5 4]
Resulting array:
[[13 8 14]
[13 14 6]
[ 9 14 12]]
Original array:
[[ 6 9 7 9]
```

```
[ 4 6 8 8]
[ 7 8 8 2]
[ 2 10 5 8]]
1D array:
[2 1 6 4]
Resulting array:
[[ 4 7 5 7]
[ 3 5 7 7]
[ 1 2 2 -4]
[-2 6 1 4]]
```

6. Linear Algebra

This section covers linear algebra operations using NumPy.

```
# Create a NumPy array of shape (3, 3) representing a matrix
matrix = np.random.randint(1, 11, size=(3, 3))
print('Original matrix:')
print(matrix)
# Compute the determinant
determinant = np.linalg.det(matrix)
print('Determinant:', determinant)
# Compute the inverse
inverse = np.linalg.inv(matrix)
print('Inverse:')
print(inverse)
# Compute the eigenvalues
eigenvalues = np.linalg.eigvals(matrix)
print('Eigenvalues:', eigenvalues)
# Create two NumPy arrays of shape (2, 3) and (3, 2)
array1 = np.random.randint(1, 11, size=(2, 3))
array2 = np.random.randint(1, 11, size=(3, 2))
print('Array 1:')
print(array1)
print('Array 2:')
print(array2)
# Perform matrix multiplication
result = np.dot(array1, array2)
print('Matrix multiplication result:')
print(result)
```

Output:

```
Original matrix:
[[ 3 9 8]
[ 1 10 6]
[ 7 1 6]]
Determinant: -66.00000000000003
Inverse:
[[-0.81818182 0.6969697 0.39393939]
[-0.54545455 0.57575758 0.15151515]
[ 1.04545455 -0.90909091 -0.31818182]]
```

```
Eigenvalues: [16.51814752 -1.11183881 3.59369129]
Array 1:
[[6 8 9]
[1 2 3]]
Array 2:
[[1 7]
[3 2]
[6 4]]
Matrix multiplication result:
[[84 94]
[25 23]]
```

7. Advanced Array Manipulation

This section demonstrates advanced techniques for manipulating arrays.

```
# Create a NumPy array of shape (3, 3) with values from 1 to 9
array = np.arange(1, 10).reshape((3, 3))
print('Original array:')
print(array)
# Reshape the array to shape (1, 9)
reshaped_array_1 = array.reshape((1, 9))
print('Reshaped array (1, 9):')
print(reshaped_array_1)
# Reshape the array to shape (9, 1)
reshaped_array_2 = reshaped_array_1.reshape((9, 1))
print('Reshaped array (9, 1):')
print(reshaped_array_2)
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print('Original array:')
print(array)
# Flatten the array
flattened_array = array.flatten()
print('Flattened array:')
print(flattened_array)
# Reshape the array back to (5, 5)
reshaped_array = flattened_array.reshape((5, 5))
print('Reshaped array:')
print(reshaped_array)
```

Output:

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

```
[3]
[4]
[5]
[6]
[7]
[8]
[9]]
Original array:
[[ 7 17 3 12 20]
[14 13 14 11 2]
[13 6 5 20 17]
[18 18 4 2 1]
[ 3 1 13 12 1]]
Flattened array:
[ 7 17 3 12 20 14 13 14 11 2 13 6 5 20 17 18 18 4 2 1 3 1 13 12
1]
Reshaped array:
[[ 7 17 3 12 20]
[14 13 14 11 2]
[13 6 5 20 17]
[18 18 4 2 1]
[ 3 1 13 12 1]]
```

8. Fancy Indexing and Boolean Indexing

This section covers advanced indexing techniques.

```
# Create a NumPy array of shape (5, 5) filled with random integers
array = np.random.randint(1, 21, size=(5, 5))
print('Original array:')
print(array)
# Use fancy indexing to extract the elements at the corners of the array
corners = array[[0, 0, -1, -1], [0, -1, 0, -1]]
print('Corner elements:')
print(corners)
# Create a NumPy array of shape (4, 4) filled with random integers
array = np.random.randint(1, 21, size=(4, 4))
print('Original array:')
print(array)
# Use boolean indexing to set all elements greater than 10 to 10
array[array > 10] = 10
print('Modified array:')
print(array)
```

Output:

```
Original array:
[[ 7 7 1 17 12]
[15 7 3 4 9]
[13 13 8 11 6]
[12 20 14 4 6]
[17 1 1 1 1]]
```

```
Corner elements:
[ 7 12 17 1]
Original array:
[[ 1 2 3 4]
[ 5 6 7 8]
[ 9 10 11 12]
[13 14 15 16]]
Modified array:
[[ 1 2 3 4]
[ 5 6 7 8]
[ 9 10 10 10]
[10 10 10 10]]
```

9. Structured Arrays

This section demonstrates how to create and manipulate structured arrays.

```
# Create a structured array with fields 'name', 'age', and 'weight'
data_type = [('name', 'U10'), ('age', 'i4'), ('weight', 'f4')]
data = np.array([('Alice', 25, 55.5), ('Bob', 30, 85.3), ('Charlie', 20,
65.2)], dtype=data_type)
print('Original array:')
print(data)
# Sort the array by age
sorted_data = np.sort(data, order='age')
print('Sorted array by age:')
print(sorted_data)
# Create a structured array with fields 'x' and 'y'
data_type = [('x', 'i4'), ('y', 'i4')]
data = np.array([(1, 2), (3, 4), (5, 6)], dtype=data_type)
print('Original array:')
print(data)
# Compute the Euclidean distance between each pair of points
distances = np.sqrt((data['x'][:, np.newaxis] - data['x']) ** 2 +
(data['y'][:, np.newaxis] - data['y']) ** 2)
print('Euclidean distances:')
print(distances)
```

Output:

```
Original array:
[('Alice', 25, 55.5) ('Bob', 30, 85.3) ('Charlie', 20, 65.2)]
Sorted array by age:
[('Charlie', 20, 65.2) ('Alice', 25, 55.5) ('Bob', 30, 85.3)]
Original array:
[(1, 2) (3, 4) (5, 6)]
Euclidean distances:
[[0. 2.82842712 4.47213595]
[2.82842712 0. 2.82842712]
[4.47213595 2.82842712 0. ]]
```

10. Masked Arrays

This section covers the use of masked arrays in NumPy.

```
import numpy.ma as ma
# Create a masked array of shape (4, 4) with random integers
array = np.random.randint(1, 21, size=(4, 4))
masked_array = ma.masked_greater(array, 10)
print('Original array:')
print(array)
print('Masked array:')
print(masked_array)
# Compute the sum of the unmasked elements
sum_unmasked = masked_array.sum()
print('Sum of unmasked elements:', sum_unmasked)
# Create a masked array of shape (3, 3) with random integers
array = np.random.randint(1, 21, size=(3, 3))
masked _array = ma.masked_array(array, mask=np.eye(3, dtype=bool))
print('Original array:')
print(array)
print('Masked array:')
print(masked_array)
# Replace the masked elements with the mean of the unmasked elements
mean_unmasked = masked_array.mean()
masked_array = masked_array.filled(mean_unmasked)
print('Modified masked array:')
print(masked_array)
```

Output:

```
Original array:
[[6 19 5 10]
[15 19 13 11]
[12 4 18 19]
[18 5 19 16]]
Masked array:
[[6 -- 5 10]
[-- -- -- --]
[-- 4 -- --]
[-- -- -- --]]
Sum of unmasked elements: 30
Original array:
[[ 4 16 2]
[ 12 15 20]
[ 14 1 16]]
Masked array:
[[-- 16 2]
[ 12 -- 20]
[ 14 1 --]]
Modified masked array:
[[10 16 2]
[12 10 20]
[14 1 10]]
```