NumPy

A note by

Shah Md. Arshad Rahman Ziban

Chapter 00: Basic Attributes and Operations

Declare (or import) system

```
import numpy as np
# Creating a 2D NumPy array
a = np.array([[1, 2, 3], [4, 5, 6]])
b = a.view()
```

Operation	Code	Output	Explanation
Creating a 2D NumPy Array	print(a)	# [[1 2 3] # [4 5 6]]	Creates a 2D array with two rows and three columns.
Number of Dimensions	print(a.ndim)	# 2	Returns the number of dimensions of the array
Shape of the Array	print(a.shape)	# (2,3)	Returns the shape of the array as a tuple (rows, columns).
Size of the Array	print(a.size)	# 6	Returns the total number of elements in the array.
Data Type of the Array Elements	print(a.dtype)	#int32 or int64 (may vary based on your system)	Returns the data type of the elements in the array.
Element Size in Bytes	print(a.itemsize)	4 or 8 (may vary based on your system)	Returns the size in bytes of each element in the array.
Bytes Consumed by the Array	print(a.nbytes)	#24 or 48 (may vary)	Returns the total number of bytes consumed by the elements
Transposed Array	print(a.T)	# [[1 4] # [2 5] # [3 6]]	Returns the transposed version of he array. Rows become columns and vice versa.
Base Object	print(b.base is a)	# True	Checks if the array b shares data with array a (i.e., they share the same base object).
Memory Layout Information	print(a.flags)	#Output will vary	Returns a dictionary-like object with memory layout information of the array.
Strides	print(a.strides)	#Output will vary	Returns a tuple of bytes to step in each dimension when traversing an array.
Real Part	print(a.real)	# [[1 2 3] # [4 5 6]]	Returns the real part of the elements in the array (same as original for real numbers).
Imaginary Part	print(a.imag)	# [[0 0 0] # [0 0 0]]	Returns the imaginary part of the elements in the array (zero for real numbers).
Flat Iterator	print(list(a.flat))	# [1, 2, 3, 4, 5, 6]	Returns a 1D iterator over the elements of the array.
Array as List	print(a.tolist())	# [[1, 2, 3], [4, 5, 6]]	Converts the array into a nested list.

Chapter 01: Create an Array

Operation	Code	Output	Explanation
Creating an Array of Zeros	zeros_arr = np.zeros((3, 3)) print(zeros_arr)	# [[0. 0. 0.] # [0. 0. 0.] # [0. 0. 0.]]	Creates a 3x3 array filled with zeros.
Creating an Array of Ones	ones_arr = np.ones((2, 4)) print(ones_arr)	# [[1. 1. 1. 1.] # [1. 1. 1. 1.]]	Creates a 2x4 array filled with ones.
Create an array of ones with a specific data type	array_int = np.ones((2, 3), dtype=int) print(array_int)	#[[1 1 1] #[1 1 1]]	Creates a 2x3 array of ones with ones with a specific data type.
Creating an Empty Array (Values are Not Initialized)	empty_arr = np.empty((2, 3)) print(empty_arr)	# Output varies	Creates a 2x3 array with uninitialized values (values are whatever is already in memory).
Creating an Array (1D) with a Range of Values	arange_arr = np.arange(0, 10, 2) print(arange_arr)	# [0 2 4 6 8]	Creates a 1D array with values from 0 to 8, with a step of 2.
Creating an Array (2D) with a Range of Values	array_2d = np.arange(1, 13).reshape(3, 4) print(array_2d)	# [[1 2 3 4] # [5 6 7 8] # [9 10 11 12]]	np.arange(1, 13) generates numbers from 1 to 12, and .reshape(3, 4) formats them into a 3x4 array.
Creating an Array with Evenly Spaced Values	linspace_arr = np.linspace(0, 1, 5) print(linspace_arr)	# [0. 0.25 0.5 0.75 1.]	Creates a 1D array with 5 evenly spaced values between 0 and 1.
Creating an Array with Random Values	random_arr = np.random.random((2, 3)) print(random_arr)	# Output will vary due to random values	Creates a 2x3 array with random values between 0 and 1.
Creating an Array with Random Integers	random_int_arr = np.random.randint(0, 10, (3, 3)) print(random_int_arr)	# Output will vary due to random values	Creates a 3x3 array with random integers between 0 and 9.
Creating an Identity Matrix	identity_matrix = np.eye(3) print(identity_matrix)	# [[1. 0. 0.] # [0. 1. 0.] # [0. 0. 1.]]	Creates a 3x3 identity matrix (1s on the diagonal, 0s elsewhere).

Indexing

Example Type	Code
1D Array	arr = np.array([1, 2, 3, 4, 5]) print(arr[0]) # Output: 1 print(arr[-1]) # Output: 5
2D Array	arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) print(arr_2d[1, 2]) # Output: 6 print(arr_2d[-2, -1]) # Output: 6

Boolean	arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
	# Creating a boolean array
	bool_idx = (arr > 5)
	print("Boolean array:\n", bool_idx)
	# Output:
	# Boolean array:
	# [[False False False]
	# [False False True]
	# [True True True]]
	
	# Using the boolean array to index the original array
	filtered_arr = arr[bool_idx]
	print("Elements greater than 5:", filtered_arr)
	# Output: Elements greater than 5: [6 7 8 9]

Fancy	arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
	# Using integer arrays to index
	rows = np.array([0, 1, 2])
	cols = np.array([2, 1, 0])
	fancy_indexed_arr = arr[rows, cols]
	print("Fancy indexed array:", fancy_indexed_arr)
	# Output: Fancy indexed array: [3 5 7]

slicing/subsetting

Example Type	Code
1D Array	arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) sliced = arr[2:6] print(sliced) # Output: [2 3 4 5]
2D Array	arr = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]]) sliced = arr[0:2, 0:2] print(sliced) # Output: # [[0 1] # [3 4]]
Using Step (1D)	arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) sliced = arr[1:9:2] print(sliced) # Output: [1 3 5 7]
Using Step (2D)	arr = np.array([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]]) sub_array = arr[1, ::2] print(sub_array) # Output: [5 7]
Negative Indexing	arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) sliced = arr[-5:] print(sliced) # Output: [5 6 7 8 9]
Modifying a Slice	arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]) arr[2:6] = [20, 30, 40, 50] print(arr) #Output: [0 1 20 30 40 50 6 7 8 9]
Slicing with Negative Step	arr = np.array([[1, 2, 3, 4],[5, 6, 7, 8],[9, 10, 11, 12]]) sub_array = arr[1, ::-1] print(sub_array) # Output: [8 7 6 5]
Modifying Slices	arr = np.array([[1, 2, 3],[4, 5, 6],[7, 8, 9]]) arr[:2, 1:3] = 99 print(arr) # Output: # [[1 99 99] # [4 99 99] # [7 8 9]]

Operation	Code	Description
Reshape	arr1 = np.array([1, 2, 3, 4, 5, 6]) reshaped_arr = arr1.reshape((2, 3)) print(reshaped_arr) #Output: #[[1 2 3] #[4 5 6]]	Reshapes the array without modifying the original array.
Resize (in-place)	arr1 = np.array([1, 2, 3, 4, 5, 6]) arr1.resize((2, 3)) print(arr1) # Output: # [[1 2 3] # [4 5 6]]	Changes the shape of the array in-place.

Resize (larger shape)	arr1 = np.array([1, 2, 3, 4, 5, 6])
	# refcheck=False allows resizing even if the array
	#is referenced elsewhere
	arr1.resize((3, 3), refcheck=False)
	print(arr1)
	# Output may vary
	# Example output:
	# [[1 2 3]
	# [4 5 6]
	# [1 2 3]]
		Resizes the array to a larger shape, potentially filling with repeated data.

Flatten	arr1 = np.array([[1, 2, 3], [4, 5, 6]]) flattened_arr = arr1.flatten() print(flattened_arr) # Output: # [1 2 3 4 5 6]	Returns a new 1D array that is a copy of the original array's elements.
Ravel	arr1 = np.array([[1, 2, 3], [4, 5, 6]]) raveled_arr = arr1.ravel() print(raveled_arr) # Output: # [1 2 3 4 5 6]	Returns a 1D array that is a view of the original array whenever possible. Modifications to the raveled array can affect the original array if a view is returned.

Chapter 04: Array Stacking

Operation	Code	Description

Horizontal Stack (hstack)	`import numpy as np`
	`a = np.array([1, 2, 3])`
	`b = np.array([4, 5, 6])`
	`c = np.hstack((a, b))`
	`print(c)`
	`# Output:`
	`[1 2 3 4 5 6]`
		Stacks arrays in sequence horizontally (column-wise).

Vertical Stack (vstack)	`import numpy as np`
	`a = np.array([1, 2, 3])`
	`b = np.array([4, 5, 6])`
	`c = np.vstack((a, b))`
	`print(c)`
	`# Output:`
	`# [[1 2 3]`
	`#  [4 5 6]]`
		Stacks arrays in sequence vertically (row-wise).

Column Stack	`import numpy as np`
	`a = np.array([1, 2, 3])`
	`b = np.array([4, 5, 6])`
	`c = np.column_stack((a, b))`
	`print(c)`
	`# Output:`
	`# [[1 4]`
	`#  [2 5]`
	`#  [3 6]]`
		Stacks 1-D arrays as columns into a 2-D array. For 2-D arrays, behaves like hstack.

Row Stack	`import numpy as np`
	`a = np.array([1, 2, 3])`
	`b = np.array([4, 5, 6])`
	`c = np.row_stack((a, b))`
	`print(c)`
	`# Output:`
	`# [[1 2 3]`
	`#  [4 5 6]]`
		Equivalent to vstack for 1-D arrays, stacking them along a new row axis.

Depth Stack (dstack)	`import numpy as np`
	`a = np.array([[1, 2, 3], [4, 5, 6]])`
	`b = np.array([[7, 8, 9], [10, 11, 12]])`
	`c = np.dstack((a, b))`
	`print(c)`
	`# Output:`
	`# [[[ 1  7]`
	`#  [ 2  8]`
	`#  [ 3  9]]`
	`#`
	`#  [[ 4 10]`
	`#  [ 5 11]`
	`#  [ 6 12]]]`
		Stacks arrays in sequence depth-wise (along the third dimension).

General Stack (stack)	`import numpy as np`
	
	`a = np.array([1, 2, 3])`
	`b = np.array([4, 5, 6])`
	`c = np.stack((a, b), axis=0)`
	`print(c)`
	`# Output:`
	`# [[1 2 3]`
	`#  [4 5 6]]`
	
	`c = np.stack((a, b), axis=1)`
	`print(c)`
	`# Output:`
	`# [[1 4]`
	`#  [2 5]`
	`#  [3 6]]`
	
	`import numpy as np`
	
	`a = np.array([[1, 2, 3], [4, 5, 6]])`
	`b = np.array([[7, 8, 9], [10, 11, 12]])`
	
	`c = np.stack((a, b), axis=2)`
	`print(c)`
	`# Output:`
	`# [[[ 1  7]`
	`#  [ 2  8]`
	`#  [ 3  9]]`
	`#`
	`#  [[ 4 10]`
	`#  [ 5 11]`
	`#  [ 6 12]]]`
		Joins a sequence of arrays along a new axis. The axis parameter specifies the index of the new axis.

Chapter 05: Adding and removing Elements

Adding

Operation	Code	Description

Appending Elements	`import numpy as np`
	
	`arr = np.array([1, 2, 3])`
	`values_to_append = [4, 5, 6]`
	`appended_arr = np.append(arr, values_to_append)`
	
	`print(appended_arr)`
	`# Output: [1 2 3 4 5 6]`
		Appends values to the end of an array. Does not modify the original array but returns a new array with the appended values. Can be used with 1D and multi-dimensional arrays.

	`import numpy as np`
	
	`arr = np.array([[1, 2, 3], [4, 5, 6]])`
	`values_to_append = np.array([[7, 8, 9], [10, 11, 12]])`
	
	`appended_arr = np.append(arr, values_to_append, axis=0)`
	`print(appended_arr)`
	`# Output:`
	`# [[ 1  2  3]`
	`#  [ 4  5  6]`
	`#  [ 7  8  9]`
	`#  [10 11 12]]`
	
	`values_to_append = np.array([[13], [14]])`
	`appended_arr = np.append(arr, values_to_append, axis=1)`
	`print(appended_arr)`
	`# Output:`
	`# [[ 1  2  3 13]`
	`#  [ 4  5  6 14]]`	

Inserting Elements	`import numpy as np`
	
	`arr = np.array([1, 2, 3, 4, 5])`
	`values_to_insert = [9, 8, 7]`
	`inserted_arr = np.insert(arr, 2, values_to_insert)`
	
	`print(inserted_arr)`
	`# Output: [1 2 9 8 7 3 4 5]`
		Inserts values into an array before the specified index. Does not modify the original array but returns a new array with the inserted values. Can be used with 1D and multi-dimensional arrays. The index specifies the position in the array where the new values]should be inserted.

	`import numpy as np`
	
	`arr = np.array([[1, 2, 3], [4, 5, 6]])`
	`values_to_insert = np.array([[7, 8, 9]])`
	
	`# Inserting values into the array before index 1 along axis 0`
	`inserted_arr = np.insert(arr, 1, values_to_insert, axis=0)`
	`print(inserted_arr)`
	`# Output:`
	`# [[1 2 3]`
	`#  [7 8 9]`
	`#  [4 5 6]]`
	
	`# Inserting values into the array before index 2 along axis 1`
	`values_to_insert = np.array([[10], [11]])`
	`inserted_arr = np.insert(arr, 2, values_to_insert, axis=1)`
	`print(inserted_arr)`
	`# Output:`
	`# [[ 1  2 10  3]`
	`#  [ 4  5 11  6]]`	

Removing

Operation	Code	Description
Deleting Rows	import numpy as np arr_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) deleted_element_arr_2d = np.delete(arr_2d, 4) print(deleted_element_arr_2d) # Output: # [1 2 3 4 6 7 8 9] arr_1d = np.array([1, 2, 3, 4, 5, 6, 7, 8]) deleted_elements_arr_1d = np.delete(arr_1d, [2, 4]) print(deleted_elements_arr_1d) # Output: # [1 2 4 5 7 8]	In the 2D array example, a specific element was deleted after flattening the array. In the 1D array example, specific elements were removed based on their indices.

	import numpy as np
	arr_2d = np.array([[1, 2, 3],
	[4, 5, 6],
	[7, 8, 9],
	[10, 11, 12]])
	
	# Deleting the row at index 3
	deleted_row_arr = np.delete(arr_2d, 3, axis=0)
	print(deleted_row_arr)
	
	# Output:
	# [[1 2 3]
	# [4 5 6]
	# [7 8 9]]
		The original array had 4 rows. The row at index 3 (4th row) [10, 11, 12] was deleted. The resulting array now has 3 rows.

Deleting Columns	import numpy as np
	
	arr_2d = np.array([[1, 2, 3, 4],
	[5, 6, 7, 8],
	[9, 10, 11, 12]])
	
	# Deleting the column at index 3 (4th column)
	deleted_col_arr = np.delete(arr_2d, 3, axis=1)
	
	print(deleted_col_arr)
	
	# Output:
	# [[1 2 3]
	# [5 6 7]
	# [9 10 11]]
		This deletes the column at index 3 in the original array. The result is an array with the last column removed.

Chapter 06: Importing/exporting

Operation	Code	Description

Loading from a text file	`import numpy as np`
	
	`# Loading data from a text file`
	`data = np.loadtxt('file.txt')`
	
	`# Printing the loaded data`
	`print(data)`
		Loads data from a text file (file.txt) and prints it. Assumes the file contains space -separated values.

Loading from a CSV file	`import numpy as np`
	
	`# Loading data from a CSV file`
	`data = np.genfromtxt('file.csv', delimiter=',')`
	
	`# Printing the loaded data`
	`print(data)`
		Loads data from a CSV file (file.csv) using a comma as a delimiter and prints it.
		
		The delimiter=',' parameter specifies that the values in the file are separated by commas.

Writing to a text file	`import numpy as np`
	
	`# Data to be written to the text file`
	`arr = np.array([[1, 2, 3], [4, 5, 6]])`
	
	`np.savetxt('file.txt', arr, delimiter=' ')`
		Writes a NumPy array to a text file (file.txt) with space as the delimiter.

Writing to a CSV file	`import numpy as np`
	
	`# Data to be written to the CSV file`
	`arr = np.array([[1, 2, 3], [4, 5, 6]])`
	
	`np.savetxt('file.csv', arr, delimiter=',')`
		Writes a NumPy array to a CSV file (file.csv) with a comma as the delimiter.

Operation	Code	Description

Concatenating Arrays Along Rows	`import numpy as np`
	
	`arr1 = np.array([[1, 2], [3, 4]])`
	`arr2 = np.array([[5, 6], [7, 8]])`
	`result = np.concatenate((arr1, arr2), axis=0)`
	`print(result)`
	
	`# Output:`
	`# [[1 2]`
	`#  [3 4]`
	`#  [5 6]`
	`#  [7 8]]`
		Combines arrays along rows (vertically)

Concatenating Arrays Along Columns	`import numpy as np`
	
	`arr1 = np.array([[1, 2], [3, 4]])`
	`arr2 = np.array([[5, 6], [7, 8]])`
	
	`result = np.concatenate((arr1, arr2), axis=1)`
	`print(result)`
	
	`# Output:`
	`# [[1 2 5 6]`
	`#  [3 4 7 8]]`
		Combines arrays along columns (horizontally)

Splitting Array into Sub-arrays	`import numpy as np`
	
	`arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])`
	
	`result = np.split(arr, 3)`
	`print(result)`
	
	`# Output:`
	`# [array([1, 2, 3]), array([4, 5, 6]), array([7, 8, 9])]`
		Splits a 1D array into 3 equal-sized sub-arrays

Splitting Horizontally at Index	`import numpy as np`
	`arr = np.array([[1, 2, 3, 4, 5],`
	`[6, 7, 8, 9, 10]])`
	
	`result = np.hsplit(arr, 5)`
	`print(result)`
	
	`# Output:`
	`# [array([[1],`
	`#  [6]]),`
	`#  array([[2],`
	`#  [7]]),`
	`#  array([[3],`
	`#  [8]]),`
	`#  array([[4],`
	`#  [9]]),`
	`#  array([[ 5],`
	`# [10]])]
		Splits a 2D array horizontally into sub-arrays at columns

Operation	Code	Description
Elementwise Addition	arr1 = np.array([4, 5, 6]) arr2 = np.array([1, 2, 3]) result = np.add(arr1, arr2) print(result) # Output: [5 7 9]	Adds elements of two arrays elementwise
Elementwise Subtraction	result = np.subtract(arr1, arr2) print(result) # Output: [3 3 3]	Subtracts elements of two arrays elementwise
Elementwise Multiplication	result = np.multiply(arr1, arr2) print(result) # Output: [4 10 18]	Multiplies elements of two arrays elementwise
Elementwise Division	result = np.divide(arr1, arr2) print(result) # Output: [4. 2.5 2.]	Divides elements of two arrays elementwise
Elementwise Power	result = np.power(arr1, arr2) print(result) # Output: [4 25 216]	Raises elements of one array to powers of another
Square Root of Elements	arr = np.array([1, 4, 9, 16]) result = np.sqrt(arr) print(result) # Output: [1. 2. 3. 4.]	Finds square root of each element
Sine of Elements	arr = np.array([0, np.pi / 2, np.pi]) result = np.sin(arr) print(result) # Output: [0. 1. 0.] 0, π/2 (90 degrees), and π (180 degrees).	Computes sine of each element
Natural Log of Elements	arr = np.array([1, np.e, np.e**2]) result = np.log(arr) print(result) # Output: [0. 1. 2.] log(1) is 0. log(e) is 1 log(e^2) is 2	Computes natural log of each element
Absolute Value of Elements	arr = np.array([-1, -2, 3, -4]) result = np.abs(arr) print(result) # Output: [1 2 3 4]	Computes absolute value of each element
Round Up to Nearest Integer	arr = np.array([1.1, 2.5, 3.7]) result = np.ceil(arr) print(result) # Output: [2. 3. 4.]	Rounds up each element to nearest integer
Round Down to Nearest Integer	result = np.floor(arr) print(result) # Output: [1. 2. 3.]	Rounds down each element to nearest integer
Round to Nearest Integer	arr = np.array([1.1, 2.5, 3.7]) result = np.round(arr) print(result) # Output: [1. 2. 4.] 1.1 is closer to 1 than to 2, so it rounds to 1. When a number is exactly halfway between two integers (e.g., 2.5), it rounds to the nearest even integer. This is known as "round half to even".	Rounds each element to nearest integer

Operation	Code	Description
Add a scalar to each element	arr = np.array([1, 2, 3]) result = np.add(arr, 1) print(result) # Output: [2 3 4]	Adds a scalar to each element of the array
Subtract a scalar from each element	arr = np.array([4, 5, 6]) result = np.subtract(arr, 2) print(result) # Output: [2 3 4]	Subtracts a scalar from each element of the array
Multiply each element by a scalar	arr = np.array([1, 2, 3]) result = np.multiply(arr, 3) print(result) # Output: [3 6 9]	Multiplies each element by a scalar
Divide each element by a scalar	arr = np.array([4, 5, 6]) result = np.divide(arr, 4) print(result) # Output: [1. 1.25 1.5]	Divides each element by a scalar
Raise each element to the specified power	arr = np.array([1, 2, 3]) result = np.power(arr, 5) print(result) # Output: [1 32 243]	Raises each element to the specified power

Operation	Code	Description
Mean along specific axis	arr = np.array([[1, 2, 3], [4, 5, 6]]) mean = np.mean(arr, axis=0) print(mean) # Output: [2.5 3.5 4.5]	Computes the mean along the specified axis (0 for columns)
Sum of elements	arr = np.array([1, 2, 3, 4]) total = arr.sum() print(total) # Output: 10	Computes the sum of all elements in the array
Minimum value	arr = np.array([1, 2, 3, 4]) min_val = arr.min() print(min_val) # Output: 1	Finds the minimum value in the array
Maximum value along specific axis	arr = np.array([[1, 2, 3], [4, 5, 6]]) max_val = arr.max(axis=0) print(max_val) # Output: [4 5 6]	Computes the maximum value along the specified axis (0 for columns)
Variance of array	arr = np.array([1, 2, 3, 4]) variance = np.var(arr) print(variance) # Output: 1.25	Computes the variance of the array
Standard deviation along specific axis	arr = np.array([[1, 2, 3], [4, 5, 6]]) std_dev = np.std(arr, axis=1) print(std_dev) # Output: [0.81649658 0.81649658]	Computes the standard deviation along the specified axis (1 for rows)
Correlation coefficient of array	arr = np.array([[1, 2, 3], [4, 5, 6]]) corr_coef = np.corrcoef(arr) print(corr_coef) # Output: # [[1. 1.] # [1. 1.]]	Computes the correlation coefficient matrix of the array