



ENGR 1330

**Computational Thinking with
Data Science**

NumPy



Topic Outline



- NumPy library
 - ✓ Data representation: Arrays - vectors and matrices
 - ✓ Data operations: Mathematical operations, indexing, selection, and copying



Objectives



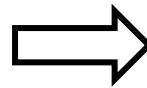
- To be able to represent data in different forms via the NumPy library
- To be able to access data within a NumPy array
- To be able to perform basic mathematical functions on the NumPy arrays



Computational Thinking Concepts

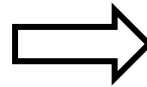


NumPy arrays:
Vectors and matrices



Data representation

Data interpretation,
manipulation, and
analysis of NumPy arrays



Decomposition

Algorithm



NumPy in Python



NumPy



- NumPy: Numerical Python
- Foundational library for scientific computing
- All data science libraries rely on NumPy as one of their building blocks



Features of NumPy



- Features:
 - ✓ Provides a fast and efficient multi-dimensional array object called 'ndarray' (n-dimensional array) – NumPy arrays
 - ✓ Functions for performing computations with arrays and mathematical operations between arrays
 - ✓ Linear algebra operations and random number generation



Multi-dimensional Array

- 1D

10	15	7	2
----	----	---	---

- 2D

4	20	13	9
5	9	12	0
10	15	7	2

- 3D

...



NumPy Arrays

- NumPy arrays can be 1-dimensional (1D) or 2-dimensional (2D)
- Creating a 1D array: Vector

In [1]: `import numpy as np` → Importing NumPy library

In [2]: `list1 = [1,2,3,4,5,6,7,8,9]`
`np.array(list1)` (Demo)

Function to create a NumPy



NumPy Arrays

- Creating a 2D array: Matrix

```
In [5]: list2 = [[1,2,3],[4,5,6],[7,8,9]]  
np.array(list2)
```

Function to create a NumPy
array

- What will be the shape of the above 2D NumPy array?

(Demo)



Numpy Arrays

```
# Create a length-10 integer array filled with zeros  
np.zeros(10, dtype=int)
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

```
# Create a 3x5 floating-point array filled with ones  
np.ones((3, 5), dtype=float)
```

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

```
# Create a 3x5 array filled with 3.14  
np.full((3, 5), 3.14)
```

```
array([[ 3.14,  3.14,  3.14,  3.14,  3.14],  
       [ 3.14,  3.14,  3.14,  3.14,  3.14],  
       [ 3.14,  3.14,  3.14,  3.14,  3.14]])
```



Numpy Arrays

```
# Create an array filled with a linear sequence  
# Starting at 0, ending at 20, stepping by 2  
# (this is similar to the built-in range() function)  
np.arange(0, 20, 2)
```

```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

```
# Create an array of five values evenly spaced between 0 and 1  
np.linspace(0, 1, 5)
```

```
array([ 0.   ,  0.25,  0.5  ,  0.75,  1.   ])
```

```
# Create a 3x3 array of uniformly distributed  
# random values between 0 and 1  
np.random.random((3, 3))
```

```
array([[ 0.99844933,  0.52183819,  0.22421193],  
       [ 0.08007488,  0.45429293,  0.20941444],  
       [ 0.14360941,  0.96910973,  0.946117   ]])
```



Numpy Arrays

```
import numpy as np
```

```
np.random.seed(0) # seed for reproducibility
```

```
x1 = np.random.randint(10, size=6) # One-dimensional array
```

```
x2 = np.random.randint(10, size=(3, 4)) # Two-dimensional array
```

```
x3 = np.random.randint(10, size=(3, 4, 5)) # Three-dimensional array
```



NumPy Arrays

- Other functions to create NumPy arrays easily
 - ✓ `arange()`: Returns evenly spaced array elements
 - ✓ `linspace()`: Returns evenly spaced array elements
 - ✓ `zeros()`: Returns an array of zeros
 - ✓ `ones()`: Returns an array of ones
 - ✓ `random.randint()`: Returns random integers

(Demo)



Arrays: Indexing

- Indexing: Accessing elements of array

```
In [10]: x2
```

```
Out [10]: array([[3, 5, 2, 4],  
                [7, 6, 8, 8],  
                [1, 6, 7, 7]])
```

```
In [11]: x2[0, 0]
```

```
Out [11]: 3
```

```
In [12]: x2[2, 0]
```

```
Out [12]: 1
```

```
In [13]: x2[2, -1]
```

```
Out [13]: 7
```

Numpy array is:

- Mutable
- Unique data type
- Assigning to other datatype will be truncated silently.

```
x1[0] = 3.14159 # this will be truncated!  
x1
```

```
array([3, 0, 3, 3, 7, 9])
```



Arrays: Slicing & Striding

```
x2
```

```
array([[12,  5,  2,  4],  
       [ 7,  6,  8,  8],  
       [ 1,  6,  7,  7]])
```

1D: `x[start:stop:step]`

2D: `x[start1:stop1:step1, start2:stop2:step2]`

```
x2[:2, :3] # two rows, three columns
```

```
array([[12,  5,  2],  
       [ 7,  6,  8]])
```

```
x2[:3, ::2] # all rows, every other column
```

```
array([[12,  2],  
       [ 7,  8],  
       [ 1,  7]])
```




Array: Comparison



Operator	Meaning
==	Equal
<	Less than
>	Greater than
!=	Different
<=	Less than or equal
>=	Greater than or equal

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

```
x < 3 # Less than
```

```
array([ True,  True, False, False, False], dtype=bool)
```

```
x > 3 # greater than
```

```
array([False, False, False,  True,  True], dtype=bool)
```

```
x <= 3 # Less than or equal
```

```
array([ True,  True,  True, False, False], dtype=bool)
```

```
x >= 3 # greater than or equal
```

```
array([False, False,  True,  True,  True], dtype=bool)
```

```
x != 3 # not equal
```

```
array([ True,  True, False,  True,  True], dtype=bool)
```



Array: Counting



```
print(x)
```

```
[[5 0 3 3]  
 [7 9 3 5]  
 [2 4 7 6]]
```

```
# how many values less than 6 in each row?  
np.sum(x < 6, axis=1)
```

```
array([4, 2, 2])
```

Note: axis=0 is column

Count elements below six

```
# how many values less than 6?  
np.count_nonzero(x < 6)
```

```
8
```

```
np.sum(x < 6)
```

```
8
```



Array: sorting



1D array

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

```
array([1, 2, 3, 4, 5])
```

```
x.sort()  
print(x)
```

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



Array: sorting

2D array

```
rand = np.random.RandomState(42)
X = rand.randint(0, 10, (4, 6))
print(X)
```

```
[[6 3 7 4 6 9]
 [2 6 7 4 3 7]
 [7 2 5 4 1 7]
 [5 1 4 0 9 5]]
```

```
# sort each column of X
np.sort(X, axis=0)
```

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

```
# sort each row of X
np.sort(X, axis=1)
```

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



Discussion Exercise



```
mat1 = 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]])
```

- How would you index and slice the elements within the red-dashed box above from the matrix named 'mat1'?

```
In [19]: mat1[2:,1:4]
```

(Demo)



Arrays: Functions

- Functions to do basic operations on NumPy arrays
- ✓ `np.append(arr, values, axis=None)`: Returns appended array

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

(Demo)



Arrays: Basic Operations

- ✓ `np.min()`: Returns minimum value in an array
- ✓ `np.max()`: Returns maximum value in an array

```
In [2]: np.min([100, 20, 10, 1])
```

```
Out[2]: 1
```

```
In [3]: np.max([100, 20, 10, 1])
```

```
Out[3]: 100
```

```
In [4]: np.sum([100, 20, 10, 1])
```

```
Out[4]: 131
```

```
In [5]: np.mean([100, 20, 10, 1])
```

```
Out[5]: 32.75
```

✓ `mean()`: Returns mean value of an array

✓ `sum()`: Summing the array elements

(Demo)



Arrays: Mathematical Operations



- Functions to do mathematical operations on NumPy arrays
 - ✓ `sqrt(.)`: Returns square root of array elements
 - ✓ `exp(.)`: Returns exponential of array elements
 - ✓ `sin(.)`: Returns trigonometric sine of array elements
 - ✓ `cos(.)`: Returns trigonometric cosine of array elements
 - ✓ `log(.)`: Returns natural logarithm of array elements

(Demo)



Summary



- Concepts of representing data in the form of NumPy arrays are covered
- Concepts of interpreting, manipulating, and analyzing data within NumPy arrays are covered