



# Overview

- 1 What is Python?
  - Origins, Features, Framework for Scientific Computing
- 2 Program Development with Python
  - Working with the Terminal
  - Integrated Development Environments
- 3 Data Types, Variables, Arithmetic Expressions, Program Control, and Functions
- 4 First Program (Evaluate and Plot Sigmoid Function)
- 5 Builtin Collections (Lists, Dictionaries, and Sets)
- 6 Numerical Python (NumPy)
- 7 Tabular Data and Dataset Transformation (Pandas)
- 8 Spatial Data and Dataset Transformation (GeoPandas)

Part 5



# Numerical Python (NumPy)

## Introduction to NumPy

**Numerical Python** (NumPy) is an open source Python library that contains computational support for n-dimensional array objects, along with mathematical methods to operate on them.

### Key Features:

- Create 0-d, 1-d and 2-d arrays. 3-d blocks.
- Operations on array elements (e.g., min, max, sort).
- Operations on arrays (e.g., reshape, stack).
- Compute matrix properties. Solve matrix equations.

### Installation

```
prompt >> pip3 install numpy
```



# Working with NumPy

## Example 1: Create 0-d, 1-d, and 2-d arrays ...

```
a = np.array(101);    # <-- create 0-d array.
print (a)

a = np.array( [ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] ); # <-- create 1-d array of
print (a)

a = np.array( ["The", "Brown", "Fox"] ); # <-- array of character strings.
a = np.append(a, "!")

for i in a:                          # <-- loop over array indices ...
    print(i)
```

## Output:

```
101
[ 1  2  3  4  5  6  7  8  9 10]
The
Brown
Fox
!
```

# Working with NumPy

**Example 2:** Print each array element and its index ...

```
# Create array of character strings ...  
  
a = np.array( ["The", "Brown", "Fox", "!"] );  
  
for i,e in enumerate(a):  
    print("--- Index: {}, was: {}".format(i, e))
```

**Output:**

```
--- Index: 0, was: The  
--- Index: 1, was: Quick  
--- Index: 2, was: Brown  
--- Index: 3, was: Fox  
--- Index: 4, was: !
```

# Working with NumPy

## Example 3: Sort array elements ...

```
# Sort array of floating point numbers ...

a = np.array( [ 2.3, 1.0, 4.5, -13.0, 100.0, 43, -15.0, 0.0 ] )
print(a);
print(np.sort(a));

# Sort array of state abbreviations ...

a = np.array( ["MD", "CA", "RI", "UT", "LA", "AL", "WA", "OR", "CO"] )
print(a);
print(np.sort(a))
```

## Output:

```
--- Sort array of floating-point numbers ...
[ 2.3  1.  4.5 -13. 100.  43. -15.  0. ]
[-15. -13.  0.  1.  2.3  4.5 43. 100. ]
--- Sort array of state abbreviations ...
['MD' 'CA' 'RI' 'UT' 'LA' 'AL' 'WA' 'OR' 'CO']
['AL' 'CA' 'CO' 'LA' 'MD' 'OR' 'RI' 'UT' 'WA']
```



# Working with NumPy

## Example 4: Create two-dimensional array ...

```
c = np.array( [ ( 0, 1, 4, 3, 2), ( 3, 4, 5, 6, 7),
                ( 6, 7, 8, 9,10), ( 9,10,11,12,13) ] );

PrintMatrix("C", c);          # <-- print formatted matrix ....

print("   Min: {}".format(np.min(c)))
print("   Max: {}".format(np.max(c)))
print(" Average: {}".format(np.average(c)))
print(" Max array index: {}".format(np.argmax(c)))
```

## Output:

```
Matrix: C
  0.000   1.000   4.000   3.000   2.000
  3.000   4.000   5.000   6.000   7.000
  6.000   7.000   8.000   9.000  10.000
  9.000  10.000  11.000  12.000  13.000

Min: 0           Average: 6.5
Max: 13         Max array index: 19
```



# Working with NumPy

## Example 6: Reshape 1-d array → 2-d matrix ...

```
d1 = np.arange(20);      # <-- create 1-d test array ...
print(d1);

d1 = d1.reshape(4,5);   # <-- reshape to (4x5) array ...
PrintMatrix("(4x5)", d1 );
```

### Output:

--- 1-d test array:

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

--- Reshape to (4x5) matrix ...

Matrix: (4x5)

```
0.000    1.000    2.000    3.000    4.000
5.000    6.000    7.000    8.000    9.000
10.000   11.000   12.000   13.000   14.000
15.000   16.000   17.000   18.000   19.000
```

# Working with NumPy

## Example 7: Create horizontal and vertical array stacks ...

```

d1 = np.array( [ ( 0, 1), ( 3, 4) ] ); # <-- create test arrays ...
d2 = np.array( [ ( 5, 6), ( 7, 8) ] );

PrintMatrix("d1", d1 ); PrintMatrix("d2", d2 );

h1 = np.hstack((d1, d2)); # <-- create horizontal stack ...
PrintMatrix( "np.hstack(d1, d2)", h1 );
h2 = np.vstack((d1, d2)); # <-- create vertical stack ...
PrintMatrix( "np.vstack(d1, d2)", h2 );

```

### Output:

```

Matrix: d1
0.000  1.000
3.000  4.000

Matrix: np.hstack(d1, d2)
0.000  1.000  5.000  6.000
3.000  4.000  7.000  8.000

Matrix: d2
5.000  6.000
7.000  8.000

Matrix: np.vstack(d1, d2)
0.000  1.000
3.000  4.000
5.000  6.000
7.000  8.000

```

# Working with NumPy

## Example 8: Exercise np.zeros() and np.eye() ...

```
matrix02 = np.zeros(shape=(4, 4)) # <-- create (4x4) array of zeros.
PrintMatrix("matrix02", matrix02 );

matrix03 = np.eye(4, dtype = float) # <-- create (4x4) identity matrix.
PrintMatrix("matrix03", matrix03 );
```

### Output:

```
Matrix: matrix02
0.000    0.000    0.000    0.000
0.000    0.000    0.000    0.000
0.000    0.000    0.000    0.000
0.000    0.000    0.000    0.000
```

```
Matrix: matrix03
1.000    0.000    0.000    0.000
0.000    1.000    0.000    0.000
0.000    0.000    1.000    0.000
0.000    0.000    0.000    1.000
```

# Working with NumPy

## Example 9: Reshape arrays of random numbers

```

matrix06 = np.random.random((20,1)); # <-- create (20x1) array
PrintMatrix("matrix06", matrix06 ); # of random numbers.

PrintMatrix ( "matrix06 (reshaped)", # <-- reshape to (10x2).
              matrix06.reshape(10,2) )

```

## Abbreviated Output:

```

--- Original (20x1) matrix      --- Reshape to (10x2) matrix ...

Matrix: matrix06                Matrix: matrix06 (reshaped)
 0.326                          0.326   0.459
 0.459                          0.545   0.419
 0.545                          0.537   0.632
 .....                          .....
 0.803                          .....
 0.014                          0.165   0.803
 0.291                          0.014   0.291

```







# Working with NumPy

**Example 11:** Solve family of matrix equations:

$$\begin{bmatrix} 3 & -6 & 7 \\ 9 & 0 & -5 \\ 5 & -8 & 6 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ -4 \end{bmatrix} \quad (4)$$

## Part I: Theoretical Considerations:

- A unique solution  $\{X\} = [A^{-1}] \cdot \{B\}$  exists when  $[A^{-1}]$  exists (i.e.,  $\det[A] \neq 0$ ). Expanding  $\det(A)$  about the first row gives:

$$\begin{aligned} \det(A) &= 3\det \begin{bmatrix} 0 & -5 \\ -8 & 6 \end{bmatrix} + 6\det \begin{bmatrix} 9 & -5 \\ 5 & 6 \end{bmatrix} + 7\det \begin{bmatrix} 9 & 0 \\ 5 & -8 \end{bmatrix}, \\ &= 3(0 - 40) + 6(54 + 25) + 7(-72 - 0) = -150. \end{aligned} \quad (5)$$

# Working with NumPy

## Part II: Program Source Code:

```
1 # =====
2 # TestMatrixEquations01.py: Compute solution to matrix equations.
3 #
4 # Written by: Mark Austin                               November 2022
5 # =====
6
7 import numpy as np
8 from numpy.linalg import matrix_rank
9
10 # Function to print two-dimensional matrices ...
11
12 def PrintMatrix(name, a):
13     print("Matrix: {:s} ".format(name) );
14     for row in a:
15         for col in row:
16             print("{:8.3f}".format(col), end=" ")
17         print("")
18
19 # main method ...
20
21 def main():
22     print("--- Enter TestMatrixEquations01.main()      ... ");
23     print("--- ===== ... ");
24
25     print("--- Part 1: Create test matrices ... ");
```

# Working with NumPy

## Part II: Program Source Code: (Continued) ...

```

27     A = np.array( [ [ 3, -6,  7],
28                   [ 9,  0, -5],
29                   [ 5, -8,  6] ] );
30     PrintMatrix("A", A);
31
32     B = np.array([ [3], [3], [-4] ]);
33     PrintMatrix("B", B);
34
35     print("--- Part 2: Check properties of matrix A ... ");
36
37     rank = matrix_rank(A)
38     det  = np.linalg.det(A)
39
40     print("--- Matrix A: rank = {:f}, det = {:f} ...".format(rank, det) );
41
42     print("--- Part 3: Solve A.x = B ... ");
43
44     x = np.linalg.solve(A, B)
45     PrintMatrix("x", x);
46
47     print("--- ===== ... ");
48     print("--- Leave TestMatrixEquations01.main() ... ");
49
50     # call the main method ...
51
52     main()

```

# Working with NumPy

## Part III: Program Output:

```
# Part 1: Create test matrices ...
```

```
Matrix: A
```

```
 3.000  -6.000   7.000
 9.000   0.000  -5.000
 5.000  -8.000   6.000
```

```
Matrix: B
```

```
 3.000
 3.000
-4.000
```

```
# Part 2: Check properties of matrix A ...
```

```
Matrix A: rank = 3.000000, det = -150.000000 ...
```

```
# Part 3: Solve A.x = B ...
```

```
Matrix: x
```

```
 2.000
 4.000
 3.000
```