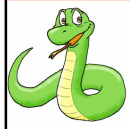


Python 3x Programing

Lesson Python 3.x

Tutor MR. Huynh Nam

Time 90 mins

1

Content

- Python Arrays
- Python Matrices
- NumPy Arrays
- Generating Data Sample
- How to solve System of Linear Equation
- How to calculate derivative of function

2

Python Arrays

- An array is a collection of elements of the **same type**.
- Can treat lists as arrays
  - List is OK

```
a = [1, 3.5, "Hello"]
```

  - Array constrain the type of elements stored in a list

```
a = arr.array('d', [1, 3.5, "Hello"])    // Error
```

- Create arrays: Need to import array module

```
import array as arr
a = arr.array('d', [1.1, 3.5, 4.5])
print(a)
```

What is 'd' ?

3

Python Arrays

- Commonly used type codes:

Type code	C Type	Python Type	Minimum size in bytes
'b'	signed char	int	1
'B'	unsigned char	int	1
'u'	Py_UNICODE	Unicode character	2
'h'	signed short	int	2
'H'	unsigned short	int	2
'i'	signed int	int	2
'I'	unsigned int	int	2
'l'	signed long	int	4
'L'	unsigned long	int	4
'f'	float	float	4
'd'	double	float	8

4

Python Arrays

- Access array element

```
import array as arr
a = arr.array('i', [2, 4, 6, 8])

print("First element:", a[0])
print("Second element:", a[1])
print("Second last element:", a[-1])
```

- Slice array

```
import array as arr

numbers_list = [2, 5, 62, 5, 42, 52, 48, 5]
numbers_array = arr.array('i', numbers_list)

print(numbers_array[2:5]) # 3rd to 5th
print(numbers_array[:-5]) # beginning to 4th
print(numbers_array[5:])  # 6th to end
print(numbers_array[:])   # beginning to end
```

```
array('i', [62, 5, 42])
array('i', [2, 5, 62])
array('i', [52, 48, 5])
array('i', [2, 5, 62, 5, 42, 52, 48, 5])
```


5

Python Arrays - Change or add elements

- Arrays are mutable: Elements can be changed in a similar way like lists.

```
import array as arr

numbers = arr.array('i', [1, 2, 3, 5, 7, 10])

# changing first element
numbers[0] = 0
print(numbers)      # Output: array('i', [0, 2, 3, 5, 7, 10])

# changing 3rd to 5th element
numbers[2:5] = arr.array('i', [4, 6, 8])
print(numbers)      # Output: array('i', [0, 2, 4, 6, 8, 10])
```

```
import array as arr

numbers = arr.array('i', [1, 2, 3])

numbers.append(4)
print(numbers)      # Output: array('i', [1, 2, 3, 4])

# extend() appends iterable to the end of the array
numbers.extend([5, 6, 7])
print(numbers)      # Output: array('i', [1, 2, 3, 4, 5, 6, 7])
```


6

Python Arrays - concatenate two arrays

- Using + operator

```
import array as arr

odd = arr.array('i', [1, 3, 5])
even = arr.array('i', [2, 4, 6])

numbers = arr.array('i')    # creating empty array of integer
numbers = odd + even

print(numbers)
```


7

Python Arrays - Remove/delete elements

- Delete one or more items from an array using Python's del

```
import array as arr

number = arr.array('i', [1, 2, 3, 3, 4])

del number[2] # removing third element
print(number) # Output: array('i', [1, 2, 3, 4])

del number # deleting entire array
print(number) # Error: array is not defined
```

- Use the remove() method to remove the given item, and pop()

```
import array as arr

numbers = arr.array('i', [10, 11, 12, 12, 13])

numbers.remove(12)
print(numbers)    # Output: array('i', [10, 11, 12, 13])

print(numbers.pop(2))    # Output: 12
print(numbers)    # Output: array('i', [10, 11, 13])
```


8

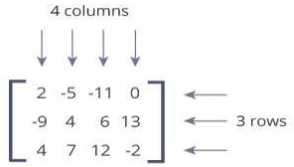
Python Arrays

- Lists are much more flexible than arrays. They can store elements of different data types including string. Also, lists are faster than arrays.
- If you need to do mathematical computation on arrays and matrices, you are much better off using something like NumPy library.
- Unless you don't really need arrays (array module may be needed to interface with C code), don't use them.

9

Python Matrices

- A matrix is a two-dimensional data structure where numbers are arranged into rows and columns.



10

Python Matrices

- Python doesn't have a built-in type for matrices. However, we can treat list of a list as a matrix.

```
A = [[1, 4, 5],
     [-5, 8, 9]]
```

$$\begin{bmatrix} 1 & 4 & 5 \\ -5 & 8 & 9 \end{bmatrix}$$

11

Access Matrix Elements

```
A = [[1, 4, 5, 12],
     [-5, 8, 9, 0],
     [-6, 7, 11, 19]]

print("A =", A)
print("A[1] =", A[1])          # 2nd row
print("A[1][2] =", A[1][2])    # 3rd element of 2nd row
print("A[0][-1] =", A[0][-1])  # Last element of 1st Row

column = [];                   # empty list
for row in A:
  column.append(row[2])

print("3rd column =", column)
```

```
A = [[1, 4, 5, 12], [-5, 8, 9, 0], [-6, 7, 11, 19]]
A[1] = [-5, 8, 9, 0]
A[1][2] = 9
A[0][-1] = 12
3rd column = [5, 9, 11]
```


12

Exercises

- Add two matrices
- Transpose a Matrix
- Multiply two matrices

13

Add two matrices

```
# Program to add two matrices using nested loop

X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]

Y = [[5,8,1],
    [6,7,3],
    [4,5,9]]

result = [[0,0,0],
          [0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[i][j] = X[i][j] + Y[i][j]

for r in result:
   print(r)
```

Initial result matrix

14

Transpose a Matrix

```
# Program to transpose a matrix using nested loop

X = [[12,7],
    [4 ,5],
    [3 ,8]]

result = [[0,0,0],
          [0,0,0]]

# iterate through rows
for i in range(len(X)):
   # iterate through columns
   for j in range(len(X[0])):
       result[j][i] = X[i][j]

for r in result:
   print(r)
```

Initial result matrix

15

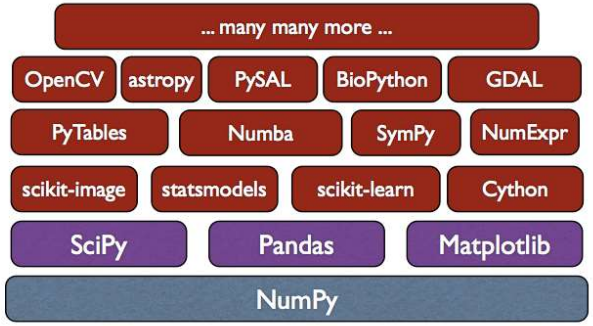
Multiply two matrices

```
# Program to multiply two matrices using nested loops

# 3x3 matrix
X = [[12,7,3],
    [4 ,5,6],
    [7 ,8,9]]
# 3x4 matrix
Y = [[5,8,1,2],
    [6,7,3,0],
    [4,5,9,1]]
# result is 3x4
result = [[0,0,0,0],
          [0,0,0,0],
          [0,0,0,0]]

# iterate through rows of X
for i in range(len(X)):
   # iterate through columns of Y
   for j in range(len(Y[0])):
       # iterate through rows of Y
       for k in range(len(Y)):
           result[i][j] += X[i][k] * Y[k][j]

for r in result:
   print(r)
```

Initial result matrix

16

What is Python Architecture



17

NumPy in Advances

- Introduction
- Environment
- NdArray Object

18

INTRODUCTION

- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.
- Using NumPy, mathematical and logical operations on arrays can be performed.

19

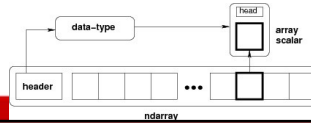
Environment

- Try to import it from Python prompt.
- **import numpy**

20

NdArray Object

- The most important object defined in NumPy is an N-dimensional array type called ndarray.
- It describes the collection of items of the same type.
- Items in the collection can be accessed using a zero-based index.
- Every item in an ndarray takes the same size of block in the memory.
- Each element in ndarray is an object of data-type object (called **dtype**).
- Any item extracted from ndarray object (by slicing) is represented by a Python object of one of array scalar types.
- An instance of ndarray class can be constructed by different array creation routines



21

CREATE NDARRAY

- numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)

```
# simple way to create  array
import numpy as np
a = np.array([1,2,3])
print(a)

# more than one dimensions
a = np.array([[1, 2], [3, 4]])
print(a)

# minimum dimensions
a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a)

# dtype parameter
a = np.array([1, 2, 3], dtype = complex)
print(a)
```

Sr.No.	Parameter & Description
1	**object** Any object exposing the array interface method returns an array, or any (nested) sequence.
2	**dtype** Desired data type of array, optional
3	**copy** Optional. By default (true), the object is copied
4	**order** C (row major) or F (column major) or A (any) (default)
5	**subok** By default, returned array forced to be a base class array. If true, sub-classes passed through
6	**ndmin** Specifies minimum dimensions of resultant array

22

NumPy - Data Types

- NumPy supports a much greater variety of numerical types than Python does.

```
# EX1: using array-scalar type
import numpy as np
dt = np.dtype(np.int32)
print(dt)

# EX2: int8, int16, int32, int64 can be replaced
# by equivalent string 'i1', 'i2','i4', etc.

dt = np.dtype('i4')
print(dt)

# EX3: now apply it to ndarray object

dt = np.dtype([('age',np.int8)])
a = np.array([(10,),(20,),(30,)], dtype = dt)
print(a)

# EX4: Advanced data type as struct

student = np.dtype([('name','S20'), ('age', 'i1'), ('marks', 'f4')])
a = np.array([('abc', 21, 50),('xyz', 18, 75)], dtype = student)
print(a)
print(a["name"])
print(a["age"])
print(a["marks"])
```

23

NumPy - Array Attributes

- numpy.itemsize: This array attribute returns the length of each element of array in bytes.

```
# dtype of array is int8 (1 byte)
import numpy as np
x = np.array([1,2,3,4,5], dtype = np.int8)
print(x.itemsize)

# dtype of array is now float32 (4 bytes)

y = np.array([1,2,3,4,5], dtype = np.float32)
print(y.itemsize)
```


26

NumPy - Array Creation Routines

- numpy.empty: It creates an uninitialized array of specified shape and dtype.

```
import numpy as np

#The elements in an array show random values as they are not initialized.
x = np.empty([3,2], dtype = int)
print(x)

# array of five zeros. Default dtype is float
x = np.zeros(5)
print(x)

x = np.zeros((5,), dtype = np.int)
print(x)

# custom type
import numpy as np
x = np.zeros((2,2), dtype = [('x', 'i4'), ('y', 'f4')])
print(x)
```


27

NumPy - Array From Existing Data

- numpy.asarray: This function is similar to numpy.array except for the fact that it has fewer parameters. This routine is useful for converting Python sequence into ndarray.

```
# convert list to ndarray
import numpy as np

x = [1,2,3]
a = np.asarray(x)
print(a)

# dtype is set
x = [1,2,3]
a = np.asarray(x, dtype = float)
print(a)

# ndarray from tuple
x = (1,2,3)
a = np.asarray(x)
print(a)

# ndarray from list of tuples
x = [(1,2,3),(4,5)]
a = np.asarray(x)
print(a)
```


28

NumPy - Array From Numerical Ranges

- numpy.arange: This function returns an ndarray object containing evenly spaced values within a given range.
- numpy.arange(**start, stop, step, dtype**)

```
import numpy as np
x = np.arange(5)
print x

# dtype set
x = np.arange(5, dtype = float)
print(x)

# start and stop parameters set
x = np.arange(10,20,2)
print(x)
```


29

NumPy - Broadcasting

```
import numpy as np

#Example 1
a = np.array([1,2,3,4])
b = np.array([10,20,30,40])
c = a * b
print(c)


#Example 2
a = np.array([[0.0,0.0,0.0],[10.0,10.0,10.0],[20.0,20.0,20.0]
    ,[30.0,30.0,30.0]])
b = np.array([1.0,2.0,3.0])

print('First array:')
print(a)
print('\n')

print('Second array:')
print(b)
print('\n')

print('First Array + Second Array')
print(a + b)
```

```
[ 10  40  90 160]
First array:
[[  0.   0.   0.]
 [ 10.  10.  10.]
 [ 20.  20.  20.]
 [ 30.  30.  30.]]


Second array:
[ 1.  2.  3.]


First Array + Second Array
[[  1.   2.   3.]
 [ 11.  12.  13.]
 [ 21.  22.  23.]
 [ 31.  32.  33.]]
```


36

NumPy - Iterating Over Array

- NumPy package contains an iterator object numpy.nditer. It is an efficient multidimensional iterator object using which it is possible to iterate over an array. Each element of an array is visited using Python's standard Iterator interface.

```
import numpy as np
a = np.arange(0,60,5)
a = a.reshape(3,4)

print('Original array is:')
print(a)
print('\n')

print('Iterator array is:')
for x in np.nditer(a):
    print(x)
```

```
Original array is:
[[ 0  5 10 15]
 [20 25 30 35]
 [40 45 50 55]]


Iterator array is:
0
5
10
15
20
25
30
35
40
45
50
55
```


37

NumPy - Array Manipulation

- Changing Shape
- Transpose Operations
- Changing Dimensions
- Joining Arrays
- Splitting Arrays
- Adding / Removing Elements
- Binary Operators: bitwise_and, bitwise_or, invert, left_shift, right_shift
- String Functions
  - split(): Returns a list of the words in the string, using separatordelimiter
  - strip(): Returns a copy with the leading and trailing characters removed
  - join(): Returns a copy with the leading and trailing characters removed
  - replace(): Returns a copy of the string with all occurrences of substring replaced by the new string

38

How to solve System of Linear Equation

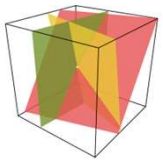
- Definition
- View in Mathematic
- Using Numpy

52

System of Linear Equation

- In mathematics, the theory of linear systems is the basis and a fundamental part of linear algebra, a subject which is used in most parts of modern mathematics. Computational algorithms for finding the solutions are an important part of numerical linear algebra, and play a prominent role in engineering, physics, chemistry, computer science, and economics.
- A system of non-linear equations can often be approximated by a linear system (see linearization), a helpful technique when making a mathematical model or computer simulation of a relatively complex system.

$$\begin{aligned} 3x + 2y - z &= 1 \\ 2x - 2y + 4z &= -2 \\ -x + \tfrac{1}{2}y - z &= 0 \end{aligned}$$



53

System of Linear Equation

- A general system of m linear equations with n unknowns can be written as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}$$

where $x_1, x_2, \ldots, x_n$ are the unknowns, $a_{11}, a_{12}, \ldots, a_{mn}$ are the coefficients of the system, and $b_1, b_2, \ldots, b_m$ are the constant terms.

54

Vector Equation Perspective

- One extremely helpful view is that each unknown is a weight for a column vector in a linear combination.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m, \end{aligned}$$

$$x_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} + x_2 \begin{bmatrix} a_{12} \\ a_{22} \\ \vdots \\ a_{m2} \end{bmatrix} + \cdots + x_n \begin{bmatrix} a_{1n} \\ a_{2n} \\ \vdots \\ a_{mn} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

55

Matrix Equation Perspective

- The vector equation is equivalent to a matrix equation of the form:

$$A\mathbf{x} = \mathbf{b}$$

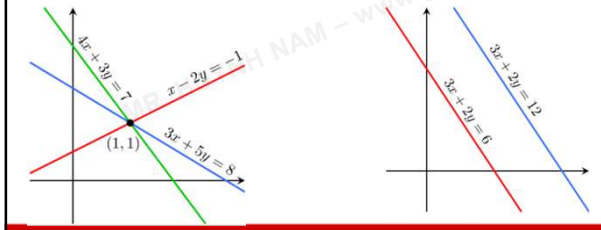
- where A is an **m×n** matrix, **x** is a column vector with **n** entries, and **b** is a column vector with **m** entries.

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

56

Solution of Linear System

- A solution of a linear system is an assignment of values to the variables: $x_1, x_2, \ldots, x_n$ such that each of the equations is satisfied. The set of all possible solutions is called the solution set.
- A linear system may behave in any one of three possible ways:
  - The system has infinitely many solutions.
  - The system has a single unique solution.
  - The system has no solution.



57

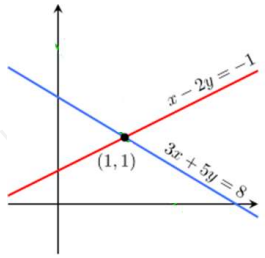
How to solve System of Linear Equation

- Matrix solution $\mathbf{x} = A^{-1}\mathbf{b}$
  - In there, **A^{-1} is inversion of matrix A**
- Solving with Numpy in simple way
  - Using **linalg.solve()**

58

Example

$$\begin{cases} x - 2y = -1 \\ 3x + 5y = 8 \end{cases}$$



59

How to calculate derivative of function

- Remind calculus
- Apply python to calculate derivative of function

61

Derivative in Calculus

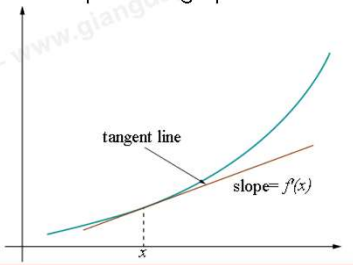
- The derivative of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value). Derivatives are a fundamental tool of calculus. For example, the derivative of the position of a moving object with respect to time is the object's velocity: this measures how quickly the position of the object changes when time advances.
- The derivative of a function of a single variable at a chosen input value, when it exists, is the slope of the tangent line to the graph of the function at that point. The tangent line is the best linear approximation of the function near that input value. For this reason, the derivative is often described as the "instantaneous rate of change", the ratio of the instantaneous change in the dependent variable to that of the independent variable.

62

Differentiation

- Differentiation is the action of computing a derivative. The derivative of a function y = f(x) of a variable x is a measure of the rate at which the value y of the function changes with respect to the change of the variable x. It is called the derivative of f with respect to x. If x and y are real numbers, and if the graph of f is plotted against x, the derivative is the slope of this graph at each point.

$$m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x},$$



63

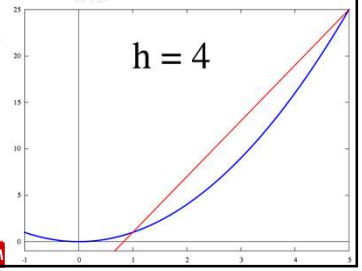
How to calculate derivative of function

- This expression is Newton's difference quotient. Passing from an approximation to an exact answer is done using a limit. Geometrically, the limit of the secant lines is the tangent line. Therefore, the limit of the difference quotient as h approaches zero, if it exists, should represent the slope of the tangent line to (a, f(a)). This limit is defined to be the derivative of the function f at a:

- **m =** $f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a)}{h}$.

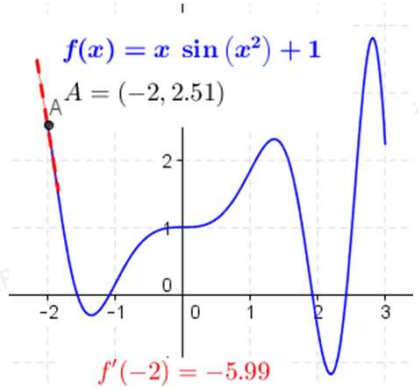
- When the limit exists,

f is said to be differentiable at a.



64

Example



65

Exercise

$$g(x) = \sin(x) * \cos(x) + e^{2x} + 2x^4 - 10$$

- Calculate derive function at a value of x = -1

67

Appendix

Descriptive Statistics Functions

Self-Study

68

Content

- Mathematical Functions
- Arithmetic Operations
- Statistical Functions
- Sort, Search & Counting Functions
- Byte Swapping
- Copies & Views
- Matrix Library
- Linear Algebra
- NumPy – Matplotlib
- NumPy - Histogram Using Matplotlib
- I/O with NumPy

69

Mathematical Functions: Trigonometric Functions

```
import numpy as np
a = np.array([0,30,45,60,90])

print ('Sine of different angles:')
# Convert to radians by multiplying with pi/180
print (np.sin(a*np.pi/180))
print ('\n')

print ('Cosine values for angles in array:')
print (np.cos(a*np.pi/180))
print ('\n')

print ('Tangent values for given angles:')
print (np.tan(a*np.pi/180))
```


70

Trigonometric Functions

```
import numpy as np
a = np.array([0,30,45,60,90])

print 'Array containing sine values:'
sin = np.sin(a*np.pi/180)
print (sin)
print ('\n')

print ('Compute sine inverse of angles. Returned values are in radians.')
inv = np.arcsin(sin)
print (inv)
print ('\n')

print ('Check result by converting to degrees:')
print (np.degrees(inv))
print ('\n')

print ('arccos and arctan functions behave similarly:')
cos = np.cos(a*np.pi/180)
print (cos)
print ('\n')

print ('Inverse of cos:')
inv = np.arccos(cos)
print (inv)
print ('\n')

print ('In degrees:')
print (np.degrees(inv))
print ('\n')

print ('Tan function:')
tan = np.tan(a*np.pi/180)
print (tan)
print ('\n')

print ('Inverse of tan:')
inv = np.arctan(tan)
print (inv)
print ('\n')

print ('In degrees:')
print (np.degrees(inv))
```


71

Trigonometric Functions - Functions for Rounding

- numpy.around(): This is a function that returns the value rounded to the desired precision.

```
import numpy as np
a = np.array([1.0,5.55, 123, 0.567, 25.532])

print ('Original array:')
print (a)
print ('\n')

print ('After rounding:')
print (np.around(a))
print (np.around(a, decimals = 1))
print (np.around(a, decimals = -1))
```


72

Trigonometric Functions - Functions for Rounding

- numpy.floor(): This function returns the largest integer not greater than the input parameter.

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print('The given array:')
print (a)
print ('\n')

print ('The modified array:')
print (np.floor(a))
```


73

Trigonometric Functions - Functions for Rounding

- numpy.ceil(): The ceil() function returns the ceiling of an input value

```
import numpy as np
a = np.array([-1.7, 1.5, -0.2, 0.6, 10])

print ('The given array:')
print (a)
print ('\n')

print ('The modified array:')
print (np.ceil(a))
```


74

Arithmetic Operations

```
import numpy as np
a = np.arange(9, dtype = np.float_).reshape(3,3)

print ('First array:')
print (a)
print ('\n')

print ('Second array:')
b = np.array([10,10,10])
print (b)
print ('\n')

print ('Add the two arrays:')
print (np.add(a,b))
print ('\n')

print ('Subtract the two arrays:')
print (np.subtract(a,b))
print ('\n')

print ('Multiply the two arrays:')
print (np.multiply(a,b))
print ('\n')

print ('Divide the two arrays:')
print (np.divide(a,b))
```


75

Arithmetic Operations

- numpy.reciprocal(): This function returns the reciprocal of argument, element-wise.

```
import numpy as np
a = np.array([0.25, 1.33, 1, 0, 100])

print ('Our array is:')
print (a)
print ('\n')

print ('After applying reciprocal function:')
print (np.reciprocal(a))
print ('\n')

b = np.array([100], dtype = int)
print ('The second array is:')
print (b)
print ('\n')

print ('After applying reciprocal function:')
print (np.reciprocal(b))
```


Arithmetic Operations

- numpy.power(): This function treats elements in the first input array as base and returns it raised to the power of the corresponding element in the second input array.

```
import numpy as np
a = np.array([10,100,1000])

print ('Our array is:')
print (a)
print ('\n')

print ('Applying power function:')
print (np.power(a,2))
print ('\n')

print ('Second array:')
b = np.array([1,2,3])
print (b)
print ('\n')

print ('Applying power function again:')
print (np.power(a,b))
```


Arithmetic Operations

- numpy.mod(): This function returns the remainder of division of the corresponding elements in the input array. The function numpy.remainder() also produces the same result.

```
import numpy as np
a = np.array([10,20,30])
b = np.array([3,5,7])

print ('First array:')
print (a)
print ('\n')

print ('Second array:')
print (b)
print ('\n')

print ('Applying mod() function:')
print (np.mod(a,b))
print ('\n')

print ('Applying remainder() function:')
print (np.remainder(a,b))
```


Arithmetic Operations

- numpy.real() – returns the real part of the complex data type argument.
- numpy.imag() – returns the imaginary part of the complex data type argument.
- numpy.conj() – returns the complex conjugate, which is obtained by changing the sign of the imaginary part.
- numpy.angle() – returns the angle of the complex argument. The function has degree parameter. If true, the angle in the degree is returned, otherwise the angle is in radians.

```
import numpy as np
a = np.array([-5.6j, 0.2j, 11. , 1+1j])

print ('Our array is:')
print (a)
print ('\n')

print ('Applying real() function:')
print (np.real(a))
print ('\n')

print ('Applying imag() function:')
print (np.imag(a))
print ('\n')

print ('Applying conj() function:')
print (np.conj(a))
print ('\n')

print ('Applying angle() function:')
print (np.angle(a))
print ('\n')

print ('Applying angle() function again (result in degrees)')
print (np.angle(a, deg = True))
```

Statistical Functions

- numpy.amin() and numpy.amax(): These functions return the minimum and the maximum from the elements in the given array along the specified axis.

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print ('Our array is:')
print (a)
print ('\n')

print ('Applying amin() function:')
print (np.amin(a,1))
print ('\n')

print ('Applying amin() function again:')
print (np.amin(a,0))
print ('\n')

print ('Applying amax() function:')
print (np.amax(a))
print ('\n')

print ('Applying amax() function again:')
print (np.amax(a, axis = 0))
```


80

Statistical Functions

- numpy.ptp(): The numpy.ptp() function returns the range (maximum-minimum) of values along an axis.

```
import numpy as np
a = np.array([[3,7,5],[8,4,3],[2,4,9]])

print ('Our array is:')
print (a)
print ('\n')

print ('Applying ptp() function:')
print (np.ptp(a))
print ('\n')

print ('Applying ptp() function along axis 1:')
print (np.ptp(a, axis = 1))
print ('\n')

print ('Applying ptp() function along axis 0:')
print (np.ptp(a, axis = 0))
```


81

numpy.percentile()

- Function used to compute the n^{th} percentile of the given data (array elements) along the specified axis.

Syntax : *numpy.percentile(arr, n, axis=None, out=None)*

Parameters :

arr : *input array.*

n : *percentile value.*

axis : *axis along which we want to calculate the percentile value. Otherwise, it will consider arr to be flattened(works on all the axis). axis = 0 means along the column and axis = 1 means working along the row.*

out : *Different array in which we want to place the result. The array must have same dimensions as expected output.*

Return : *nth Percentile of the array (a scalar value if axis is none)or array with percentile values along specified axis.*

82

numpy.percentile() (cont)

```
import numpy as np

# 1D array
arr = [20, 2, 7, 1, 34]
print("arr : ", arr)
print("50th percentile of arr : ",
       np.percentile(arr, 50))
print("25th percentile of arr : ",
       np.percentile(arr, 25))
print("75th percentile of arr : ",
       np.percentile(arr, 75))
```

```
arr :  [20, 2, 7, 1, 34]
30th percentile of arr :  7.0
25th percentile of arr :  2.0
75th percentile of arr :  20.0
```


83

numpy.percentile() (cont)

```
import numpy as np

# 2D array
arr = [[14, 17, 12, 33, 44],
       [15, 6, 27, 8, 19],
       [23, 2, 54, 1, 4,]]
print("\narr : \n", arr)

# Percentile of the flattened array
print("\n50th Percentile of arr, axis = None : ",
      np.percentile(arr, 50))
print("0th Percentile of arr, axis = None : ",
      np.percentile(arr, 0))

# Percentile along the axis = 0
print("\n50th Percentile of arr, axis = 0 : ",
      np.percentile(arr, 50, axis =0))
print("0th Percentile of arr, axis = 0 : ",
      np.percentile(arr, 0, axis =0))
```

```
arr :
 [[14, 17, 12, 33, 44], [15, 6, 27, 8, 19], [23, 2, 54, 1, 4]]

50th Percentile of arr, axis = None :  15.0
0th Percentile of arr, axis = None :  1.0

50th Percentile of arr, axis = 0 :  [15.  6. 27.  8. 19.]
0th Percentile of arr, axis = 0 :  [14.  2. 12.  1.  4.]

50th Percentile of arr, axis = 1 :  [17. 15.  4.]
0th Percentile of arr, axis = 1 :  [12.  6.  1.]
```


84

numpy.percentile() (cont)

```
import numpy as np

# 2D array
arr = [[14, 17, 12, 33, 44],
       [15, 6, 27, 8, 19],
       [23, 2, 54, 1, 4,]]
print("\narr : \n", arr)

# Percentile along the axis = 1
print("\n50th Percentile of arr, axis = 1 : ",
      np.percentile(arr, 50, axis =1))
print("0th Percentile of arr, axis = 1 : ",
      np.percentile(arr, 0, axis =1))

print("\n0th Percentile of arr, axis = 1 : \n",
      np.percentile(arr, 50, axis =1, keepdims=True))
print("\n0th Percentile of arr, axis = 1 : \n",
      np.percentile(arr, 0, axis =1, keepdims=True))
```

```
arr :
 [[14, 17, 12, 33, 44], [15, 6, 27, 8, 19], [23, 2, 54, 1, 4]]

0th Percentile of arr, axis = 1 :
 [[17.]
 [15.]
 [ 4.]]

0th Percentile of arr, axis = 1 :
 [[12.]
 [ 6.]
 [ 1.]]
```


85

NumPy - Sort, Search & Counting Functions

```
import numpy as np
a = np.array([[3,7],[9,1]])

print 'Our array is:'
print a
print '\n'

print 'Applying sort() function:'
print np.sort(a)
print '\n'

print 'Sort along axis 0:'
print np.sort(a, axis = 0)
print '\n'

# Order parameter in sort function
dt = np.dtype([('name', 'S10'),('age', int)])
a = np.array([("raju",21),("anil",25),("ravi", 17), ("amar",27)], dtype = dt)

print 'Our array is:'
print a
print '\n'

print 'Order by name:'
print np.sort(a, order = 'name')
```


86

numpy.argmax() and numpy.argmin()

```
import numpy as np
a = np.array([[30,40,70],[80,20,10],[50,90,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying argmax() function:'
print np.argmax(a)
print '\n'

print 'Index of maximum number in flattened array'
print a.flatten()
print '\n'

print 'Array containing indices of maximum along axis 0 - axis Oy:'
maxindex = np.argmax(a, axis = 0)
print maxindex
print '\n'

print 'Array containing indices of maximum along axis 1 - axis Ox:'
maxindex = np.argmax(a, axis = 1)
print maxindex
print '\n'
```


87

numpy.argmax() and numpy.argmin()

```
print 'Applying argmin() function:'
minindex = np.argmin(a)
print minindex
print '\n'

print 'Flattened array:'
print a.flatten()[minindex]
print '\n'

print 'Flattened array along axis 0:'
minindex = np.argmin(a, axis = 0)
print minindex
print '\n'

print 'Flattened array along axis 1:'
minindex = np.argmin(a, axis = 1)
print minindex
```


88

numpy.nonzero()

```
import numpy as np
a = np.array([[30,40,0],[0,20,10],[50,0,60]])

print 'Our array is:'
print a
print '\n'

print 'Applying nonzero() function:'
print np.nonzero (a)
```

```
Our array is:
[[30 40  0]
 [ 0 20 10]
 [50  0 60]]


Applying nonzero() function:
(array([0, 0, 1, 1, 2, 2]), array([0, 1, 1, 2, 0, 2]))
```


89

numpy.where()

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

print 'Indices of elements > 3'
y = np.where(x > 3)
print y

print 'Use these indices to get elements satisfying the condition'
print x[y]
```

```
Our array is:
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]]
Indices of elements > 3
(array([1, 1, 2, 2, 2]), array([1, 2, 0, 1, 2]))
Use these indices to get elements satisfying the condition
[ 4.  5.  6.  7.  8.]
```


90

numpy.extract()

```
import numpy as np
x = np.arange(9.).reshape(3, 3)

print 'Our array is:'
print x

# define a condition
condition = np.mod(x,2) == 0

print 'Element-wise value of condition'
print condition

print 'Extract elements using condition'
print np.extract(condition, x)
```

```
Our array is:
[[ 0.  1.  2.]
 [ 3.  4.  5.]
 [ 6.  7.  8.]]
Element-wise value of condition
[[ True False  True]
 [False  True False]
 [ True False  True]]
Extract elements using condition
[ 0.  2.  4.  6.  8.]
```


91


Data Visualization
www.huynhnam.com
THANK YOU
Q & A
MR. HUYNH NAM
9/19/2020
WRITTEN BY MR. HUYNH NAM
www.giangdayit.com
92

92