NumPy

Numpy

ย่อมาจาก Numerical Python เป็นไลบลารีแบบ Open source ของ Python ซึ่งเป็นไลบลารีที่ทำงานแต่นรูปแบบออบเจ็คของ Array ชื่อ ndarray ที่อาจคล้ายครึงกับ List แต่เร็วกว่าจัดการหน่วยความจำได้ดีกว่าและมี Operation ให้ใช้สะดวกกว่า

การใช้งาน Numpy

1. การนำเข้า Numpy เพื่อใช้งานและตั้งชื่อว่า np

```
In [1]: import numpy as np
```

2. การสร้าง ndarray จากข้อมูลอื่นของ python (List, tuple)

```
In [8]: lst = [2,3,4,5]
        a = np.array(lst)
        print(a)

        [2 3 4 5]
```

การใช้งาน Numpy

3. การใช้ฟังก์ชั่นต่างๆของ Numpy สร้าง ndarray

3.1 zeros() คือ สร้างแบบกำหนดมิติ และให้ทุกตัวเป็น 0

```
In [6]: b = np.zeros([3,4])
        print(b)
```

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
In [7]: b = np.zeros([3,4],dtype=int)
        print(b)
```

```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

dtype จะใส่หรือไม่ก็ได้

การใช้งาน Numpy

3.2 ones() สร้างแบบกำหนดมิติ และให้ทุกตัวเป็น 1

In [10]:
```
c = np.ones([2,5])
print(c)
```

```
[[1. 1. 1. 1. 1.]
 [1. 1. 1. 1. 1.]]
```

In [11]:
```
c = np.ones([2,5],dtype=int)
print(c)
```

```
[[1 1 1 1 1]
 [1 1 1 1 1]]
```

การใช้งาน Numpy

3.3 full() สร้างแบบกำหนดมิติ และให้ทุกตัวเป็นค่าที่กำหนด

```
In [13]: d = np.full([3,2],5,dtype=int)
         print(d)
```

```
[[5 5]
 [5 5]
 [5 5]]
```

การใช้งาน Numpy

3.4 arange() สร้างจากค่าเริ่มต้นไปยังค่าที่กำหนด

สามารถกำหนดการก้าวได้

In [16]:
```
a = np.arange(10)
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

In [18]:
```
b = np.arange(2,10)
print(b)
```

```
[2 3 4 5 6 7 8 9]
```

In [20]:
```
c = np.arange(2,10,2)
print(c)
```

```
[2 4 6 8]
```

In [21]:
```
d = np.arange(2,10,dtype=float)
print(d)
```

```
[2. 3. 4. 5. 6. 7. 8. 9.]
```

In [22]:
```
e = np.arange(5,1)
print(e)
```

```
[]
```

In [23]:
```
f = np.arange(5,1,-2)
print(f)
```

```
[5 3]
```

การใช้งาน Numpy

3.5 linspace() เป็นการสร้างตัวเลขจากค่าเริ่มต้นไปค่าสิ้นสุดแบบระบุจำนวนที่ต้องการสร้างได้ (ถ้าไม่ระบุจำนวนจะเป็น 50) โดยฟังก์ชั่นจะสร้างแบบเว้นระยะให้เท่าๆกัน

In [24]:
```
a = np.linspace(1,10,20)
print(a)
```

```
[ 1.          1.47368421  1.94736842  2.42105263  2.89473684  3.36842105
  3.84210526  4.31578947  4.78947368  5.26315789  5.73684211  6.21052632
  6.68421053  7.15789474  7.63157895  8.10526316  8.57894737  9.05263158
  9.52631579 10.        ]
```

In [25]:
```
b = np.linspace(1,10)
print(b)
```

```
[ 1.          1.18367347  1.36734694  1.55102041  1.73469388  1.91836735
  2.10204082  2.28571429  2.46938776  2.65306122  2.83673469  3.02040816
  3.20408163  3.3877551   3.57142857  3.75510204  3.93877551  4.12244898
  4.30612245  4.48979592  4.67346939  4.85714286  5.04081633  5.2244898
  5.40816327  5.59183673  5.7755102   5.95918367  6.14285714  6.32653061
  6.51020408  6.69387755  6.87755102  7.06122449  7.24489796  7.42857143
  7.6122449   7.79591837  7.97959184  8.16326531  8.34693878  8.53061224
  8.71428571  8.89795918  9.08163265  9.26530612  9.44897959  9.63265306
  9.81632653 10.        ]
```

In [26]:
```
c = np.linspace(1,10,10)
print(c)
```

```
[ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10.]
```

การใช้งาน Numpy

3.6 eye() สร้างแนวทแยงมุมที่มีค่าเป็น 1 ที่เหลือเป็น 0

สามารถกำหนดตำแหน่งที่เริ่มทแยงมุมได้

In [27]:
```
a = np.eye(3)
print(a)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
```

In [29]:
```
b = np.eye(4,3)
print(b)
```

```
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]]
```

In [31]:
```
c = np.eye(4,3,1)
print(c)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

In [34]:
```
d = np.eye(4,3,-2)
print(d)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]]
```

การใช้งาน Numpy

3.6 empty() สร้าง ndarray แบบค่าสุ่ม

```
In [49]: a = np.empty([3,3],dtype=int)
         print(a)
```

```
[[5505024 5046341 3997776]
 [3801155 5570652 6619251]
 [7536754 6881372 7274580]]
```

การใช้งาน Numpy

4. สร้างจากการสุ่มของโมดูล Random

การ import มี 2 วิธี

1. import numpy as np

   วิธีใช้ a = np.random.randint(100)

2. from numpy import random as rd

   วิธีใช้ a = rd.randint(100)

ในที่นี้จะใช้แบบที่ 2 เพราะสั้นกว่า

การใช้งาน Numpy

4.1 rand() สร้างแบบค่าสุ่ม 0-1

In [50]:
```
from numpy import random as rd
```

In [51]:
```
a = rd.rand()
print(a)
```

```
0.3373319371028397
```

In [54]:
```
b = rd.rand(3,4)
print(b)
```

```
[[0.87472663 0.09027681 0.77816701 0.46195189]
 [0.8536297  0.81690255 0.27414085 0.6601884 ]
 [0.48847713 0.42016485 0.97806863 0.4135736 ]]
```

การใช้งาน Numpy

4.2 randint() สร้างแบบกำหนดค่าสูงสุดไว้ด้วย และกำหนดขนาดได้ (ค่าเป็น int เท่านั้น)

```
In [55]: a = rd.randint(100)
         print(a)

         84

In [57]: b = rd.randint(100,size=[2,3])
         print(b)

         [[87 69  1]
          [82 71 91]]
```

การใช้งาน Numpy

4.3 choice() สร้างแบบสุ่มจากข้อมูลที่ตั้งไว้ และสามารถตั้งค่าความน่าจะเป็นให้ไม่เท่ากันได้

In [60]:
```
lst = [5,7,8,20]
a = rd.choice(lst)
print(a)
```

```
7
```

In [62]:
```
b = rd.choice(lst,size=[2,3])
print(b)
```

```
[[ 5  7  8]
 [20  8 20]]
```

In [63]:
```
c = rd.choice(lst,p=[0.1,0.1,0.3,0.5],size=[2,3])
print(c)
```

```
[[20 20  8]
 [ 7  8 20]]
```

Shuffle & Permutation

shuffle() คือการสลับตำแหน่ง

permutation() คือการสลับตำแหน่งแบบไปสร้างออบเจ็คใหม่ ไม่ให้ต้นทางเปลี่ยนค่า

In [65]:
```
a = np.arange(10)
print(a)
rd.shuffle(a)
print(a)
```

```
[0 1 2 3 4 5 6 7 8 9]
[1 5 7 0 9 2 8 6 3 4]
```

In [67]:
```
a = np.arange(10)
print(a)
b = rd.permutation(a)
print(a)
print(b)
```

```
[0 1 2 3 4 5 6 7 8 9]
[0 1 2 3 4 5 6 7 8 9]
[5 0 7 2 6 9 3 4 8 1]
```

reshape

คือการเปลี่ยนขนาดของ ndarray

In [79]:
```
a = np.arange(20)
print(a)
```

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

In [80]:
```
a = a.reshape(4,5)
print(a)
```

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

In [81]:
```
a = a.reshape(-1)
print(a)
```

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

Vectorization

คือการนำ ndarray สองตัวที่มีมิติเท่ากัน มาทำ Operation ต่างๆ

In [82]:
```
a = np.array([1,2,3,4])
b = np.array([2,2,4,5])
c = a+b
print(c)
```

```
[3 4 7 9]
```

Broadcasting

คือการนำ ndarray สองตัวที่มี 1 สามาชิก กับ n สมาชิกมาทำ Operation ต่างๆ

```
In [85]: a = np.array([1,2,3,4])
         b = np.array([5])
         c = a+b
         print(c)
```

```
[6 7 8 9]
```

Vectorization & Broadcasting

ในกรณีที่มิติมีเท่า แต่สมาชิกไม่เท่าสามารถทำ Vectorization & Broadcasting ได้เหมือนกัน เช่น

In [88]:
```
a = np.arange(50).reshape(5,10)
print(a)
b = np.arange(5).reshape(5,1)
print(b)
```

```
[[ 0  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 37 38 39]
 [40 41 42 43 44 45 46 47 48 49]]
[[0]
 [1]
 [2]
 [3]
 [4]]
```

In [89]:
```
c = a*b
print(c)
```

```
[[  0   0   0   0   0   0   0   0   0   0]
 [ 10  11  12  13  14  15  16  17  18  19]
 [ 40  42  44  46  48  50  52  54  56  58]
 [ 90  93  96  99 102 105 108 111 114 117]
 [160 164 168 172 176 180 184 188 192 196]]
```

Array Indexing


```
In [94]: a = np.arange(12).reshape(3,4)
         print(a)

[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

In [95]: a[1][2]

Out[95]: 6

In [96]: a[1,2]

Out[96]: 6

In [97]: a[-1,-3]

Out[97]: 9

In [98]: a[0,-2]

Out[98]: 2
```

Index Slicing


```
In [99]: a = np.arange(12)
         print(a)

         [ 0  1  2  3  4  5  6  7  8  9 10 11]

In [100]: a[:]

Out[100]: array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])

In [101]: a[:5]

Out[101]: array([0, 1, 2, 3, 4])

In [102]: a[3:]

Out[102]: array([ 3,  4,  5,  6,  7,  8,  9, 10, 11])

In [103]: a[3:6]

Out[103]: array([3, 4, 5])

In [104]: a[1:10:2]

Out[104]: array([1, 3, 5, 7, 9])
```

Index Slicing 2d


```
In [106]: a = np.arange(20).reshape(4,5)
          print(a)

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

```
In [107]: a[:,:]

Out[107]: array([[ 0,  1,  2,  3,  4],
                 [ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14],
                 [15, 16, 17, 18, 19]])
```

```
In [108]: a[1:3,:]

Out[108]: array([[ 5,  6,  7,  8,  9],
                 [10, 11, 12, 13, 14]])
```

```
In [109]: a[:,1:3]

Out[109]: array([[ 1,  2],
                 [ 6,  7],
                 [11, 12],
                 [16, 17]])
```

```
In [110]: a[1:3,1:3]

Out[110]: array([[ 6,  7],
                 [11, 12]])
```

```
In [111]: a[1:4:2,1:4:2]

Out[111]: array([[ 6,  8],
                 [16, 18]])
```

Index Slicing 2d


```
In [112]: print(a)
```

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

```
In [118]: a[[0,3,3,2,0]]
```

```
Out[118]: array([[ 0,  1,  2,  3,  4],
                 [15, 16, 17, 18, 19],
                 [15, 16, 17, 18, 19],
                 [10, 11, 12, 13, 14],
                 [ 0,  1,  2,  3,  4]])
```

```
In [120]: a[[0,3,3,2,0],[3]]
```

```
Out[120]: array([ 3, 18, 18, 13,  3])
```

```
In [122]: a[[0,3,3,2,0],[3,1,3,0,0]]
```

```
Out[122]: array([ 3, 16, 18, 10,  0])
```

Index Slicing 2d

In [128]:
```
a = np.arange(20).reshape(4,5)
print(a)
```

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

In [132]:
```
cond = [True,False,False,True]
a[cond]
```

Out[132]:
```
array([[ 0,  1,  2,  3,  4],
       [15, 16, 17, 18, 19]])
```

Array Join

In [140]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate([arr1, arr2])
print(arr)
```

```
[1 2 3 4 5 6]
```

In [141]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=1)
print(arr)
```

```
[[1 2 5 6]
 [3 4 7 8]]
```

In [142]:
```
arr1 = np.array([[1, 2], [3, 4]])
arr2 = np.array([[5, 6], [7, 8]])
arr = np.concatenate((arr1, arr2), axis=0)
print(arr)
```

```
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
```

Array Join

In [143]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=1)
print(arr)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

In [144]:
```
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.stack((arr1, arr2), axis=0)
print(arr)
```

```
[[1 2 3]
 [4 5 6]]
```

Array Split

In [147]:
```
arr = np.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]])
print(arr)
newarr = np.array_split(arr, 3)
print(newarr)
```

```
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]
 [17 18 19 20]
 [21 22 23 24]]
[array([[1, 2, 3, 4],
       [5, 6, 7, 8]]), array([[ 9, 10, 11, 12],
       [13, 14, 15, 16]]), array([[17, 18, 19, 20],
       [21, 22, 23, 24]])]
```

In [148]:
```
newarr[1]
```

Out[148]:
```
array([[ 9, 10, 11, 12],
       [13, 14, 15, 16]])
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

1. round ปัดเศษ

```
In [149]: a = np.array([2.0, 7.55, 98, 0.697, 3.14])

          print(np.round(a))
          print(np.round(a,1))
          print(np.round(a,-1))
```

```
[ 2.  8. 98.  1.  3.]
[ 2.   7.6 98.   0.7  3.1]
[  0.  10. 100.   0.   0.]
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

2. abs ค่าสมบูรณ์

```
In [151]: a = np.array([-1,-2,-3])

          print(np.abs(a))
```

```
[1 2 3]
```

3. power , log , exp

```
In [166]: a = np.array([1,1,3,4])

          print(np.log(a))
          print(np.exp(a))
          print(np.power(a,2))
```

```
[0.         0.         1.09861229 1.38629436]
[ 2.71828183  2.71828183 20.08553692 54.59815003]
[ 1  1  9 16]
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

4. sum

```
In [154]: a = np.arange(12).reshape(4,3)
          print(a)
          print("sum by column : ",a.sum(axis=0))
          print("sum by row : ",a.sum(axis=1))
```

```
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
sum by column :  [18 22 26]
sum by row :  [ 3 12 21 30]
```

5. sqrt

```
In [155]: a = np.array([1,4,9,16])

          print(np.sqrt(a))
```

```
[1. 2. 3. 4.]
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

6. logical

```
In [157]: a = np.array([1,1,0,4])
          b = np.array([1,0,0,4])

          print("NOT a",np.logical_not(a))
          print("a XOR b",np.logical_xor(a,b))
          print("a OR b",np.logical_or(a,b))
          print("a AND b",np.logical_and(a,b))
```

```
NOT a [False False  True False]
a XOR b [False  True False False]
a OR b [ True  True False  True]
a AND b [ True False False  True]
```

7. all , any

```
In [158]: a = np.array([1,4,9,16])

          print(np.all(a > 5))
          print(np.any(a > 5))
```

```
False
True
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

8. min max mean median

In [161]:
```
a = rd.randint(4,12,size=10)

print(a)
print("min : ",np.min(a))
print("max : ",np.max(a))
print("median : ",np.median(a))
print("mean : ",np.mean(a))
```

```
[ 6 10 10  4  5  8  6  8  5  6]
min :  4
max :  10
median :  6.0
mean :  6.8
```

In [162]:
```
a = rd.randint(4,12,size=10)
a = a.reshape(2,5)
print(a)
print("min : ",np.min(a, axis=1))
print("max : ",np.max(a, axis=1))
print("median : ",np.median(a, axis=1))
print("mean : ",np.mean(a, axis=1))
```

```
[[ 9  6  9  8 10]
 [ 6  6  4  5  6]]
min :  [6 4]
max :  [10  6]
median :  [9. 6.]
mean :  [8.4 5.4]
```

ฟังก์ชั่นการทำงานอื่นๆของ Numpy

9. sort

In [163]:
```
a = rd.randint(4,12,size=12)

print(a)
print(np.sort(a))
```

```
[ 9  6  5  4  5 10  4 11  9  9  7  9]
[ 4  4  5  5  6  7  9  9  9  9 10 11]
```

In [165]:
```
a = rd.randint(4,12,size=12)
a = a.reshape(4,3)
print(a)
print(np.sort(a,axis=1))
```

```
[[ 9  9  4]
 [11 11  4]
 [ 4  7 10]
 [ 8 10  6]]
[[ 4  9  9]
 [ 4 11 11]
 [ 4  7 10]
 [ 6  8 10]]
```

Array Iterating

In [168]:
```
arr = np.array([[1, 2, 3], [4, 5, 6]])

for x in arr:
    for y in x:
        print(y)
```

```
1
2
3
4
5
6
```

Searching Arrays

In [174]:
```
arr = np.array([6, 7, 8, 9])
x = np.searchsorted(arr, 7)
print(x)
```

```
1
```

In [175]:
```
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8])
x = np.where(arr%2 == 0)
print(x)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

Filter Array

In [176]:
```
arr = np.array([41, 42, 43, 44])
filter_arr = arr > 42
newarr = arr[filter_arr]

print(filter_arr)
print(newarr)
```

```
[False False  True  True]
[43 44]
```