

Class period 5

บทที่ 3 โปรแกรมวนซ้ำและการใช้เงื่อนไขในภาษาไพธอน (ต่อ)
More_advanced_data_structure 2

1

zeros คำสั่ง numpy.zeros()

- เป็นคำสั่งที่ใช้สร้าง matrix ที่ทุกค่าเท่ากับ 0

```
• import numpy as np
• np.zeros(2)
• array([0., 0.])

• np.zeros((2,3))
• array([[0., 0., 0.],
        [0., 0., 0.]])
```

numpy.zeros

```
numpy.zeros(shape, dtype=float, order='C', *, like=None)
Return a new array of given shape and type, filled with zeros.

Parameters: shape : int or tuple of ints
Shape of the new array, e.g., (2, 3) or 2.
dtype : data-type optional
The desired data-type for the array, e.g., numpy.float64,
numpy.float32, etc.
order : {'C', 'F'}, optional, default: 'C'
Whether to store multidimensional data in row-major (C-style) or
column-major (Fortran-style) order in memory.
like : array-like, optional
Reference object to allow the creation of arrays which are not Numpy
arrays, but have the same layout. If an array-like passed in as 'like' supports
the __array__ protocol, the result will be defined by it. In this
case, it ensures the creation of an array object compatible with that
passed in via this argument.

Returns: out : ndarray
Array of zeros with the given shape, dtype, and order.
```

1

2

ones คำสั่ง numpy.ones()

- เป็นคำสั่งที่ใช้สร้าง matrix ที่ทุกค่าเท่ากับ 1

```
• np.ones((2,3))
• array([[1., 1., 1.],
        [1., 1., 1.]])
```

numpy.ones

```
numpy.ones(shape, dtype=float, order='C', *, like=None)
Return a new array of given shape and type, filled with ones.

Parameters: shape : int or sequence of ints
Shape of the new array, e.g., (2, 3) or 2.
dtype : data-type optional
The desired data-type for the array, e.g., numpy.float64,
numpy.float32, etc.
order : {'C', 'F'}, optional, default: 'C'
Whether to store multidimensional data in row-major (C-style) or
column-major (Fortran-style) order in memory.
like : array-like, optional
Reference object to allow the creation of arrays which are not Numpy
arrays, but have the same layout. If an array-like passed in as 'like' supports
the __array__ protocol, the result will be defined by it. In this
case, it ensures the creation of an array object compatible with that
passed in via this argument.

Returns: out : ndarray
Array of ones with the given shape, dtype, and order.
```

2

3

Matrix Operation (scalar multiplication)

- scalar multiplication คือ การคูณค่าคงที่ 1 ค่าที่เป็นเลขตัวเดียวคูณเข้าไปใน matrix เช่น

```
• M_one = np.ones((2,3))
• มุมมองภายใน
• array([[1., 1., 1.],
        [1., 1., 1.]])

• 2*M_one
• ผลลัพธ์จะได้
• array([[2., 2., 2.],
        [2., 2., 2.]])
```

3

4

Random

- คำสั่ง numpy.random เป็นคำสั่งที่ใช้สร้าง matrix ที่สุ่มค่าภายใน matrix
- คำสั่ง numpy.random มีหลายประเภท โดยคำสั่งที่พบบ่อยมี 3 ประเภท คือ
- 1. numpy.random.rand
- 2. numpy.random.randn
- 3. numpy.random.choice

4

5

คำสั่ง numpy.random.rand()

- โดย random.rand จะเป็น uniform random เป็นการสุ่มค่าที่ทุกค่ามีโอกาสที่จะสุ่มได้เท่ากัน ซึ่งค่าที่สุ่มได้จะมีค่าตั้งแต่ 0 ถึง 1

numpy.random.rand

```
numpy.random.rand(*args)
Random values in a given shape.

Note
This is a convenience function for users porting code from Matlab, and wraps
random.randn. That function takes a tuple to specify the size of the output, which is
consistent with other NumPy functions like numpy.linspace and numpy.linspace.

Parameters: args : tuple of ints, optional
The dimensions of the returned array, must be non-negative. If no
argument is given a single Python float is returned.

Returns: out : ndarray, shape (args[0], args[1], ...)
Random values.
```

5

6

คำสั่ง numpy.random.randn()

- โดย random.randn จะเป็น normal distribution mean=0 std=1 เป็นการสุ่มค่าที่เข้าใกล้ 0 จะมีโอกาสที่จะสุ่มได้มากกว่าค่าที่อยู่ห่าง 0 ซึ่งค่าที่สุ่มได้จะมีค่าตั้งแต่ 0 ถึง 1

```

np.random.randn(3,2)
array([[ 2.06762285,  0.91239845],
       [-2.08011942, -0.46261935],
       [ 0.66804796,  1.19419422]])

```

numpy.random.randn

Return a sample (or samples) from the "standard normal" distribution.

Parameters: `size` : int or tuple of ints, optional
The dimensions of the returned array, must be non-negative. If no argument is given, a single float is returned.

Returns: `randn` : float or ndarray of floats
A float or ndarray of floats. If size is None, then a single float is returned. Otherwise, ndarray of floats of the given shape is returned.

7

คำสั่ง numpy.random.choice()

- โดย random.choice จะเป็นการสุ่มค่าที่กำหนดเอง

```

np.random.choice([1,2,3,'a','b','c'])
'a'

```

numpy.random.choice

random.choice(a, size=None, replace=True, p=None)
Generates random sample(s) from an array.

Parameters: `a` : 1-D array-like
An array-like object from which to generate samples. If an int, the random sample is generated as if a was np.arange(a).

size : int or tuple of ints, optional
Output shape. If the given shape is, e.g., (m, n, k), then a $m \times n \times k$ array of samples is generated. If not given, the sample is a single value.

replace : bool, optional
Whether the sample is with or without replacement. Default is True, meaning that a value of a can be selected multiple times.

p : 1-D array-like, optional
The probabilities associated with each entry in a. If not given, the sample assumes a uniform distribution over all entries in a.

Returns: `sample` : single value or ndarray
The generated random sample(s).

8

ตัวอย่างการใช้งาน Parameter: size numpy.random.choice()

- Parameter: size เป็นการกำหนดจำนวนค่าที่ต้องการสุ่มออกมา โดย size สามารถ input ค่าเป็น int (ตัวเลขเดียว) หรือ tuple of ints (ใช้สำหรับสุ่มออกมาเป็น matrix) เช่น

```

np.random.choice([1,2,3,28,11,100],size = 2)
array([28,  3])

np.random.choice([1,2,3,28,11,100],size = (2,3))
array([[ 3,  3, 11],
       [28,  1, 11]])

```

9

ตัวอย่างการใช้งาน Parameter: replace numpy.random.choice()

- Parameter: replace การใส่คืน เป็นการกำหนดว่า จะให้ค่าที่สุ่มออกมาใส่คืนหรือไม่ใส่คืน คือ การสุ่มค่าซ้ำหรือไม่ซ้ำ
- โดยค่า default ของ replace = True คือการสุ่มแบบใส่คืนมีค่าซ้ำกันได้ ถ้าต้องการสุ่มแบบไม่ใส่คืน คือไม่ให้ค่าซ้ำกัน ให้กำหนด replace = False

```

np.random.choice([1,2,3,28,11,100], size = (2,3), replace=False)
array([[ 3,  11,  2],
       [100, 28,  1]])

```

10

ตัวอย่างการใช้งาน Parameter: p numpy.random.choice()

- Parameter: p เป็นการกำหนดความน่าจะเป็นที่แต่ละค่าจะถูกรandom
- เช่น มีนักศึกษา 35 คนต้องการสุ่มเกรดให้แต่ละคนตามความน่าจะเป็นที่กำหนด
- กำหนดให้ 'A' ให้ความน่าจะเป็นที่จะถูกสุ่ม 10%
- กำหนดให้ 'B' ให้ความน่าจะเป็นที่จะถูกสุ่ม 20%
- กำหนดให้ 'C' ให้ความน่าจะเป็นที่จะถูกสุ่ม 40%
- กำหนดให้ 'D' ให้ความน่าจะเป็นที่จะถูกสุ่ม 29%
- กำหนดให้ 'F' ให้ความน่าจะเป็นที่จะถูกสุ่ม 1%

```

np.random.choice(['A','B','C','D','F'],size=35,
p = [0.1,0.2,0.4,0.29,0.01])
array(['D', 'C', 'C', 'C', 'C', 'C', 'B', 'C', 'B', 'A', 'C', 'C',
       'D', 'A', 'C', 'C', 'D', 'B', 'B', 'A', 'C', 'C', 'C', 'D',
       'A', 'D', 'C', 'D', 'B', 'B', 'D', 'C', 'D', 'C'], dtype=object)

```

11

เฉลย HW Matrix Multiplication class period 4

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \end{bmatrix}$$

- เขียน function คูณ matrix ให้ผลลัพธ์เหมือน dot product (ไม่ใช่ dot product)
- แล้ว test กับ matrix ขนาด
- (2,3)*(3,2)
- (4,4)*(4,1)
- (2,2)*(2,2)

12

การสร้างฟังก์ชัน คูณ matrix

```
def mat_mul(A,B):
    C = np.zeros((A.shape[0],B.shape[1]))
    for r_a in range(A.shape[0]):
        for c_b in range(B.shape[1]):
            for every_element in range(A.shape[1]):
                C[r_a,c_b] = C[r_a,c_b] + (A[r_a,every_element] *
                B[every_element,c_b])

    return C
```

12

13

ขั้นตอนการสร้างฟังก์ชัน คูณ matrix

- def mat_mul(A,B):
- คือการสร้างและกำหนดชื่อฟังก์ชัน, input ที่ต้องการ matrix A * matrix B
- C = np.zeros((A.shape[0],B.shape[1]))
- หมายความว่า ให้สร้าง matrix ที่มีทุกค่าเป็น 0 โดยกำหนดจำนวนแถวและหลัก จากการอ่านค่าจำนวนแถวของตัวหน้า(A.shape[0])และจำนวนหลักของตัวหลัง(B.shape[1])จากตัวแปรที่ input เพื่อเตรียมขนาดของ matrix ผลลัพธ์

13

14

ขั้นตอนการสร้าง loop1 สำหรับ คูณ matrix

- for r_a in range(A.shape[0]):
 - A.shape[0] คือจำนวนแถวของตัวหน้า
 - range(A.shape[0]) คือสร้าง list ตัวเลขตามจำนวนแถวของตัวหน้า
 - ถ้า A.shape[0] = 2
 - range(A.shape[0]) = range(2)
 - จะเท่ากับ range(0, 2) ในรูปแบบ list คือ[0, 1]
 - ดังนั้น loop1 ใช้รับอ่านค่าใน range(A.shape[0]) ที่ละตัว เก็บไว้ในตัวแปร r_a
- range คือคำสั่งที่ใช้สร้าง list ตัวเลข
shape คือคำสั่งในการตรวจสอบขนาดของ matrix

14

15

ขั้นตอนการสร้าง loop2 สำหรับ คูณ matrix

- for c_b in range(B.shape[1]):
- B.shape[1] คือจำนวนหลักของตัวหลัง
- range(B.shape[1]) คือสร้าง list ตัวเลขตามจำนวนหลักของตัวหลัง
- ถ้า B.shape[1] = 2
- range(B.shape[1]) = range(2)
- จะเท่ากับ range(0, 2) ในรูปแบบ list คือ[0, 1]
- ดังนั้น loop2 ใช้รับอ่านค่าใน range(B.shape[1]) ที่ละตัว เก็บไว้ในตัวแปร c_b

15

16

ขั้นตอนการสร้าง loop3 สำหรับ คูณ matrix

- for every_element in range(A.shape[1]):
- A.shape[1] คือจำนวนหลักของตัวแรก
- range(A.shape[1]) คือสร้าง list ตัวเลขตามจำนวนหลักของตัวแรก
- ถ้า A.shape[1] = 3
- range(A.shape[1]) = range(3)
- จะเท่ากับ range(0, 3) ในรูปแบบ list คือ[0, 1, 2]
- ดังนั้น loop3 ใช้รับอ่านค่าใน range(A.shape[1]) ที่ละตัว เก็บไว้ในตัวแปร every_element

16

17

ลำดับการทำงาน loop for ในฟังก์ชัน

- Loop1 คือ for r_a in range(A.shape[0]):
- Loop2 คือ for c_b in range(B.shape[1]):
- Loop3 คือ for every_element in range(A.shape[1]):
- ถ้า
- range(A.shape[0]) = [0, 1]
- range(B.shape[1]) = [0, 1]
- range(A.shape[1]) = [0, 1, 2]
- Loop1
- Loop2
- Loop3
- print(f'row : (r_a) column : (c_b) every_element : (every_element)')

17

18

ลำดับการทำงาน loop for ในฟังก์ชัน

- ผลลัพธ์จะได้

row : 0	column : 0	every_element : 0
row : 0	column : 0	every_element : 1
row : 0	column : 0	every_element : 2
row : 0	column : 1	every_element : 0
row : 0	column : 1	every_element : 1
row : 0	column : 1	every_element : 2
- หมายความว่า loop3 จะทำงานลูปของตัวเองจนจบรอบก่อน ถึงนับเป็นลูป 1 รอบของ loop2 และ loop2 จะทำงานลูปของตัวเองจนจบรอบก่อน ถึงนับเป็นลูป 1 รอบของ loop1

19

การคำนวณภายใน loop3

- $C[r_a, c_b] = C[r_a, c_b] + (A[r_a, \text{every_element}] * B[\text{every_element}, c_b])$
- ในการทำงานครั้งแรกของฟังก์ชันค่าของ $C[r_a, c_b]$ จะเท่ากับ 0
- เพราะสร้าง $C = \text{np.zeros}((A.\text{shape}[0], B.\text{shape}[1]))$
- $A[r_a, \text{every_element}]$ คือ ค่าของ matrix ตัวหนึ่งตามตำแหน่ง $[r_a, \text{every_element}]$ เช่น
- ถ้า A คือ

array([[7, 8],
[9, 10],
[11, 12]])
- $A[0,0]$ จะเท่ากับ 7
- $A[0,1]$ จะเท่ากับ 8

20

การคำนวณภายใน loop3

- $B[\text{every_element}, c_b]$ คือ ค่าของ matrix ตัวหนึ่งตามตำแหน่ง $[\text{every_element}, c_b]$ เช่น
- ถ้า B คือ

array([[7, 8],
[9, 10],
[11, 12]])
- $B[0,0]$ จะเท่ากับ 7
- $B[1,0]$ จะเท่ากับ 9

21

การคำนวณการคูณ matrix

- ดังนั้น
 - $C[r_a, c_b] = C[r_a, c_b] + (A[r_a, \text{every_element}] * B[\text{every_element}, c_b])$
 - ในการทำงานรอบแรกของ loop1
 - $C[r_a, c_b]$ จะเป็นตัวแปรที่เข้ามาเก็บค่าผลลัพธ์จากการคำนวณเพื่อนำค่าไปบวกใน loop รอบต่อไป เช่น
 - $C[0,0] = C[0,0] + (A[0,0] * B[0,0])$
 - $C[0,0] = 0.0 + (1 * 7)$
 - $C[0,0] = 7.0$ และเอาไปคำนวณใน loop รอบต่อไป
 - $C[0,0] = C[0,0] + (A[0,1] * B[1,0])$
 - $C[0,0] = 7.0 + (2 * 9)$
 - $C[0,0] = 25.0$ และเอาไปคำนวณใน loop รอบต่อไปจนจบ $C[0,0]$ และไปเริ่ม $C[0,1]$

22

วิธีใช้งานฟังก์ชัน

- สร้าง matrix input ตามตัวอย่าง

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 & 8 \\ 9 & 10 \\ 11 & 12 \end{bmatrix} = \begin{bmatrix} 58 & 64 \end{bmatrix}$$
- ```
mat1 = np.array([[1,2,3],[4,5,6]])
array([[1, 2, 3],
 [4, 5, 6]])
mat2 = np.array([[7,8],[9,10],[11,12]])
array([[7, 8],
 [9, 10],
 [11, 12]])
```

23

### ผลลัพธ์ของฟังก์ชัน

- `mat_mul(mat1,mat2)` ชื่อฟังก์ชันตามด้วย input (A,B) ที่ตั้งไว้
- ผลลัพธ์จะได้
 

|                      |
|----------------------|
| array([[ 58.,  64.], |
| [139., 154.]])       |
- ซึ่งถ้าเปรียบเทียบกับตัวแปรในฟังก์ชันคือตัวแปรที่เข้ามาเก็บผลลัพธ์การคำนวณในแต่ละรอบของ loop1
  - $C[0,0]$ ,  $C[0,1]$ ,
  - $C[1,0]$ ,  $C[1,1]$

24