Class period 5

บทที่ 3 โปรแกรมวนซ้ำและการใช้เลียนไขในภาษาไพธอน (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

2

ones คำสั่ง numpy.ones()

- เป็นคำสั่งที่ใช้สร้าง matrix ที่ทุกค่าเท่ากับ 1

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

numpy.ones

3

Matrix Operation (scalar multiplication)

- scalar multiplication คือ การคูณค่าคงที่ 1 ค่าที่เป็นสเกลาตัวเดียวคูณเข้าใน matrix เช่น

```
M_one = np.ones((2,3))
```

- มุมมองภาายใน

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

2*M_one
```

- ผลลัพธ์จะได้

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

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

5

คำสั่ง numpy.random.rand()

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

```
np.random.rand(3,2)
array([[0.12461684, 0.63204405],
       [0.240901  , 0.34341953],
       [0.22536518, 0.86663463]])
```

numpy.random.rand

6

คำสั่ง numpy.random.randn()

- โดย random.randn จะเป็น normal distribution mean=0 std=1 เป็นการสุ่มค่าที่ค่าเข้าใกล้ 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

7

คำสั่ง numpy.random.choice()

- โดย random.choice จะเป็นการสุ่มค่าที่กำหนดเอง

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

numpy.random.choice

8

ตัวอย่างการใช้งาน Parameter: size numpy.random.choice()

- Parameter: size เป็นการกำหนดจำนวนค่าที่ต้องการสุ่มออกมาโดย size สามารถ input ค่าเป็น int (ตัวเลขตัวเดียว) หรือ tuple of ints (ใช้สำหรับสุ่มค่าออกมาเป็น 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 จะให้นำค่าที่สุ่มออกมาใส่คืนหรือไม่ใส่คืน คือ การสุ่มค่าซ้ำหรือไม่ซ้ำ
- โดยค่า default ของ replace = True คือการสุ่มแบบใส่คืนมีค่าซ้ำกันได้ ถ้าต้องการสุ่มแบบไม่ใส่คืน คือให้ไม่มีค่าซ้ำกัน ให้กำหนด replace = Fasle

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

9

10

ตัวอย่างการใช้งาน Parameter: p numpy.random.choice()

- Parameter: p เป็นการกำหนดความน่าจะเป็นที่ค่าแต่ละค่าจะถูกสุ่ม
- เช่น มีนักศึกษา 35 คนต้องการสุ่มแกรดให้แต่ละคนตามความน่าจะเป็นที่กำหนด
- กำหนดให้ 'A' ให้มีความน่าจะเป็นที่จะถูกสุ่ม 10%
- กำหนดให้ 'B' ให้มีความน่าจะเป็นที่จะถูกสุ่ม 20%
- กำหนดให้ 'C' ให้มีความน่าจะเป็นที่จะถูกสุ่ม 40%
- กำหนดให้ 'D' ให้มีความน่าจะเป็นที่จะถูกสุ่ม 29%
- กำหนดให้ 'F' ให้มีความน่าจะเป็นที่จะถูกสุ่ม 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', 'D', 'A', 'C', 'C',
       'D', 'A', 'C', 'C', 'D', 'D', 'A', 'C', 'A', 'C', 'C', 'C', 'D',
       'A', 'D', 'C', 'D', 'B', 'D', 'C', 'D', 'C'], dtype='<U1')
```

10

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 ให้พ้องลักษณ์เหมือน dot product (ไม่ให้ใช้ dot product)
- แล้ว test กับ matrix ขนาด
- (2,3)*(3,2)
- (4,4)*(4,1)
- (2,2)*(2,2)

11

12

การสร้างฟังก์ชั่น 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

ขั้นตอนการสร้างฟังก์ชั่น คูณ matrix

```
def mat_mul(A,B):
```

- คือการสร้างและกำหนดเครื่องฟังก์ชั่น, input ที่ต้องการ matrix A * matrix B

```
C = np.zeros((A.shape[0],B.shape[1]))
```

- หมายความว่า ให้สร้าง matrix ที่มีทุกตาเป็น 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 ใช้วนลูปง่ายหน้าใน range(A.shape[0]) ที่ละตัว เก็บไว้ในตัวแปร r_a

range คือคำสั่งที่ใช้สร้าง list ตัวแรก
.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 ใช้วนลูปง่ายหน้าใน 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 ใช้วนลูปง่ายหน้าใน range(A.shape[1]) ที่ละตัว เก็บไว้ในตัวแปร every_element

16

17

ลำดับการทำงาน loop for ในฟังก์ชั่น

- 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 ในฟังก์ชั่น

- 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 จะทำงานวนลูปของตัวเองจนจบทุกรอบก่อนถึงนับเป็นวนลูป 1 รอบของ loop2 และ loop2 จะทำงานวนลูปของตัวเองจนจบทุกรอบก่อนถึงนับเป็นวนลูป 1 รอบของ loop1

18

19

การคำนวณภายใน loop3

```
C[r_a,c_b] = C[r_a,c_b] + (A[r_a,every_element] * B[every_element,c_b]
```

- ในการทำงานครั้งแรกของฟังก์ชั่นค่าของ `C[r_a,c_b]` จะเท่ากับ 0
- เพราะสร้าง `C = np.zeros((A.shape[0],B.shape[1]))`
- `A[r_a,every_element]` คือ ค่าของ matrix ตัวหน้าตามตำแหน่ง `[r_a,every_element]` เช่น
- ถ้า A คือ

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

- `A[0,0]` จะเท่ากับ 1
- `A[0,1]` จะเท่ากับ 2

19

20

การคำนวณภายใน loop3

- `B[every_element,c_b]` คือ ค่าของ matrix ตัวหลังตามตำแหน่ง `[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

20

21

การคำนวณการคูณ matrix

- ดังนั้น

```
C[r_a,c_b] = C[r_a,c_b] + (A[r_a,every_element] * B[every_element,c_b]
```

- ในการทำงานรอบแรกของ loop1
- `C[r_a,c_b]` จะเป็นตัวแปรที่เข้ามารับค่าผลลัพธ์จากการคำนวณเพื่อนำค่าไปบวกใน loop รอบถัดไป 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 รอบถัดไป
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 รอบถัดไปจนจบ C[0,0] และไปเริ่ม C[0,1]
```

21

22

วิธีใช้งานฟังก์ชั่น

- 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]])
```

22

23

ผลลัพธ์ของฟังก์ชั่น

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

```
array([[ 58.,  64.],
       [139., 154.]])
```

- ซึ่งถ้าเปรียบเทียบตัวแปรในฟังก์ชั่นคือตัวแปรที่เข้ามารับผลลัพธ์การคำนวณในแต่ละรอบของ loop1

```
[[C[0,0], C[0,1]],
 [C[1,0], C[1,1]]]
```

23

24