Class period 5

More_advanced_data_structure 2

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.int8`. Default is `numpy.float64`.

order : *{'C', 'F'}, optional, default: 'C'*

Whether to store multi-dimensional 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. If an array-like passed in as `like` supports the `__array_function__` 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.

New in version 1.20.0.

Returns: **out** : *ndarray*

Array of zeros with the given shape, dtype, and order.

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=None, order='C', *, like=None)` [source]

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.int8`. Default is `numpy.float64`.

order : *{'C', 'F'}, optional, default: C*

Whether to store multi-dimensional 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. If an array-like passed in as `like` supports the `__array_function__` 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.

New in version 1.20.0.

Returns:

out : *ndarray*

Array of ones with the given shape, dtype, and order.

Matrix Operation (scalar multiplication)

- scalar multiplication คือ การคูณค่าคงที่ 1 ค่าที่เป็นเลขตัวเดียวคูณเข้าใน matrix เช่น
- `M_one = np.ones((2,3))`
- มุมมองค่าภายใน
- `array([[1., 1., 1.],`
- `[1., 1., 1.]])`

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

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

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

`random.rand(d0, d1, ..., dn)`

Random values in a given shape.

> **Note**
>
> This is a convenience function for users porting code from Matlab, and wraps `random_sample`. That function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like `numpy.zeros` and `numpy.ones`.

Create an array of the given shape and populate it with random samples from a uniform distribution over `[0, 1)`.

Parameters: d0, d1, ..., dn : *int, 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* `(d0, d1, ..., dn)`

Random values.

> **See also**
>
> `random`

คำสั่ง numpy.random.randn

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

`random.randn(d0, d1, ..., dn)`

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

> **Note**
>
> This is a convenience function for users porting code from Matlab, and wraps `standard_normal`. That function takes a tuple to specify the size of the output, which is consistent with other NumPy functions like `numpy.zeros` and `numpy.ones`.

> **Note**
>
> New code should use the `standard_normal` method of a `Generator` instance instead; please see the Quick Start.

If positive int_like arguments are provided, `randn` generates an array of shape `(d0, d1, ..., dn)`, filled with random floats sampled from a univariate "normal" (Gaussian) distribution of mean 0 and variance 1. A single float randomly sampled from the distribution is returned if no argument is provided.

Parameters: **d0, d1, ..., dn** : *int, optional*

The dimensions of the returned array, must be non-negative. If no argument is given a single Python float is returned.

Returns: **Z** : *ndarray or float*

A `(d0, d1, ..., dn)`-shaped array of floating-point samples from the standard normal distribution, or a single such float if no parameters were supplied.

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 a random sample from a given 1-D array

New in version 1.7.0.

Note

New code should use the `choice` method of a `Generator` instance instead; please see the Quick Start.

Parameters: a : *1-D array-like or int*

If an ndarray, a random sample is generated from its elements. If an int, the random sample is generated as if it were `np.arange(a)`

size : *int or tuple of ints, optional*

Output shape. If the given shape is, e.g., `(m, n, k)`, then `m * n * k` samples are drawn. Default is None, in which case a single value is returned.

replace : *boolean, 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: samples : *single item or ndarray*

The generated random samples

ตัวอย่างการใช้งาน Parameter: size numpy.random.choice

- Parameter: size เป็นการกำหนดจำนวนค่าที่ต้องการสุ่มออกมา โดย 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]])`

ตัวอย่างการใช้งาน Parameter: replace numpy.random.choice

- Parameter: replace การใส่คืน เป็นการกำหนดว่า จะให้นำค่าที่สุ่มออกมาใส่คืนหรือไม่ใส่คืน คือการสุ่มค่าซ้ำหรือไม่ซ้ำ
- โดยค่า default ของ replace = True คือการสุ่มแบบใส่คืนมีค่าซ้ำกันได้ ถ้าต้องการสุ่มแบบไม่ใส่คืน คือให้ไม่ค่าซ้ำกัน ให้กำหนด replace = Fasle

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

ตัวอย่างการใช้งาน Parameter: p numpy.random.choice

- Parameter: p เป็นการกำหนดความน่าจะเป็นที่ค่าแต่ละค่าจะถูกสุ่ม
- เช่น มีนักศึกษา 35 คนต้องการสุ่มเกรดให้แต่ละคนตามความน่าจะเป็นที่กำหนด
- 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', 'B', 'C', 'A', 'C', 'C', 'C', 'D',
•        'A', 'D', 'C', 'D', 'B', 'D', 'C', 'D', 'C'], dtype='<U1')
```

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

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
```

ขั้นตอนการสร้างฟังก์ชั่น คูณ matrix

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

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 ใช้วนลูปอ่านค่าใน `range(A.shape[0])` ทีละตัว เก็บไว้ในตัวแปร `r_a`

`range` คือคำสั่งที่ใช้สร้าง `list` ตัวเลข
`.shape` คือคำสั่งในการตรวจสอบขนาดของ `matrix`

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 ใช้วนลูปอ่านค่าใน `range(B.shape[1])` ทีละตัว เก็บไว้ในตัวแปร `c_b`

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 ใช้วนลูปอ่านค่าใน `range(A.shape[1])` ทีละตัว เก็บไว้ในตัวแปร `every_element`

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}')`

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 จะทำงานวนลูปของตัวเองจนจบทุกรอบก่อน ถึงนับเป็นวนลูป 1 รอบของ loop2 และ loop2 จะทำงานวนลูปของตัวเองจนจบทุกรอบก่อน ถึงนับเป็นวนลูป 1 รอบของ loop1

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

การคำนวณภายใน loop3

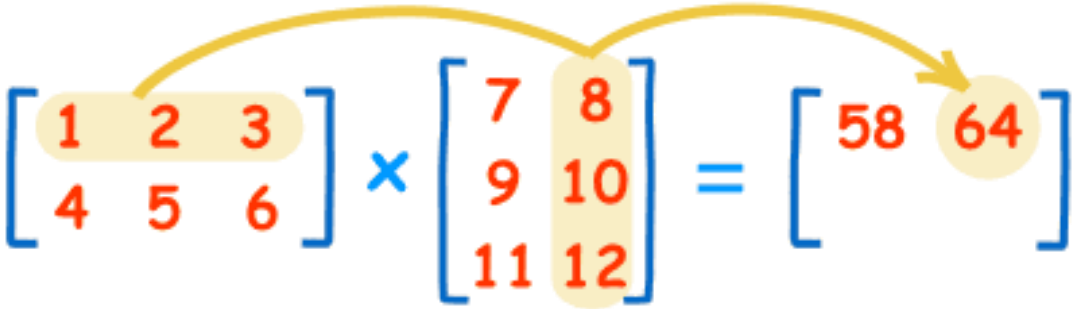
- B[every_element,c_b] คือ ค่าของ matrix ตัวหลังตามตำแหน่ง [every_element,c_b] เช่น
- ถ้า B คือ
- array([[7, 8],
- [9, 10],
- [11, 12]])
- B[0,0] จะเท่ากับ 7
- B[1,0] จะเท่ากับ 9

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]

วิธีใช้งานฟังก์ชั่น

- สร้าง matrix input ตามตัวอย่าง




```
• 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]])
```

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]]]