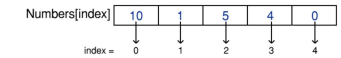
# Class period 4

บทที่ 3 โปรแกรมวนซ้ำและการใช้เงื่อนไขในภาษาไพธอน (ต่อ)

More_advanced_data_structure 1

1

## ทบทวน List (เรียนไปแล้ว)




- List ดีกว่า Array ยังไง
- 1. เก็บข้อมูลได้หลายประเภท int str float
- 2. มี list slicing



2

## Dictionary

สามารถกำหนด index ให้ค่าที่ต้องการได้ และสามารถกำหนด index เป็นอะไรก็ได้ และลำดับจะหายไป

```
ex_dict = {'a':10, '1':1, 'stat':5}
#curly brackets {} ใช้ครอบหัวนิ dict
ex_dict['stat']
```

ผลลัพธ์จะได้

```
5
```



3

## ข้อกำหนดในการใช้งาน Dictionary

- 2:'two'
- ตัวหน้า : คือ key หรือ index (ตัวชี้)
- ตัวหลัง : คือ value (ค่า)
- ไม่สามารถใช้ value ในการชี้กลับไปที่ index

```
ex2_dict['two']
```

- ผลลัพธ์จะได้

```
KeyError: 'two'
```



4

## ตัวอย่างการใช้งาน Dictionary

```
ex2_dict = {1:'one', 2:'two', 3:'three'}
ex2_dict[2]
```

ผลลัพธ์จะได้

```
'two'
```



5

## การเพิ่มสมาชิกใน dict

- สามารถกำหนด index (ตัวชี้) และ value (ค่า) ที่ต้องการได้เลย เช่น

```
ex2_dict[0] = 'zero'
```

- ต้องการเพิ่ม index 0 ชี้ไปที่ value 'zero' เข้าไปใน dict ex2_dict

```
ex2_dict
```

- ผลลัพธ์จะได้

```
{0: 'zero', 1: 'one', 2: 'two', 3: 'three'}
```



6

## คำสั่งที่ใช้บ่อยของ dict

- **.keys()** คือคำสั่งที่ใช้ตรวจสอบ keys ใน dict ว่ามี index (ตัวชี้) อะไรบ้าง

```
ex2_dict.keys()
```

- ผลลัพธ์จะได้

```
dict_keys([1, 2, 3, 0])
```

- **.values()** คือคำสั่งที่ใช้ตรวจสอบ values ใน dict ว่ามี values (ค่า) อะไรบ้าง

```
ex2_dict.values()
```

- ผลลัพธ์จะได้

```
dict_values(['one', 'two', 'three', 'zero'])
```



7

## ตัวอย่างการใช้งานคำสั่ง .keys()และ .values()

- สามารถใช้งานร่วมกับ loop for เพื่อนำ keys หรือ values ไปใช้งานต่อ เช่น

```
for index in ex2_dict.keys():
    print(index)
```

- ผลลัพธ์จะได้

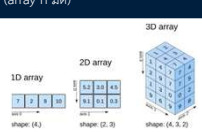
```
1
2
3
0
```

***dict เอาไปใช้ร่วมกับ Dataframe ของ pandas



8

## Numpy Array (array n มิติ)




- numpy คือ package ที่ทำงานเกี่ยวกับ array ของตัวเลขเท่านั้น (ที่มีคนเขียนขึ้นมาและนิยมใช้)
- numpy ย่อมาจาก number python



9

## วิธีเรียกใช้งาน package

- สามารถใช้ import ตามด้วยชื่อ package ของ python ที่ต้องการใช้งาน เช่น
- เรียกใช้งาน numpy
- import numpy

- สามารถใช้ as ต่อท้ายชื่อ package เพื่อตั้งชื่อใหม่ให้ง่ายต่อการใช้งาน เช่น
- import numpy as np

9

10

## การแปลง list ให้เป็น numpy array

- เปลี่ยน list ให้เป็น array 2 มิติ ด้วยคำสั่ง numpy.array()
- `ex_2d_array=numpy.array([[5.2,3.0,4.5],[9.1,0.1,0.3]])`
- `print(ex_2d_array)`

- ผลลัพธ์จะได้ การสร้าง matrix
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

10

11

## การตรวจสอบขนาดของ matrix

- ใช้คำสั่ง .shape ตามท้ายชื่อตัวแปรที่ใช้เก็บ matrix ในการตรวจสอบขนาด เช่น
- `ex_2d_array=numpy.array([[5.2,3.0,4.5],[9.1,0.1,0.3]])`

- `ex_2d_array.shape`
- ผลลัพธ์จะได้
- `(2, 3)`
- หมายความว่า เป็น matrix 2 แถว 3 หลัก
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

11

12

## การชี้ค่าใน numpy array

- `ex_2d_array= [[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

- ถ้าต้องการชี้ไปที่ค่า 0.3 ใน ex_2d_array
- โดยเข้าดูจากรูปหรือ matrix ค่า 0.3
- จะอยู่ในแถวที่ 1 หลักที่ 2 (เริ่มนับจาก 0) ดังนั้น
- `ex_2d_array[1,2]`
- ผลลัพธ์จะได้
- `0.3`

2D array


shape: (2, 3)

12

13

## การชี้ค่าใน list ก่อนที่จะเปลี่ยนเป็น numpy array

- `list_x = [[5.2,3.0,4.5],[9.1,0.1,0.3]]`

- ถ้าต้องการชี้ไปที่ค่า 0.3 ใน list_x
- `print(len(list_x))` ผลลัพธ์จะได้ 2 หมายความว่า list_x มีสมาชิก 2 ตัว คือ
- [5.2,3.0,4.5] และ [9.1,0.1,0.3]
- `print(list_x[1])` ผลลัพธ์จะได้ [9.1, 0.1, 0.3] ดังนั้น
- `print(list_x[1][2])` ผลลัพธ์จะได้ 0.3 คือชี้ไปที่สมาชิกตัวที่ 1 ของ list_x และชี้ไปที่สมาชิกตัวที่ 2 ของ list_x[1] ก็จะได้ list_x[1][2] คือ 0.3

13

14

## Operations

- numpy array หรือ matrix สามารถนำมา บวก ลบ คูณ หาร กันได้
- สร้าง matrix ใหม่อีกอันเก็บไว้ในตัวแปร ex2_2d_array
- `ex2_2d_array = numpy.array([[1,0,0],[0,0,1]])`

- `print(ex_2d_array)` (matrix#1)
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`
- `print(ex2_2d_array)`
- `[[1 0 0]`
- `[0 0 1]]`

14

15

## ตัวอย่างใช้งานการบวก matrix

- การบวก matrix คือการเอาค่าตำแหน่งเดียวกันมาบวกกัน
- `ex_2d_array + ex2_2d_array`
- มุมมองทางภาษาใน
- `[[5.2 3.  4.5] + [[1 0 0]`
- `[9.1 0.1 0.3]]  [0 0 1]]`

- ผลลัพธ์จะได้
- `array([[6.2, 3. , 4.5],`
- `[9.1, 0.1, 1.3]])`

15

16

## การบวกค่าใน list ก่อนที่จะเปลี่ยนเป็น matrix

- `list_x = [[5.2,3.0,4.5],[9.1,0.1,0.3]]`
- `list_x2 = [[1,0,0],[0,0,1]]`
- `list_x + list_x2`

- ผลลัพธ์จะได้
- `[[5.2, 3.0, 4.5], [9.1, 0.1, 0.3], [1, 0, 0], [0, 0, 1]]`

- จะเห็นว่า list ไม่สามารถบวกเลขตำแหน่งเดียวกันแบบบวก matrix ใน numpy array
- งานปกติทั่วไปสามารถใช้ List ได้ แต่งานที่เกี่ยวกับตัวเลขจะใช้ numpy array

16

17

## ตัวอย่างใช้งานการลบ matrix

- `ex_2d_array - ex2_2d_array`
- มุมมองทางภาษาใน
- `[[5.2 3.  4.5] - [[1 0 0]`
- `[9.1 0.1 0.3]]  [0 0 1]]`

- ผลลัพธ์จะได้
- `array([[ 4.2,  3. ,  4.5],`
- `[ 9.1,  0.1, -0.7]])`

17

18

## ตัวอย่างใช้งานการคูณ matrix

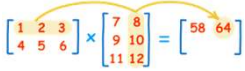
- การคูณ matrix ใน numpy array จะคูณในแบบ array เอาตำแหน่งเดียวกันมาคูณกัน
- `ex_2d_array * ex2_2d_array`
- มุมมองภายใน
- `[[5.2 3.  4.5] * [[1 0 0]`
- `[9.1 0.1 0.3]]    [0 0 1]]`

- ผลลัพธ์จะได้
- `array([[5.2, 0. , 0. ],`
- `[0. , 0. , 0.3]])`

18

19

## matix multiplication (dot product)




- การคูณ matrix ใน numpy array ที่ถูกต้อง จะใช้คำสั่ง numpy.dot ตามด้วยตัวแปรที่เก็บค่า matrix ที่ต้องการคูณ(ตัวแปร1, ตัวแปร2)

19

20

## ตัวอย่างการคูณ matrix (dot product)

- `numpy.dot(ex_2d_array,ex2_2d_array)`
- มุมมองภายใน
- `[[5.2 3.  4.5] * [[1 0 0]`
- `[9.1 0.1 0.3]]    [0 0 1]]`

- ผลลัพธ์จะได้
- `ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)`

20

21

## ตัวอย่างการคูณ matrix (dot product)

- หลักการการคูณ matrix คือ หลักของตัวหน้าต้องเท่ากับแถวของตัวหลัง ดังนั้นจะต้อง transpose matrix (การกลับหลักเป็นแถวกลับแถวเป็นหลัก)
- สามารถ transpose matrix ที่ต้องการได้โดยใช้คำสั่ง ชื่อตัวแปรที่เก็บค่า matrix ตามด้วย .T
- `print(ex_2d_array)`
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

- `print(ex_2d_array.T)`
- `[[5.2 9.1]`
- `[3.  0.1]`
- `[4.5 0.3]]`

21

22

## ตัวอย่างการคูณ matrix (dot product)

- ดังนั้น การคูณ matrix ใน numpy array ที่ถูกต้อง
- `dot_mat = numpy.dot(ex_2d_array,ex2_2d_array.T)`
- มุมมองภายใน
- `[[5.2 3.  4.5] * [[1 0]`
- `[9.1 0.1 0.3]]    [0 0]`
- `[0 1]]`
- `print(dot_mat)`
- ผลลัพธ์จะได้
- `[[5.2 4.5]`
- `[9.1 0.3]]`

22

23

## การหา det ของ matrix

- ใช้คำสั่ง `numpy.linalg.det` ตามด้วย (ตัวแปรที่ต้องการหา)
- `numpy.linalg.det(dot_mat)`
- มุมมองภายใน
- `[[5.2 4.5]`
- `[9.1 0.3]]`

- ผลลัพธ์จะได้
- `-39.389999999999986`

23

24

## matrix slicing

- `print(ex_2d_array)`
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

- สามารถตัดได้เหมือน list
- `ex_2d_array[1,1:]` หมายความให้เอาค่าสมาชิกแถวที่1 หลักที่1 ไปจนถึงหลักสุดท้าย
- ผลลัพธ์จะได้
- `array([0.1, 0.3])`

24

25

## ตัวอย่าง matrix slicing

- `print(ex_2d_array)`
- `[[5.2 3.  4.5]`
- `[9.1 0.1 0.3]]`

- `ex_2d_array[:,:2]` หมายความให้เอาค่าสมาชิกแถวแรกหลักแรกไปจนถึงหลักที่ 1
- ผลลัพธ์จะได้
- `array([[5.2, 3. ],`
- `[9.1, 0.1]])`

25

26

## Homework class period 4

- 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)

26

27