

LECTURE 2-3_3

PYTHON PROGRAMMING

CODES AND STRUCTURES

DR. PRAPASSORN TANTIPHANWADI

INDUSTRIAL ENGINEERING, FACULTY OF ENGINEERING AT KHAMPAENGSAEN

DECEMBER 2565

การเรียกใช้งาน BUILT-IN MODULES

➤ **Built-in module:** Panda, Numpy,...,etc

➤ **import**

```
import ชื่อโมดูล  
ชื่อโมดูล.ชื่อฟังก์ชันในโมดูล
```

ตัวอย่างเช่น

```
import pandas  
  
pandas.read_csv("custcontact.csv")
```

➤ **From Import module**

เป็นการเลือกโหลดเฉพาะฟังก์ชันที่ต้องการใช้งาน
ของโมดูลเข้ามายังโปรแกรม ก่อนเรียกใช้งาน
ฟังก์ชันนั้นๆ ให้ทำงาน โดยมีรูปแบบดังนี้

```
from ชื่อโมดูล import ชื่อฟังก์ชันที่ต้องการใช้งาน  
ชื่อฟังก์ชันในโมดูล
```

ตัวอย่างเช่น

```
from pandas import read_csv  
  
read_csv("custcontact.csv")
```

LIBRARY

➤ NumPy

NumPy จะมีการทำงานในรูปแบบออบเจ็คของ Array ที่เรียกว่า ndarray แทนรูปแบบอาร์เรย์ หลายมิติของชุดข้อมูลชนิดเดียวกันที่กำลังนำมาคำนวณ คล้ายคลึงกับออบเจ็ค List ซึ่งเป็น ออบเจ็คแบบอาร์เรย์ของภาษา Python พื้นฐาน แต่ต่างกันตรงที่ ndarray ไม่สามารถเพิ่มขนาดได้ อัตโนมัติหลังจากสร้างแล้วซึ่งแตกต่างจาก List โดยการเพิ่มหรือลดสมาชิกใน ndarray จะทำให้เกิด การสร้างออบเจ็ค ndarray ตัวใหม่และลบตัวเก่าไป

➤ การติดตั้ง NumPy

โดยปกติไลบรารี NumPy ไม่ใช่ไลบรารีพื้นฐานของไพธอน หากต้องการใช้งานจะต้องติดตั้ง ไลบรารี NumPy เพิ่มเติมเสียก่อน แต่ถ้าติดตั้งโปรแกรม Anaconda แล้ว ก็จะมีไลบรารี NumPy ติดตั้งมาให้โดยอัตโนมัติ สามารถเรียกใช้งานได้ทันที

➤ การสร้าง ndarray

โครงสร้างพื้นฐานของการทำงานด้วย NumPy คือ ndarray ซึ่งแทนข้อมูลประเภทอาร์เรย์ตั้งแต่ หนึ่งมิติจนถึงหลายมิติ หากต้องการสร้างโมเดลหรือคำนวณชุดข้อมูลเพื่อวิเคราะห์ค่าต่างๆ จึงต้อง นำชุดข้อมูลมาสร้างเป็น ndarray เสียก่อน

การสร้าง ndarray ทำได้โดยเรียกใช้ฟังก์ชัน `array()` ซึ่งสามารถนำข้อมูลในรูปแบบ List หรือ Tuple มาสร้าง ndarray ได้ เช่น

ตัวอย่างที่ 4.1

```
import numpy as np

a = np.array((1,2,3,4)) # สร้าง ndarray จากข้อมูลแบบ Tuple
list_a = [1,2,3,4]       # List
b = np.array(list_a)      # สร้าง ndarray จากข้อมูลแบบ List

print(a)
print(b)
```

ซึ่งจะได้ผลลัพธ์เช่นเดียวกัน คือ

```
[1 2 3 4]    # a
[1 2 3 4]    # b
```

ทั้งนี้เรายสามารถสร้าง ndarray จากข้อมูลแบบ List และ Tuple รวมกันได้ หรือจะสร้างจากอาร์เรย์หลายมิติก็ได้ เช่น

ตัวอย่างที่ 4.2

```
import numpy as np

a = np.array([[1,2,3],[10,20,30]]) # อาร์เรย์ 2 มิติ (List ช้อน List)
b = np.array(('a','b','c',))       # Tuple
c = np.array([[1.5, 3.0],[2.5,5.0],(3.5,7.0)]) # List และ Tuple

print(a)
print(b)
print(c)
```

ผลลัพธ์

```
[[ 1  2  3]          # a
 [10 20 30]]
 ['a' 'b' 'c']         # b
 [[1.5 3. ]           # c
 [2.5 5. ]
 [3.5 7. ]]
```

นอกจากฟังก์ชัน array() แล้ว NumPy มีฟังก์ชันชื่อร่วงการสร้างออบเจ็ค ndarray โดยให้ระบุสร้างมิติและกำหนดค่าเริ่มต้นของข้อมูลในอาร์เรย์ในแบบต่างๆ ดังนี้

zeros() และ ones()

ฟังก์ชัน zeros() และ ones() จะรับพารามิเตอร์เป็นมิติของอาร์เรย์ที่ต้องการสร้าง โดย zeros() จะสร้าง ndarray ที่มีมิติตามที่ระบุซึ่งสมาชิกทุกตัวจะมีค่าเป็น 0 ส่วน ones() จะมีการทำงานเช่นเดียวกับ zeros() แต่สมาชิกทุกตัวจะมีค่าเป็น 1

ฟังก์ชัน zeros() หรือ ones() มีรูปแบบดังนี้

```
zeros( มิติของอาร์เรย์ [ , ชนิดข้อมูล])
```

```
ones( มิติของอาร์เรย์ [ , ชนิดข้อมูล])
```

ทั้งนี้ค่าของสมาชิกใน ndarray ผลลัพธ์ที่สร้างจาก zero() และ ones() จะมีชนิดข้อมูลเป็น float

ตัวอย่างที่ 4.3

```
import numpy as np

a = np.zeros((3,3)) # สร้าง ndarray 3 แก้ 3 คอลัมน์
b = np.ones((2,5)) # สร้าง ndarray 2 แก้ 5 คอลัมน์

print(a)
print(b)

ผลลัพธ์

[[0. 0. 0.] # a
 [0. 0. 0.]
 [0. 0. 0.]]
[[1. 1. 1. 1. 1.] # b
 [1. 1. 1. 1. 1.]]
```

ตัวอย่างที่ 4.4

```
import numpy as np

a = np.zeros((3,3),dtype=int)

print(a)

ผลลัพธ์

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

full()

`full()` จะมีหลักการทำงานเช่นเดียวกับ `zeros()` และ `ones()` แต่ค่าของสมาชิกทุกตัวใน ndarray ผลลัพธ์จะเป็นตามที่กำหนดไว้ แต่หากกำหนดชนิดข้อมูลเป็นพารามิเตอร์ตัวสุดท้าย ค่าของสมาชิกจะเปลี่ยนเป็นชนิดข้อมูลตามที่กำหนด

ฟังก์ชัน `full()` มีรูปแบบดังนี้

```
full( มิติของอาร์เรย์, ค่าข้อมูลของอาร์เรย์ [, ชนิดข้อมูล])
```

ตัวอย่างที่ 4.5

```
import numpy as np  
  
a = np.full((3,3),5.0,dtype=int)  
  
print(a)
```

ผลลัพธ์

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

arange()

arange() เป็นฟังก์ชันสำหรับการสร้าง ndarray มิติเดียวให้มีสมาชิกเป็นตัวเลขเรียงลำดับนับจากค่าเริ่มต้นไม่เกินค่าสิ้นสุดที่กำหนด โดยค่าสิ้นสุดจะไม่รวมใน ndarray ที่เป็นผลลัพธ์ ซึ่งหากต้องการทำให้เป็น ndarray หลายมิติสามารถใช้ฟังก์ชัน reshape() ในการปรับเป็น ndarray หลายมิติในภายหลังได้

เราสามารถสร้าง ndarray จากฟังก์ชัน arange() โดยกำหนดค่าพารามิเตอร์ในรูปแบบดังนี้

```
arrange( [ค่าเริ่มต้น,] ค่าสิ้นสุด [, ผลต่างระหว่างข้อมูลสมาชิก] [, ชนิดข้อมูล])
```

ทั้งนี้ สมาชิกตัวสุดท้ายของ ndarray ผลลัพธ์จะไม่เท่ากับและไม่เกินค่าสิ้นสุดเสมอ

ตัวอย่างที่ 4.6

คำอธิบาย	ตัวอย่าง	ผลลัพธ์
arange(ค่าสิ้นสุด)		
สร้างสมาชิกตั้งแต่ 0 จนถึงค่าก่อนค่าสิ้นสุด	np.arange(10)	[0 1 2 3 4 5 6]
arange(ค่าเริ่มต้น, ค่าสิ้นสุด)		
สร้างสมาชิกตั้งแต่ค่าเริ่มต้นจนถึง ค่าก่อนค่าสิ้นสุด	np.arange(3,15)	[3 4 5 6 7 8 9 10]
arange(ค่าเริ่มต้น, ค่าสิ้นสุด, ผลต่างระหว่างข้อมูลสมาชิก)		
สร้างสมาชิกตั้งแต่ค่าเริ่มต้นจนถึง ค่าก่อนค่าสิ้นสุด แต่ละสมาชิกจะ มีผลต่างตามที่กำหนด	np.arange(2,10,2)	[2 4 6 8]
arange(ค่าสิ้นสุด, ชนิดข้อมูล)		
สร้างสมาชิกตั้งแต่ 0 จนถึงค่า ก่อนค่าสิ้นสุด สมาชิกจะมีชนิด ข้อมูลตามที่กำหนด	np.arange(4,dtype=float)	[0. 1. 2. 3.]
คำอธิบาย	ตัวอย่าง	ผลลัพธ์
arange(ค่าเริ่มต้น, ค่าสิ้นสุด, ชนิดข้อมูล)		
สร้างสมาชิกตั้งแต่ค่าเริ่มต้นจนถึง ค่าก่อนค่าสิ้นสุด สมาชิกจะมีชนิด ข้อมูลตามที่กำหนด	np.arange(1,10,dtype=float)	[1. 2. 3. 4. 5. 6. 7. 8. 9.]
arange(ค่าเริ่มต้น, ค่าสิ้นสุด, ผลต่างระหว่างข้อมูลสมาชิก, ชนิดข้อมูล)		
สร้างสมาชิกตั้งแต่ค่าเริ่มต้นจนถึง ค่าก่อนค่าสิ้นสุด แต่ละสมาชิก จะมีชนิดข้อมูลและผลต่างตามที่ กำหนด	np.arange(1,10,2,dtype=float)	[1. 3. 5. 7. 9.]

ทั้งนี้การกำหนดค่าในการเรียกใช้งานฟังก์ชัน `arrange()` มีข้อกำหนดดังนี้

- นอกจากพารามิเตอร์ ค่าสิ้นสุด แล้ว ค่าอื่นจะได้หรือไม่ก็ได้
- ผลต่างระหว่างข้อมูลสมาชิก ไม่สามารถเป็น 0 ได้ ซึ่งจะทำให้เกิด `ZeroDivisionError`
- ผลต่างระหว่างข้อมูลสมาชิก หากไม่กำหนดจะมีค่าเป็น 1
- ชนิดข้อมูล จะต้องเป็นประเภทตัวเลขเท่านั้น ได้แก่ `uint`, `int` และ `float` ขนาด 8, 16, 32 หรือ 64 bit ได้ ซึ่งหากไม่กำหนดจะถือเป็น 64
- หากไม่กำหนดชนิดข้อมูล NumPy จะสร้างสมาชิกตามชนิดข้อมูล อย่างเช่น กำหนดค่าเริ่มต้นและค่าผลต่างระหว่างสมาชิกเป็น

```
np.arange(9,15,5,2)
```

ผลลัพธ์ จะได้สมาชิกใน ndarray เป็น float ดังนี้

```
[ 9. 11. 13. 15.]
```

- ค่าตั้งต้น และค่าสิ้นสุด สามารถเป็นค่าลบได้ ตัวอย่างเช่น

```
np.arange(-3,2)
```

ผลลัพธ์

```
[-3 -2 -1  0  1]
```

- ค่าตั้งต้นต้องน้อยกว่าค่าสิ้นสุดเสมอ หากค่าตั้งต้นมากกว่าหรือเท่ากับค่าสิ้นสุด จะทำให้ได้ผลลัพธ์เป็น ndarray ว่าง ตัวอย่างเช่น

```
np.arange(5,2)
```

ผลลัพธ์

```
[]
```

- หากต้องการให้สมาชิกเรียงลำดับจากมากไปน้อยลดลงไปเรื่อยๆ ให้กำหนดค่าผลต่างระหว่างชนิดข้อมูลมีค่าเป็นลบ ตัวอย่างเช่น

```
np.arange(5,2,-2)
```

ผลลัพธ์

```
[5 3]
```

linspace()

`linspace()` เป็นฟังก์ชันสำหรับการสร้าง ndarray มิติเดียวให้มีสมาชิกเป็นตัวเลขเรียงลำดับตั้งแต่ค่าเริ่มต้นจนถึงค่าสิ้นสุดคล้าย `arange()` แต่จะมีข้อแตกต่างคือ `linspace()` จะสร้างจำนวนสมาชิกใน ndarray ตามจำนวนที่กำหนดและค่าสิ้นสุดจะเป็นสมาชิกตัวสุดท้ายของผลลัพธ์ด้วย

`linspace()` จะมีประโยชน์ในการใช้สร้าง ndarray ที่การันตีให้แต่ละสมาชิกใน ndarray มีระยะห่างเท่าๆ กัน และมีจำนวนตามที่กำหนด

เราสามารถสร้าง ndarray จากฟังก์ชัน `linspace()` โดยกำหนดค่าพารามิเตอร์ในรูปแบบดังนี้

`linspace(ค่าเริ่มต้น, ค่าสิ้นสุด [, จำนวนข้อมูลสมาชิก] [, ชนิดข้อมูล])`

การใช้งาน `linspace()` จะ

- ต้องระบุค่าเริ่มต้นและค่าเรียงลำดับจากมากไปน้อย
 - หากระบุจำนวนข้อมูล
 - จำนวนข้อมูลสมาชิก หากไม่ระบุ จะมีค่าเริ่มต้นเป็น 50 และไม่สามารถใส่ค่าลบได้
- สมาชิกของ ndarray อาจมีชนิดข้อมูลเปลี่ยนแปลงได้ หากค่าเริ่มต้นหรือค่าสิ้นสุดเป็น int แต่หารไม่ลงตัวกับจำนวนข้อมูลที่ต้องการ
 - หากผลลัพธ์เป็น float แต่ระบุชนิดข้อมูลเป็น int จะทำให้ NumPy ปัดเศษเป็นค่า int และมีสมาชิกที่ค่าซ้ำกันได้

➤ การใช้

ตัวอย่างที่ 4.7

ตัวอย่าง	ผลลัพธ์
<code>np.linspace(1,4)</code>	[1. 1.06122449 1.12244898 1.18367347 1.24489796 1.30612245 1.36734694 1.42857143 1.48979592 1.55102041 1.6122449 1.67346939 1.73469388 1.79591837 1.85714286 1.91836735 1.97959184 2.04081633 2.10204082 2.16326531 2.2244898 2.28571429 2.34693878 2.40816327 2.46938776 2.53061224 2.59183673 2.65306122 2.71428571 2.7755102 2.83673469 2.89795918 2.95918367 3.02040816 3.08163265 3.14285714 3.20408163 3.26530612 3.32653061 3.3877551 3.44897959 3.51020408 3.57142857 3.63265306 3.69387755 3.75510204 3.81632653 3.87755102 3.93877551 4.]
<code>np.linspace(2,5,7)</code>	[2. 2.5 3. 3.5 4. 4.5 5.]
<code>np.linspace(2,5,7,dtype=int)</code>	[2 2 3 3 4 4 5]
<code>np.linspace(11,3,7)</code>	[11. 9.66666667 8.33333333 7. 5.66666667 4.33333333 3.]

eye()

`eye()` เป็นฟังก์ชันสำหรับการสร้าง `ndarray` แบบ $m \times n$ มิติ ซึ่งจะได้ผลลัพธ์เป็นอาร์เรย์แบบ เมตริกซ์ทแยงมุน (Diagonal Matrix) ที่มีค่าแนวทแยงมุนเป็น 1.0 และสมาชิกอื่นๆ มีค่าเป็น 0

`eye()` มีการกำหนดค่าพารามิเตอร์ในรูปแบบดังนี้

```
eye(จำนวนมิติหรือแถว [, จำนวนคอลัมน์] [, ตำแหน่งเริ่มทแยงมุน] [, ชนิดข้อมูล])
```

การใช้งาน `eye()` มีเงื่อนไขดังนี้

- หากไม่ระบุจำนวนคอลัมน์ จะใช้จำนวนแถวเป็นจำนวนคอลัมน์ด้วย ได้ผลลัพธ์เป็นอาร์เรย์ ที่มีมิติ $n \times n$
- หากกำหนดตำแหน่งเริ่มทแยงมุนเป็นค่ามากกว่า 0 จะหมายถึงให้เริ่มกำหนดค่าเป็น 1.0 ที่คอลัมน์ของแถวแรกก่อน จากนั้นໄล่กำหนดค่า 1.0 ตามแนวทแยงมุนลงไปด้านล่าง
- หากกำหนดตำแหน่งเริ่มทแยงมุนเป็นค่าน้อยกว่า 0 จะหมายถึงให้เริ่มกำหนดค่าเป็น 1.0 ที่คอลัมน์ของแถวสุดท้ายก่อน จากนั้นໄล่กำหนดค่า 1.0 ตามแนวทแยงมุนขึ้นไปด้านบน
- หากไม่ระบุจำนวนคอลัมน์ จะไม่สามารถระบุตำแหน่งเริ่มทแยงมุนได้

ตัวอย่าง	ผลลัพธ์
	<code>[[1. 0. 0.]</code>
	<code>[0. 1. 0.]</code>
	<code>[0. 0. 1.]]</code>
<code>(4,3,dtype=int)</code>	<code>[[1. 0 0 0]</code>
	<code>[0 1 0 0]</code>
	<code>[0 0 1 0]</code>
	<code>[0 0 0 1]]</code>
	<code>[[1. 0. 0.]</code>
	<code>[0. 1. 0.]</code>
	<code>[0. 0. 1.]]</code>

ตัวอย่าง	ผลลัพธ์
<code>4,3,2)</code>	<code>[[0. 1. 0.]</code>
ค่า 1.0 ที่คอลัมน์ที่ 2 รากก่อน	<code>[0. 0. 1.]</code>
	<code>[0. 0. 0.]</code>
	<code>[0. 0. 0.]]</code>
<code>4,3,-3,dtype=int)</code>	<code>[[0 0 0]</code>
ค่า 1 ที่คอลัมน์ที่ 3 ด้วยก่อน	<code>[1 0 0]</code>
	<code>[0 1 0]</code>
	<code>[0 0 1]]</code>

empty()

`empty()` เป็นฟังก์ชันสำหรับ `ndarray` แบบเลขสุ่ม โดยฟังก์ชัน `empty()` จะมีการใช้งานคล้ายฟังก์ชัน `zeros()` คือ รับพารามิเตอร์เพื่อกำหนดมิติของอาร์เรย์ผลลัพธ์ที่ต้องการและสามารถระบุชนิดข้อมูลได้ ซึ่ง NumPy จะใส่เลขแบบสุ่มตามจำนวนมิติและชนิดข้อมูลให้เอง

ฟังก์ชัน `empty()` มีรูปแบบดังนี้

```
empty( ฟิติของอาร์เรย์ [, ชนิดข้อมูล])
```

ตัวอย่างที่ 4.9

```
import numpy as np  
  
a = np.empty((3,3))  
b = np.empty((3,3), dtype=int)  
c = np.empty((3,3), dtype=int)
```

```
print(a)  
print(b)  
print(c)
```

ผลลัพธ์

```
[[1.24612081e-306 3.44898671e-307 4.00536382e-307] # a  
 [1.61323712e-307 2.67008863e-307 3.56035245e-307]  
 [1.61323966e-307 9.34609111e-307 3.22651328e-307]]  
 [[6815860 7209071 3604531] # b  
 [3342381 3866674 3801155]  
 [4849756 7733345 6029409]]  
 [[7536749 5046364 6488169] # c  
 [7274610 7274611 7602278]  
 [5636128 2097235 7274563]]
```

การเรียกใช้โมดูล Random

การเรียกใช้โมดูล Random สามารถเรียกใช้ได้สองแบบคือ

- เรียกใช้ผ่านออบเจ็ค Numpy ตั้งนี้

```
import numpy as np
```

```
a = np.random.randint(100)
```

```
print(a)
```

- Import โมดูล Random จากนั้นจึงเรียกใช้จากออบเจ็ค random ที่

```
from numpy import random as rd
```

```
a = rd.randint(100)
```

```
print(a)
```

rand()

เป็นฟังก์ชันที่ใช้สำหรับการสุ่มตัวเลขชนิด float โดยจะให้เลขสุ่มจำนวนหนึ่งตัวในช่วง 0 ถึง 1 นอกจากนี้สามารถใช้สร้างขอบเขต ndarray จากฟังก์ชัน rand() ได้โดยกำหนดจำนวนมิติที่ต้องการ 'ได้โดยใส่เป็นพารามิเตอร์'

ฟังก์ชัน rand() มีรูปแบบดังนี้

```
rand( [จำนวนมิติ ndarray ที่ต้องการ] )
```

ตัวอย่างที่ 4.10

```
from numpy import random as rd
a = rd.rand()
b = rd.rand(10,5,3) # ndarray มีติ 10x5x3
print(a)
print(b)
```

ผลลัพธ์

```
0.2665616784706025 # a
[[[0.27525795 0.14489728 0.05780794] # b
 [0.26281548 0.47250465 0.1956632 ]
 [0.8247254 0.55363337 0.99592935]
```

randint()

เป็นฟังก์ชันที่ใช้สำหรับการสร้าง ndarray จำนวนเต็มภายในช่วงค่าตั้งแต่ 0 ถึงค่าสูงสุดที่กำหนดได้
ฟังก์ชัน randint() มีรูปแบบดังนี้

```
randint( [ค่าสูงที่สุดที่เป็นไปได้] )
```

นอกจากนี้ยังสามารถสร้าง ndarray โดยกำหนดค่าในรูปแบบ Tuple เพื่อให้ NumPy ทำการ
สร้างขึ้นมาให้คล้ายเป็นสมาชิกของ ndarray เต็มความจุของ ndarray นั้น ซึ่งสามารถเรียกใช้
ในรูปแบบดังนี้

```
randint( [ค่าสูงที่สุดที่เป็นไปได้], size=[จำนวนมิติของ ndarray ที่ต้องการ] )
```

ตัวอย่างการใช้ฟังก์ชัน randint() เช่น หากต้องการสร้างออบเจ็ค ndarray ที่มีมิติ 10x5x3 จาก
ฟังก์ชัน randint() สามารถทำได้ดังนี้

ตัวอย่างที่ 4.11

```
from numpy import random as rd

a = rd.randint(100, size=(10,5,3))

print(a)
```

ผลลัพธ์

```
[[[23 45 52]
 [82 53 70]
 [23 75 80]
 [45 12 34]
 [34 65 23]
 [12 34 56]
 [56 78 90]
 [78 90 56]
 [90 56 78]
 [56 78 90]]]
```

choice()

เป็นฟังก์ชันที่ใช้สำหรับการสร้าง ndarray ด้วยการสุ่มตัวเลขขึ้นมาจากการเรียกดันทางข้อมูลให้
ครบตามจำนวนหรือมิติของ ndarray ที่กำหนด โดยสมาชิกของ ndarray ที่ได้จะเป็นตัวเลขที่เป็น
สมาชิกของอาร์เรย์ดันทางเท่านั้น ซึ่งสามารถใช้เดียวกับข้อมูลชนิด int หรือ float ได้ รวมไปถึงสามารถ
กำหนดมิติของ ndarray ที่ต้องการก็ได้ ซึ่งหากไม่กำหนดจะได้ผลลัพธ์เป็นเลขตัวเดียว

ฟังก์ชัน choice() มีรูปแบบดังนี้

```
choice( อาร์เรย์ดันทางข้อมูล, [จำนวนมิติ ndarray ที่ต้องการ])
```

ตัวอย่างที่ 4.12

```
from numpy import random as rd

a = rd.choice([1,3,5,7,11,13,17,19])
b = rd.choice([1,3,5,7,11,13,17,19], size=(3, 4))

print(a)
print(b)
```

ผลลัพธ์

```
7                                     # a
[[13  1  7  3]                      # b
 [ 1 19  1 11]
 [ 5  3  1  7]]
```

ทั้งนี้ฟังก์ชัน choice() สามารถใช้ร่วมกับ rand() หรือ randint() ได้เพื่อใช้สร้างอาร์เรย์ข้อมูล
ต้นทางของสมาชิก เช่น

```
from numpy import random as rd

a = rd.choice(rd.rand(10), size=(3, 5))

print(a)
```

NUMPY Pg 107 – 134 อ่านเอง (บทที่ 4)

หัวข้อ

อ่านเอง (บทที่ 5-10)

- Choice() – Random Distribution
- Shuffling & Permutation
- Vectorization & Broadcasting
- Ndarray operation
- Ndarray – Array Indexing
- Index Slicing
- NumPy – Other functions

THE END