

สารบัญ

สารบัญ	1
การเขียนโปรแกรมด้วยภาษา Python เนื้องต้น	7
Data Type, Variable, and Expression	7
○ Data Type	7
● Numeric Types	7
○ int	7
○ float	7
○ complex	8
● Text Type	8
○ string	8
● Sequence Types	13
○ list	13
○ tuple	17
○ range	18
● Mapping Type	18
○ dict	18
● Set Type	21
○ set	21
● Boolean Type	24
○ bool	24
○ Variable and Expression	24
● ตัวดำเนินการกำหนดค่า	25
● ตัวดำเนินการเปรียบเทียบ	25
● ลำดับความสำคัญ	26
Condition/Selection/Branching (if-elif-else)	27
○ if-elif-else	27
● การรับข้อมูลผ่านคีย์บอร์ด	27
● การกำหนดเงื่อนไขที่มีสองทางเลือก	28
● การกำหนดเงื่อนไขแบบซับซ้อน	28
○ break and continue	29
Iteration/Repetition (while, for)	31
○ for-loop	31
● กฎการย่อหน้าของภาษาไพธอน	31
● ช่วงจำนวน (range)	32
● ลูปซ้อนลูป (loop embedding)	33
○ while-loop	35
File	37
○ การเปิดและปิดไฟล์	37

○ การเขียนข้อมูลลงไฟล์	37
○ การอ่านข้อมูลที่ละ 1 บรรทัดจากไฟล์	38
○ การอ่านไฟล์ทุกบรรทัดในคราวเดียว	39
Function and Recursion	39
○ การนิยามฟังก์ชัน	40
○ การคืนค่าฟังก์ชันด้วยคำสั่ง return	40
○ การหยุดการทำงานของฟังก์ชันด้วยคำสั่ง return	41
○ การทำให้โค้ดอ่านง่ายขึ้น	43
○ การรีเฟคเตอร์โค้ด	44
○ ฟังก์ชันเรียกเกิด (Recursion)	44
○ การจดจำรายทาง (Memorization)	45
การโปรแกรมเชิงวัตถุ	46
○ คลาส	47
○ การถ่ายทอดคุณสมบัติของคลาส	49
แบบฝึกหัดเพิ่มเติม	51
1. Data Type, Variable, and Expression	51
2. Condition/Selection/Branching	52
3. Iteration/Repetition	53
4. File	54
5. Function and Recursion	54
6. การโปรแกรมเชิงวัตถุ	55
การนำเสนอภาพข้อมูลเบื้องต้น	58
ทำไม่ต้องนำเสนอข้อมูลด้วยภาพ?	59
• ตัวอย่างการทำ Data Visualization	59
Data Visualization What-Why-How loop	60
○ What?: What data do we have? - data abstraction	60
★ Semantics	60
★ Dataset Component	60
○ Why?: Do we have to see it? - Task Abstraction	61
★ Actions	61
○ How?: Do we make it happen? - Construction	61
★ Encode	61
★ Manipulate	61
★ Numpy	62
● พื้นฐาน Numpy	62
● การติดตั้ง Numpy	62
● การ import numpy มาใช้	63
● คำสั่งที่จะทำให้เราเข้าใจ numpy ง่ายขึ้น	63
● การสร้าง Array ด้วย np.array	63
★ np.zeros()	64
★ np.ones()	64

★ np.identity()	64
★ np.full()	65
★ np.eye()	65
★ np.random.random()	65
★ np.zeros_like()	65
★ np.arange()	66
● Mathematics Operations	66
★ Element-wise operation	66
★ Boardcasting	70
★ Indexing	72
★ Pandas	74
● การติดตั้ง Pandas	74
● การ import pandas มาใช้	74
● วิธีเช็ค Version Pandas	74
● โหลดไฟล์ CSV (Import)	74
● วิธีเช็คจำนวนแถว และจำนวนคอลัมน์	75
● วิธีสุ่มข้อมูลสำหรับเช็ค (Sample)	75
● วิธีเช็คข้อมูลหาความผิดปกติใน DataFrame เป็นต้น	75
● วิธีแปลงประเภทข้อมูล (Data Type) ใน Data Frame	76
● วิธีเช็ค Summary ของแต่ละคอลัมน์ (count, min, max, mean)	76
● วิธีเช็ค Summary (count, min, max, mean) แบบแยกกลุ่ม	76
● วิธีสร้าง DataFrame ใหม่	76
● วิธีเลือกหลายคอลัมน์จาก DataFrame	77
● วิธีเลือกคอลัมน์ตามเงื่อนไขที่ต้องการ	77
● วิธีเลือกแถวตามเงื่อนไขที่ต้องการ	77
● วิธีเพิ่มคอลัมน์ใหม่	77
● การสลับ Row <-> Column (Transpose)	77
● การต่อ DataFrame	78
● การต่อ DataFrame แบบ Join	78
● การหาค่า Mean, Sum, Max (Aggregate) แบบทั้ง DataFrame	78
● การ Aggregate แบบตามกลุ่มที่ต้องการ	78
● การรัน Function เดียวกันทุกแถว หรือทุกคอลัมน์	78
● รันคำสั่งที่เขียนเองกับทุกแถวใน 1 คอลัมน์	79
● รันคำสั่งที่เขียนเองกับทุกค่า	79
● คำนวน Correlation & Covariance	79
● คำนวน Cross Tabulation	79
● วิธีหาค่า Unique ในแต่ละคอลัมน์	79
● วิธีเช็คว่ามีเดาไว้ในข้อมูลซ้ำมั้ย (Duplicated)	80
● วิธีการนับจำนวน Duplicate	80
● วิธีการลบ Duplicate	80
● วิธีการลบเดา และลบคอลัมน์	80

• วิธีการลบแถวที่มี Missing Value	80
• วิธีแทนค่า Missing Value ด้วยค่าเฉลี่ย (Mean Imputation)	81
• การลูปข้อมูลแต่ละคอลัมน์ และแต่ละแถว	81
• การลูปข้อมูลแต่ละแถว	81
• วิธีเปลี่ยน DataFrame จากแบบ Wide เป็น Long (Melt)	81
• วิธีการเปลี่ยนชื่อคอลัมน์ (Rename)	81
• วิธีการใส่คำนำหน้าคอลัมน์ (Prefix)	81
• วิธีการแทนค่าใน DataFrame	82
• วิธีการ Export DataFrame เป็นไฟล์ CSV	82
★ Facet	82
★ Reduce	82
★ Package	82
★ Matplotlib	82
● การติดตั้ง Matplotlib	83
● การ import matplotlib มาใช้	83
● วิธีเช็ค Version Matplotlib	83
● Anatomy of a Figure	83
● Plotting	84
● Markers	85
● Line	88
● Labels and Title	90
● Adding Grid Lines	92
● Subplot	93
● Scatter Plot	94
● Bars Plot	97
● Histogram	99
● Pie Chart	99
● Cheatsheets	102
Good Data Visualization?	102
○ เพื่อนำเสนอจาก (กราฟ?) นี้	102
○ แนวทางการเลือก chart ควรให้เข้าใจง่าย	102
● plot เพื่อเข้าใจการแจกแจงของ data	103
★ 1 variable	103
- Frequency Plot/Density Plot	103
- Histogram	103
- Cumulative Frequency Diagram	104
- Box and Whisker Plot	104
- Violin Plot	105
- Stem and Leaf Plot	105
- Dot Matrix Chart	105
★ 2 variable	106

- Scatter Plot	106
- Dot Map	106
● plot เพื่อถูกความสัมพันธ์ระหว่าง variable	106
★ 1 variable	107
- Venn Diagram	107
★ 2 variable	107
- Line Chart	107
- Scatter Plot	107
★ ≥ 3 variable	108
- Bubble Chart	108
- Network Diagram	108
- Arc Diagram	109
- Tree Diagram	109
● plot เพื่อเปรียบเทียบค่า	109
★ ถูกความแตกต่างระหว่างกลุ่ม	110
- Bar Chart/Column Chart	110
- Mekko Chart/Marimekko Chart	110
- Bullet Graph	111
- Population Pyramid	111
- Multi-set Bar Chart	112
- Chord Diagram	112
- Pictogram Chart	112
- Cholopleth Map	113
★ ถูกการเปลี่ยนแปลงตามเวลา	113
- Line Chart	113
- Column Chart	114
- Radar Chart	114
● plot เพื่อถูกล่วนประกอบของ data นั้น ๆ	114
★ ไม่เปลี่ยนแปลงตามเวลา	114
- Pie Chart/Doughnuts Chart	114
- Waterfall Chart	115
- Stacked Bar Chart	115
- Tree Map	116
- Word Cloud	116
- Sunburst Diagram	116
- Bubble Map	117
★ เปลี่ยนแปลงตามเวลา	117
- Stacked Area Chart	117
- Stacked Bar Chart	118
● plot เพื่อดูการเคลื่อนไหว	118
- Sankey Diagram	118

- Connection Map	118
- Flow Map	119
Precautions	119
○ Lack of Data/Missing Data	119
○ Color/Contrast	119
○ Opacity	120
○ Misleading Scaling	120
Case Study	121
○ Dataset	121
○ แนวทางการทำ Data Visualization	121
★ ความเร็วของ Pokemon สัมพันธ์กับปัจจัยพื้นฐานต่าง ๆ อย่างไร?	121
★ แต่ละ Generation มี Pokemon อยู่เท่าไหร?	121
★ Pokemon แต่ละ Generation มีกี่ชนิด?	121
★ Pokemon ประเภทใดที่แพร่หลายมากที่สุดทั้งประเภท primary และ secondary?	121
★ ส่วนสูงและน้ำหนักของ Pokemon สัมพันธ์กับค่าพลังพื้นฐานต่าง ๆ อย่างไร?	121
★ เราจะหา Pokemon ที่แข็งแกร่งที่สุดได้หรือไม่?	121
Books	122
DATA + DESIGN	122
Fundamental of Data Visualization	122
Reference	123

การเขียนโปรแกรมด้วยภาษา Python เป็องตัน

Data Type, Variable, and Expression

- Data Type

Data Type หรือ ประเภทของข้อมูล คือ ลักษณะการจัดเก็บข้อมูลที่สามารถเก็บข้อมูล และใช้งานตามลักษณะที่ได้ระบุไว้ ใน Python สามารถนำมาร่วมผลได้มีหลายประเภท เช่น

- Numeric Types

- int

เลขจำนวนเต็ม หรือ int เป็นชนิดข้อมูลที่เก็บตัวเลขที่เป็นจำนวนเต็ม ซึ่งตัวเลขสามารถมีค่า ติดลบได้ (ขนาด 4 bytes)

```
x = 37  
  
print(x)  
print(type(x))
```

```
37  
<class 'int'>
```

- float

เลขทศนิยม หรือ float เป็นชนิดข้อมูลที่เก็บตัวเลขที่เป็นทศนิยม และ float สามารถเก็บค่าที่ เป็น e/E ได้ (ขนาด 8 bytes)

```
x = 37.54  
y = 1.74e-3  
  
print(x)  
print(y)  
print(type(x))  
print(type(y))
```

```
37.54  
0.00174  
<class 'float'>  
<class 'float'>
```

- **complex**

เลขจำนวนเชิงซ้อน หรือ complex เป็นชนิดข้อมูลที่เก็บตัวเลขที่เป็นจำนวนเชิงซ้อน (ขนาด 80 bytes)

```
x = 3 + 5j
```

```
print(x)
print(type(x))
```

```
(3+5j)
<class 'complex'>
```

- **Text Type**

- **string**

ข้อความ หรือ String เป็นชนิดของข้อมูลที่ใช้เก็บข้อความที่เป็นตัวอักษร และ ตัวเลข โดย ข้อความ จะต้องอยู่ใน Single quote('') หรือ Double quote("")

```
x = "Hello Gift!"
```

```
print(x)
print(type(x))
```

```
Hello Gift!
<class 'str'>
```

ที่จริงแล้วเราสามารถจัดรูปแบบของสตริงก่อนที่จะแสดงผลได้ วิธีการคือ เราจะใช้สัญลักษณ์ f นำหน้า สตริงที่เราต้องการจะให้จัดรูปแบบ ซึ่งสัญลักษณ์ f ย่อมาจากคำว่า format (รูปแบบ) นั่นเอง ข้อดีของการใช้สัญลักษณ์ f นำหน้าสตริงก็คือ เราสามารถแทรกค่าตัวแปรลงไปตำแหน่งใดของสตริงก็ได้ เพียง แค่ครอบตัวแปรด้วยปีกกา { } เท่านั้น นับว่าสะดวกมาก

```
name = 'Pannawit'
telno = '533-7855'
age = 18
degree = 'high-school'

print(f'My name is {name} and I am {age} years old. I am still a
{degree} student and my phone number is {telno}.')
```

```
My name is Pannawit and I am 18 years old. I am still a
high-school student and my phone number is 533-7855.
```

นอกจากนี้เรายังสามารถกำหนดวิธีการแสดงผลตัวแปรในเครื่องหมายปีกกาแต่ละตำแหน่งได้อีกด้วย

การกำหนดความกว้างของสตริง ในกรณีของสตริง เราสามารถระบุความกว้างสูงสุดของสตริงได้ด้วยคำสั่ง {ตัวแปรสตริง:ความกว้าง} โดยสตริงจะถูกแสดงผลแบบชิดซ้าย การกำหนดความกว้างของสตริงได้ทำให้เราสามารถพิมพ์ตารางออกมาได้สวยงามมาก โดยจะจัดชิดซ้ายเสมอ

```
user_database = [
    ('Pannawit', '533-7855', 18, 'high-school'),
    ('Purinut', '536-2313', 19, 'university'),
    ('Santipab', '524-0935', 18, 'university')
]

print(60 * '-')
for name, telno, age, degree in user_database:
    print(f'| {name:20} | {telno:15} | {degree:15} |')
print(60 * '-')
```

```
-----
| Pannawit           | 533-7855          | high-school      |
| Purinut            | 536-2313          | university        |
| Santipab           | 524-0935          | university        |
-----
```

การจัดสตริงไปทางซ้าย กลาง หรือขวา เราสามารถกำหนดให้จัดสตริงไปทางซ้าย กลาง หรือขวาได้ด้วยอพชั่น <, ^, และ > ที่ด้านหน้าความกว้างของสตริง

```
print(60 * '-')
for name, telno, age, degree in user_database:
    print(f'| {name:<20} | {telno:^15} | {degree:>15} |')
print(60 * '-')
```

```
-----
| Pannawit           | 533-7855          | high-school      |
| Purinut            | 536-2313          | university        |
| Santipab           | 524-0935          | university        |
-----
```

การเข้าถึงติกขันนารีและลิสต์ หากกรณีที่ค่าที่เราจะแสดงเป็นค่าที่ต้องเข้าถึงด้วยคีย์หรือ index เราถึงสามารถใช้เครื่องหมาย indexing () ได้ตามปกติเลย อย่างไรก็ตาม โปรดสังเกตว่า หากคีย์ที่ใช้จำเป็นต้องมีเครื่องหมายคำพูดเดียว (single quote) ให้เราเปลี่ยนเครื่องหมายนั้นเป็นเครื่องหมายคำพูดคู่ (double quote) " " แทน ไม่งั้นเติ่ยวจะซ้ำกับเครื่องหมายคำพูดเดียวที่ครอบอยู่ด้านนอกคอมพิวเตอร์จะงงเอาระ

```
user_database = [
    {'name': 'Pannawit', 'telno': '533-7855', 'age': 18, 'degree':
     'high-school'},
    {'name': 'Purinut', 'telno': '536-2313', 'age': 19, 'degree':
     'university'},
    {'name': 'Santipab', 'telno': '524-0935', 'age': 18, 'degree':
```

```

'university'}
]

print(60 * '-')
for user in user_database:
    print(f'| {user["name"]:<20} | {user["telno"]:^15} |
{user["degree"]:>15} |')
print(60 * '-')

```

```

-----
| Pannawit           | 533-7855      | high-school |
| Purinut            | 536-2313      | university   |
| Santipab           | 524-0935      | university   |
-----
```

การจัดฟอร์แมตให้ตัวเลข ในการจัดฟอร์แมตให้ตัวเลข เราสามารถระบุชนิดของตัวเลขได้ที่หลังความกว้างของสตริง

- ตัวเลขจำนวนเต็มจะใช้ออพชั่น d ตามหลังตัวเลข
- ตัวเลขจำนวนจริงจะใช้ออพชั่น f ตามหลังตัวเลข

เช่น เราสามารถจัดฟอร์แมตให้ตัวเลขใน user['age'] มีความกว้าง 3 ตัวอักษรและซิดขวาได้ดังนี้

```

user_database = [
    {'name': 'Pannawit', 'telno': '533-7855', 'age': 18, 'degree':
'high-school'},
    {'name': 'Purinut', 'telno': '536-2313', 'age': 19, 'degree':
'university'},
    {'name': 'Santipab', 'telno': '524-0935', 'age': 18, 'degree':
'university'}
]

print(66 * '-')
for user in user_database:
    print(f'| {user["name"]:<20} | {user["telno"]:^15}|{user["age"]:>3d}
| {user["degree"]:>15} |')
print(66 * '-')

```

```

-----
| Pannawit           | 533-7855      | 18 | high-school |
| Purinut            | 536-2313      | 19 | university   |
| Santipab           | 524-0935      | 18 | university   |
-----
```

ในกรณีที่เป็นตัวเลขจำนวนจริง เราสามารถระบุได้ทั้งความกว้างและจำนวนหลักทศนิยม โดยใช้รูปแบบเป็น ความกว้าง.จำนวนหลักทศนิยม f

ข้อควรระวัง: ความกว้างตรงนี้เป็นความกว้างทั้งหมด ซึ่งรวมເອງຈຸດທຄນິຍມແລະຫລັກທຄນິຍມເຂົ້າໄປແລ້ວ ຈຶ່ງຄວາມມັດຮຽນຕອນກຳຫຼັດຄວາມກວ້າງທັງໝົດໃຫ້ດືນນະ

เช่น เราสามารถแสดงคะแนนของนักเรียน score ให้มีความกว้าง 8 ตัวอักษร และมีทศนิยม 2 ตำแหน่งได้โดยใช้ฟอร์แมต {score:8.2f}

```
student_dict = {
    'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0,
    'Barry': 81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5
}

print(35 * '-')
for name, score in student_dict.items():
    print(f'| {name:<20} | {score:>8.2f} |')
print(35 * '-')
```

```
-----
| John           | 35.50 |
| Mary          | 73.00 |
| Cindy         | 83.50 |
| Sean          | 65.00 |
| Barry         | 81.00 |
| Mark          | 79.50 |
| Eugene        | 87.00 |
| Bob           | 49.50 |
-----
```

การแยกสตริง (tokenization) เราสามารถแบ่งสตริงออกเป็นลิสต์ของคำด้วยเครื่องหมายเว้นวรรคได้ด้วยคำสั่ง สตริง.split()

```
tokens = 'hello this is pannawit and i am glad to be here'.split()
print(tokens)
```

```
['hello', 'this', 'is', 'pannawit', 'and', 'i', 'am', 'glad',
'to', 'be', 'here']
```

ไม่ว่าจะเว้นวรรคสักกี่ครั้ง คำสั่ง split() ก็ยังสามารถแบ่งสตริงออกเป็นลิสต์ของคำได้

```
tokens = ' hello    this is    pannawit    and i    am glad    to be    here    '.split()
print(tokens)
```

```
['hello', 'this', 'is', 'pannawit', 'and', 'i', 'am', 'glad',
'to', 'be', 'here']
```

นอกจากนี้เรายังสามารถระบุเครื่องหมายตัดสตริง (delimiter) ได้อีกด้วย เช่น หากเราต้องการตัดสตริงด้วยเครื่องหมาย , เราจะใช้คำสั่ง ดังนี้

```
tokens = '533-7855,536-2313,524-7306'.split(',')
print(tokens)
```

```
['533-7855', '536-2313', '524-7306']
```

แต่หากจะบุเครื่องหมายตัดสตริง คำสั่ง `split()` จะยอมปล่อยให้มีสตริงว่างเกิดขึ้นได้ด้วย หากว่ามีเครื่องหมายตัดสตริงปรากฏขึ้นติดกันหลายครั้ง

```
tokens = '533-7855,,536-2313,,,524-7306'.split(',')
print(tokens)
```

```
['533-7855', '', '536-2313', '', '', '', '524-7306']
```

การเชื่อมลิสต์ของสตริง เราสามารถเชื่อมลิสต์ของสตริงเข้าด้วยกันโดยใช้คำสั่ง `สตริงเชื่อม.join(ลิสต์ของสตริง)`

```
words = ['hello', 'this', 'is', 'pannawit', 'and', 'i', 'am', 'glad',
'to', 'be', 'here']
print(' '.join(words))
```

```
hello this is pannawit and i am glad to be here
```

หากเราเชื่อมลิสต์ของสตริงด้วยสตริงว่าง ผลที่ได้คือทุกสตริงในลิสต์จะเชื่อมติดกันหมด

```
words = ['hello', 'this', 'is', 'pannawit', 'and', 'i', 'am', 'glad',
'to', 'be', 'here']
print(''.join(words))
```

```
hellothisisp洋洋洒洒andiamgladtobehere
```

การกำจัดช่องว่างที่ด้านซ้ายและขวาของสตริง เราสามารถกำจัดช่องว่างที่อยู่ด้านซ้ายและขวาของสตริงได้ด้วยคำสั่ง `สตริง.strip()`

```
text = '      In the beginning, there was darkness.      '
print(f'|{text}|')
print(f'|{text.strip()}|')
```

```
|      In the beginning, there was darkness.      |
|In the beginning, there was darkness.|
```

คำสั่งนี้จะมีประโยชน์มาก เวลาที่เราแยกสตริง แล้วนำแต่ละคำที่ได้มาแปลงเป็นตัวเลข ในกรณีนี้เราต้องกำจัดช่องว่างซ้ายขวาด้วย `strip()` ก่อนเสมอ ไม่เช่นนั้นเราจะไม่สามารถแปลงแต่ละคำเป็นตัวเลขได้เลย เช่น เราจะหาผลรวมของตัวเลขจำนวนเต็มที่อยู่ในสตริง `number_text`

```
number_text = '    10    ,    20    ,    30    ,    40    ,    50    '
tokens = number_text.split(',')
print(tokens)

my_sum = 0

for token in tokens:
    num = token.strip()
```

```
my_sum = my_sum + int(num)

print(my_sum)
```

```
[ 10 , 20 , 30 , 40 , 50 ]
150
```

- Sequence Types
 - list

ลิสต์ (list) คือ รายการสิ่งของที่เราสามารถปรับเปลี่ยนรายการได้ตลอดเวลา ใช้สัญลักษณ์ [] คั่นด้วยเครื่องหมายคอมมา ,

```
x = ["ComCamp35", "cpe37"]

print(x)
print(type(x))
```

```
['ComCamp35', 'cpe37']
<class 'list'>
```

- คำสั่งเข้าถึงสมาชิก (indexing) เราสามารถเข้าถึงสมาชิกแต่ละตัวในลิสต์ด้วยการระบุตำแหน่งของสมาชิกที่เราต้องการ ด้วยคำสั่ง [i] เมื่อ i คือ index ที่เราต้องการเข้าถึง ซึ่งเริ่มนับจาก 0 เสมอ
- คำสั่งเลือกบางส่วน (slicing) ลิสต์[a:b] จะเลือกสมาชิกตั้งแต่ index ที่ a จนถึง index ที่ b-1

- การเพิ่มสมาชิกเข้าไปในลิสต์

คำสั่ง **append** เราจะนำสมาชิกใหม่ต่อท้ายลิสต์ด้วยคำสั่ง ลิสต์.append(e) เมื่อ e คือสมาชิกใหม่ตัวนั้น เช่น เราสามารถต่อท้ายตัวเลข 1 ถึง 5 ที่ท้ายลิสต์ my_numbers ได้ดังนี้

```
my_numbers = [11, 12, 13, 14, 15]

for i in range(1, 6):
    my_numbers.append(i)

print(my_numbers)
[11, 12, 13, 14, 15, 1, 2, 3, 4, 5]
```

คำสั่ง **insert** เราจะแทรกสมาชิกใหม่เข้าไปใน index ที่กำหนดด้วยคำสั่ง ลิสต์.insert(i, e) เมื่อ i คือ index ที่เราจะแทรกสมาชิกใหม่ลงไป และ e คือสมาชิกใหม่ตัวนั้น

```
my_numbers = [20, 40, 60, 80, 100]
my_numbers.insert(2, 70)
print(my_numbers)
[20, 40, 70, 60, 80, 100]
```

คำสั่ง **extend** นอกจากนี้เรายังสามารถเพิ่มเติมสมาชิกจากลิสต์อื่นเข้าไปต่อท้ายลิสต์ที่เราสนใจได้ด้วยคำสั่ง **ลิสต์.append(ลิสต์ต้นทาง)** ข้อแตกต่างสำคัญระหว่าง extend กับ append คือ extend จะนำสมาชิกทุกตัวในลิสต์ต้นทางมาต่อท้ายลิสต์ปลายทาง ส่วน append จะนำสมาชิกหนึ่งตัวมาต่อท้ายลิสต์ เช่น เราสามารถเพิ่มสมาชิกจากลิสต์ [11, 12, 13, 14, 15] ไปต่อท้ายลิสต์ my_list ได้ด้วยโปรแกรมนี้

```
my_list = [1, 2, 3, 4, 5]
new_list = [11, 12, 13, 14, 15]

my_list.extend(new_list)
print(my_list)
```

[1, 2, 3, 4, 5, 11, 12, 13, 14, 15]

ซึ่งต่างจากการใช้คำสั่ง append ดังนี้

```
my_list = [1, 2, 3, 4, 5]
new_list = [11, 12, 13, 14, 15]

my_list.append(new_list)
print(my_list)
```

[1, 2, 3, 4, 5, [11, 12, 13, 14, 15]]

- การลบสมาชิกในลิสต์

คำสั่ง **del** เราสามารถลบสมาชิกในลิสต์ได้ด้วยคำสั่ง **del** เช่น เราสามารถลบสมาชิกที่ index ที่ 2 ออกจากลิสต์ my_numbers ได้ด้วยคำสั่ง **del my_numbers[2]** ดังนี้

```
my_numbers = [11, 12, 13, 14, 15]
del my_numbers[2]
print(my_numbers)
```

[11, 12, 14, 15]

นอกจากนี้เรายังสามารถลบสมาชิกออกจากลิสต์แบบเป็นช่วงๆ ได้ด้วย เช่น เราสามารถลบสมาชิก index ที่ 1 ถึง 3 ออกจากลิสต์ my_numbers ได้ดังนี้

```
my_numbers = [11, 12, 13, 14, 15]
del my_numbers[1:4]
print(my_numbers)
```

[11, 15]

- การเรียงลำดับสมาชิกในลิสต์

ในภาษา Python จะมีคำสั่งสำหรับเรียงลำดับสมาชิกในลิสต์อยู่ 2 คำสั่งคือ **sort** และ **sorted** คำสั่ง **ลิสต์.sort()** จะเรียงลำดับสมาชิกภายในลิสต์โดยจะทับลิสต์ตัวเดิมไปเลย เราจะใช้คำสั่ง **sort** นี้ต่อท้ายลิสต์ที่เราต้องการ

```
items = [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
items.sort()
print(items)
```

```
[1, 2, 3, 3, 4, 5, 6, 7, 8, 8, 9, 10]
```

ในขณะที่ คำสั่ง `sorted()` จะสร้างลิสต์ใหม่ที่เป็นผลลัพธ์จากการเรียงลำดับสมาชิกในลิสต์ตัวเดิม เราจะใช้คำสั่ง `sorted` คร่าวมลิสต์ที่เราต้องการ

```
old_items = [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
new_items = sorted(old_items)
print('Old items:', old_items)
print('New items:', new_items)
```

```
Old items: [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
New items: [1, 2, 3, 3, 4, 5, 6, 7, 8, 8, 9, 10]
```

คำสั่ง `sorted` นี้จะมีประโยชน์มากในการวนลูปตามการเรียงลำดับของสมาชิกภายในลิสต์ โดยที่ไม่ส่งผลข้างเคียงกับลิสต์ดังกล่าว

```
items = [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
print('The item list is:', items)
print()

print('Sort and print')
for item in sorted(items):
    print('Item =', item)
print()

print('Reversed sort and print')
for item in sorted(items, reverse=True):
    print('Item =', item)
print()

print('The item list is still the same:', items)
```

```
The item list is: [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
```

```
Sort and print
Item = 1
Item = 2
Item = 3
Item = 3
Item = 4
Item = 5
Item = 6
Item = 7
```

```
Item = 8
Item = 8
Item = 9
Item = 10

Reversed sort and print
Item = 10
Item = 9
Item = 8
Item = 8
Item = 7
Item = 6
Item = 5
Item = 4
Item = 3
Item = 3
Item = 2
Item = 1
```

The item list is still the same: [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]

- การตรวจสอบการเป็นสมาชิก

คำสั่ง **in** เราสามารถตรวจสอบการเป็นสมาชิกของลิสต์ด้วยคำสั่ง **in** เช่น หากเราต้องการตรวจสอบว่า มีสตริง 'name' อยู่ในลิสต์ **my_list** หรือไม่ เราสามารถใช้คำสั่งว่า 'name' **in my_list** ได้ดังตัวอย่างด้านล่าง

```
my_list = ['hello', 'my', 'name', 'is', 'kongtap']
if 'name' in my_list:
    print('It is in the list!')
else:
    print('It is not in the list.')
```

It is in the list!

ส่วนการตรวจสอบการไม่เป็นสมาชิก สามารถทำได้โดยใช้คำสั่ง **not in** เช่น หากเราต้องการตรวจสอบว่า 'doraemon' อยู่ในลิสต์ **my_list** หรือไม่ เราจะใช้คำสั่ง 'doraemon' **not in my_list** ได้ดังตัวอย่างด้านล่าง

```
my_list = ['hello', 'my', 'name', 'is', 'kongtap']
if 'doraemon' in my_list:
    print('It is in the list!')
else:
    print('It is not in the list.')
```

It is not in the list.

- tuple

ทูเพิล (tuple) คือลิสต์ที่มีขนาดคงที่ ไม่สามารถเพิ่มหรือลดสมาชิกได้อีก ทูเพิลใช้สัญลักษณ์ () และคั่นด้วยเครื่องหมายคอมมา ,

```
x = ("Ground", "Fire", "Water")
```

```
print(x)  
print(type(x))
```

```
('Ground', 'Fire', 'Water')
```

```
<class 'tuple'>
```

```
# คู่ลำดับเป็น tuple ที่มีสมาชิก 2 ตัว
```

```
coor1 = (3, 5)  
coor2 = (4, 7)  
print(coor1)  
print(coor2)
```

```
# คู่ลำดับสามารถมีสมาชิกได้หลายตัว
```

```
my_tuple = (3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4)  
print(my_tuple)
```

```
(3, 5)
```

```
(4, 7)
```

```
(3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4)
```

เราสามารถใช้ตัวดำเนินการ len เพื่อหาขนาดของทูเพิลได้

```
len(my_tuple)
```

12

เราสามารถนำค่าแต่ละตัวในทูเพิลมาใส่ตัวแปรได้อีกด้วย โดยตัวแปรที่นำมารับค่า ต้องมีจำนวนเท่ากับจำนวนสมาชิกในทูเพิลนั้น ๆ

```
coor_2d = (3, 5)  
coor_3d = (1, 2, 3)
```

```
x1, y1 = coor_2d  
x2, y2, z2 = coor_3d
```

```
print(x1, y1)  
print(x2, y2, z2)
```

```
3 5  
1 2 3
```

นอกจากนี้ เราสามารถใช้ indexing เพื่อเข้าถึงสมาชิกในทูเพิลได้อีกด้วย โดยถ้าเราทำ slicing ผลลัพธ์จะออกมาเป็นทูเพิลด้วย

```
my_tuple[2:7]
```

```
(5, 10, 2, 9, 7)
```

เราสามารถเปรียบเทียบทูเพิลได้ โดยคอมพิวเตอร์จะเริ่มเปรียบเทียบจากสมาชิกที่ละคู่ หากว่าสมาชิกคู่นั้นยังเท่ากัน ก็จะเปรียบเทียบสมาชิกตัวอื่นต่อไปเรื่อยๆ

```
(1, 2, 3) < (1, 2, 4) # ทูเพิลแรกน้อยกว่าทูเพิลหลัง เพราะเปรียบเทียบที่ละคู่จนไปเจอกับ 3 < 4
```

True

```
(1, 2, 3) < (1, 2, 3, 0) # ทูเพิลแรกน้อยกว่าทูเพิลหลัง เพราะทูเพิลแรกหมดสมาชิกก่อน
```

True

- range

range เป็นคำสั่งสำหรับมีไว้ในการลำดับจำนวนตัวเลข เช่น แสดงตัวเลข 0 - 3 หรือแบบກ้าวกระโดด 3,6,9 มีทั้งไปทาง + และ - ตามระบบเล่นจำนวนจริง ใช้ได้โดย มีหลักในการใช้งานหลักนี้ range(เริ่ม, จบ, การเพิ่มขึ้น)

```
x = range(0, 35, 2)  
  
print(x)  
print(type(x))
```

```
range(0, 35, 2)  
<class 'range'>
```

- Mapping Type

- dict

ติกชันนารี (dictionary) คือโครงสร้างข้อมูลที่เชื่อมโยงคีย์ (key) ไปยังค่า (value) ที่เก็บเอาไว้ ในคีย์นั้น คีย์ของติกชันนารีอาจจะเป็นตัวเลข สตริง หรือแม้แต่ทูเพิลก็ได้ และค่าที่เก็บไว้ในคีย์นั้นจะเป็นชนิดใดก็ได้ ติกชันนารีแตกต่างจากลิสต์และทูเพิลตรงที่ ลิสต์และทูเพิลจะเข้าถึงสมาชิกภายใต้ด้วยการใช้ค่า index ซึ่งเป็นจำนวนเต็ม ในขณะที่ในติกชันนารี เราจะเข้าถึงค่าภายใต้ด้วยการใช้คีย์ที่อาจเป็นข้อมูลชนิดอื่นก็ได้

- การสร้างติกชันนารี

เราสามารถสร้างติกชันนารีได้ 2 วิธี

- เราสร้างติกชันนารีเปล่าได้ด้วยคำสั่ง {}

- หรือเราสามารถสร้างติเกชันนารีที่มีค่าเริ่มต้นได้ด้วยคำสั่ง {k1: v1, k2: v2, k3: v3 , ... } โดยจะมีคุ่ของคีย์ ki และค่า vi เรียงกันไป คันตัวเครื่องหมายคอมมา ,

```
# ติเกชันนารีเปล่า
my_empty_dict = {}

# ติเกชันนารีที่มีค่าเริ่มต้น
x = {"name": "ComCamp35", "organizer": "cpe37", "place": "S14"}

print(my_empty_dict)
print(type(my_empty_dict))
print(x)
print(type(x))
```

```
{ }
<class 'dict'>
{'name': 'ComCamp35', 'organizer': 'cpe37', 'place': 'S14'}
<class 'dict'>
```

- การเข้าถึงค่าในคีย์

เราสามารถเข้าถึงค่าที่อยู่ในคีย์ได้ด้วยการใช้คำสั่ง indexing [] เมื่อันกับลิสต์และทูเพิล

```
student_dict = {
    'John': 35.0, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0,
    'Barry': 81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5
}

print(student_dict['Mary'])
print(student_dict['Eugene'])
```

```
73.0
87.0
```

หากเราพยายามอ่านค่าในคีย์ซึ่งไม่มีในติเกชันนารี คอมพิวเตอร์จะฟ้องด้วยข้อความว่า KeyError ซึ่งแปลว่า หาคีย์ไม่เจอ นั่นเอง

```
print(student_dict['Lucy'])
```

```
KeyError: 'Lucy'
```

ดังนั้นเพื่อป้องกันไม่ให้เกิดเหตุการณ์แบบนี้ จึงจำเป็นต้องตรวจสอบความเป็นสมาชิกของคีย์เล็กก่อน เราสามารถตรวจสอบความเป็นสมาชิกของคีย์ในติเกชันนารีได้ด้วยคำสั่ง in

```
print('Lucy' in student_dict)
print('Mary' in student_dict)
```

False

True

- การเพิ่มและแทนที่คีย์ในติกชั้นนารี

เราสามารถเติมคีย์ใหม่เข้าไปในติกชั้นนารีได้ด้วยคำสั่ง indexing เช่นกัน

```
student_dict = {  
    'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0,  
    'Barry': 81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5  
}  
  
student_dict['Rupert'] = 91.0  
  
print(student_dict)
```

```
{'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0, 'Barry':  
81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5, 'Rupert': 91.0}
```

หากเราพยายามเติมคีย์ใหม่ด้วยคำสั่ง indexing แต่ปรากฏว่าคีย์ตัวนั้นมีอยู่แล้ว ค่าของคีย์นั้นจะถูกลบทับไปด้วยค่าใหม่

```
student_dict = {  
    'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0,  
    'Barry': 81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5  
}  
  
student_dict['Mary'] = 89.0  
student_dict['Barry'] = student_dict['Barry'] + 5  
  
print(student_dict)
```

```
{'John': 35.5, 'Mary': 89.0, 'Cindy': 83.5, 'Sean': 65.0, 'Barry':  
86.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5}
```

- การลบคีย์ในติกชั้นนารี

เราสามารถลบคีย์ออกจากติกชั้นนารีด้วยคำสั่ง **del**

```
student_dict = {  
    'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0,  
    'Barry': 81.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5  
}  
  
del student_dict['Barry']  
print(student_dict)
```

```
{'John': 35.5, 'Mary': 73.0, 'Cindy': 83.5, 'Sean': 65.0, 'Mark': 79.5, 'Eugene': 87.0, 'Bob': 49.5}
```

- Set Type

- set

ภาษา Python ยังมีโครงสร้างข้อมูลอีกชนิดหนึ่งซึ่งทำหน้าที่เหมือนกับเซ็ตในทางคณิตศาสตร์ โครงสร้างนี้ก็เรียกว่า เซ็ต (set) นั่นเอง เซ็ตจะใช้สัญลักษณ์ { } สามารถแต่งตัวจะคุณด้วยเครื่องหมาย คอมมา , ข้อแตกต่างระหว่างเซ็ตกับลิสต์ก็คือ เซ็ตจะเก็บสมาชิกเฉพาะตัวที่ไม่ซ้ำและไม่จำเป็นว่าจะต้องเรียงลำดับตามการเพิ่มสมาชิก ในขณะที่ลิสต์จะเก็บสมาชิกทุกตัวและเรียงลำดับตามการเพิ่มสมาชิก

```
x = {"Grass", "Fighting", "Poison"}  
  
print(x)  
print(type(x))
```

```
{'Poison', 'Grass', 'Fighting'}  
<class 'set'>
```

เราสามารถตรวจสอบได้ว่า ค่าด้านนี้เป็นสมาชิกของเซ็ตหรือเปล่า ด้วยคำสั่ง **in**

```
'Grass' in x  
'Fire' in x
```

True

False

หากเราต้องการสร้างตัวแปรสำหรับเซ็ตว่าง Ø เราสามารถใช้คำสั่ง **set()** ได้

```
my_empty_set = set()  
print(my_empty_set)
```

set()

- การเพิ่มสมาชิกในเซ็ตด้วยคำสั่ง **add** และ **update**

เราสามารถเพิ่มสมาชิกในเซ็ตได้ 2 วิธี

- เพิ่มสมาชิกที่ละตัวด้วยคำสั่ง **เซ็ต.add(e)** เมื่อ e คือสมาชิกตัวที่เราจะเพิ่มเข้าไปในเซ็ต
- เพิ่มสมาชิกที่ละหลายตัวด้วยคำสั่ง **เซ็ต.update(ลิสต์หรือเซ็ตต้นทาง)** โดยคำสั่งนี้จะเพิ่มสมาชิกทุกตัวในลิสต์หรือเซ็ตต้นทางเข้าไปในเซ็ตปลายทางนั่นเอง

```
my_set = {300, 200, 100}  
print('my_set = ', my_set, end='\n\n')  
  
print('Adding 400')  
my_set.add(400)  
print('my_set = ', my_set, end='\n\n')
```

```

print('Adding 200')
my_set.add(200)
print('my_set =', my_set, end='\n\n')

print('Updating with [400, 500, 600]')
my_set.update([300, 400, 500, 600])
print('my_set =', my_set)

```

my_set = {200, 100, 300}

Adding 400
my_set = {200, 100, 400, 300}

Adding 200
my_set = {200, 100, 400, 300}

Updating with [400, 500, 600]
my_set = {100, 200, 300, 400, 500, 600}

- ตัวดำเนินการบนเซ็ต

ตัวดำเนินการ	Python	ความหมาย
Union (\cup)	$s1 \cup s2$	นำสมาชิกทั้งสองเซ็ตมารวมกัน
Intersection (\cap)	$s1 \cap s2$	เลือกสมาชิกที่อยู่ในทั้งสองเซ็ต
Difference (-)	$s1 - s2$	เลือกเฉพาะสมาชิกที่อยู่ในเซ็ตแรกแต่ไม่อยู่ในเซ็ตสอง
Symmetric difference (Δ)	$s1 \Delta s2$	เลือกเฉพาะสมาชิกที่อยู่ในเซ็ตใดเซ็ตหนึ่งเท่านั้น
Membership (\in)	$e \in s$	อยู่ในเซ็ต หรือไม่
Subset (\subseteq)	$s1 \subseteq s2$	เซ็ต $s1$ เป็นสับเซ็ตของ $s2$ หรือไม่
Superset (\supseteq)	$s1 \supseteq s2$	เซ็ต $s1$ เป็นซูเปอร์เซ็ตของ $s2$ หรือไม่

```

s1 = {1, 2, 3}
s2 = { 2, 3, 4}
print('s1 =', s1)
print('s2 =', s2, end='\n\n')

print('Union:           s1 ∪ s2 =', s1 | s2)      # vertical
bar = union
print('Intersection:    s1 ∩ s2 =', s1 & s2)      # ampersand
= intersection
print('Difference:      s1 - s2 =', s1 - s2)      # minus =
set difference (except)
print('                     s2 - s1 =', s2 - s1)
print('Symmetric difference: s1 Δ s2 =', s1 ^ s2)  # caret =
symmetric difference
print('Membership:       3 ∈ s1 =', 3 in s1)
print('Subset:           {2, 3} ⊆ s1 =', {2, 3} <= s1)   # '<='
= subset
print('                     {1, 2, 3} ⊆ s1 =', {1, 2, 3} <= s1)
print('Pure subset:       {2, 3} ⊂ s1 =', {2, 3} < s1)
print('                     {1, 2, 3} ⊂ s1 =', {1, 2, 3} < s1)
print('Superset:          s1 ⊇ {2, 3} =', s1 >= {2, 3})
print('                     s1 ⊇ {1, 2, 3} =', s1 >= {1, 2, 3})
print('Pure superset:     s1 ⊃ {2, 3} =', s1 > {2, 3})
print('                     s1 ⊃ {1, 2, 3} =', s1 > {1, 2, 3})

```

```

s1 = {1, 2, 3}
s2 = {2, 3, 4}

Union:           s1 ∪ s2 = {1, 2, 3, 4}
Intersection:   s1 ∩ s2 = {2, 3}
Difference:     s1 - s2 = {1}
                s2 - s1 = {4}
Symmetric difference: s1 Δ s2 = {1, 4}
Membership:     3 ∈ s1 = True
Subset:          {2, 3} ⊆ s1 = True
                {1, 2, 3} ⊆ s1 = True
Pure subset:    {2, 3} ⊂ s1 = True
                {1, 2, 3} ⊂ s1 = False
Superset:       s1 ⊇ {2, 3} = True
                s1 ⊇ {1, 2, 3} = True
Pure superset: s1 ⊃ {2, 3} = True
                s1 ⊃ {1, 2, 3} = False

```

- Boolean Type
 - **bool**

ตรรกะ หรือ Boolean เป็นชนิดข้อมูลที่เก็บข้อมูลระหว่าง True/False เท่านั้น และตัวแปรที่เป็น boolean จะสามารถนำไปใช้กับการกำหนดเงื่อนไขต่าง ๆ ได้

```

x = bool(5 > 10)

print(x)
print(type(x))

```

```

False
<class 'bool'>

```

- Variable and Expression

ตัวแปรเป็นที่เก็บข้อมูลในโปรแกรม ต้องมีชื่อกำกับ ชื่อตัวแปรประกอบด้วยตัวอักษร ตัวเลข หรือเครื่องหมายขีดเส้นใต้ (_) ตัวอังกฤษใหญ่เมื่อตอนตัวเล็ก ห้ามเขียนต้นชื่อด้วยตัวเลข และอย่าตั้งชื่อตัวแปรซ้ำกับชื่อฟังก์ชันใน Python เช่น *int, str, max, sum, abs, ...*

การให้ค่ากับตัวแปร

```

a = b = c = 0.0 # ให้ตัวแปร a b และ c เก็บจำนวนจริง 0.0
a = 5; b = 6; a,b = b,a # สับค่าตัวแปร a และ ตัวแปร b → a เก็บ 6 และ b เก็บ
5
a = x # ถ้า x ไม่เคยมีการให้ค่ามาก่อน คำสั่งนี้จะผิด เพราะไม่ว่า x มีค่าเท่าใด

```

- ตัวดำเนินการทางคณิตศาสตร์

ตัวดำเนินการที่ใช้สำหรับการคำนวณทางคณิตศาสตร์ในพื้นฐาน เช่น การบวก การลบ การคูณ การหาร และการหารเอาเศษ เป็นต้น

Operator	Name	$x = 10, y = 5$
+	Addition	$x + y = 15$
-	Subtraction	$x - y = 5$
*	Multiplication	$x * y = 50$
/	Division	$x / y = 2.0$
**	Exponential	$x^{**}y = 100000$
%	Modulus	$x \% y = 0$
//	Floor division	$x // y = 2.0$

- ตัวดำเนินการกำหนดค่า

ตัวดำเนินการที่ใช้สำหรับกำหนดค่าให้กับตัวแปร

Operator	Name	Example	Same As	Result
=	Assignment	$x = 10$	$x = 10$	$x = 10$
+=	Addition Assignment	$x += 3$	$x = x + 3$	$x = 13$
-=	Subtraction Assignment	$x -= 3$	$x = x - 3$	$x = 10$
*=	Multiply Assignment	$x *= 3$	$x = x * 3$	$x = 30$
/=	Divide Assignment	$x /= 3$	$x = x / 3$	$x = 10$
%=	Modulus Assignment	$x \%= 20$	$x = x \% 20$	$x = 10$
//=	Floor Division Assignment	$x // = 2$	$x = x // 2$	$x = 5$
**=	Exponential Assignment	$x **= 2$	$x = x ** 2$	$x = 25$

- ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการที่ใช้สำหรับเปรียบเทียบค่าในตัวแปร หากเงื่อนไขเป็นจริงผลลัพธ์จะเป็น True และเป็น False หากเงื่อนไขไม่เป็นจริง โดยทั่วไปมักใช้กับคำสั่งตรวจสอบเงื่อนไข if และคำสั่งวนซ้ำ for while เพื่อควบคุมการทำงานของโปรแกรม

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

```

x = 10
y = 5
x == y # false
x != y # true
x > y # true
x < y # false
x >= y # true
x <= y # false

```

- ลำดับความสำคัญ

Precedence	Operators	Precedence	Operators	Note
1	()	6	== , !=	ถ้าเครื่องหมายมีลำดับการดำเนินการระดับเดียวกันให้ดำเนินการจากซ้ายไปขวา
2	! , ++ , --	7	&&	
3	* , / , %	8		
4	+ , -	9	*= , /= , %= , += , -=	
5	< , <= , > , >=			

Example:

```

15 + 3 * 10 + 5 = 15 + 30 + 5 = 50
10 - 40 / 4 + 80 - 10 = 10 - 10 + 80 - 10 = 70
50 * 10 - 50 = 500 - 50 = 450
10 % 50 + 20 - 10 = 10 + 20 - 10 = 20

```

Condition/Selection/Branching (if-elif-else)

○ if-elif-else

การกำหนดเงื่อนไข (branching/selection/condition) ในภาษา Python จะใช้คำสั่ง `if` (แปลว่า 'ถ้า') โดยจะมีรูปแบบดังนี้

```
if <เงื่อนไข>:  
    <กระบวนการ>
```

- คอมพิวเตอร์จะทำการวนการที่อยู่ในด้านใน ถ้าหากว่าเงื่อนไขที่กำหนดนี้เป็นจริง
- การเงินวรครายอ่อนน้ำจะเป็นการบอกสโคป (scope) ของคำสั่งว่าเป็นกระบวนการที่อยู่ภายใต้คำสั่งที่อยู่ด้านนอก การวนลูปก็จะวนภายใน scope ของตัวเอง
- การกำหนดเงื่อนไขจะทำงานเฉพาะภายใน scope ของตัวเองเท่านั้น

เช่น โปรแกรมด้านล่างนี้จะพิมพ์ข้อความ 'Hurray!' ถ้าหากว่าเงินเดือนในตัวแปร salary มีค่ามากกว่า 10,000 บาท

```
salary = 50000  
  
if salary > 10000:  
    print('Hurray!')
```

Hurray!

ทดลองเปลี่ยนค่าตัวเลขของตัวแปร salary และรันใหม่ดู

- การรับข้อมูลผ่านคีย์บอร์ด

เราสามารถรับอินพุตจากคีย์บอร์ดได้ด้วยคำสั่ง `input` แต่เวลาสั่งคำสั่งนี้แล้ว จะต้องหาตัวแปรมาเก็บค่าอินพุตที่รับมาด้วยเสมอ

```
my_name = input('Enter your name: ')      # ใส่ข้อความเดือนผู้ใช้ที่ตรงนี้  
print('Hello,', my_name)
```

```
Enter your name: Pannawit  
Hello, Pannawit
```

ค่าที่รับมาได้จะอยู่ในรูปข้อความ (string) เสมอ หากเราต้องการรับค่าตัวเลข เราจำเป็นต้องแปลงข้อความกลับมาเป็นตัวเลขด้วยคำสั่ง `int` หรือ `float`

- คำสั่ง `int` จะแปลงข้อความเป็นจำนวนเต็ม เช่น 10, 20, 30
- คำสั่ง `float` จะแปลงข้อความเป็นจำนวนจริง เช่น 0.25, 3.5, 10.0

```
my_salary_str = input('Enter your salary: ')  
my_salary = int(my_salary_str) # ต้องแปลงข้อความให้เป็นจำนวนเต็มก่อนเปรียบเทียบ  
if my_salary > 10000:  
    print('Your salary is quite high.')
```

```
Enter your salary: 15000
Your salary is quite high.
```

- ลองรันโปรแกรมด้านบนใหม่อีกรอบ แต่คราวนี้ใส่เป็นข้อความที่ไม่ใช่จำนวนเต็มดูบ้าง เช่น 'Pannawit' หรือ 100.5 โปรแกรมจะระเบิดทันที
- ที่คอมพิวเตอร์ฟ้องว่า ValueError นั้นหมายความว่า คอมพิวเตอร์ไม่สามารถแปลงข้อความให้กลายเป็นตัวเลขได้นั่นเอง

- การกำหนดเงื่อนไขที่มีสองทางเลือก

ในกรณีที่เราต้องการกำหนดเงื่อนไขที่มี 2 ทางเลือก เราสามารถใช้คำสั่ง if ... else ... (else แปลว่า 'ไม่เช่นนั้น') ได้
รูปแบบของเงื่อนไขที่มี 2 ทางเลือก

```
if <เงื่อนไข> :
    <กระบวนการ 1>
else:
    <กระบวนการ 2>
```

หมายความว่าเงื่อนไขที่กำหนดมา ถ้าเงื่อนไขเป็นจริง ก็ให้ทำการบวนการที่ 1 ไม่เช่นนั้นก็ให้ทำการบวนการที่ 2 แทน เช่น โปรแกรมด้านล่างนี้จะพิมพ์ข้อความ 'Hurray!' ถ้าหากว่าเงินเดือนในตัวแปร salary มีค่ามากกว่า 10,000 บาท หรือไม่เช่นนั้นก็พิมพ์ข้อความว่า 'Poor you.' แทน

```
salary = 5000

if salary > 10000:
    print('Hurray!')
else:
    print('Poor you.')
```

Poor you.

- การกำหนดเงื่อนไขแบบซับซ้อน

นอกจากนี้ หากเงื่อนไขของโปรแกรมมีความ слับซับซ้อน เช่น การตัดเกรดของนักเรียนที่ต้องทำแบบเป็นช่วงชั้น เราสามารถใช้การกำหนดเงื่อนไขแบบซับซ้อน if ... elif ... else ... ได้ (elif ย่อมาจากคำว่า 'else if' แปลว่า 'ไม่เช่นนั้น ถ้าหาก')
รูปแบบของการกำหนดเงื่อนไขแบบซับซ้อน

```
if <เงื่อนไข 1> :
    <กระบวนการ 1>
elif <เงื่อนไข 2> :
    <กระบวนการ 2>
elif <เงื่อนไข 3> :
    <กระบวนการ 3>
:
:
:
```

```
else:
```

```
    <กระบวนการสุดท้าย>
```

ความหมายของคำสั่งนี้ คือ

- ถ้าเงื่อนไข 1 นี้เป็นจริง ก็ให้ทำการวนการที่ 1
- ไม่ เช่นนั้น ก็ให้เช็คเงื่อนไข 2 ถ้าเป็นจริง ก็ให้ทำการวนการที่ 2
- ไม่ เช่นนั้น ก็ให้เช็คเงื่อนไข 3 ถ้าเป็นจริง ก็ให้ทำการวนการที่ 3
- เรา จะเช็คเงื่อนไขแบบนี้ไปเรื่อยๆ จนกระทั่งไม่เข้าเงื่อนไขใดเลย จึงทำการวนการสุดท้าย

เช่น ถ้าเราจะกำหนดเงื่อนไขเพื่อตัดเกรดของนักเรียน จากคะแนนเต็ม 100 คะแนน

- ถ้าได้คะแนน 80 ขึ้นไป จะได้เกรด 4
- ไม่ เช่นนั้น ถ้าได้คะแนน 70 ขึ้นไป จะได้เกรด 3
- ไม่ เช่นนั้น ถ้าได้คะแนน 60 ขึ้นไป จะได้เกรด 2
- ไม่ เช่นนั้น ถ้าได้คะแนน 50 ขึ้นไป จะได้เกรด 1
- ไม่ เช่นนั้น จะได้เกรด 0

```
score = int(input('Enter your score: '))

if score >= 80:           # score >= 80
    grade = 4
elif score >= 70:          # 70 <= score <= 79
    grade = 3
elif score >= 60:          # 60 <= score <= 69
    grade = 2
elif score >= 50:          # 50 <= score <= 59
    grade = 1
else:                      # score < 50
    grade = 0

print('Your grade = ', grade)
```

Enter your score: 55

Your grade = 1

- break and continue

เราสามารถออกจากลูปได้ด้วยคำสั่ง break เช่น เราสามารถสั่งให้คอมพิวเตอร์พิมพ์ชื่อนักเรียน และความสูงตามลิสต์ด้านล่างออกมา จนกว่าความสูงจะมากกว่า 165 เซนติเมตรจึงจะหยุด ได้ด้วยคำสั่ง break

```
student_names = ['Kongtap', 'Nano', 'Phupha']
student_heights = [163.5, 200.5, 173.0]

for i in range(len(student_heights)):
    # พิมพ์ชื่อนักเรียนและความสูง
    print(student_names[i], ':', student_heights[i])
    # ถ้าความสูงของนักเรียนคนนั้น > 165 เซนติเมตร ให้ออกจากลูป
```

```
if student_heights[i] > 165.0:  
    break  
  
print('That is all.')
```

Kongtap : 163.5
Nano : 200.5
That is all.

ส่วนคำสั่ง continue จะสั่งให้คอมพิวเตอร์เลิกทำกระบวนการที่กำลังทำกับสมาชิกตัวปัจจุบัน แล้วกระโดดไปสมาชิกตัวถัดไปทันที เช่น เราสามารถสั่งให้คอมพิวเตอร์พิมพ์รายชื่อนักเรียนและความสูง แต่หากนักเรียนคนใดที่มีชื่อขึ้นต้นด้วยตัว 'K' หรือตัว 'P' ก็จะไม่ต้องพิมพ์ต่อท้ายว่า blacklisted ได้ดังนี้

```
student_names = ['Kongtap', 'Nano', 'Phupha', 'Arm']  
student_heights = [163.5, 200.5, 173.0, 165.0]  
  
for i in range(len(student_heights)):  
    # พิมพ์ชื่อนักเรียนและความสูง  
    print(student_names[i], ':', student_heights[i], end=' ')  
    # ถ้าชื่อนักเรียนขึ้นต้นด้วยตัว K หรือตัว P ให้ตัดจบ และไปสมาชิกตัวถัดไปทันที  
    if student_names[i][0] == 'K' or student_names[i][0] == 'P':  
        print()  
        continue  
    print(' --> blacklisted')
```

Kongtap : 163.5
Nano : 200.5 --> blacklisted
Phupha : 173.0
Arm : 165.0 --> blacklisted

Iteration/Repetition (while, for)

สังเกตใหม่ว่า ในตัวอย่างก่อนหน้านี้เราต้องใช้วิธี copy-paste แล้วแก้หมายเลขตำแหน่งของสมาชิกเอาที่ลงทะเบียน ถ้าเกิดลิสต์มีสมาชิกจำนวนมากขึ้นมา เราอาจจะ copy-paste กันมือหิว แน่นอน ดังนั้น เราสามารถใช้วิธีการวนลูป (loop) เพื่อทำการวนการเดิมซ้ำ ๆ ได้

○ for-loop

ลูปฟอร์ (for-loop) ใช้สำหรับทำการวนการเดิมซ้ำๆ กับสมาชิกแต่ละตัวในลิสต์ ตามลำดับ รูปแบบคำสั่งของลูปฟอร์ คือ

```
for <ตัวแปร> in <ลิสต์>:  
    <กระบวนการ>
```

คำสั่งนี้จะแทนสมาชิกแต่ละตัวในลิสต์ด้วยตัวแปรที่กำหนด แล้วจึงทำขั้นตอนการนี้จนกว่า จะใช้สมาชิกครบถ้วนตามลำดับ เช่น ถ้าเราต้องการพิมพ์สมาชิกแต่ละตัวในลิสต์ student_heights ของเรา เราจะสั่งคำสั่ง ดังนี้

```
# ลิสต์ความสูงของนักเรียนในกลุ่ม  
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]  
# สมาชิกแต่ละตัวของลิสต์ student_heights จะแทนด้วยตัวแปร height  
for height in student_heights:  
    # พิมพ์ค่าของตัวแปร height ออกหน้าจอ  
    print(height)  
    print('hey!')  
  
print('That is all.')
```

```
163.5  
hey!  
150.0  
hey!  
167.0  
hey!  
161.25  
hey!  
170.0  
hey!  
That is all.
```

สิ่งที่อยากรู้สังเกตตรงนี้คือ กระบวนการที่เราต้องการทำซ้ำคือ print(height) จะต้องมีอยู่หน้าตัดเข้าไป

- กฎการย่อหน้าของภาษาไพธอน
 - กระบวนการที่เราต้องการทำซ้ำในลูปทุกชนิด จะต้องย่อหน้าตัดเข้าไปทางขวาเสมอ
 - ขนาดของย่อหน้าที่นิยมกันคือ เดอะ spacebar 4 ครั้ง
 - เมื่อไรก็ตามที่ย่อหน้ากลับออกมาทางซ้าย จะถือว่าจบลูปแล้ว

- ช่วงจำนวน (range)

- บางครั้งเราจำเป็นต้องวนลูปฟอร์บันเลขตำแหน่ง index ด้วย
- เราสามารถใช้คำสั่ง range(a, b) เพื่อสร้างช่วงจำนวนในลูปฟอร์ได้ โดย a จะเป็นค่า index เริ่มต้นและ b-1 จะเป็นค่าสุดท้าย เช่น

```
for i in range(1, 11):
    print('i =', i)
```

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

หากเราละค่า a ไป เหลือเพียง range(b) ช่วงจำนวนจะเริ่มต้นจาก 0 และสิ้นสุดที่ b-1

```
for i in range(20):
    print('i =', i)
```

```
i = 0
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
i = 11
i = 12
i = 13
i = 14
i = 15
i = 16
i = 17
i = 18
i = 19
```

นอกจากนี้เรายังสามารถกำหนดขนาดของก้าวในการสร้างช่วงจำนวนได้ โดยใช้คำสั่ง range(a, b, c) เมื่อค่า c คือขนาดของก้าว

```
# ก้าวมีขนาดเท่ากับ 2
for i in range(0, 21, 2):
    print('i =', i)
```

```
i = 0
i = 2
i = 4
i = 6
i = 8
i = 10
i = 12
i = 14
i = 16
i = 18
i = 20
```

ข้อดีของการใช้คำสั่ง range ที่กำหนดขนาดของก้าวได้ ก็คือ เราสามารถใส่ index ย้อนหลังได้ด้วย เช่น

```
for i in range(10, 0, -1):
    print(i)
```

**ก้าวมีขนาดติดลบ แปลว่าเดินถอยหลัง

**สังเกตว่าลูปฟอร์จะหยุดทำงานเมื่อค่า index i = 0 ทำให้ไม่พิมพ์ค่า 0 ออกมากทางหน้าจอ

- ลูปซ้อนลูป (loop embedding)

เราสามารถเขียนลูปซ้อนลูปได้ด้วยหากว่าโปรแกรมของเรามีความซับซ้อนกว่าลูปเดียว เช่น เราสามารถวนลูปเพื่อไล่สูตรคูณแม่ 3 ถึงแม่ 5 ได้ดังนี้

```
# วนลูป index m เพื่อไล่แม่สูตรคูณจากแม่ 3 ถึงแม่ 5
for m in range(3, 5):
    print('Multiplication of', m)
    # วนลูป index i เพื่อไล่ตัวคูณจาก 1 ถึง 12
    for i in range(1, 13):
        print(m, '*', i, '=', m * i)
    # พิมพ์บรรทัดใหม่คั่นระหว่างแม่
    print()
```

Multiplication of 3

```
3 * 1 = 3
3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
```

```
3 * 9 = 27
3 * 10 = 30
3 * 11 = 33
3 * 12 = 36
```

Multiplication of 4

```
4 * 1 = 4
4 * 2 = 8
4 * 3 = 12
4 * 4 = 16
4 * 5 = 20
4 * 6 = 24
4 * 7 = 28
4 * 8 = 32
4 * 9 = 36
4 * 10 = 40
4 * 11 = 44
4 * 12 = 48
```

Multiplication of 5

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
5 * 11 = 55
5 * 12 = 60
```

ข้อสังเกตจากการวนลูปforในการไล่แม่สูตรคูณคือ ในลูปนอก เราชวนลูป index ของแม่สูตรคูณ ก่อน เช่น แม่ 3 ถึงแม่ 5 เป็นต้น ส่วนลูปข้างใน เราชวนลูป index ของตัวคูณในแต่ละแม่ คือตั้งแต่ 1 ถึง 12 นั่นแสดงว่า ถ้าเราจะเขียนโปรแกรมที่ลับซับซ้อนได้ เราจะต้องแยกระยะและถอดลูปออกเป็นชั้นนอก และชั้นในให้ออก

!Tips: เราสามารถระบุให้คำสั่ง print ไม่ต้องขึ้นบรรทัดใหม่หลังจากพิมพ์ค่าอุกหนาจ่อได้ด้วยการระบุ end=" " ไว้ในวงเล็บของ print

```
for i in range(4):      # จำนวนจตุรัส
    for j in range(3):    # จำนวนแคล
        for k in range(3): # จำนวนหลัก
            print(9*i + 3*j + k + 1, ' ', end=' ')
        print()
    print()
```

```
1 2 3  
4 5 6  
7 8 9  
  
10 11 12  
13 14 15  
16 17 18  
  
19 20 21  
22 23 24  
25 26 27  
  
28 29 30  
31 32 33  
34 35 36
```

○ while-loop

ข้อจำกัดสำคัญของลูปฟอร์คือ มันจะไม่ลูปสมาชิกในลิสต์เท่านั้น ทำให้บางครั้งก็ไม่สะดวก หากเราต้องทำซ้ำกระบวนการบันสมាជิกตัวเดิมในลิสต์ตามเงื่อนไขที่กำหนด

คำสั่งลูปໄว์ล (while-loop) จะตรวจสอบเงื่อนไขที่กำหนด หากเป็นไปตามเงื่อนไข ก็จะทำซ้ำกระบวนการนั้นไปเรื่อยๆ จนกว่าเงื่อนไขนั้นจะไม่เป็นจริง
รูปแบบของลูปໄว์ลเป็นดังนี้

```
while <เงื่อนไข>:
```

```
    <กระบวนการ>
```

ความหมายของคำสั่งนี้คือ ตราบเท่าที่ เงื่อนไขนี้เป็นจริง ก็ให้ทำการวนการนี้ไปเรื่อยๆ นั่นเอง เช่น หากเราต้องการหาผลรวมของจำนวนที่ index ที่ 0, 2, 4, ... ไปเรื่อยๆ ตราบเท่าที่จำนวนนั้นไม่เท่ากับ 7 เราจะเขียนโปรแกรมได้ดังนี้

```
items = [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]  
idx = 0  
jumping_sum = 0  
  
while idx < len(items) and items[idx] != 7:  
    print('Adding', items[idx])  
    jumping_sum = jumping_sum + items[idx]  
    idx = idx + 2  
  
print('The jumping sum is', jumping_sum)
```

```
Adding 3  
Adding 5  
Adding 2  
The jumping sum is 10
```

เราสามารถออกจากการลูปໄว์ลได้ด้วยคำสั่ง break เช่น หากเราต้องการหาผลรวมของจำนวนในลิสต์ items จากตัวแรกไปเรื่อยๆ จนกว่า จำนวนนั้นจะน้อยกว่าหรือเท่ากับ 2 เราจะเขียนโปรแกรมได้ดังนี้

```

items = [3, 8, 5, 10, 2, 9, 7, 6, 3, 8, 1, 4]
idx = 0
my_sum = 0

while idx < len(items):
    print('Adding', items[idx])
    my_sum = my_sum + items[idx]
    # สังเกตว่าคำสั่ง break จะทำงานหลังจากที่ได้บวกผลรวมเรียบร้อยแล้ว
    if items[idx] <= 2:
        break

    idx = idx + 1

print(my_sum)

Adding 3
Adding 8
Adding 5
Adding 10
Adding 2
28

```

จากตัวอย่างด้านบน จะสังเกตว่าคำสั่ง break จะทำงานหลังจากที่ได้บวกผลรวมเรียบร้อยแล้ว

```

my_numbers = []

# วนลูปตลอดไปจนกว่าจะถูกสั่ง break
while True:
    num = int(input('Enter an integer: '))

    # หากค่า num เป็น 0 ให้ออกจากลูปด้วยคำสั่ง break
    if num == 0:
        break

    # นำค่า num ไปต่อท้ายลิสต์ my_numbers
    my_numbers.append(num)

# เรียงลำดับสมาชิกในลิสต์ my_numbers จากมากไปน้อยด้วยคำสั่ง sort
my_numbers.sort(reverse=True)

print(my_numbers)

```

Input: 1\n2\n3\n4\n0\n

Output: [4, 3, 2, 1]

สังเกตว่าคำสั่ง while True จะใช้ในการวนลูปตลอดไป (forever loop) ซึ่งจะมีประโยชน์ในกรณีที่จำเป็นต้องตรวจสอบเงื่อนไขในขณะทำการนั่นเอง

File

○ การเปิดและปิดไฟล์

เราสามารถเปิดไฟล์ได้โดยใช้คำสั่ง `open(ชื่อไฟล์, ข้ออนุญาต)` โดยเราจะต้องนำตัวแปร 1 ตัว (ซึ่งเรียกว่า file handler) มารับตัวแทนไฟล์นั้นเอาไว้ เราจะได้นำไฟล์ไปใช้งานต่อได้

ข้ออนุญาตในการใช้ไฟล์ที่ใช้กันทั่วไป มีอยู่ด้วยกัน 3 ประเภท

- ข้ออนุญาตประเกท 'r' คืออ่านข้อมูลอย่างเดียวเท่านั้น
- ข้ออนุญาตประเกท 'w' คือเขียนข้อมูลทั้งหมดอย่างเดียวเท่านั้น
- ข้ออนุญาตประเกท 'a' คือเขียนข้อมูลต่อท้ายอย่างเดียวเท่านั้น
- เมื่อใช้ไฟล์เสร็จแล้ว เรายังต้องปิดไฟล์ได้ด้วยคำสั่ง `ไฟล์.close()`

ข้อควรระวัง: ถ้าไม่ปิดไฟล์หลังใช้งานเสร็จ ข้อมูลที่เราสั่งเขียนลงไปอาจจะหายไปหมดก็ได้

```
fhdl = open('test_file_1.txt', 'w') # fhdl ย่อมาจากคำว่า file handler
fhdl.close()
```

○ การเขียนข้อมูลลงไฟล์

เราสามารถเขียนข้อมูลลงไฟล์ได้ด้วยคำสั่ง `ไฟล์.write(สตริง)` โปรดสังเกตว่า ข้อมูลที่เขียนลงไฟล์จะต้องอยู่ในรูปแบบของสตริงเสมอ หากเราต้องการเขียนข้อมูลลงไฟล์ 1 บรรทัด เราต้องใส่เครื่องหมายขึ้นบรรทัดใหม่ `\n` เองด้วย

```
fhdl = open('test_file_1.txt', 'w')

for i in range(1, 11):
    fhdl.write(f'{10 * i}\n')    # '\n' = newline

fhdl.close()
```

○ การอ่านข้อมูลทีละ 1 บรรทัดจากไฟล์

เราสามารถอ่านข้อมูลทีละ 1 บรรทัดจากไฟล์ได้ด้วยการวนลูปฟอร์

```
for <บรรทัด> in <ไฟล์> :
    <กระบวนการ>
```

จะสังเกตได้ว่า ทุกบรรทัดที่อ่านออกมาจากไฟล์ด้วยลูปฟอร์ จะมีเครื่องหมายขึ้นบรรทัด `\n` ติดมาด้วยเสมอ

```
fhdl = open('test_file_1.txt', 'r')

for line in fhdl:
    print(line)

fhdl.close()
```

10
20
30
40
50
60
70
80
90
100

ดังนั้นปกติแล้ว เราจะสิ่ง สตริง.strip() แต่ละบรรทัดด้วย เพื่อกำจัดเครื่องหมายเว้นวรรคและขั้นบรรทัดใหม่ที่ด้านซ้ายและขวาออกไป เช่น ถ้าเราต้องการจะอ่านตัวเลขในไฟล์ test_file_1.txt ออกมากแล้วหาผลรวม เราจะเขียนโปรแกรมได้ดังนี้

```
fhdl = open('test_file_1.txt', 'r')

my_sum = 0

for line in fhdl:
    numstr = line.strip()      # กำจัดเว้นวรรคและขั้นบรรทัดใหม่ซ้ายขวาทิ้งไป
    my_sum = my_sum + int(numstr)

fhdl.close()

print(my_sum)
```

550

○ การอ่านไฟล์ทุกบรรทัดในคราวเดียว

นอกจากนี้เรายังสามารถอ่านไฟล์ทุกบรรทัดได้ในคราวเดียวด้วยคำสั่ง ไฟล์.readlines() จะสังเกตได้ว่าทุกบรรทัดก็ยังคงติดเครื่องหมายขั้นบรรทัดใหม่เอาไว้ **โปรดระมัดระวังการใช้งานข้อมูลแต่ละบรรทัดด้วย** ทั้งนี้ เราสามารถกำจัดเครื่องหมายเว้นวรรคและขั้นบรรทัดใหม่ที่อยู่ด้านซ้ายและขวาในแต่ละบรรทัดได้ด้วยคำสั่ง strip() เช่นเดย

```
fhdl = open('test_file_1.txt', 'r')
lines = fhdl.readlines()
```

```

fhdl.close()
print(lines)

for i in range(len(lines)):
    lines[i] = lines[i].strip()

print(lines)

```

```

['10\n', '20\n', '30\n', '40\n', '50\n', '60\n', '70\n', '80\n', '90\n', '100\n']
['10', '20', '30', '40', '50', '60', '70', '80', '90', '100']

```

Function and Recursion

ในบางครั้งเราจำเป็นต้องทำการวนการบางอย่างซ้ำๆ กัน แต่ว่าเปลี่ยนค่าตัวแปรที่ควบคุมกระบวนการ (ชื่oreียกว่า พารามิเตอร์ – parameter) เหล่านั้นไปตามสถานการณ์ ทำให้เราต้องก็อปปี้ แปะโค้ดและแก้โค้ดซ้ำๆไปซ้ำมา การกระทำดังกล่าวอาจก่อให้บก (bug) ได้ในภายหลัง หากเราต้องแก้ไขกระบวนการหนึ่งจุด เราจะต้องไล่ตามไปแก้ไขทุกจุดที่เรา ก็อปปี้ แปะมา ไม่เช่นนั้นโปรแกรมก็จะทำงานไม่เหมือนกัน ปัญหานี้เรียกว่า ปัญหาความไม่เป็นหน่วยเดียวกัน (modularity issue)

กระบวนการที่เรา ก็อปปี้ แปะมานั้น มักจะต่อ ya กันเป็นพรีด หากมาอ่านโค้ดเองในภายหลัง ก็อาจจะลืมไปแล้วว่าโค้ดส่วนนี้ทำหน้าที่อะไร ปัญหานี้เรียกว่า ปัญหาการอ่านโค้ดไม่ออก (code readability issue)

การ ก็อปปี้ แปะโค้ดซ้ำๆไปซ้ำมา ทำให้ไม่สามารถเรียกตัวเองซ้ำแบบเวียนเกิดได้ ปัญหานี้เรียกว่า ปัญหาการเรียกตัวเองซ้ำ (self-callability issue)

เพื่อแก้ปัญหาดังกล่าว เราจะนำกระบวนการที่ต้องทำซ้ำๆ แต่เปลี่ยนค่าตัวแปรที่ควบคุมกระบวนการเหล่านั้น มารวมกลุ่มเป็นฟังก์ชัน (function)

- การนิยามฟังก์ชัน

เราสามารถนิยามฟังก์ชันได้ด้วยคำสั่ง def (ย่อมาจากคำว่า define) ดังนี้

```

def <ชื่อฟังก์ชัน> ( <รายการพารามิเตอร์> ) :
    <กระบวนการ>

```

เช่น เราสามารถนิยามฟังก์ชันสำหรับรับพารามิเตอร์ ได้แก่ ชื่อนักเรียน อายุ และความสูง แล้วแสดงผลออกหน้าจอ ได้ ดังนี้

```

def print_student(name, age, height):
    print(f'Name : {name}')
    print(f'Age : {age:d} y/o')
    print(f'Height : {height:6.2f} cm')
    print()

```

เมื่อเรานิยามฟังก์ชันแล้ว เรา ก็สามารถเรียกฟังก์ชันนั้นได้ทันที

```

student_scores = [
    ('John', 13, 165.5), ('Mary', 14, 173.0), ('Cindy', 12, 163.5)
]

```

```
for name, age, height in student_scores:  
    print_student(name, age, height)
```

```
Name      : John  
Age       : 13 y/o  
Height    : 165.50 cm
```

```
Name      : Mary  
Age       : 14 y/o  
Height    : 173.00 cm
```

```
Name      : Cindy  
Age       : 12 y/o  
Height    : 163.50 cm
```

○ การคืนค่าฟังก์ชันด้วยคำสั่ง return

นอกจากฟังก์ชันจะรวมกระบวนการให้เป็นหน่วยเดียวได้แล้ว เราจึงสามารถกำหนดให้ฟังก์ชันทำการคำนวนบางอย่างในกระบวนการ แล้วคืนค่าผลลัพธ์กลับออกมายังตัวเราได้อีกด้วย เราสามารถสั่งให้ฟังก์ชันหยุดการทำงานและคืนค่ากลับออกมายังตัวเราด้วยคำสั่ง `return` โดยมีรูปแบบ ดังนี้

```
return <ค่าที่คืนกลับไป>
```

เช่น เราสามารถเขียนฟังก์ชันให้คำนวนค่าเฉลี่ย (average) ของสมาชิกในลิสต์ แล้วคืนค่าเฉลี่ยนั้นกลับออกไปได้ ดังนี้

```
def average(items):  
    my_sum = 0.0  
    for item in items:  
        my_sum = my_sum + item  
    return my_sum / len(items)
```

เราสามารถนำค่าที่ฟังก์ชันคืนกลับออกมายังตัวเราไปใช้ต่อได้ทันที

```
student_heights = [163.5, 150.0, 167.0, 161.25, 170.0]  
  
avg = average(student_heights)  
print(f'The average of student heights is {avg:.2f}.')
```

The average of student heights is 162.35.

○ การหยุดการทำงานของฟังก์ชันด้วยคำสั่ง return

เราสามารถหยุดการทำงานของฟังก์ชันกลางคันได้ ด้วยคำสั่ง `return` โดยโปรแกรมจะออกจากฟังก์ชันทันที จริงๆ แล้วในคำสั่ง `return` นี้เราอาจจะใส่ค่าที่คืนกลับไปก็ได้ หรือไม่ใส่ลงไปก็ได้

หากไม่ใส่ค่าคืนกลับ โปรแกรมก็เพียงหยุดการทำงานของฟังก์ชัน แล้วย้อนกลับไปที่จุดเดิมที่เรียกฟังก์ชันนั้น

แต่ถ้าหากใส่ค่าคืนกลับ โปรแกรมก็จะหยุดการทำงานของฟังก์ชัน และย้อนกลับไปที่จุดเดิม พร้อมกับใส่ค่าคืนกลับที่ทำແเน່ນของฟังก์ชันนั้น

เรามาลองสังเกตฟังก์ชันดังต่อไปนี้กัน ฟังก์ชันนี้จะคำนวณค่า factorial ของจำนวนเต็ม n ซึ่งແเน່ນด้วยสูตรการคำนวณจะเป็นดังนี้

$$0! = 1$$

$$1! = 1$$

$$n! = 1 \times 2 \times 3 \times \dots \times n ; \text{ เมื่อ } n > 1$$

หากเราใส่พารามิเตอร์ n เป็น 0 หรือ 1 ฟังก์ชันนี้จะคืนค่า 1 อกไไปแล้วหยุดการทำงานทันที

```
# คำนวณค่า factorial ของจำนวนเต็ม n
def factorial(n):

    # หากค่า n เป็น 0 หรือ 1
    if n in [0, 1]:
        # ให้คืนค่า 1 กลับไป แล้วหยุดการทำงานทันที
        return 1

    # วนลูปคูณเลข
    result = 1.0
    for i in range(2, n + 1):
        result = result * i

    # คืนผลลัพธ์กลับออกไไป
    return result
```

เรามาลองทดสอบฟังก์ชัน factorial กัน

```
print(factorial(0))
print(factorial(20))
```

```
1
2.43290200817664e+18
```

ในกรณีที่ฟังก์ชันเป็นเพียงการรวมกระบวนการให้เป็นหน่วย ไม่ได้มีจุดประสงค์จะคืนค่า กลับออกไไป เรายังสามารถใช้คำสั่ง return ได้ด้วย เพียงแต่เราไม่ต้องระบุค่าคืนกลับเท่านั้นเอง

ในฟังก์ชันด้านล่างนี้ ถ้าหากพารามิเตอร์ age มีค่า 60 ขึ้นไป ฟังก์ชันจะพิมพ์ข้อความ --Senior-- แล้วจบการทำงานทันที

```
def print_student(name, age, height):
    # หาก age มีค่า 60 ขึ้นไป
    if age >= 60:
        print('--Senior--')
        print()
    # หยุดการทำงานของฟังก์ชัน
    return

    print(f'Name : {name}')
```

```

print(f'Age    : {age:d} y/o')
print(f'Height : {height:.2f} cm')
print()

student_scores = [
    ('John', 13, 165.5), ('Mary', 14, 173.0), ('Cindy', 12, 163.5),
    ('Magdalene', 65, 165.5), ('Eugene', 15, 177.0)]

for name, age, height in student_scores:
    print_student(name, age, height)

```

Name : John
 Age : 13 y/o
 Height : 165.50 cm

Name : Mary
 Age : 14 y/o
 Height : 173.00 cm

Name : Cindy
 Age : 12 y/o
 Height : 163.50 cm

--Senior--

Name : Eugene
 Age : 15 y/o
 Height : 177.00 cm

○ การทำให้โค้ดอ่านง่ายขึ้น

โปรแกรมเมอร์ที่ดีจะมีความสามารถที่จะทำให้โปรแกรมเมอร์คนอื่นสามารถอ่านโค้ดของเขารู้เรื่องได้ในภายหลัง การที่จะทำอย่างนั้นได้ เราควรเรียนโค้ดตามคำแนะนำดังต่อไปนี้

- พยากรณ์กระบวนการที่ก็อบปี้แปะช้าๆ กัน ให้กลายเป็นฟังก์ชัน การกระทำเช่นนี้เรียกว่า การรวมเป็นหน่วย (modularization)
- พยากรณ์ตั้งชื่อตัวแปร ค่าคงที่ พารามิเตอร์ และชื่อฟังก์ชันต่างๆ ให้มีสื่อความหมายและสะท้อนหน้าที่ของมัน การกระทำเช่นนี้เรียกว่า การตั้งชื่อให้มีความหมาย (meaningful naming)
- หากฟังก์ชันมีรายการพารามิเตอร์ยาวผิดปกติ หรือฟังก์ชันมีความยาวผิดปกติ ให้ลองแยกฟังก์ชันให้กลายเป็นฟังก์ชันย่อย และเรียกใช้ฟังก์ชันเหล่านั้นแทน การกระทำเช่นนี้เรียกว่า การรีเฟคเตอร์โค้ด (code refactoring)
- พยากรณ์ให้หมายเหตุลงในโค้ด การกระทำเช่นนี้เรียกว่า การใส่หมายเหตุที่สมเหตุผล (appropriate documentation)
- อย่าเพิ่งพยากรณ์ปรับโค้ดให้เร็วขึ้นในตอนต้น ขอให้เขียนโค้ดให้ทำงานตรงตามความคิดเสียก่อน

สิ่งที่ใส่ในหมายเหตุได้มีอะไรบ้าง

1. ภาพรวมของกระบวนการ ซึ่งสามารถอธิบายให้คนทั่วไปเข้าใจได้ด้วยภาษา�นุษย์
2. ชื่อเต็มของตัวแปร ค่าคงที่ พารามิเตอร์ และฟังก์ชัน ในกรณีที่ชื่อเหล่านี้เป็นตัวย่อหรือมีความหมายพิเศษที่ต้องการคำอธิบายเพิ่มเติม
3. วิธีการใช้งานฟังก์ชัน หากว่าฟังก์ชันนั้นมีรายการพารามิเตอร์ที่ค่อนข้าง слับซับซ้อน
4. เคล็ดลับแม่บ้าน (procedural protocols) ทั้งหลาย
5. ชื่อของผู้พัฒนา และไลเซนส์ของโค้ด

○ การรีเฟคเตอร์โค้ด

หากเราพบว่าฟังก์ชันใดมีจำนวนพารามิเตอร์เยอะๆ (เช่น เกิน 5 ตัว) ให้เราลองลังเลตความล้มพันธุ์ของพารามิเตอร์และการทำงานของโค้ดดูก่อน หากเราสามารถจัดกลุ่มพารามิเตอร์และโค้ดที่เกี่ยวข้องได้ ให้เราจารณาที่จะแยกหั้งกลุ่มนี้ไปตั้งเป็นฟังก์ชันใหม่แล้วเรียกใช้ฟังก์ชันนี้แทน โค้ดที่ได้น่าจะอ่านง่ายขึ้นเยอะ

○ ฟังก์ชันเรียก返 (Recursion)

ฟังก์ชันเรียก返 (recursive function) คือฟังก์ชันที่มีการเรียกตัวเองซ้ำไปซ้ำมา เช่น ฟังก์ชันคำนวณค่าแฟคทอเรียล (factorial) $n!=1\times2\times3\times\ldots\times n$ สามารถเขียนให้อยู่ในรูปฟังก์ชันเรียก返ได้ดังนี้

$$\text{factorial}(n) = \begin{cases} n \times \text{factorial}(n-1) & \text{เมื่อ } n > 0 \\ 1 & \text{มิฉะนั้น} \end{cases}$$

ฟังก์ชันเรียก返จะมีองค์ประกอบ 2 ส่วน คือ

1. ส่วนเรียก返 (recursive part) เป็นส่วนของฟังก์ชันที่มีการเรียกตัวเองซ้ำ
2. เงื่อนไขการหยุดทำงาน (termination condition) เป็นเงื่อนไขที่ทำให้ฟังก์ชันเรียก返หยุดทำงาน ในส่วนนี้จะไม่มีการเรียกตัวเองซ้ำอีก

ยกตัวอย่างเช่น เวลาที่เราจะคำนวณค่า factorial(5) เราจะทำการส่วนเรียก返และเงื่อนไขการหยุดทำงานดังต่อไปนี้

$$\begin{aligned} \text{factorial}(5) &= 5 \times \text{factorial}(4) \\ &= 5 \times 4 \times \text{factorial}(3) \\ &= 5 \times 4 \times 3 \times \text{factorial}(2) \\ &= 5 \times 4 \times 3 \times 2 \times \text{factorial}(1) \\ &= 5 \times 4 \times 3 \times 2 \times 1 \times \text{factorial}(0) \\ &= 5 \times 4 \times 3 \times 2 \times 1 \times 1 \end{aligned}$$

ฟังก์ชันแฟคทอเรียลนี้สามารถเขียนเป็นโปรแกรมภาษา Python ได้ดังนี้

```
# ฟังก์ชันแฟคทอเรียล
def factorial(n):
    # เงื่อนไขการหยุดทำงาน
    if n == 0:
```

```

        return 1
    # สวนเวียนเกิด
    else:
        return n * factorial(n - 1) # เรียกฟังก์ชัน factorial ข้ามที่ตรงนี้

factorial(30)

```

265252859812191058636308480000000

- การจดจำรายทาง (Memorization)

ในบางครั้งการคำนวณฟังก์ชันเวียนเกิดจะมีลิ่งที่ต้องคำนวณซ้ำกันด้วย เช่น อนุกรมของพีโบนัชชีจะเป็น ดังนี้ 0,1,2,3,5,8,13,21,... โดยตัวเลขลัดไปจะเกิดจากการเราตัวเลข 2 ตัวก่อนหน้ามาบวกกัน

ฟังก์ชัน Fibonacci (พีโบนัชชี) จะมีนิยามดังนี้

$$\text{fibonacci}(n) = \begin{cases} \text{fibonacci}(n-1) + \text{fibonacci}(n-2) & \text{เมื่อ } n > 1 \\ 0 & ; n=0 \\ 1 & ; n=1 \end{cases}$$

สมมติว่า เราจะคำนวณค่า $\text{fibonacci}(4)$ เราจะต้องทำดังนี้

$$\begin{aligned} \text{fibonacci}(4) &= \text{fibonacci}(3) + \text{fibonacci}(2) \\ &= [\text{fibonacci}(2) + \text{fibonacci}(1)] + [\text{fibonacci}(0) + \text{fibonacci}(1)] \\ &= [(\text{fibonacci}(1) + \text{fibonacci}(0)) + \text{fibonacci}(1)] + [\text{fibonacci}(0) + \text{fibonacci}(1)] \\ &= ((1+0)+1) + (0+1) \\ &= 3 \end{aligned}$$

ฟังก์ชันพีโบนัชชีสามารถเขียนด้วย Python ได้ดังนี้

```

def fibonacci(n):
    if n == 0 or n == 1:
        return n
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)
fibonacci(10)

```

55

อย่างไรก็ตาม ในการคำนวณค่า $\text{fibonacci}(4)$ เราต้องคำนวณค่าของ $\text{fibonacci}(2)$ ซ้ำสองรอบ เราไม่จำเป็นต้องคำนวณลิ่งที่เคยคำนวณไปแล้วก็ได้ เราควรจะบันทึกลิ่งที่เราเคยคำนวณไว้แล้วเก็บไว้ในหน่วยความจำ เราเรียกกระบวนการนี้ว่า การจดจำรายทาง (memoization) ในตัวอย่างด้านล่างนี้เราใช้การจดจำรายทางค่าตอบของฟังก์ชันพีโบนัชชีด้วยการเก็บลงดิกชันnaire

```

fibonacci_memory = {}

```

```

def fibonacci_memo(n):
    if n not in fibonacci_memory:
        if n == 0 or n == 1:
            fibonacci_memory[n] = n
        else:
            fibonacci_memory[n] = fibonacci_memo(n - 1) +
fibonacci_memo(n - 2)
    return fibonacci_memory[n]

fibonacci_memo(4) # 3
fibonacci_memo(100) # 354224848179261915075

```

การจดจำรายทางจะช่วยให้เราสามารถคำนวณฟังก์ชันเวียนเกิดได้รวดเร็วขึ้นมาก

การโปรแกรมเชิงวัตถุ

ในหัวข้อที่แล้ว เราได้เรียนรู้การรวมกระบวนการให้เป็นหน่วยด้วยฟังก์ชัน นั่นคือเราพยายามออกแบบโปรแกรมโดยเน้นที่กระบวนการของการทำ แต่ในบางกรณี การออกแบบโปรแกรมโดยเน้นกระบวนการอาจจะทำให้การเขียนโปรแกรมยุ่งยากมากขึ้น เช่น เราต้องการใช้โปรแกรมเพื่อจัดการกับรูปทรงต่างๆ เช่น สามเหลี่ยม สี่เหลี่ยมจัตุรัส ฯลฯ โดยมีการกระทำดังต่อไปนี้

- A. หาพื้นที่ได้
- B. หาเส้นรอบวงได้
- C. หาสีของรูปทรงได้
- D. เปลี่ยนข้อมูลจำเพาะของรูปทรงได้ เช่น รัศมี ความกว้าง ความยาว ฯลฯ
- E. เปลี่ยนสีของรูปทรงได้

ถ้าเราเขียนโปรแกรมด้วยฟังก์ชันทั้งหมด เราจะเขียนโปรแกรมได้ดังนี้

```

pi = 3.1415926

# คำนวณพื้นที่วงกลม
def area_circle(circle):
    return pi * circle['radius']**2

# คำนวณเส้นรอบวงของวงกลม
def circumference_circle(circle):
    return 2 * pi * circle['radius']

# หาค่าสีของวงกลม
def color_circle(circle):
    return circle['color']

# คำนวณพื้นที่จัตุรัส
def area_square(square):
    return square['side']**2

```

```

# คำนวณเส้นรอบวงจัตุรัส
def circumference_square(square):
    return 4 * square['side']

# หาค่าสีของจัตุรัส
def color_square(square):
    return square['color']

```

จะสังเกตได้ว่า เรายังต้องเขียนฟังก์ชันซ้ำกันอยู่หลายอัน แม้ว่าฟังก์ชันเหล่านี้จะทำงานคล้ายคลึงกันก็ตาม เช่น color_circle กับ color_square

นอกจากนี้ เวลาที่เราสร้างวัตถุใหม่ขึ้นมา เราจะสร้างด้วยดิจิทัลนารี และเราก็จะต้องก็อปปี้ โครงสร้างของดิจิทัลนารีให้เหมือนเป็นบด้วย มิเช่นนั้นโปรแกรมก็อาจทำงานผิดพลาดได้ เพราะอ้างพิเศษหรืออ้างคีย์ไม่ครบ เช่น

```

circle1 = {'radius': 30.0, 'color': 'red'}
circle2 = {'radius': 15.5, 'color': 'green'}
circle3 = {'radius': 10.0, 'color': 'blue'}

square1 = {'side': 10.0, 'color': 'yellow'}
square2 = {'side': 15.0, 'color': 'orange'}
square3 = {'side': 5.75, 'color': 'pink'}

print(f'Area of circle1 = {area_circle(circle1)}')
print(f'Circumference of circle2 = {circumference_circle(circle2)}')
print(f'Color of circle3 = {color_circle(circle3)}')
print()

print(f'Area of square1 = {area_square(square1)}')
print(f'Circumference of square2 = {circumference_square(square2)}')
print(f'Color of square3 = {color_square(square3)}')

```

```

Area of circle1 = 2827.43334
Circumference of circle2 = 97.3893706
Color of circle3 = blue

```

```

Area of square1 = 100.0
Circumference of square2 = 60.0
Color of square3 = pink

```

○ คลาส

เพื่อให้อ่านโค้ดได้ง่ายขึ้น เรายังสามารถรวมกระบวนการทางตามชนิดของวัตถุได้อีกด้วย โดยวัตถุแต่ละชนิดจะมีฟังก์ชันหน้าตาเหมือนกัน แต่ใช้ในของแต่ละฟังก์ชันจะแตกต่างกันไปตามชนิดของวัตถุ

การรวบรวมกระบวนการตามชนิดของวัตถุนี้
(object-oriented programming หรือ OOP) ในภาษา Python เราสามารถสร้างวัตถุชนิดใหม่ได้ด้วยคำสั่ง class ดังรูปแบบต่อไปนี้

```
class <ชื่อคลาส> :  
    <รายการฟังก์ชัน>
```

เราสามารถนิยามฟังก์ชันภายในคลาสได้ด้วยคำสั่ง def เมื่อกับฟังก์ชันปกติเลย ฟังก์ชันภายในคลาสนี้มีชื่อเรียกอย่างเป็นทางการว่า เมธอด (method)

เมธอดสำคัญของคลาสคือ คอนสตรักเตอร์ (constructor) มีไว้สำหรับสร้างวัตถุ (object) จากคลาส โดยคอนสตรักเตอร์จะใช้ชื่อเมธอดว่า __init__ (โปรดสังเกตว่าชื่อ init จะมีจุดเล่นใต้ 2 อันที่ด้านหน้าและด้านหลัง)

คอนสตรักเตอร์จะรับพารามิเตอร์ 2 ชุด

- ★ self สำหรับอ้างถึงวัตถุที่เรากำลังจะสร้าง
- ★ พารามิเตอร์ที่จะใส่ในวัตถุนั้น

เช่น เราสามารถนิยามคลาส Circle และคอนสตรักเตอร์ของคลาสนี้ได้ดังนี้

```
class Circle:  
  
    def __init__(self, color, radius):  
        self.color = color  
        self.radius = radius
```

คอนสตรักเตอร์ของคลาส Circle จะรับพารามิเตอร์สำหรับคลาส 2 ตัวคือ color และ radius เมื่อเรารับพารามิเตอร์เหล่านี้มาแล้ว เราอาจจะนำมาใส่เป็นคุณสมบัติ (property) ของคลาส โดยเราจะกำหนดคุณสมบัติของคลาสด้วยการอ้างถึงตัวแปร self เช่น

```
def __init__(self, color, radius):  
    self.color = color  
    self.radius = radius
```

นอกจากนี้เรายังสามารถนิยามเมธอดอื่นๆ สำหรับวัตถุของคลาส Circle ได้อีกด้วย ในตัวอย่างข้างล่างนี้ เราจะนิยามฟังก์ชันเพิ่มอีก 3 ตัว ได้แก่ area, circumference, และ get_color

```
pi = 3.1415926  
  
class Circle:  
  
    # คอนสตรักเตอร์  
    def __init__(self, color, radius):  
  
        # กำหนดสีจากพารามิเตอร์  
        self.color = color  
  
        # กำหนดรัศมีจากพารามิเตอร์  
        self.radius = radius
```

```

# คำนวณพื้นที่
def area(self):
    return pi * self.radius**2

# คำนวณเส้นรอบวง
def circumference(self):
    return 2 * pi * self.radius

# หาค่าสี
def get_color(self):
    return self.color

```

เราสามารถประกาศวัตถุของคลาสได้โดยเรียกชื่อคลาส ตามด้วยพารามิเตอร์ในคอนสตรัคเตอร์

ในตัวอย่างด้านล่างนี้ เราประกาศวัตถุของคลาส Circle ออกมา 3 ตัว สังเกตว่าคราวนี้เราจะสร้างวัตถุได้ง่ายกว่าการสร้างติกขั้นนารีเองมาก

จริงๆ แล้วการสร้างคลาสเปรียบเสมือนการสร้างแม่พิมพ์ (template) เพื่อบีบปั๊วัตถุจำนวนมาก ๆ ที่มีหน้าตาคล้ายคลึงกันออกมา

```

circle1 = Circle('red', 30.0)
circle2 = Circle('green', 15.5)
circle3 = Circle('blue', 10.0)

```

และเราก็สามารถเรียกเมธอดของวัตถุนี้ได้ตามรูปแบบ <วัตถุ.<เมธอด>(<พารามิเตอร์>)

```

print('Area of circle1 =', circle1.area())
print('Circumference of circle2 =', circle2.circumference())
print('Color of circle3 =', circle3.get_color())

```

```

Area of circle1 = 2827.43334
Circumference of circle2 = 97.3893706
Color of circle3 = blue

```

○ การถ่ายทอดคุณสมบัติของคลาส

ข้อดีอีกข้อหนึ่งของการสร้างคลาสก็คือ เราสามารถถ่ายทอดคุณสมบัติของคลาสให้กับคลาสลูกหลานได้อีกด้วย

เช่น ถ้าเราสร้างคลาส Circle และ Square ให้ดี เราจะพบว่าทั้งสองคลาสนี้มีคุณสมบัติร่วมกันคือ มี color (สี) เมื่อกัน และมีเมธอดร่วมกันคือ get_color (หาค่าสี)

ที่เป็นเช่นนี้ก็เพราะว่า จริงๆ แล้วคลาส Circle (วงกลม) และ Square (สี่เหลี่ยมจัตุรัส) ต่างก็เป็นรูปทรงด้วยกันทั้งสิ้น เราจึงมองได้ว่าทั้งสองคลาสนี้เป็นคลาสลูกของคลาส Shape ด้วยกันทั้งสิ้น

```

class Shape:

    def __init__(self, color):
        self.color = color

    # หาค่าสี

```

```
def get_color(self):
    return self.color
```

หากเราต้องการนิยามคลาสใหม่ให้เป็นคลาสลูก เราสามารถใช้คำสั่ง class ได้ตามรูปแบบนี้

```
class <คลาสลูก> (<คลาสแม่> ) :
    <รายการเมธอด>
```

เมื่อเราประกาศคลาสลูกขึ้นมาแล้ว เราจะต้องนิยามคุณสมบัติที่ต้องการให้เรียกใช้คุณสมบัติ
ของคลาสแม่ด้วย

วิธีการเรียกใช้คุณสมบัติที่ต้องการของคลาสแม่คือใช้คำสั่ง

```
super(<คลาสลูก>, self).__init__(<พารามิเตอร์สำหรับคลาสแม่>)
```

เช่น เราสามารถนิยามคลาส Circle ให้เป็นคลาสลูกของ Shape ได้ดังนี้

```
class Circle(Shape):
```

```
    def __init__(self, color, radius):
```

นี่คือวิธีการเรียกคุณสมบัติที่ต้องการของคลาสแม่ โดยเราจะต้องป้อนพารามิเตอร์ที่จำเป็น
สำหรับคลาสแม่เข้าไปด้วย

```
        super(Circle, self).__init__(color)
```

หลังจากนั้นให้เรากำหนดคุณสมบัติของคลาสลูกตามปกติ
 self.radius = radius

```
# คำนวณพื้นที่
```

```
    def area(self):
        return pi * self.radius**2
```

```
# คำนวณเส้นรอบวง
```

```
    def circumference(self):
        return 2 * pi * self.radius
```

จะสังเกตว่า คราวนี้เราจะไม่ต้องนิยามฟังก์ชัน get_color อีกแล้ว เพราะฟังก์ชันนี้ได้รับการ
ถ่ายทอดมาจากคลาส Shape อยู่แล้วนั่นเอง

```
my_circle = Circle('red', 30.0)
print('Area =', my_circle.area())
print('Circumference =', my_circle.circumference())
print('Color =', my_circle.get_color())
```

Area = 2827.43334

Circumference = 188.495556

Color = red

แบบฝึกหัดเพิ่มเติม

1. Data Type, Variable, and Expression

- a. Input: รับจำนวนเต็ม 3 จำนวนจากแป้นพิมพ์ (บรรทัดละจำนวน) เก็บในตัวแปร h, m และ s ซึ่งแทนจำนวนชั่วโมง นาที และ วินาที
Process: คำนวณจำนวนวินาทีรวมที่ได้จากการบวกของ h, m และ s
Output: จำนวนวินาทีรวมทั้งหมดที่คำนวณได้
- b. Input: รับจำนวนจริง 1 จำนวนจากแป้นพิมพ์ เก็บใน x
Process: คำนวณ $y = 2 - x + \frac{3}{7}x^2 - \frac{5}{11}x^3 + \log_{10}(x)$
Output: ค่า y ที่คำนวณได้
- c. Input: รับจำนวนจริง 1 จำนวนจากแป้นพิมพ์เก็บใน a
Process: ให้ x มีค่าเป็น 1 จากนั้นทำคำสั่ง $x = (x - a/x)/2$ จำนวน 4 ครั้ง
Output: ค่า x ที่ได้จากการทำงานข้างบนนี้
- d. Input: มี 2 บรรทัด แต่ละบรรทัดมีจำนวนจริง 3 จำนวน คั่นด้วย ช่องว่าง อ่านบรรทัดแรกเก็บใน v1, v2, v3 แทนเวกเตอร์ $v = (v1, v2, v3)$
อ่านบรรทัดที่สองเก็บใส่ u1, u2, u3 แทน เวกเตอร์ $u = (u1, u2, u3)$
Process: คำนวณ dot product ของเวกเตอร์ v กับ u
Output: ค่า dot product ที่คำนวณได้
- e. Input: อ่านจำนวนจริง 4 จำนวนคั่นด้วยช่องว่างจากแป้นพิมพ์ เก็บใน x1, y1, x2 และ y2 ค่าของ x1, y1 แทนพิกัดของจุดที่ 1 และ x2, y2 แทนพิกัดของจุดที่ 2 บนระบบ x-y
Process: คำนวณระยะห่างลั่นสูตรระหว่างจุดทั้งสอง
Output: ระยะห่างที่หาได้
- f. Input: อ่านพิกัดเชิงข้อของจุดบนระบบ直角坐标系 ซึ่งเป็นจำนวนจริง 2 จำนวนคั่นด้วยช่องว่าง เก็บในตัวแปร r และ theta (เป็นเรเดียน)
Process: คำนวณค่า x และ y ซึ่งเป็นพิกัดคาร์ทีเซียน ของจุด (r, theta) ที่อ่านเข้ามา
Output: ค่า x และ y (คั่นด้วยช่องว่าง)
- g. Input: อ่านพิกัดคาร์ทีเซียนของจุดบนระบบ直角坐标系 ซึ่งเป็นจำนวนจริง 2 จำนวนคั่นด้วยช่องว่าง เก็บในตัวแปร x และ y
Process: คำนวณค่า r, theta (เป็นเรเดียน) ซึ่งเป็น พิกัดเชิงข้อของจุด (x, y)
Output: ค่า r และ theta (คั่นด้วยช่องว่าง)
- h. Input: อ่านจำนวนจริง 5 จำนวน คั่นด้วยช่องว่าง
Process: คำนวณค่าเฉลี่ยของจำนวนทั้ง 5
Output: ค่าเฉลี่ยที่หาได้

- i. Input: รับข้อมูล 3 ตัว a, b กับ c คืนด้วยช่องว่าง a และ b เป็นตัวอักษร ส่วน c เป็นจำนวนเต็ม
 Output: ตัวอักษรใน a ต่อกับตัวอักษรใน b ต่อ กับ ค่าของจำนวนเต็มใน c ต่อ กับ ชุดของตัวอักษรใน a ต่อ กับ ตัวอักษร ใน b ที่ซ้ำ ๆ กันเป็นจำนวน c ชุด เช่น ผู้ใช้ป้อน v o 5 จะแสดง vo5vovovovo

2. Condition/Selection/Branching

- a. Input: รับจำนวนเต็ม 3 จำนวน คืนด้วยช่องว่าง
 Process: หากยืนยันของจำนวนทั้ง 3
 Output: มัตยฐานที่หาได้
- b. Input: รับข้อมูลของวงกลม 2 วง บรรทัดละหนึ่งวง ประกอบด้วยจำนวนจริง 3 จำนวนคืนด้วยช่องว่าง แทน พิกัด x กับ y ของจุดศูนย์กลาง และรัศมี ของวงกลม
 Process: ตรวจสอบว่าวงกลมสองวงที่รับมาทับกันหรือแตะกัน หรือไม่
 Output: แสดงคำว่า touch เมื่อขอบของทั้งสองวงแตะกัน พอตี แสดงคำว่า overlap เมื่อสองวงทับกัน ถ้าไม่แตะ หรือทับ ให้แสดงคำว่า free
- c. Input: รับจำนวนจริง 2 จำนวน คืนด้วยช่องว่าง แทนพิกัด (x, y) บนระนาบ สอนมิติ
 Process: ตรวจสอบว่าพิกัด (x, y) อยู่บนริเวณใดในระนาบ
 Output: ตำแหน่งของพิกัด (x, y) ว่า อยู่ในจตุภาคใด หรืออยู่บนแกน x หรือ y หรืออยู่ที่จุดกำเนิด
- d. Input: รับจำนวนเต็ม 5 จำนวน คืนด้วยช่องว่าง
 Process: ตรวจสอบว่าลำดับจากซ้ายไปขวาของจำนวนที่รับมา เรียงจากน้อยไปมากหรือไม่
 Output: ผลการตรวจว่า True หรือ False
- e. Input: รับจำนวนเต็ม 4 จำนวน คืนด้วยช่องว่าง
 Process: หากรวมของจำนวนที่รับมา โดยไม่รวมจำนวนที่มากสุดหนึ่งจำนวน และจำนวนที่น้อยสุดหนึ่งจำนวน
 Output: ผลรวมที่หาได้
- f. Input: รับจำนวนเต็มหนึ่งจำนวนเก็บในตัวแปร a
 Process: ตรวจสอบว่ามีจำนวนเต็ม x ที่ค่า $x \times 3$ เท่ากับ a หรือไม่
 Output: ถ้ามี แสดงค่าของ x ถ้าไม่มี แสดง Not Found
- g. Input: รับจำนวนเต็มแทนรอบอก (หน่วยเป็นนิ้ว)
 Process: หากขนาดของเลือดปอดตามรอบอกดังนี้ น้อยกว่า 37 นิ้ว ขนาด XS ตั้งแต่ 37 แต่ไม่ถึง 41 นิ้ว ขนาด S ตั้งแต่ 41 แต่ไม่ถึง 43 นิ้ว ขนาด

M ตั้งแต่ 43 แต่ไม่ถึง 46 นิ้ว ขนาด L ตั้งแต่ 46 นิ้วเป็นต้นไป ขนาด XL

Output: ขนาดเลือปโภคภารกษาที่ได้รับ

3. Iteration/Repetition

a. Input: ไม่มี

Process: หาจำนวนเต็มบวก k ที่มีค่าน้อยสุดที่ทำให้มี $\left(\frac{1}{k}\right)k$ ค่าไม่เท่ากับ 1
(เนื่องจากความไม่แม่นยำ 100% ของจำนวนจริงในคอมพิวเตอร์)
Output: จำนวนเต็ม k ที่หาได้

b. Input: ไม่มี

Process: หาจำนวนเต็มบวก k ที่มีค่าน้อยที่สุดที่ทำให้
$$1 - \left(\frac{365}{365}\right)\left(\frac{365-1}{365}\right)\left(\frac{365-2}{365}\right) \cdots \left(\frac{365-k}{365}\right) \geq 0.5$$
 เป็นจริง
Output: จำนวนเต็ม k ที่หาได้

c. Input: ไม่มี

Process: คำนวณค่าประมาณของ π จากสูตร

$$4\left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} + \cdots + \frac{1}{399997} - \frac{1}{399999}\right)$$

Output: ค่าประมาณของ π ที่หาได้

d. Input: อ่านจำนวนเต็ม 2 จำนวน a กับ b

Process: คำนวณค่าจากสูตร $\sum_{i=a}^{b-1} (-1)^i \sum_{j=i+1}^b (i+j)$

Output: ค่าที่คำนวณได้

e. Input: อ่านจำนวนเต็ม 2 จำนวน a กับ b

Process: คำนวณค่าจากสูตร $\sum_{a \leq i < j \leq b} (-1)^i (i+j)$

Output: ค่าที่คำนวณได้

f. Input: บรรทัดแรกรับจำนวนเต็มเก็บในตัวแปร n (n จะมีค่ามากกว่า 0) และ
อีก n บรรทัด เป็นจำนวนเต็ม บรรทัดละจำนวน

Process: หากลั่งของค่ามากสุดกับค่าน้อยสุด และหาว่ามีกี่จำนวนที่เป็น
เลขลบ

Output: ผลลั่ง และจำนวนเลขลบที่หาได้

g. Input: จำนวนเต็มเก็บในตัวแปร n

Process: หาจำนวนเต็มบวก w, x, y, z ทั้งหมดที่ $w^3 = x^2 + y^2 + z^2$ โดยที่
 $1 \leq x \leq y \leq z \leq n$

Output: ค่าของ w, x, y, z ที่หาได้

4. File

- a. Input: หนังบรรทัดเป็นชื่อแฟ้ม

Process: อ่านข้อความในแฟ้มมากลับลำดับบรรทัด Output: แสดงข้อความในแฟ้มแบบกลับลำดับบรรทัด ออกทางจอภาพ เช่น

ข้อมูลในแฟ้ม	ผลลัพธ์ (ออกทางจอ)
line1	line1
line2	line2
line3	line3

- b. Input: หนังบรรทัดเป็นชื่อแฟ้ม

Process: อ่านข้อความในแฟ้มแล้วกลับลำดับบรรทัด แต่มีเงื่อนไขว่าจะไม่เอาบรรทัดที่ว่าง ๆ หรือมีแต่ blank

Output: บันทึกข้อความแบบกลับลำดับบรรทัดลงแฟ้ม ชื่อ reverse.txt

- c. Input: หนังบรรทัดเป็นชื่อแฟ้ม

Process: แสดงหัวข้อข่าวทั้งหมดในแฟ้ม หัวข้อข่าวเป็น ข้อความที่อยู่ระหว่าง `<headline>` กับ `</headline>` ในแฟ้มนี้(หัวข้อ `<headline>` กับ `</headline>` อยู่ในบรรทัดเดียวกันแน่ ๆ และแต่ละบรรทัดมีไม่เกิน 1 หัวข้อข่าว)

Output: บรรทัดละหนึ่งหัวข้อออกทางจอภาพ ให้ครบถ้วนทุกหัวข้อ

- d. Input: สองบรรทัด แต่ละบรรทัดเป็นชื่อแฟ้ม

Process: เปรียบเทียบว่าสองแฟ้มนี้มีค่าเหมือนกันหรือไม่

Output: ถ้าแฟ้มทั้งสองมีข้อมูลเหมือนกัน แสดง True ต่างกันก็แสดง False

5. Function and Recursion

- a. ชื่อฟังก์ชัน: f1

Parameter: รับข้อมูล a เป็นสตริง และ รับ b เป็นจำนวนเต็ม

Process: พิมพ์ a ออกทางหน้าจอจำนวน b บรรทัด

Return: ไม่ต้องคืนค่า

- b. ชื่อฟังก์ชัน: f2

Parameter: รับข้อมูล a เป็นสตริง และ รับ b เป็นจำนวนเต็ม

Process: สร้างลิสต์ที่เก็บสตริง a จำนวน b ตัว

Return: ลิสต์ที่สร้าง

- c. ชื่อฟังก์ชัน: g

Parameter: รับจำนวนจริงสี่จำนวน m b n และ c

Return: คืนค่าจุดตัดของเส้นตรง $y = mx+b$ และ $y = nx+c$ เป็น tuple ของจุดตัด (x,y) หากเป็นเส้นตรงที่ขนานกัน ให้คืนค่าจำนวนเต็ม 1 หากเป็นเส้นตรงเดียวกัน ให้คืนค่าจำนวนเต็ม 2

d. ชื่อฟังก์ชัน: h

Parameter: รับลิสต์ของจำนวนเต็ม x

Return: คืนลิสต์ใหม่ ที่มีสมาชิกจาก x เฉพาะที่เป็นเลขคู่เท่านั้น โดยห้ามแก้ไขค่าในลิสต์ x

e. เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้

$$a_n = \begin{cases} 1, & n = 0 \\ -2, & n = 1 \\ a_{n-2} \times n, & \text{otherwise} \end{cases}$$

f. เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้

$$\begin{aligned} k(2n) &= k(n) + (k(n)\%10) && \text{if } n > 0 \\ k(2n+1) &= k(n-1)*n && \text{if } n > 0 \\ k(0) &= 1 \\ k(1) &= 2 \end{aligned}$$

g. เขียนฟังก์ชันเวียนเกิดเพื่อคำนวณค่าดังนี้

$$\begin{aligned} s_{i,k} &= \begin{cases} 0, & i \geq k \\ k + t_{i+1,k}, & i < k \end{cases} \\ t_{j,k} &= \begin{cases} 0, & j \geq k \\ j + s_{j,k-1}, & j < k \end{cases} \end{aligned}$$

h. ชื่อฟังก์ชัน: is_palindrome

Parameter: รับข้อมูล s เป็นสตริง

Process: เขียนฟังก์ชันเวียนเกิดเพื่อตรวจสอบว่า s เป็น palindrome (อ่านจากหน้าไปหลังเหมือนหลังไปหน้า) หรือไม่

Return: ถ้า s เป็น palindrome ให้คืนค่า True ถ้าไม่เป็น ให้คืนค่า False

6. การโปรแกรมเชิงวัตถุ

a. เติมเมธอดของคลาสนิสิตตาม comment ที่กำหนดให้สมบูรณ์

```
class Nisit:
    def __init__(self, name, year, faculty):
        # n = Nisit('Krit', 4, 'Engineering')

    def __str__(self):
        # คืนสตริงของนิสิต เช่น 'Krit (year 4) Engineering'

    def __lt__(self, rhs):
        # เรียงลำดับนิสิตด้วยคณะตามพจนานุกรม ถ้าอยู่คณะเดียวกัน ให้เรียงลำดับด้วยชื่อจากน้อยไปมาก
        # ถ้าอยู่คณะและชื่อปีเดียวกัน ให้เรียงลำดับด้วยชื่อตามพจนานุกรม
        # เช่น Nisit('Krit', 4, 'Engineering') < Nisit('Boy', 3,
```

```

'Science')
    # Nisit('Prim', 2, 'Engineering') < Nisit('Krit', 4,
'Engineering')
    # Nisit('Joey', 2, 'Engineering') < Nisit('Prim', 2,
'Engineering')

```

b. เติมเมธอดของคลาสรถยนต์ตาม comment ที่กำหนดให้สมบูรณ์

```

class Car:
    def __init__(self, license, brand, color):
        # c = Car('AA1234', 'Honda', 'White')
        # มีตัวแปร report สำหรับเก็บข้อมูลประวัติการซ่อมบำรุง โดยกำหนดค่าเริ่ม
        # ต้นเป็นลิสต์ว่าง

    def __str__(self):
        # คืนสตริงของรถยนต์ เช่น 'AA1234 - White Honda'

    def __lt__(self, rhs):
        # เรียงลำดับรถยนต์โดยเปรียบเทียบป้ายทะเบียนรถแบบสตริง

    def add_report(self, new_report):
        # เพิ่มประวัติการซ่อมบำรุง โดยไม่ต้องคืนค่า
        # ตัวแปร new_report เก็บ tuple (วันที่, ค่าอธิบาย, ราคา)
        # เช่น c.add_report( ('25 May 2017', 'change tires', 1500))

    def total_payment(self):
        # คืนค่าใช้จ่ายทั้งหมดที่ใช้ในการซ่อมบำรุงที่ผ่านมา

    def max_payment(self):
        # คืนลิสต์ของประวัติการซ่อมบำรุง (วันที่, ค่าอธิบาย, ราคา) ทุกรายการ ที่มี
        # ค่าใช้จ่ายมากที่สุด
        # กรณีที่รถยนต์ไม่มีประวัติการซ่อมบำรุงเลย ให้คืนค่าลิสต์ว่าง

```

c. จาก class Book ให้เติมเมธอดของ class ShoppingCart สำหรับการซื้อหนังสือผ่านเว็บไซต์ ดังนี้

```

class ShoppingCart:
    def __init__(self, id):
        self.id = id
        self.books = []
        # books เก็บลิสต์ของหนังสือในตะกร้าร่วมจำนวน เช่น [[b1,2],[b3,7]]

    def add_book(self, book, n):
        # เพิ่มข้อมูลการซื้อหนังสือ book เพิ่มอีก n เล่ม โดยไม่ต้องคืนค่า
        # หากไม่มีหนังสือเล่มนี้ในตะกร้า ให้เพิ่mlisrt [book, n] ต่อท้าย books

```

```

# หากเดย์มีข้อมูลหนังสือเล่มนี้ในตะกร้าแล้ว ให้เพิ่มจำนวนที่ชื่ออีก n เล่ม
# เช่น ถ้า books = [[b1,2]] และเราสั่ง add_book(b1,3) จะได้
books = [[b1,5]]

def delete_book(self, book):
    # ลบข้อมูลการซื้อหนังสือ book ออกจากตะกร้า โดยไม่ต้องคืนค่า
    # ถ้าในตะกร้าไม่มีหนังสือ book ไม่ต้องทำอะไร

def get_total(self):
    # คืนค่าราคารวมของหนังสือทั้งหมดในตะกร้า

def __lt__(self, rhs):
    # ตะกร้าที่มีราคารวมของหนังสือน้อยกว่า จะเป็นตะกร้าที่น้อยกว่า

```

- d. ข้างล่างนี้แสดงคลาส Station และคลาส BTScard (อ่านคำอธิบายของแต่ละคลาสจาก comment ที่เขียน)

สถานีรถไฟเป็นอ็อบเจกต์ของคลาส Station และบัตรโดยสารแบบเติมเงินแต่ละใบเป็นอ็อบเจกต์ของคลาส BTScard จงเติมคำสั่งในเมท็อด add_value, enter, leave และ __lt__ ของคลาส BTScard ให้ทำงานตาม comment ที่เขียน (เมท็อดอื่นที่ได้เขียนคำสั่งไว้ ทำงานถูกต้องแล้ว) ดูตัวอย่างการใช้งานข้างล่างนี้ประกอบ

```

s1 = Station('Siam'); s2 = Station('Mo Chit'); s3 = Station('Asok')
c1 = BTScard(123, 5); c2 = BTScard(999, 10)
# -----
c1.add_value(100)      # c1 มีเงินในบัตร 105 บาท
p = c1.enter(s1)       # p = True
p = c1.enter(s3)       # p = False (แตะเข้าสถานีหลังจากแตะเข้าไปแล้ว)
p = c1.leave(s2)        # c1 เหลือเงินในบัตร 95 บาท โดย p = (95, 0)
# -----
p = c2.enter(s3)       # p = True
p = c2.leave(s1)        # c2 มีเงินในบัตร 10 บาทไม่พอจ่ายค่าโดยสาร โดย p = (10, -1)
c2.add_value(50)        # c2 มีเงินในบัตร 60 บาท
p = c2.leave(s1)        # c2 เหลือเงินในบัตร 40 บาท โดย p = (40, 0)
p = c2.leave(s2)        # p = (40, -2) (ยังไม่ได้แตะเข้าสถานี จึงไม่มีสถานีต้นทาง)
p = c2.enter(s2)        # p = True
p = c1 < c2             # p = False

```

```

class Station: # คลาสของสถานีรถไฟ
    def __init__(self, id, name): # สร้างสถานีที่มีหมายเลข (id) และชื่อสถานี (name)
        self.sid = int(id) # กำหนดให้หมายเลขสถานีเป็นจำนวนเต็ม โดยสถานีที่ติดกัน
        self.name = name # มีค่าห่างกัน 1

    def get_price(self, other): # คืนค่าโดยสารระหว่างสถานี self และ

```

other

```
        return abs(self.sid - other.sid)*5

#-----

class BTScard: # คลาสของบัตรโดยสารแบบเติมเงิน
    def __init__(self, id, value): # สร้างบัตรโดยสารที่มีเลขบัตร (id) และ
        เงินเริ่มต้น (value)
        self.cid = id # self.station เก็บว่าสถานีต้นทางคือสถานีอะไร
        self.value = value # โดยถ้าบัตรไม่ได้อยู่ระหว่างการเดินทาง จะเก็บค่า
        self.station = '' # สถานีต้นทางนี้เป็นสตริงว่าง ๆ

    def __str__(self):
        return '('+str(self.cid)+','+str(self.value)+')'

    def add_value(self, x): # เพิ่มเงินในบัตรโดยสารเท่ากับ x โดยไม่ต้อง
        return

    def enter(self, station):
        # แตะบัตรเพื่อเข้าสู่สถานีรถไฟฟ้า ให้เช็คว่า บัตรนี้ไม่ได้แตะเข้าที่สถานีอื่นมา
        ก่อน
        # ถ้าไม่มีการแตะเข้ามาก่อน ให้เปลี่ยนค่าสถานีต้นทางเป็น station และ
        return True
        # แต่ถ้ามีการแตะเข้าสถานีอื่นมาก่อน ให้ return False โดยไม่เปลี่ยนข้อมูล
        สถานีต้นทางของบัตรโดยสาร

    def leave(self, station):
        # แตะบัตรเพื่ออกจากสถานีรถไฟฟ้า ให้เช็คว่า บัตรนี้มีข้อมูลสถานีต้นทางอยู่
        # ถ้าไม่มีข้อมูลสถานีต้นทาง ให้return tuple ของเงินในบัตรและ -2
        # ถ้ามีสถานีต้นทาง แต่จำนวนเงินในบัตรไม่พอจ่ายค่าโดยสาร ให้ return
        tuple ของเงินในบัตรและ -1
        # ถ้ามีสถานีต้นทาง และจำนวนเงินในบัตรพอจ่ายค่าโดยสาร ให้ลบค่าโดยสาร
        ออกจากจำนวนเงินในบัตร
        # เปลี่ยนสถานีต้นทางเป็นสตริงว่าง และ return tuple ของเงินในบัตรหลัง
        หักค่าโดยสารและ 0

    def __lt__(self, rhs): # บัตรโดยสารที่มีเงินในบัตรน้อยกว่า จะถือว่าด้อยกว่า
```

การนำเสนอภาพข้อมูลเบื้องต้น

ทำไมต้องนำเสนอข้อมูลด้วยภาพ?

ข้อมูล (Data) เป็นทรัพยากรที่สำคัญซึ่งทำให้ทีมสามารถดำเนินงาน ตัดสินใจ และพัฒนาไปในทางที่ถูกต้อง

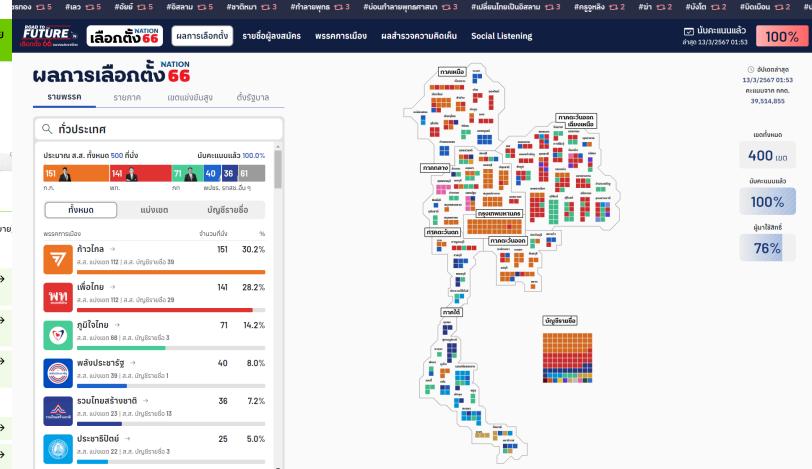
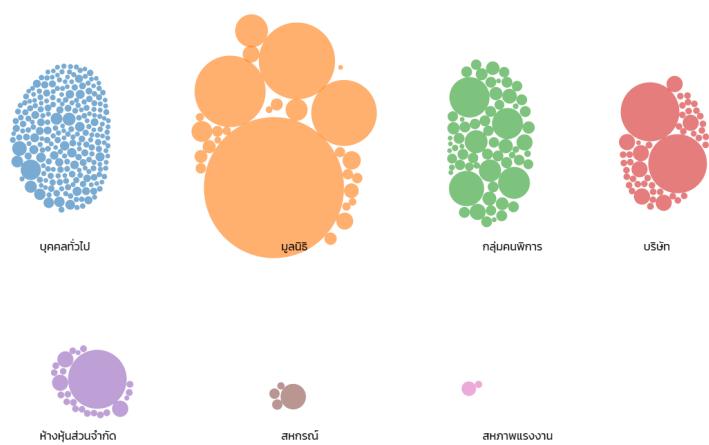
ในยุคปัจจุบันที่ข้อมูลมีจำนวนมหาศาลและถูกบันทึกเอาไว้ในฐานข้อมูลผ่านวิธีการต่างๆ (Big Data) ทำให้การนำเสนอข้อมูลเหล่านี้มาใช้ให้เป็นประโยชน์มีความสำคัญอย่างมาก ดังนั้นการทำ Data Visualization จึงเข้ามามีบทบาทสำคัญในการวิเคราะห์และสามารถนำข้อมูลเหล่านี้มาพัฒนาต่อยอดให้เกิดประโยชน์สูงสุด

Data Visualization คือหนึ่งในรูปแบบการสื่อสารสิ่งที่เราต้องการนำเสนอ เช่น ข้อมูลชนิดต่างๆ เพื่อให้ผู้รับสารเข้าใจง่าย และ เพิ่มความน่าเชื่อถือของข้อมูล การสรุปข้อมูลให้อยู่ในรูปแบบของภาพที่แสดงจากข้อมูลที่เรารอจะยังไม่รู้รูปแบบของข้อมูลนั้น เพื่อให้มนุษย์สามารถทำความเข้าใจและรับรู้ได้โดยง่ายในทันที เช่น กราฟ แผนภูมิ เป็นต้น ซึ่งง่ายและประยุกต์เวลาการวิเคราะห์ มองข้อมูลจากในตาราง ข้อมูล หรือรูปภาพ เป็นต้น นอกจากนี้ผู้ใช้ยังสามารถเห็นรูปแบบ (Trend) ของข้อมูลซึ่งจะทำให้สามารถตัดสินใจได้ง่ายมากขึ้น

ทุกข้อมูลไม่จำเป็นต้องมาทำ Data Visualization ขึ้นอยู่กับ ผู้รับสาร/message/จุดประสงค์ ของเรา บางทีอาจจะใช้แค่รูปภาพง่ายๆ ก็สามารถสื่อสารได้แล้ว

ในมุมของตัวผู้รับสารเอง เมื่อเจอข่าว infographic/Data Visualization ใด ๆ ก็อย่าเพิ่งเชื่อ 100 เปอร์เซ็นต์ทันที ต้องดูแหล่งที่มา และความน่าเชื่อถือของข้อมูลด้วย

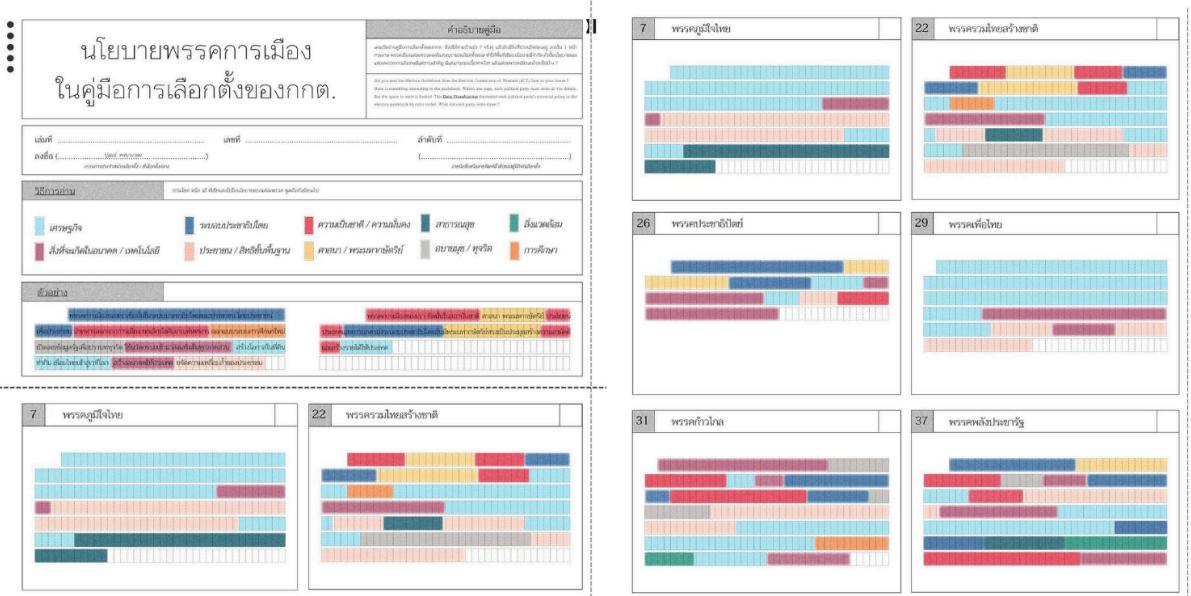
● ตัวอย่างการทำ Data Visualization



- ★ Data Journalism เรื่อง “ hairy ” ได้ร้าย ? [https://kmutt.me/DVeg1] (ช้ายบัน)
- ★ Dangerous Speech Monitoring in Thailand [https://kmutt.me/DVeg2] (ขวบัน)
- ★ Campaign website for Chadchart Sittipunt for Bangkok mayor election (ช้ายล่าง)

[https://kmutt.me/DVeg4]
- ★ Thailand’s national election [https://kmutt.me/DVeg3] (ขวลา่ง)
- ★ นโยบายพรรคการเมืองในคู่มือการเลือกตั้งของ กกต. [https://kmutt.me/DVeg5] (ล่าง ↓)

credit: instagram.com/poontany/



ภายใน 1 หน้า กระดาษ พรรคร่วมเมืองแต่ละพรรคระบุรายละเอียดทั้งหมด ทำให้พื้นที่เขียนนโยบายมีจำกัด ดังนั้นนโยบายของแต่ละพรรคร่วมเมืองจะมีแค่สาระสำคัญ มีแค่แก่นของเนื้อหาจริงๆ แล้วแต่ละพรรคระบุรายละเอียดที่สำคัญที่สุด

Data Visualization ชิ้นนี้นำ Color-coded มาแสดงหัวข้อต่างๆ ที่ถูกเขียนภายใต้นโยบายของพรรคร่วมเมืองแต่ละพรรค สีที่แตกต่างกันแสดงหัวข้อนั้นๆ

Data Visualization What-Why-How loop

○ What?: What data do we have? - data abstraction

การทำ Data Visualization เริ่มต้นจากการที่เรามีข้อมูลอะไร ประกอบไปด้วย

★ Semantics

Semantics หมายถึงความหมายของข้อมูลที่เรามี เช่น 070623 อาจหมายถึง เบอร์โทรศัพท์หรือ Zip Code/Product Code หรืออาจจะเป็นวันที่เกิด แล้วถ้าเกิดมันเป็นวันที่แล้วมันเป็นวันที่ในรูปแบบไหน เช่น เดือน-วัน-ปี/วัน-เดือน-ปี แล้ววันที่นั้นคือวันอะไร เกิดจากการคำนวณหรือการบันทึกตามข้อมูลจริง แล้วข้อมูลนี้ได้มายอย่างไร มีความแม่นยำ/ความถูกต้องแค่ไหน

ดังนั้น Semantics จึงเป็นสิ่งสำคัญในการทำ Data Visualization เพราะเป็นการที่เราจะเข้าใจชุดข้อมูลของเรางเพื่อให้สามารถวิเคราะห์หา insight ได้เพิ่มเติม

★ Dataset Types

- Tables - ประกอบไปด้วย component /item, Attributes

- Networks - ประกอบไปด้วย component *Items*, *Links*, *Attributes*(optional)
- Field - ประกอบไปด้วย component *Attributes*, *Positions*
- Geometry - ประกอบไปด้วย component *Items*, *Attributes*, *Positions*

○ Why?: Do we have to see it? - Task Abstraction

ต่อมาคือ เราต้องการทำ Data Visualization ไปเพื่ออะไร

★ Actions

- Analyze

- Consume - คือการทำ Data Visualization เพื่อค้นหา insight อะไรบางอย่างจากชุดข้อมูล แล้วนำมานำเสนอด้วยการทำให้สวยงามเข้าใจง่ายมากขึ้น
- Produce - คือการทำ Data Visualization เพื่อต้องการต่อยอดจากชุดข้อมูลเดิมเพื่อให้เข้าใจปัญหานั้น ๆ มากยิ่งขึ้น โดยการทำ Annotate ข้อมูลเพิ่มเติม / การบันทึกข้อมูลที่ชุดข้อมูลเดิมไม่ได้บันทึกไว้ / การทำให้เห็นข้อมูลชุดใหม่จากข้อมูลชุดเดิมได้

- Search

	Target Known	Tarket unknown
Location known	Lookup	Browse
Location unknown	Locate	Explore

- Query

- Query คือการทำ Data Visualization เพื่อระบุว่าข้อมูลนั้น ๆ คืออะไร หรือเพื่อเปรียบเทียบข้อมูล หรือเพื่อสรุปตอบปัญหาใดได้

○ How?: Do we make it happen? - Construction

สุดท้ายคือ เราจะทำยังไงให้มันเกิดขึ้น หลังจากที่เรามีข้อมูลอยู่ในมือ รู้ว่าข้อมูลของเราคืออะไร เรา泯์คำตามแล้วในใจว่าเราต้องการจะดูอะไร และผู้ใช้ต้องการจะมองเห็นอะไร เราเก็บมาสร้างกัน การสร้างเป็นขั้นตอนที่มีทางเลือกเยอะที่สุด เราสามารถทำได้หลากหลายมุมมอง ไม่ว่าจะเป็นการ Encode/Manipulate/Faceting/Reduce

★ Encode

การ Encode คือการที่เรานำข้อมูลที่เรามีอยู่มาจัดการทำเป็นกราฟ/bar chart เพื่อเปรียบเทียบหรือเรียงลำดับหรือเพื่อ Map Feature เข้ากับสี ขนาด หรือรูปร่างของจุด

★ Manipulate

ภาษา Python มี Python Package ที่นิยมใช้ในการจัดการข้อมูลอยู่อย่าง Numpy

★ Numpy

NumPy เป็น Python Package พื้นฐานที่ใช้สำหรับการคำนวณทางวิทยาศาสตร์ ซึ่งเป็นสิ่งจำเป็นสำหรับการวิเคราะห์ข้อมูล และการนำเสนอข้อมูลด้วยภาพ สำหรับการจัดการกับข้อมูลที่มีลักษณะเป็นเมตริกซ์และอาร์เรย์ (arrays) ต่าง ๆ n มิติ (ndarray) โดย NumPy จะมีฟังก์ชัน และเครื่องมือต่าง ๆ ที่ช่วยให้ผู้ใช้งานสามารถดำเนินการทางคณิตศาสตร์ หรือสถิติ กับเมตริกซ์และอาร์เรย์ได้อย่างสะดวก และรวดเร็วมากขึ้น เช่น การหาค่าเฉลี่ย การหาค่าสูงสุด การหาค่าต่ำสุด การหาค่า มัธยฐาน การคำนวณผลรวม การคูณ การแยกเมทริกซ์ รวมไปถึงการสุ่มค่าด้วย

Numpy นั้นได้แรงบันดาลใจมาจาก MATLAB ดังนั้นผู้ที่มีประสบการณ์จาก MATLAB อยู่แล้ว จะทำความเข้าใจ Numpy ได้ไม่ยาก โดยหลักการ คือ การนิยามตัวแปร array หลายมิติ ที่เราคุ้นเคยในคณิตศาสตร์ อาทิเช่น เวกเตอร์ (1 มิติ) เมटริกซ์ (2 มิติ) เทนเซอร์ (3 มิติขึ้นไป) และ operations ของมันในการทำความเข้าใจ Numpy นั้น เราควรมีความรู้พื้นฐาน Linear algebra พวก vector/matrix บ้าง

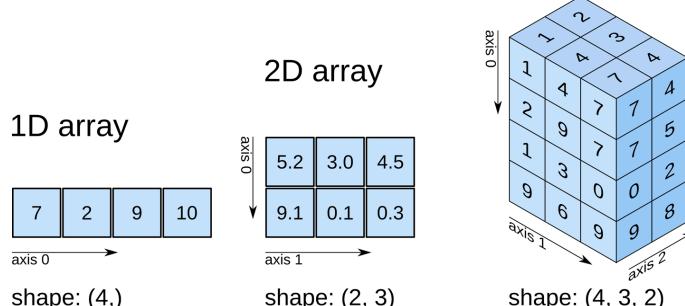
**แม้ในทางคณิตศาสตร์เวกเตอร์ควรจะแทนด้วย array 1 มิติ แต่ด้วย syntax ของ python นั้นการใช้ array 2 มิติแทนเวกเตอร์จะเหมาะสมกว่า

● พื้นฐาน Numpy

วัตถุประสงค์หลักของ Numpy คือเพื่อจัดการกับข้อมูลที่เป็น Array หลายมิติ (Table of Elements) ใน Numpy เรียก Dimensions ว่า axes โดยการใช้ Array มีข้อสังเกตอยู่เล็กน้อยอย่าง

- Array มีลักษณะคล้าย List แต่เปลี่ยนขนาดไม่ได้
- ค่าใน Array ทุกช่องต้องเป็นประเภทเดียวกันทั้งหมด เช่น int/float ทุกช่อง (ผสมกันไม่ได้ ถ้าเป็น List ผสมได้)
- เราสร้างเวกเตอร์ได้ด้วย Array 1 มิติ และสร้างเมทริกซ์ได้ด้วย Array 2 มิติ

3D array



เช่น

```
{7, 2, 9, 10}
```

คือ Array 1 มิติ และมี 4 elements ดังนั้น เราจะกล่าวได้ว่า Array นี้มีความยาว 4 หน่วย

```
[[ 5.2, 3.0, 4.5],  
 [ 9.1, 0.1, 0.3]]
```

ตัวอย่างนี้ คือ Array ที่มี 2 มิติ แกนแรก มีความยาว 2 หน่วย ส่วนแกนที่ 2 มีความยาว 3 หน่วย

● การติดตั้ง Numpy

โดยทั่วไปแล้วเราสามารถใช้คำสั่งด้านล่างนี้ เพื่อบอกให้ Google Colab ช่วยติดตั้ง Numpy ให้

```
!pip install numpy
```

- การ import numpy มาใช้

```
import numpy as np
```

- คำสั่งที่จะทำให้เราเข้าใจ numpy ง่ายขึ้น

```
type() # ใช้สำหรับเรียกดู datatype  
shape() # ใช้สำหรับเรียกดูขนาดมิติของ Array
```

- การสร้าง Array ด้วย np.array

การสร้าง Array ที่นิยมทำกันอยู่มีหลากหลายรูปแบบ ไม่ว่าจะเป็น การสร้างจาก Python List หรือ Tuple หรือในหัวข้อนี้เราจะสร้างด้วยการใช้ np.array() Function ใน Numpy

```
a = np.array([7,2,9,10])  
a # เพื่อ print array ออกมา  
print(a) # เพื่อ print ค่าใน array ออกมา  
print(type(a))  
print(a.shape)
```

```
array([ 7,  2,  9, 10])  
[ 7  2  9 10]  
<class 'numpy.ndarray'>  
(4,)
```

เราสามารถตรวจสอบชนิดของข้อมูลใน Array ได้โดย

```
print(a.dtype)
```

```
int64
```

โดยการกำหนด Datatype สามารถทำได้โดย

```
x = np.array([1,2,3],int)  
x
```

```
array([1, 2, 3])
```

หากเราลองเปลี่ยนค่าใน Array เป็นทศนิยม

```
x[0] = 4.8  
x
```

```
array([4, 2, 3])
```

จะเห็นได้ว่า Numpy จะบัดค่าหลังทศนิยมทิ้งไปเลย

! ระวัง np.array() กับ np.ndarray() ไม่เหมือนกัน

```
a = np.array([4, 5])
a
```

```
array([4, 5])
```

```
a = np.ndarray([2,3])
a
```

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

จะเห็นว่า การใช้ `np.ndarray([a, b])` จะเป็นการสร้าง Array ที่มีมิติ $a \times b$ ซึ่งอาจจะทำให้เราลับสนได้

เรียงสามารถสร้าง Array ได้ด้วยคำสั่งอื่น ๆ อีก ไม่ว่าจะเป็น

★ `np.zeros()`

```
np.zeros((2,3))
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

ใช้สำหรับสร้าง Array ที่ทุกค่าใน Array มีค่าเท่ากับ 0

★ `np.ones()`

```
np.ones((3,2), int)
```

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

ใช้สำหรับสร้าง Array ที่ทุกค่าใน Array มีค่าเท่ากับ 1

★ `np.identity()`

```
np.identity(4)
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

จะเห็นว่า เราจะได้ identity Matrix ออกมา ซึ่งสามารถที่จะกำหนดมีค่าเป็น float แต่หากเราต้องการให้สามารถมีค่าเป็นจำนวนเต็ม สามารถเพิ่ม attribute ไปได้ ดังนี้ `np.identity(4, int)`

★ *np.full()*

```
c = np.full((2,2), 7)
c
```

```
array([[7, 7],
       [7, 7]])
```

ใช้สร้าง array 2 มิติขนาด 2×2 ที่มีสมาชิกเป็นค่าคงที่เท่ากับ 7 ทั้งหมด

★ *np.eye()*

```
d = np.eye(2)
d
```

```
array([[1., 0.],
       [0., 1.]])
```

ใช้สร้าง "เมตริกซ์เอกลักษณ์" ขนาด 2×2 นั่นคือมีสมาชิกในลั้นแทนยงมุ่นเป็น 1 นอกนั้นเป็น 0 ทั้งหมด

★ *np.random.random()*

```
e = np.random.random((2,2))
e
```

```
array([[0.37331921, 0.42245879],
       [0.42507462, 0.63182221]])
```

ใช้สร้าง array 2 มิติขนาด 2×2 ที่มีสมาชิกเป็นค่าสุ่ม (random numbers) ในช่วง (0,1)

★ *np.zeros_like()*

```
x = np.array([[2, 5, 6, 7], [5, 9, 8, 1], [7, 3, 4, 8]])
x

y = np.zeros_like(x, int)
y
```

```
array([[2, 5, 6, 7],
       [5, 9, 8, 1],
       [7, 3, 4, 8]])
```

```
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [0, 0, 0, 0]])
```

ใช้สร้าง Array ที่มีขนาดเท่ากับ Array x และเป็นจำนวนเต็ม 0 ทุกช่อง และในทำนองเดียวกัน หากต้องการให้สมาชิกทุกตัวมีค่าเป็นจำนวนเต็มที่มีค่าเป็น 1 ก็สามารถใช้คำสั่ง `np.ones_like(x, int)`

★ `np.arange()`

ใช้สร้าง Array โดยการกำหนดช่วงโดยมี arguments ดีอ `np.arange(start, stop, step, dtype)`

```
np.arange(3, 7, 2, float)
```

```
array([3., 5.])
```

- Mathematics Operations

★ *Element-wise operation*

การดำเนินการ Array กับ Array (เช่น A+B) การดำเนินการหรือทำ Function กับ Array จะเป็นแบบช่องต่อช่อง (element-wise) เช่น

การบวก ลบ คูณ หารในสมาชิกแต่ละตัวของ array สอง array เฉพาะสมาชิกในตำแหน่งเดียวกัน สามารถทำได้อย่างตรงไปตรงมาในกรณีที่ array ทั้งสองมีขนาดเท่ากัน

```
x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

# [[ 1  2] + [[ 5  6] =  [[ 6  8]
#   [3 4]]      [ 7  8]]      [10 12]]
print(x + y)

# [[-4 -4]
#  [-4 -4]]
print(x - y)

# [[ 5 12]
#  [21 32]]
print(x * y)

# ตัวอย่างการหาร เมื่อเลขไม่ลงตัว สมาชิกใน array จะเปลี่ยนเป็นจำนวนจริงโดยอัตโนมัติ
# กรณีการหารตัวเลข
# [[ 0.2          0.33333333]
#  [ 0.42857143  0.5          ]]
print(x / y)
```

```

[[ 6  8]
 [10 12]]
[[ -4 -4]
 [-4 -4]]
[[ 5 12]
 [21 32]]
[[0.2          0.33333333]
 [0.42857143 0.5        ]]

```

การยกกำลัง และถอดรากในสมາชิกแต่ละตัว ก็ทำได้ตรงไปตรงมา

```

# ยกกำลัง
print(x ** 2)

# ถอดรากที่สองมีสองวิธีคือเรียกฟังก์ชันของ numpy หรือ ยกกำลังด้วย 1/2
print(np.sqrt(x))
print(x ** (1/2))

```

```

[[ 1  4]
 [ 9 16]]
[[1.          1.41421356]
 [1.73205081 2.          ]]
[[1.          1.41421356]
 [1.73205081 2.          ]]

```

การคูณแบบ Matrix/Vector สามารถทำได้โดยใช้ฟังก์ชัน np.dot

ถ้ากำหนดให้ A มีขนาด $n \times m$

และ B มีขนาด $m \times p$

การคูณแบบ Matrix จะให้ผลลัพธ์ดังนี้

$C = AB$ จะมีขนาด $n \times p$ ซึ่งจะใช้คำสั่ง $C = np.dot(A, B)$

$C = BA$ จะไม่สามารถคูณได้กรณี n ไม่เท่ากับ p และเมื่อ $C = np.dot(B, A)$ จะแจ้ง error

```

A = np.random.random((2,3))
B = np.random.random((3,4))
print(A.shape, B.shape)

```

```

C = np.dot(A,B)
print(C.shape)

```

```

(2, 3) (3, 4)
(2, 4)

```

```

C = np.dot(B,A)
print(C.shape)

```

```
ValueError: shapes (3,4) and (2,3) not aligned: 4 (dim 1) != 2 (dim 0)
```

Warning

การอ่านแบบโปรแกรมที่เกี่ยวข้องกับ matrix/vector operations ไม่ควรใช้ array 1 มิติ เนื่องจาก Numpy syntax จะไม่สอดคล้องกับ linear algebra และจะทำให้เราลับสนได้ เช่น

ถ้ากำหนดให้เมทริกซ์ A มีขนาด $m \times m$

และเวกเตอร์ x มีขนาด $m \times 1$

การคูณแบบ Matrix/Vector ควรจะให้ผลลัพธ์ดังนี้

$y = Ax$ จะได้ y เป็นเวกเตอร์ที่มีขนาด $m \times 1$ เช่นเดียวกับตัวย่อ $y = np.dot(A, x)$

$y = xA$ จะไม่สามารถคูณได้ตามหลัก linear algebra และ $y = np.dot(x, A)$ จะ error

อย่างไรก็ตี array 1 มิติ จะไม่เหมือนกับเวกเตอร์ในทางคณิตศาสตร์และจะคูณสำเร็จทั้งสองกรณี (ไม่แจ้ง error ในกรณีที่สอง) ซึ่งทำให้ได้ผลลัพธ์ที่ไม่พึงประสงค์ ทั้งนี้เนื่องมาจากการของเราอ่านแบบสมการ เราจะอ่านแบบสมการทางคณิตศาสตร์โดยเฉพาะ Linear Algebra มาก่อน ดังนั้นผลลัพธ์ที่ไม่ตรงกับ Linear Algebra นี้จะทำให้เราลับสนได้ (อาจเป็น bug ที่เราไม่รู้ว่ามันไม่ถูก) ดังตัวอย่างต่อไปนี้

```
A = np.ones((3,3))
x = np.array([1, 1, 1]) # นี่คือ array 1 มิติที่ควรหลีกเลี่ยง

print(np.dot(A,x))
print(np.dot(A,x).shape, '\n')
print(np.dot(x,A))
print(np.dot(x,A).shape)
# ทั้งสองกรณีให้ผลลัพธ์เท่ากันทั้งที่ควรจะ error ในกรณีที่สอง
```

```
[3. 3. 3.]
(3, )
```

```
[3. 3. 3.]
(3, )
```

วิธีแก้ก็คือ กำหนดให้เวกเตอร์เป็น array 2 มิติ เช่นเดียวกับเมทริกซ์ โดยมีขนาดของมิติที่ 2 เป็น 1 (การกำหนดให้ ขนาดของมิติที่ 2 เป็น 1 คือเวกเตอร์แนวตั้งหรือที่เรียกว่าค่าคงล้มเวกเตอร์เป็นมาตรฐานที่คนส่วนใหญ่ใช้ อย่างไรก็ตี นาน ๆ ครั้งเราจะพบเวกเตอร์แนวอนหือ array 2 มิติที่มีขนาดของมิติแรกเป็น 1 บ้างเหมือนกัน)

การเปลี่ยนมิติของ array ทำได้โดยใช้ method reshape โดยมี argument เป็นขนาดของมิติที่ต้องการ ดังตัวอย่างข้างล่าง ซึ่งจะทำให้ $np.dot(x, A)$ ไม่สามารถทำงานได้และแจ้ง error ตาม linear algebra

```
x = x.reshape(3,1)
print('Now x is 2D array with shape: ', x.shape, '\n')

print('Now np.dot(A,x) is :')
print(np.dot(A,x))
print(np.dot(A,x).shape, '\n')
```

```
Now x is 2D array with shape: (3, 1)
```

```
Now np.dot(A,x) is :
```

```
[ [3.]  
[3.]  
[3.]]  
(3, 1)
```

```
print('And np.dot(x,A) will cause error :')  
print(np.dot(x,A))  
print(np.dot(x,A).shape)  
# error ในกรณีที่สองตามที่ควรจะเป็น
```

```
And np.dot(x,A) will cause error :
```

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

Numpy มีฟังก์ชันในการคำนวนอื่นๆ ที่มีประโยชน์อีกมาก many เช่น `np.sum` ซึ่งเป็นฟังก์ชันที่จะรวมผลบวกของ array ในมิติที่กำหนด โดยเราใช้ argument `axis = d` สำหรับมิติ d ที่ต้องการ โดย index d นั้นเริ่มต้นจาก 0 ดังแสดงในตัวอย่างข้างล่าง

```
A = np.array([[1,2],[3,4]]) # สร้างเมตริกซ์ขนาด 2x2  
print(A)  
  
print(np.sum(A)) # ผลบวกของทุกสมาชิกในเมตริกซ์; พิมพ์ "10"  
print(np.sum(A, axis=0)) # ผลบวกในมิติแรก (axis = 0) หรือผลบวกตามแกนของทุกสมาชิกในเมตริกซ์; พิมพ์ "[4 6]"  
print(np.sum(A, axis=1)) # ผลบวกในมิติที่สอง (axis = 1) หรือผลบวกตามคอลัมน์ของทุกสมาชิกในเมตริกซ์; พิมพ์ "[3 7]"
```

```
[ [1 2]  
[3 4]]  
10  
[4 6]  
[3 7]
```

Operator อีกตัวที่เราใช้บ่อยใน linear algebra คือ `transpose` ซึ่งเป็นการ `reshape` เมตริกซ์ชนิดหนึ่ง เช่นกัน (สลับจากແແວເປັນຄອລັມນີ້ແລະຄອລັມນີ້ເປັນແດວ) ซึ่งเราสามารถเรียกใช้ได้ง่าย ๆ เนื่องจากเป็น array method ที่ชื่อว่า `T` ดังตัวอย่าง

```
A = np.array([[1,2], [3,4]])  
print(A)      # พิมพ์ "[[1 2]  
#                  [3 4]]"  
print(A.T)    # พิมพ์ "[[1 3]  
#                  [2 4]]"
```

```
[ [1 2]
[3 4] ]
[ [1 3]
[2 4] ]
```

★ Broadcasting

การดำเนินการ Array กับ Array ที่มีขนาดไม่เท่ากัน อาจมีการขยาย Array ให้มีขนาดเท่ากันก่อน (broadcasting) โดยพิจารณา มิติ ของทั้งสอง Array จากขวาไปซ้าย ให้เป็นไปตามกฎการ broadcast

การบวก ลบ คูณ หาร "รายตัว" กรณี Array ทั้งสองมีขนาดต่างกัน เรา.mักจะขอขยายสถานการณ์ ที่เราต้องการบวก ลบ คูณ หารสอง array ที่มีขนาดไม่เท่ากัน เช่น

- เรา มีเมตริกซ์ขนาด 3×3 และเวกเตอร์ขนาด 3×1 และเราต้องการจะ บวกทุกๆ คอลัมน์ของเมตริกซ์ ด้วยคอลัมน์เวกเตอร์
- เรา มีเมตริกซ์ขนาด 3×3 และเวกเตอร์แนวอนขนาด 1×3 และเราต้องการจะ บวกทุกๆ แถวของเมตริกซ์ ด้วยแถวเตอร์แนวอน

ชีส Numpy จะทำให้โดยอัตโนมัติ (ยกเว้นกรณีที่จำนวนแคลว์/คอลัมน์ไม่ตรงกันดังแสดงในตัวอย่าง เช่น

```
W = np.array([[1,2,3],[4,5,6],[7,8,9]]) #สมมติเรามีเมตริกซ์ขนาด 3x3
print('We have a 3x3 matrix : ')
print(W, '\n')

x = np.array([1,0,1])
y = x.reshape(3,1) # สร้างเวกเตอร์แนวตั้ง หรือคอลัมน์เวกเตอร์
z = x.reshape(1,3) # สร้างเวกเตอร์แนวอน
print('We also have two vectors with shapes: ',y.shape, z.shape)
print(y)
print(z, '\n')

print('Broadcasting addition with respect to the column vector : ')
print(W+y, '\n') # บวกทุกคอลัมน์ของเมตริกซ์ด้วยเวกเตอร์แนวตั้ง
print('Broadcasting addition with respect to the row vector : ')
print(W+z, '\n') # บวกทุกแคลว์ของเมตริกซ์ด้วยเวกเตอร์แนวอน

# จริงๆ แล้วการคูณเมตริกซ์ด้วยค่าคงที่ ก็คือ Broadcasting ชนิดหนึ่ง
print('Broadcasting constant multiplication example : ')
print(W*2)
```

```
We have a 3x3 matrix :
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
We also have two vectors with shapes: (3, 1) (1, 3)
```

```
[[1]
 [0]
 [1]]
 [[1 0 1]]
```

```
Broadcasting addition with respect to the column vector :
```

```
[[ 2 3 4]
 [ 4 5 6]
 [ 8 9 10]]
```

```
Broadcasting addition with respect to the row vector :
```

```
[[ 2 2 4]
 [ 5 5 7]
 [ 8 8 10]]
```

```
Broadcasting constant multiplication example :
```

```
[[ 2 4 6]
 [ 8 10 12]
 [14 16 18]]
```

อย่างไรก็ตี Broadcasting จะไม่สามารถทำได้ถ้า Numpy ไม่สามารถปรับ dimension หรือมิติของทั้งสอง array ให้ตรงกันได้ เช่น ถ้าเราต้องการบวกเมทริกซ์ขนาด 3x3 กับเวกเตอร์ขนาด 2x1 จะเกิด Error ขึ้น ดังตัวอย่าง

```
W = np.array([[1,2,3],[4,5,6],[7,8,9]]) #สมมติเรามีเมทริกซ์ขนาด 3x3
print('If we have a 3x3 matrix : ')
print(W, '\n')

print('And the following 2x1 vector : ')
x = np.array([1,10])
x = x.reshape(2,1) # สร้างเวกเตอร์แนวตั้ง 2x1
print(x, '\n')
```

```
If we have a 3x3 matrix :
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

```
And the following 2x1 vector :
```

```
[[ 1]
 [10]]
```

```

print('We are sorry that we cannot add them as broadcasting mismatched
arrays is not possible : ')
print(W+x) # error

```

We are sorry that we cannot add them as broadcasting mismatched arrays is not possible :

ValueError: operands could not be broadcast together with shapes (3, 3) (2, 1)

★ Indexing

การอ้างอิงข้อมูลใน Array (NumPy Indexing)

- สำหรับ Array 1 มิติ
 - การเข้าใช้ข้อมูล

V[k]

เหมือนการใช้ลิสต์ คือเลือกข้อมูลหนึ่งช่อง โดยที่ k เป็นจำนวนเต็ม

V[a:b:c]

เหมือนการใช้ลิสต์ คือเลือกข้อมูลเป็นช่วง ได้ผลเป็นArray

- การใส่ค่าใหม่

V[k] = 99

เหมือนกับการใช้ลิสต์

V[a:b:c] = d

แตกต่างจากลิสต์

- ถ้า d เป็น int, float หรือ list/Array ขนาด 1 ช่อง จะ broadcast ให้มีขนาดเท่ากับช่วงของ a:b:c
- ถ้า d เป็น list/Array ขนาดมากกว่า 1 ช่อง ต้องมีขนาดเท่ากับขนาดของช่วง a:b:c เช่น

- V = np.zeros(5,int)
- V[2:4] = 1 ทำได้, V เปลี่ยนเป็น [0 0 1 1 0]
- V[::2] = 3 ทำได้, V เปลี่ยนเป็น [3 0 3 1 3]
- V[::2] = [9] ทำได้, V เปลี่ยนเป็น [9 0 9 1 9]
- V[::2] = [8,8,8] ทำได้, V เปลี่ยนเป็น [8 0 8 1 8]
- V[1:5] = [1,2] ทำไม่ได้, เพราะขนาดของข้อมูลไม่ตรงกัน
- V[7:7] = [1,2] ไม่เกิดการเปลี่ยนแปลงใด ๆ

- สำหรับ Array 2 มิติ

เราสามารถอ้างอิงถึงตัวข้อมูล, Array 1 มิติที่แทนแถว (row) ของข้อมูล, Array 1 มิติที่แทนหลัก (column) ของข้อมูล หรือ Array ย่อ 2 มิติที่แทนช่วงของแถวและหลัก ได้หลากหลายแบบ ดังนี้

- M[r,c] ข้อมูล ณ แผลที่ r หลักที่ c
- M[r,:] หรือ M[r] Array 1 มิติของแผลที่ r ทั้งหมด
- M[:,c] Array 1 มิติของหลักที่ c ทั้งหลัก (ไม่ใช่Array 2 มิติที่มี 1 หลัก)

- $M[r1:r2,:]$ หรือ $M[r1:r2]$ Array 2 มิติประกอบด้วย แถวที่ $r1$ ถึง $r2-1$
- $M[:,c1:c2]$ Array 2 มิติประกอบด้วย หลักที่ $c1$ ถึง $c2-1$
- $M[r1:r2,c1:c2]$ Array ย่ออย 2 มิติ ในช่วงแถวที่ $r1$ ถึง $r2-1$ และช่วงหลักที่ $c1$ ถึง $c2-1$
- $M[r1:r2:a,c1:c2:b]$ Arrayย่ออย 2 มิติ ในช่วงแถวและหลักที่กำหนดโดย $r1:r2:a$, $c1:c2:b$

เช่น กำหนดให้ $M = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])$

- $M[1:3]$ ได้ $np.array([[5,6,7,8],[9,10,11,12]])$
- $M[1:2, 1:3]$ ได้ $np.array([[6,7]])$
- $M[:, 2]$ ได้ $np.array([3,7,11])$
- $M[1:, :2]$ ได้ $np.array([[5,6],[9,10]])$
- $M[0:3:2]$ ได้ $np.array([[1,2,3,4],[9,10,11,12]])$
- $M[0:3:2, 1:4:2]$ ได้ $np.array([[2,4],[10,12]])$
- $M[::-1]$ ได้ $np.array([[9,10,11,12],[5,6,7,8],[1,2,3,4]])$

และเรารสามารถใช้ค่ากับหลาย ๆ ช่องในArrayพร้อมกันได้ เช่น

- $M[2:4, 1:4] = [[-1, 1, -1], [-1, 1, -1]]$ ขนาดเท่ากับพอดี
- $M[2:4, 1:4] = [-1, 1, -1]$ ขนาดไม่เท่า ระบบ broadcast ให้ M ต้องมี 4 แถว
- $M[:, 0] = [1, 2, 3, 4]$ broadcast 0
- $M[:, -1] = 0$

- Boolean Indexing

เรารสามารถ index ค่าใน array ด้วยตัวแปร Boolean ได้

```
a = np.array([[1,2], [3, 4], [5, 6]])
bool_idx = (a > 2) # return array ขนาดเดียวกับ a
# ค่า bool_idx[i,j] มีค่า เป็น True ถ้า a[i,j] > 2
# นอกนั้นจะมีค่าเป็น False

print(bool_idx.shape)
print(bool_idx)      # พิมพ์      "[[False False]
#                      [ True  True]
#                      [ True  True]]"

# เราใช้ boolean index นี้สร้าง array 1 มิติ จาก a ที่จะ return เฉพาะ index ที่
bool_idx = True
print(a[bool_idx])  # พิมพ์ "[3 4 5 6]"

# เราสามารถลัดขั้นตอนทั้งหมดเหลือเพียงบรรทัดเดียว
print(a[a > 2])    # พิมพ์ "[3 4 5 6]"
```

```
(3, 2)
[[False False]
 [ True  True]
 [ True  True]]
[3 4 5 6]
[3 4 5 6]
```

นอกจากนี้เรายังมี Pandas Python Package ที่สามารถใช้จัดการกับข้อมูลจำนวนมาก ๆ ได้

★ Pandas

Pandas ถือเป็นเครื่องมือหลักในการทำ Data Wrangling บน Python และสามารถนำไปใช้ประโยชน์คู่กับ Package อื่น เช่น เอาไปเตรียมข้อมูลก่อนทำ Data Visualization โดยวิธีการใช้งาน Pandas คือ โหลดไฟล์ข้อมูล เช่น CSV เข้าไป แล้วเราจะได้ข้อมูลในรูปแบบตาราง (DataFrame) ที่แบ่งข้อมูลตามแถวและคอลัมน์ หรือเหมือน Excel ที่เราใช้กันนั่นเอง ส่วนเหตุผลที่ว่าทำไมเราถึงไม่ใช้นั่นก็ เป็นเพราะว่าใน Excel เมื่อคุณมีข้อมูลที่มากกว่า 10,000 แถว โปรแกรมมันจะเริ่มช้าลง และ Excel จะ จำกัดจำนวนแถวของข้อมูลใน 1 spreadsheet โดยมากที่สุดคือ 1,048,576 แต่ เนื่องจากข้อจำกัดของหน่วยความจำ

**pl. Pandas ไม่เกี่ยวกับหนึ่งแพนต้านะครับ จริง ๆ แล้วมาจากคำว่า PANel DAta ซึ่งหมายถึงข้อมูลที่มีหลายมิตินั่นเอง

● การติดตั้ง Pandas

โดยทั่วไปแล้วเราสามารถใช้คำสั่งด้านล่างนี้ เพื่อบอกให้ Google Colab ช่วยติดตั้ง Pandas ให้

```
!pip install pandas
```

● การ import pandas มาใช้

```
import pandas as pd
```

● วิธีเช็ค Version Pandas

คำสั่งนี้เหมือนไม่สำคัญ แต่จริง ๆ แล้วสำคัญมากเวลาเราอ่าน Documentation เพราะถ้าเกิดมีอะไรพังเราจะเทียบได้ว่า Pandas ของเรารอเวอร์ชันตามใน Documentation หรือไม่

```
print ("Pandas version",pandas.__version__)
```

Pandas version 1.5.3

● โหลดไฟล์ CSV (Import)

จุดเริ่มต้นของการทำ Data Exploration & Analysis ใน Pandas ก็คือการโหลดไฟล์ข้อมูลแบบ CSV มาใช้งานนั่นเอง

ก่อนอื่น เราจะโหลดเอา dataset เข้ามาใน colab ก่อน

```
!wget https://github.com/opalchonlapat/sample-data/raw/master/sample_big_dataset.zip
!unzip sample_big_dataset.zip
```

โดยครั้งนี้เรารอเยี่ม dataset มาลองเล่นกันจาก [@opalchonlapat](#) และเรารสามารถใช้คำสั่ง `.head()` หรือ `.tail()` เพื่อดูข้อมูลแถวนอนสุด หรือแถวล่างสุดได้

```
# Read DF
df = pd.read_csv('sample_big_dataset.csv')
# Sometimes reading CSV for Excel need encoding
df = pd.read_csv('sample_big_dataset.csv',encoding = "ISO-8859-1")
# Print head and tail
df.head()
df.tail()
```

```
0   1   2   3   4   5   6   7   8   9   ...   90  91  92  93  94  95  96  97  98  99
1999995 foo bar bar bar bar foo baz bar baz foo ... bar baz bar foo bar foo baz baz foo baz
1999996 bar bar baz foo foo bar baz foo bar bar ... baz foo foo foo baz bar foo bar baz foo
1999997 baz foo bar foo bar bar foo foo bar baz ... baz bar foo foo bar foo bar foo baz bar
1999998 baz baz foo baz baz foo foo bar foo ... bar bar baz bar bar foo bar baz bar baz
1999999 bar baz baz bar bar baz baz bar foo foo ... foo foo foo bar bar bar bar baz foo bar
5 rows * 100 columns
```

- วิธีเช็คจำนวนแ雷้า และจำนวนคอลัมน์

ใน Pandas มีฟังชั่นสำหรับนับจำนวนแ雷้า และจำนวนคอลัมน์แบบง่าย ๆ โดยใช้

```
df.shape
```

```
(2000000, 100)
```

- วิธีสุ่มข้อมูลสำหรับเช็ค (Sample)

ปกติเราเช็คข้อมูลว่าถูกต้องมั้ยด้วย `head` กับ `tail` ซึ่งเป็นการเช็คจากด้านบนหรือด้านล่าง อีกวิธีที่น่าสนใจ คือ เช็คแบบสุ่มข้อมูลขึ้นมาหนึ่งสอง ทำได้ง่าย ๆ โดยใช้

```
df.sample()
```

```
0   1   2   3   4   5   6   7   8   9   ...   90  91  92  93  94  95  96  97  98  99
17137 bar baz bar bar baz baz bar foo foo baz ... baz foo baz bar foo bar foo bar foo baz
1 rows * 100 columns
```

- วิธีเช็คข้อมูลหาความผิดปกติใน DataFrame เป็นต้น

หลังจากโหลดข้อมูลมาแล้ว เราอยากรู้ว่าข้อมูลมีกี่แ雷า, Missing value เท่าไหร่, แต่ละคอลัมน์เป็น Data Type อะไรบ้าง ก็รันคำสั่งนี้ได้เลย

```
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2000000 entries, 0 to 1999999
Data columns (total 100 columns):
 #   Column   Dtype  
----  --  
 0   0        object 
 1   1        object 
 2   2        object 
 3   3        object 
 ...
 96  96       object 
 97  97       object 
 98  98       object 
 99  99       object 
dtypes: object(100)
memory usage: 1.5+ GB

```

`df.info()` จะแสดงรูปข้อมูลมาให้

นอกจากนั้นยังมีคำสั่ง `df.dtypes` (ไม่มีวงเล็บ) สำหรับดู Data Type แต่ละ colum อย่างเดียว

- วิธีแปลงประเภทข้อมูล (Data Type) ใน Data Frame

บางครั้งประเภทข้อมูลของ colum เป็น String แต่เราต้องการ Integer หรือเราต้องการ Date เราสามารถแปลงข้อมูลได้ง่าย ๆ ดังนี้เลย

```

df['hour'] = pd.to_numeric(df['hour']) # แปลงเป็น Numeric
df['hour'] = df['hour'].astype('int') # สามารถใช้วิธีนี้แปลงเป็น float ได้เช่นกัน

```

- วิธีเช็ค Summary ของแต่ละ colum (count, min, max, mean)

ถ้าเรารอ已久 Distribution คร่าว ๆ ของแต่ละ colum ว่าเป็นอย่างไร สามารถใช้คำสั่ง `describe()` ได้

```
df.describe()
```

- วิธีเช็ค Summary (count, min, max, mean) แบบแยกกลุ่ม

บางครั้งเราไม่ได้ต้องการรูป Summary ของทั้ง colum แต่อยากให้แยกตามแต่ละค่าใน colum นั้น ๆ ซึ่งมีประโยชน์มากเวลาเราทำ Data Analysis และอย่างรู้ว่าบางกลุ่มมีอะไรผิดปกติหรือเปล่า

```

test = df.groupby(['Gender'])
test.describe()

```

- วิธีสร้าง DataFrame ใหม่

วิธีสร้างแบบง่ายที่สุด ถ้าต้องการข้อมูลหลายรูปแบบ เราสามารถใช้ Dictionary แบบนี้เลย

```

dataframe = pandas.DataFrame({
    'C1': pandas.date_range('20170101', periods=4),
    'C2' : [10, 20, 30, 40],
    'C3': pandas.Categorical(['A', 'B', 'C', 'D']),
    'C4': 1})

```

แต่ถ้าเราต้องการแค่เป็นแบบตัวเลขทั่วไป ใช้ Numpy แบบนี้ได้เลย

```
array = numpy.array([(1,2,3), (4,5,6),(7,8,9)])
dataframe = pandas.DataFrame(array,columns=['C1','C2','C3'])
```

- วิธีเลือกหลายคอลัมน์จาก DataFrame

ปกติถ้าเราต้องการเลือกแค่ 1 Column ก็เขียนแบบนี้ได้เลย

```
df['C1']
```

แต่ถ้าต้องการเลือกหลายคอลัมน์ ให้ทำแบบนี้

```
df[['C1','C2']]
```

- วิธีเลือกคอลัมน์ตามเงื่อนไขที่ต้องการ

บางทีเรายาก Filter เฉพาะคอลัมน์ที่มีค่าตามที่เราต้องการโดยใช้ .loc ได้ โดยสามารถเลือก Filter แบบ .all() (ทุกค่าในคอลัมน์ต้องตรงตามเงื่อนไข) หรือ .any() (บางค่าในคอลัมน์ต้องตรงตามเงื่อนไข)

```
dataframe2 = dataframe.loc[:,(dataframe>50).any()]
dataframe3 = dataframe.loc[:,(dataframe>50).all()]
```

เราสามารถใช้หาคอลัมน์ที่มี Missing Values หรือหาคอลัมน์ที่ไม่มี Missing Values เลยก็ได้

```
dataframe2 = dataframe.loc[:,dataframe.isnull().any()]
dataframe3 = dataframe.loc[:,dataframe.notnull().all()]
```

- วิธีเลือกเฉพาะตามเงื่อนไขที่ต้องการ

```
dataframe[dataframe['C1']>50] # เงื่อนไขแบบง่าย ๆ
dataframe2 = dataframe.loc[dataframe.C1.isin([1,2,3])] # เงื่อนไขแบบซับข้อน
ถ้ามีหลายเงื่อนไขเราสามารถใช้ & (and) หรือ | (or) ได้
```

```
dataframe[(dataframe['C1']>50) & ((dataframe['C2']<25) | (dataframe['C2']>75))]
หรือใช้ Query เป็นเงื่อนไขได้ด้วย มีประโยชน์มากเวลาเรามีเงื่อนไขเปลก ๆ ไม่ต้องเขียนลูปขึ้นมาเอง
```

```
dataframe2 = dataframe.query('C1 > C2')
```

- วิธีเพิ่มคอลัมน์ใหม่

สามารถเพิ่มคอลัมน์ใหม่ได้ 2 แบบ คือ

- เพิ่มโดยอิงจากคอลัมน์เดิม (เช่น เอาคอลัมน์เดิม + 10 หรือ เอาคอลัมน์ A - คอลัมน์ B มีประโยชน์มากตอนทำ Feature Engineering)
- เพิ่มคอลัมน์โดยตั้งค่า Fix ไปเลยสำหรับทุกແຕา ส่วนใหญ่จะใช้วิธีนี้เวลาเราอยากรีดค่าอะไร แปลก ๆ ที่ต้องเขียนลูปเพื่อใส่ค่า ก็สร้างคอลัมน์แบบ Fix ค่าก่อน และต่อด้วยลูป

```
df['new'] = dataframe['old'] + 10 # use old values
df['new2'] = 5 # apply the same value
```

- การสลับ Row <-> Column (Transpose)

ถ้าเราต้องการ Transpose (ารมณ์เหมือน Vector) เราสามารถใช้คำสั่งนี้ได้เลย

dataframe.T

- การต่อ DataFrame

การต่อ Data Frame คือการเอา Data Set 2 ซึ่ดมาต่อกันในแนวตั้งหรือแนวนอน สำหรับการต่อแบบปั๊บๆไปเลย

มี 2 คำสั่งที่เหมือนกัน คือ concat กับ append แต่ให้ใช้ concat ไปเลย เพราะ append เป็นคำสั่งที่ไม่ Memory Efficient

```
pd.concat([df1,df2], axis=1) # รวมกัน 2 คอลัมน์ (axis = 0 คือแถว, axis = 1 คือคอลัมน์)
pd.concat([df1,df2,df3]) # รวมมากกว่า 2 คอลัมน์ก็ได้
pd.concat(..., ignore_index=True) # รวมเสร็จแล้ว reset index ให้ด้วย ควรใช้ ไม่
# จำเป็นจะต้อง reset ID ข้างหน้าตอนรวมร่วง
pd.concat(..., join='inner') # รวมร่างเฉพาะคอลัมน์ที่ df1 กับ df2 มีทั้งคู่
pd.concat(..., keys=['source1', 'source2']) # เพิ่มคอลัมน์เข้าไปด้วยเพื่อระบุว่า
# Row แต่ละอันมาจาก Data Frame อันไหน
pd.concat(..., join_axes=[df2.index]) # เลือกร่วงเฉพาะ row index ที่เรา
# กำหนดได้
```

- การต่อ DataFrame แบบ Join

ถ้าต้องการต่อ DataFrame แบบ Advance หน่อย เราอาจจะ Join DataFrame ได้เหมือน Join Table ได้เช่น SQL มาถ้าอนุญาตจะสนับ烈

```
pd.merge(df1, df2, left_on="col1", right_on="col2", how="inner")
```

เราสามารถเปลี่ยนตรง how="inner" เป็น "outer", "left", "right" เพื่อเปลี่ยนเป็น Outer Join, Left Join, Right Join ได้อีกด้วย

- การหาค่า Mean, Sum, Max (Aggregate) แบบทั้ง DataFrame

Pandas สามารถสั่ง Aggregate เพื่อหาค่า Mean, Sum, และ Max ได้เลย หมายความว่าเราต้องการรวมข้อมูลก่อนเอาไป Visualize หรือต้องการทำ Feature Engineering ก็ได้

```
newdf = df.agg(['sum', 'max', 'mean'])
```

- การ Aggregate แบบตามกลุ่มที่ต้องการ

บางทีเรายาก Aggregate ข้อมูลตามการจัดกลุ่มในคอลัมน์อื่น เช่น เราอยากรายงานรายเดือนของแต่ละคน (ต้อง aggregate sum ของคอลัมน์รายจ่าย โดยแบ่งกลุ่มตามคอลัมน์ User ID) ใช้แบบนี้

```
aggregate = dataframe.groupby('C1').sum()
```

- การรัน Function เดียวกันทุกแถว หรือทุกคอลัมน์

เวลาเรายากจะรันคำสั่งอะไรมากอย่างสำหรับทุกแถว หรือทุกคอลัมน์ เราสามารถเขียนได้แบบนี้

```
# sum for columns
sum_columns = dataframe[['C1', 'C2']].apply(sum, axis=0)
# sum for rows
```

```
sum_rows = dataframe[['C1', 'C2']].apply(sum, axis=1)
```

เหมือนกับพิงก์ชั่น `apply()` ใน R นั้นเอง

- รันคำสั่งที่เขียนเองกับทุกแطرใน 1 คอลัมน์

ถ้าต้องการรันคำสั่ง (Function) ที่เขียนเอง สำหรับทุกแطرในคอลัมน์อันใดอันหนึ่ง ใช้แบบนี้ได้

```
dataframe['C1'] = dataframe['C1'].map(lambda x: x-100)
```

- รันคำสั่งที่เขียนเองกับทุกค่า

ถ้าต้องการรันคำสั่งที่เขียนเองกับทุกค่าใน DataFrame ใช้โค้ดนี้

```
function_result = dataframe.applymap(lambda x: x*10)
```

หรือใช้ `transform` ก็ได้

```
new_dataframe = dataframe.transform(lambda x: x*100)
```

- คำนวณ Correlation & Covariance

เวลาเราอยากรู้ว่าค่าต่าง ๆ ใน Data Set เรา Correlate กันมั้ย

```
dataframe.corr() # Correlation  
dataframe.cov() # Covariance
```

แต่ค่าที่ออกมากเป็นตัวเลขอาจจะดูยากนิดนึง เราสามารถพลอตสวย ๆ ด้วย Seaborn ได้ สามารถใช้โค้ดด้านล่างนี้ได้เลย

```
import seaborn as sns  
corr = modeldf.corr()  
# Set up the matplotlib figure  
f, ax = plt.subplots(figsize=(15, 8))  
# Generate a custom diverging colormap  
cmap = sns.diverging_palette(10, 10, as_cmap=True)  
# Draw the heatmap with the mask and correct aspect ratio  
sns.heatmap(corr, annot=True)
```

- คำนวณ Cross Tabulation

Cross Tabulation มีประโยชน์มากเวลาเราอยากรู้ว่ามี Data ที่ตรงกับรูป A ของคอลัมน์ 1 และรูป B ของคอลัมน์ 2 เท่าไหร่ เช่น มีนักเรียนผู้ชาย (คอลัมน์ gender) กี่คนในมัธยมปลาย (คอลัมน์ education) แบบนี้เป็นต้น หรือถ้าใครใช้ PivotTable ใน Excel มา ก่อน ก็เหมือนกัน

```
aggregate = pandas.crosstab(dataframe.C1, dataframe.C2)
```

- วิธีหาค่า Unique ในแต่ละคอลัมน์

คำสั่งนี้มีประโยชน์มาก เอาไว้ใช้เช็คว่าแต่ละคอลัมน์มีค่าเปลก ๆ มั้ย

ตัวอย่างการใช้งานก็คือ เราอยากรู้ว่า มีบ้านไหนที่มีจำนวนห้องนอนเปลก ๆ มั้ย (เช่น 50 ห้องนอน หรือ -5 ห้องนอน) ก็หาค่า unique จากคอลัมน์ "bedrooms"

```
dataframe['C1'].unique()
```

- วิธีเช็คว่ามีเดาไว้หนึ่งข้อมูลซ้ำๆ (Duplicated)

อันนี้มีประโยชน์มาก เอาไว้ใช้เช็คว่ามีข้อมูล重複 ๆ อยู่ เช่น ทุก colum นั้นซ้ำกันหมด (อันนี้มีโอกาสว่าเป็นข้อมูลซ้ำ อาจจะต้องลบออก) หรือซ้ำกันบาง colum นั้น (อันนี้ต้องเช็คอีกทีว่าคืออะไร)

```
dataframe.duplicated() # หาอันที่เหมือนกันทุก colum  
dataframe.duplicated('C1') # หาอันที่ซ้ำกันเฉพาะ colum C1  
dataframe.duplicated(['C1', 'C2']) # หาอันที่ซ้ำกันเฉพาะ colum C1 และ C2
```

ปกติแล้วถ้ามีอิฐเมหะ คำสั่งนี้จะไม่แสดงให้เห็นแรกในกลุ่มที่ซ้ำ (เช่น ถ้า C1=5 มี 2 แต่ มันจะแสดงเฉพาะเดาที่ 2) เราสามารถใส่ Argument `keep=False` เข้าไปเพื่อบังคับให้มันแสดงทุกเดาได้

นอกจากนั้นเรายังสามารถนับจำนวนเดาที่ Duplicate และลบทิ้งได้ด้วย

- วิธีการนับจำนวน Duplicate

```
len(df[ df.duplicated(['A', 'B', 'C'], keep = False) ])
```

- วิธีการลบ Duplicate

เอาไว้เช็ตตอนเราเจอว่าทุก colum นั้นซ้ำกันหมดเลย ซึ่งเป็นเคลสที่บอกรว่าข้อมูลน่าจะซ้ำ ลบออกได้ (ขึ้นอยู่กับข้อมูลด้วยนะ บางข้อมูลอาจจะไม่ได้แปลว่าซ้ำแล้วลบได้):

```
# Remove the duplicates  
df.drop_duplicates(['A', 'B', 'C'], inplace=True)  
# Reset dataframe index after drop_duplicates.  
df.reset_index(drop=True, inplace=True)  
len(df)
```

สำหรับโค้ดข้างบน จะเห็นว่าเราต้อง reset index หลังลบ duplicate

- วิธีการลบเดา และลบ colum นึง

ลบ colum นึงสามารถทำได้แบบนี้

```
dataframe = dataframe.drop('C1', axis=1)  
df.drop(['C1'], axis=1, inplace=True) # แบบนี้ก็ได้  
df.drop(['C1', 'C2', 'C3'], 1, inplace=True) # ลบทีละหลาย colum ก็ได้
```

ส่วนการลบเดาจะลำบากหน่อย เพราะต้องใส่ Row Index (เลขที่อยู่ซ้ายสุดเวลาเราปรินท์ DataFrame)

```
dataframe = dataframe.drop(5, axis=0)  
dataframe.reset_index(drop=True) # Reset index
```

ลบเดาแล้วอย่าลืมเช็คด้วยว่าที่ลบไปถูกต้องมั้ย และหลังจากลบเดาต้อง Reset Index ด้วย

- วิธีการลบเดาที่มี Missing Value

ข้อควรระวัง: การที่อยู่ ๆ เราลบเดาที่มี Missing Value ที่ไปเลยอาจจะไม่ใช่วิธีที่ดีที่สุดในการทำ Data Analysis เสมอไป บางเคสการ `Impute` (คำนวนหาค่าไปใส่) จะดีกว่า

```
dataframe2 = dataframe.dropna(axis=0)
```

- วิธีแทนค่า Missing Value ด้วยค่าเฉลี่ย (Mean Imputation)

วิธีหนึ่งในการแทนค่าที่หายไป คือการทำสิ่งที่เรียกว่า Mean Imputation หรือหาค่าเฉลี่ยของคอลัมน์นั้น แล้วนำมาแทนค่าที่หายไปนั้นเอง

ข้อดีของการทำ Mean Imputation คือ สามารถทำได้ง่าย แต่ก็ต้องระวังเรื่องข้อเสีย เช่น ทำแบบนี้จะเป็นการไม่สนใจความสัมพันธ์ระหว่างตัวแปร ทำให้เกิด Bias สูง ควรใช้เฉพาะเวลา Missing Value ไม่เยอะเท่านั้น

สามารถรันโค้ดด้านล่างเพื่อทำ Mean Imputation ได้ง่าย ๆ เลย

```
import numpy as np  
meanAge = np.mean(df.Age) # Get mean value  
df.Age = df.Age.fillna(meanAge) # Fill missing values with mean
```

- การลูปข้อมูลแต่ละคอลัมน์ และแต่ละแถว

การลูปมีประโยชน์มากถ้าเราต้องการเขียนฟังก์ชันแปลก ๆ ใช้เองที่ Pandas ไม่รองรับ (หรืออาจจะรองรับแต่เราหาไม่เจอ เช่น弄ง่ายกว่า) สามารถลูปได้ทั้งแต่ละคอลัมน์ และแต่ละแถว

```
for col_idx,data in dataframe.iteritems():  
    print ("column:",col_idx)  
    print ("column data:")  
    print (data,"\n")
```

- การลูปข้อมูลแต่ละแถว

```
for col_idx,data in dataframe.iterrows():  
    print ("row:",col_idx)  
    print ("row data:")  
    print (data,"\n")
```

- วิธีเปลี่ยน DataFrame จากแบบ Wide เป็น Long (Melt)

การ Melt Data มีประโยชน์มากเวลาเราต้องการเอาข้อมูลไปพลอต Data Visualization หรือเราต้องการ Aggregate

```
dataframe2 = dataframe.melt()
```

- วิธีการเปลี่ยนชื่อคอลัมน์ (Rename)

บางที่เราต้องการเปลี่ยนชื่อเพื่อให้สั้นลง ให้พิมพ์ slash ขึ้น สามารถทำได้ดังนี้

```
dataframe.rename(columns={'old':'new'},inplace=True)
```

- วิธีการใส่คำนำหน้าคอลัมน์ (Prefix)

อันนี้มีประโยชน์มากตอนเรามีข้อมูลหลาย ๆ ชุด และต้องการ Merge โดยอยากให้ชื่อคอลัมน์ไม่ซ้ำกัน

```
thisdata = thisdata.add_prefix('data_')
```

- วิธีการแทนค่าใน DataFrame

เหมามากเวลาต้องการแก้ Typo Error เช่น เราอยากรีดค่า Bangkok แต่เราเขียนเป็น BKK อะไรแบบนี้ (รันคำสั่ง .unique เพื่อดูก่อน)

สามารถ Replace ทั้ง DataFrame ได้โดยแบบนี้

```
dataframe2 = dataframe.replace(1, -100)
```

สามารถ Replace หลายค่าพร้อมกันได้ด้วย และสามารถกำหนด Column ที่ต้องการให้แทนค่าได้ด้วย

```
df['city'].replace({  
    'BKK': 'Bangkok',  
    'BNK': 'Bangkok'  
}, inplace=True)
```

- วิธีการ Export DataFrame เป็นไฟล์ CSV

หลังจากที่เรายัดการ Data เรียบร้อยแล้ว ก็สามารถ Export เป็น CSV เอาไปใช้ต่อกับโปรแกรมอื่น หรืองานส่วนอื่น ๆ ได้

```
dataframe.to_csv('dataframe.csv')
```

★ Facet

คือการวางแผนไว้เพื่อเทียบเคียงกัน หรือการแบ่งกราฟออกให้เข้าใจง่ายขึ้น หรือการนำกราฟมาซ้อนกันเพื่อให้เห็นความลับพันธ์ของกราฟ

★ Reduce

สามารถ Filter เพื่อให้การนำเสนอข้อมูลของเราทำได้ง่ายขึ้น เช่น การรวมช่วงอายุที่ใกล้เคียงกัน เป็นต้น

★ Package

เห็นถึงความสำคัญของ Data Visualization และใช้มันล่ะ เราลองทำ Data Visualization ด้วยตัวเองกันดีกว่า ซึ่งในการทำ Data Visualization มี Python Package อยู่หลายตัวที่นิยมใช้กันอยู่ เช่น Matplotlib, Seaborn, Plotly, Bokeh, Geoplotlib

แต่ในหัวข้อนี้เราจะยกตัวอย่าง Matplotlib Python Package ให้เข้าใจง่าย ๆ กัน

★ Matplotlib

Matplotlib เป็น Python Package สำหรับใช้ในการสร้างกราฟซึ่งมีความยืดหยุ่นสูงสามารถทำกราฟได้หลายรูปแบบ มีความสามารถในการปรับแต่งรายละเอียดกราฟได้หลากหลาย ทำให้ใช้สามารถปรับแต่งกราฟให้เหมาะสมกับการนำเสนอข้อมูลได้อย่างเหมาะสม โดยผู้สร้าง Package นี้ขึ้นมาพยายามสร้างให้เหมือนกับการทำ Data Visualization ของ MATLAB ให้สามารถทำงานได้บน Python

นอกจากนี้ Matplotlib ยังสามารถใช้งานร่วมกับไลบรารี Python อื่น ๆ เช่น NumPy, Pandas และ SciPy เพื่อการวิเคราะห์ข้อมูลที่มีความซับซ้อนขึ้นได้

- การติดตั้ง Matplotlib

โดยทั่วไปแล้วเราสามารถใช้คำสั่งด้านล่างนี้ เพื่อบอกให้ Google Colab ช่วยติดตั้ง Matplotlib ให้

```
!pip install matplotlib
```

- การ import matplotlib มาใช้

```
import matplotlib.pyplot as plt
```

โดยส่วนใหญ่เราจะนิยมใช้ตัวย่อเป็น plt แทน pyplot ซึ่งคำสั่งเกือบทั้งหมดของ Matplotlib ในหัวข้อนี้ใช้เพียงตัว pyplot เท่านั้น กราฟที่วาดได้ใน Matplotlib นั้นมีหลายชนิดด้วยกัน เช่น กราฟเส้นธรรมดា, แผนภูมิแท่ง, แผนภูมิวงกลมและอื่น ๆ และในบทความนี้เราจะพูดถึงกราฟเส้นธรรมดากันก่อน ซึ่งฟังก์ชันที่ใช้วาดกราฟเส้นธรรมดาก็คือฟังก์ชันที่ชื่อว่า plot ในส่วนของ Argument ที่ต้องใส่ในฟังก์ชันนี้คือ พิกัดในแนวแกน x และ y ตามลำดับ โดยข้อมูลที่ใส่ต้องเป็น Object ชนิดลำดับ เช่น List และสามารถใช้ NumPy Array ได้ด้วย

การวาดกราฟด้วย Matplotlib สามารถปรับแต่งแก้ไขอะไรได้มากมายตามที่ต้องการ เช่น สีเส้น, รูปแบบเส้นและอื่น ๆ

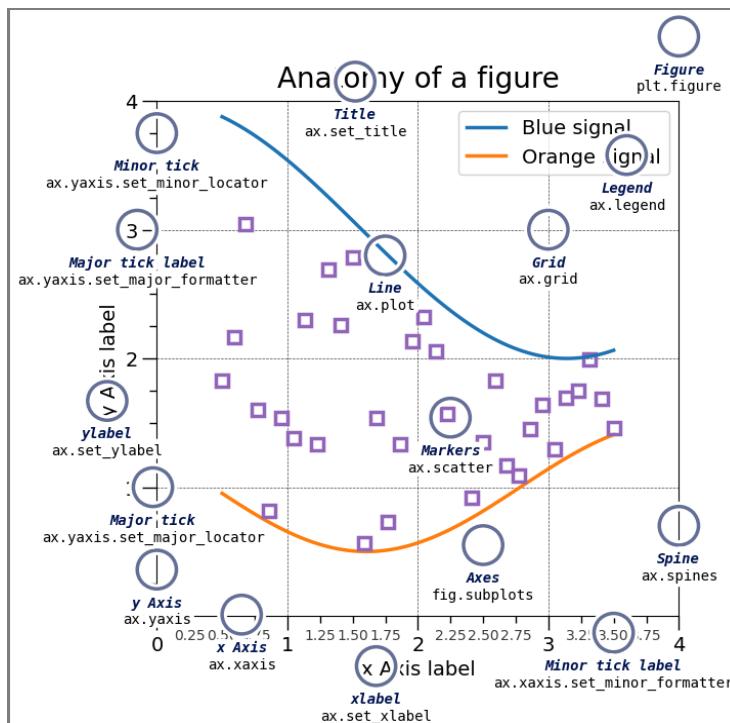
- วิธีเช็ค Version Matplotlib

```
print("Matplotlib version", matplotlib.__version__)
```

Matplotlib version 3.7.1

- Anatomy of a Figure

ก่อนอื่นเรามาดู Anatomy ของกราฟที่เราสามารถ custom ได้เพื่อให้เรานำเสนอข้อมูลได้ในแบบที่เราต้องการ ซึ่งประกอบไปด้วย Title, Major/Minor tick Label, Major/Minor tick, x/y Axis, x/y label, Line, Markers, Grid, Legend, Axes, Figure, Spines



- Plotting

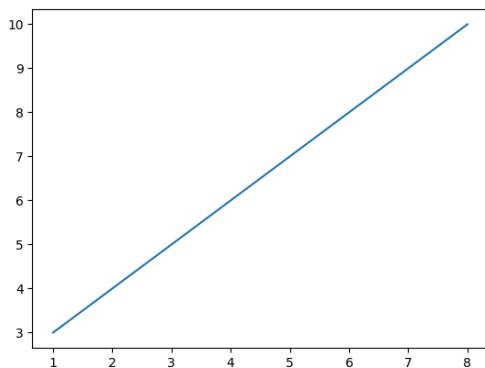
- plotting x and y points

ฟังก์ชัน `plot()` ใช้สำหรับวาดจุด (markers) ใน Figure โดยค่าเริ่มต้น ฟังก์ชัน `plot()` จะลากเส้นจากจุดหนึ่งไปยังอีกจุดหนึ่งโดยฟังก์ชันใช้ parameter เพื่อระบุจุดใน Figure โดยกำหนดให้ parameter 1 คือ Array ที่มีจุดบนแกน x (แกนนอน) และ parameter 2 คือ Array ที่มีจุดบนแกน y (แกนตั้ง)

ในตัวอย่างนี้เราต้องการ plot กราฟเส้นจากจุด (1,3) ถึงจุด (8,10) เราจึงต้องใส่ Array 2 Array คือ [1,8] และ [3,10] ไปในฟังก์ชัน `plot()`

```
xpoints = np.array([1, 8])
y whole="points = np.array([3, 10])

plt.plot(xpoints, ypoints)
plt.show()
```

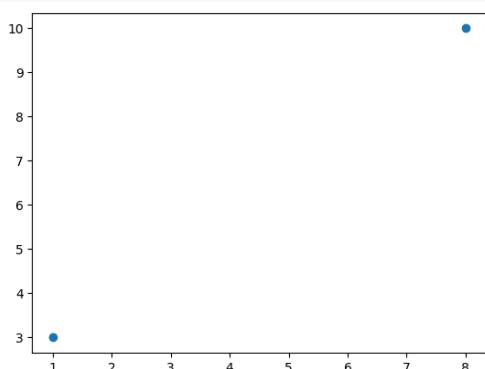


- Plotting without Line

หากเราต้องการ plot เพียง marker เราสามารถใช้ shortcut string notation parameter 'o' ที่หมายถึง marker ที่เป็นลักษณะวงกลม

```
xpoints = np.array([1, 8])
y whole="points = np.array([3, 10])

plt.plot(xpoints, ypoints, 'o')
plt.show()
```

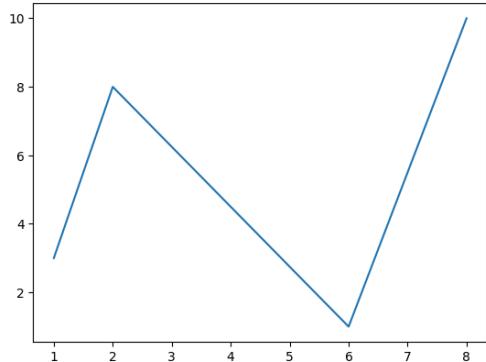


- Multiple points

เราสามารถ plot จุดได้หลายจุดเท่าที่เราต้องการ เพียงแต่เราต้องแน่ใจว่าเรามีคู่อับดับของจุดอยู่ครบ หรือกล่าวในอีกทางหนึ่งได้ว่า มี Array 2 Array ที่มีขนาดเท่ากัน

```
xpoints = np.array([1, 2, 6, 8])
```

```
ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(xpoints, ypoints)  
plt.show()
```

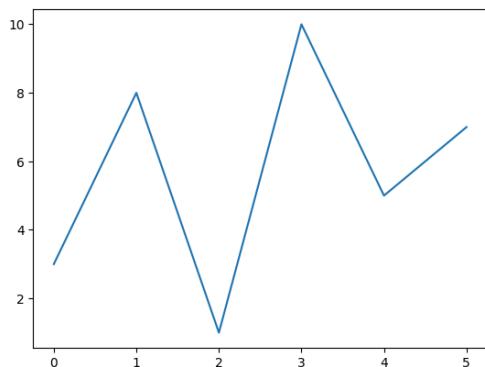


○ Default X-points

หากเราไม่ระบุค่าในแกน x matplotlib จะใช้ค่า default ซึ่งคือ 0, 1, 2, 3, ... จำนวนเท่ากับความยาวของค่าในแกน y

ดังนั้น หากเราจะใช้ค่าในแกน x ที่เริ่มจาก 0 ไปทางบวก เราต้องใส่ค่าแกน x ให้ได้

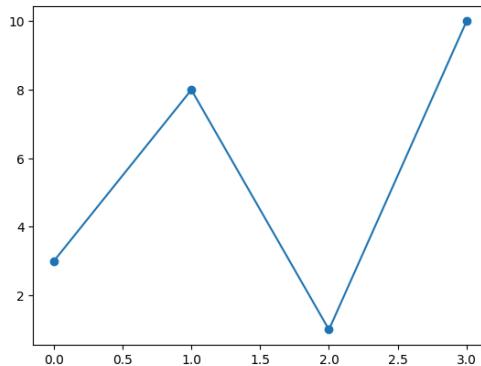
```
ypoints = np.array([3, 8, 1, 10, 5, 7])  
  
plt.plot(ypoints)  
plt.show()
```



● Markers

เราสามารถใช้ argument marker เพื่อกำหนดให้จุดมีลักษณะ marker เป็นลักษณะอย่างไรได้

```
ypoints = np.array([3, 8, 1, 10])  
  
plt.plot(ypoints, marker = 'o')  
plt.show()
```



○ Marker Reference

เรามาสามารถเลือกลักษณะของ marker เพื่อ plot ในกราฟของเราได้ตาม format ด้านล่างนี้

- Unfilled markers

'.'	1	—	—
'1'	2		
'2'	3		
'3'	4	◀	◀
'4'	5	▶	▶
'+'	6	△	△
'x'	7	◆	◆
' '	8	◀▶	◀▶
'-'	9	◀▶	◀▶
'0'	10	△△	△△
'_'	11	▼▼	▼▼

- Marker fill styles

'full'	●	●	●
'left'	◐	◑	◑
'right'	◑	◐	◑
'bottom'	◑	◑	◑
'top'	◑	◑	◑
'none'	○	○	○

- Markers from TeX symbols

'\$1\$'	1	1	1
'\$\frac{1}{2}\$'	½	½	½
'\$f\$'	f	f	f
'\$\mathcal{A}\$'	A	A	A

- Filled markers

'.'	●	●	●	'p'	◇	◇	◇
'o'	●	●	●	'*'	★	★	★
'v'	▼	▼	▼	'h'	⬡	⬡	⬡
'^'	▲	▲	▲	'H'	⬢	⬢	⬢
'<'	◀	◀	◀	'D'	⬢	⬢	⬢
'>'	▶	▶	▶	'd'	⬢	⬢	⬢
'8'	●	●	●	'P'	✚	✚	✚
's'	■	■	■	'X'	✖	✖	✖

- Markers from Paths

circle	●	●	●
cut_star	★	★	★

- Advanced marker modifications with transform

Filled marker	●	●	●	●	●	●	●
Un-filled marker	Y	Y	Y	Y	Y	Y	Y
Equation marker	½	½	½	½	½	½	½

0° 10° 20° 30° 45° 60° 90°

- Setting marker cap style and join style



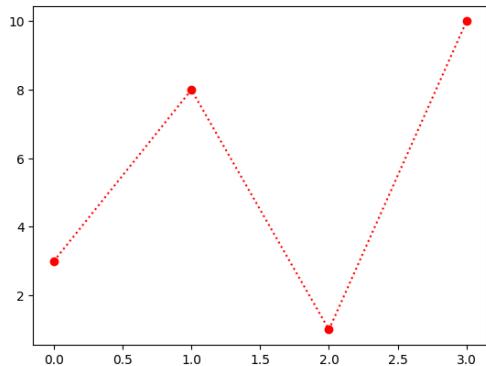
0° 10° 20° 30° 45° 60° 90°

○ Format Strings

เราสามารถใช้พารามิเตอร์ที่เรียกว่า fmt ใน การระบุลักษณะของ marker โดยมีรูปแบบ

marker|line|color

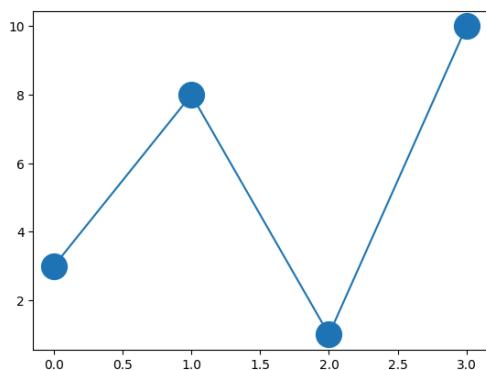
```
plt.plot(ypoints, 'o:r')
```



○ Marker Size

เราสามารถใช้ argument markersize หรือ ms เพื่อที่จะกำหนดขนาดของ markers ได้

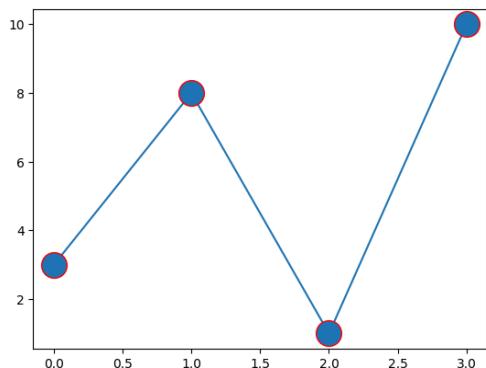
```
plt.plot(ypoints, marker = 'o', ms = 20)
```



○ Marker Color

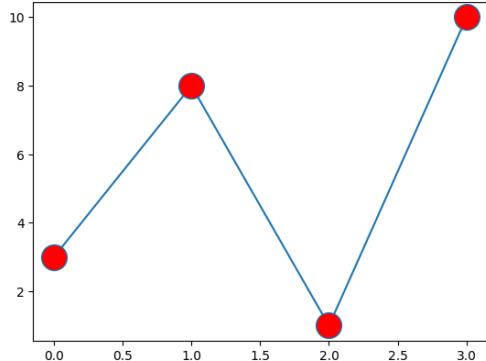
เราสามารถใช้ argument markeredgecolor หรือ mec เพื่อที่จะกำหนดสีขอบของ markers

```
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'r')
```



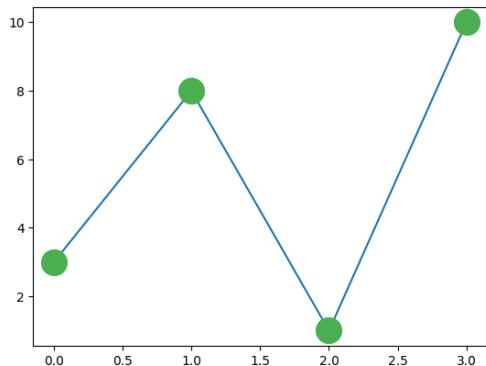
และเราสามารถใช้ argument markerfacecolor หรือ mfc เพื่อกำหนดสีข้างใน markers ได้

```
plt.plot(ypoints, marker = 'o', ms = 20, mfc = 'r')
```



เรารสามารถใช้ Hexadecimal color values เพื่อกำหนดค่าสีได้หลากหลายมากขึ้น

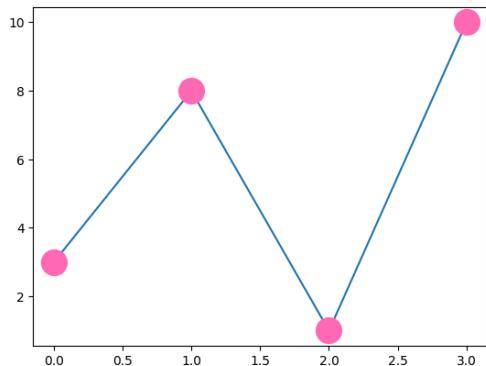
```
plt.plot(ypoints, marker = 'o', ms = 20, mec = '#4CAF50', mfc = '#4CAF50')
```



และยังมีอีกทางหนึ่งคือใช้ชื่อของสีในการกำหนดสี ซึ่งมีอยู่ด้วยกัน 140 ค่า ดูต่อไปที่

https://www.w3schools.com/colors/colors_names.asp

```
plt.plot(ypoints, marker = 'o', ms = 20, mec = 'hotpink', mfc = 'hotpink')
```



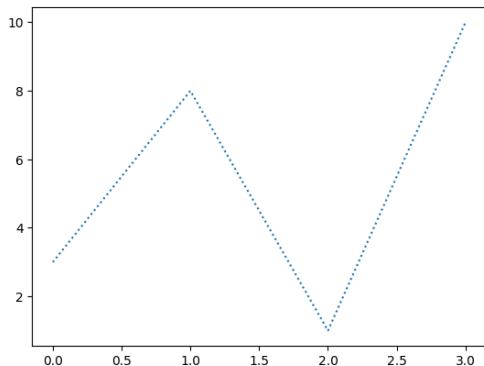
- *Line*

- *Linestyle*

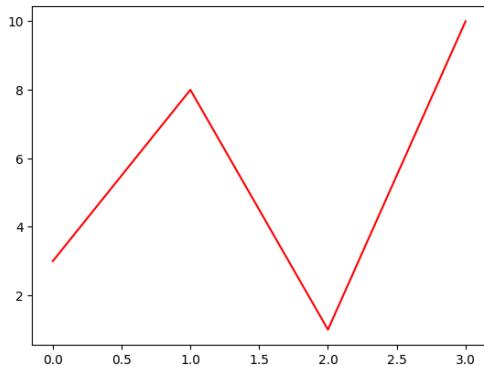
เรารสามารถใช้ argument linestyle หรือ ls เพื่อที่จะกำหนดลักษณะของเส้นในกราฟซึ่งสามารถใช้ได้ทั้งแบบเต็มและแบบย่อตามตาราง โดยสามารถใช้ควบคู่กับ argument C เพื่อที่จะเปลี่ยนสีได้ และสามารถใส่เป็น RGB code ได้เช่นกัน

Style	Description	Style	Description
'solid' (default)	'-'	'dashdot'	'-.'
'dotted'	'.'	'None'	'none', ''
'dashed'	'--'		

```
plt.plot(ypoints, linestyle = 'dotted')
```



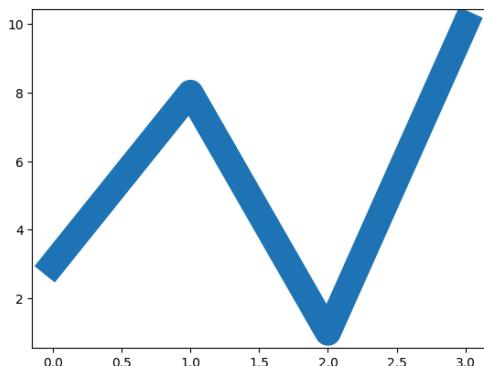
```
plt.plot(ypoints, color = 'r')
```



○ Line Width

เราสามารถใช้ argument linewidth หรือ lw เพื่อที่จะเปลี่ยนขนาดของเส้นได้ โดยสามารถใช้ argument ได้ทั้งแบบเต็มและแบบย่อ

```
plt.plot(ypoints, linewidth = '20.5')
```



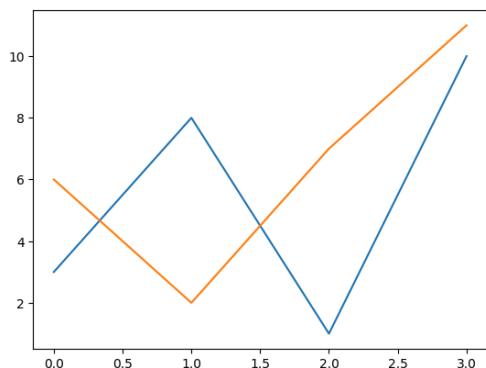
- Multiple Lines

เราสามารถ plot เลื่อนได้หลายเส้นเท่าที่เราต้องการเพียงแค่เพิ่มฟังก์ชัน plt.plot() และเรายังสามารถเพิ่มจุดสำหรับแกน x และแกน y สำหรับการ plot ได้อีกด้วย

```
y1 = np.array([3, 8, 1, 10])
y2 = np.array([6, 2, 7, 11])

plt.plot(y1)
plt.plot(y2)

plt.show()
```



- Labels and Title

- Creating

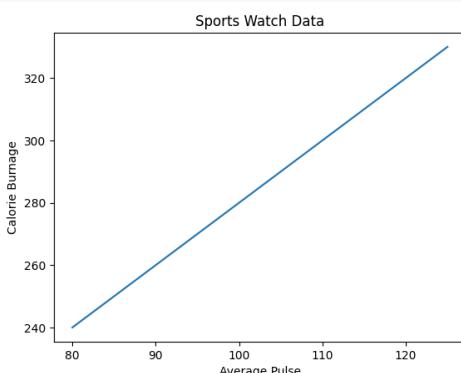
เราสามารถใช้ฟังก์ชัน xlabel() และ ylabel() เพื่อตั้งค่าป้ายกำกับแกน x และ y โดยสามารถใช้ฟังก์ชัน title() เพื่อกำหนด title ของ Figure

```
x = np.array([80, 85, 90, 95, 100, 105, 110, 115, 120, 125])
y = np.array([240, 250, 260, 270, 280, 290, 300, 310, 320, 330])

plt.plot(x, y)

plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

plt.show()
```



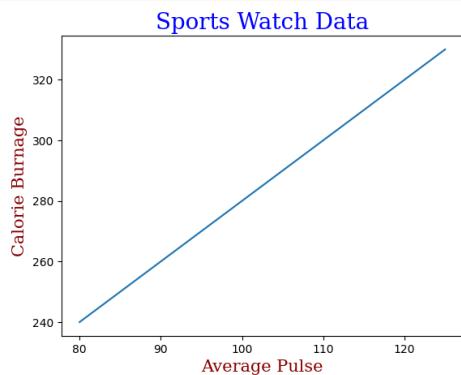
- Font Properties

เราสามารถใช้ parameter `fontdict` ใน `xlabel()`, `ylabel()` และ `title()` เพื่อตั้งค่า font สำหรับ `title` และ `labels` ได้

```
font1 = {'family':'serif','color':'blue','size':20}
font2 = {'family':'serif','color':'darkred','size':15}

plt.title("Sports Watch Data", fontdict = font1)
plt.xlabel("Average Pulse", fontdict = font2)
plt.ylabel("Calorie Burnage", fontdict = font2)

plt.plot(x,y)
```



- Setting Thai Font

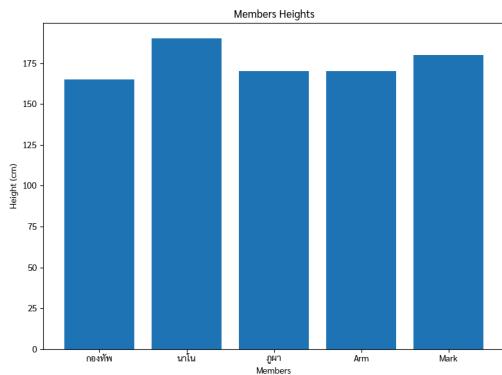
```
!wget -q https://github.com/google/fonts/raw/main/ofl/sarabun/Sarabun-Regular.ttf
import matplotlib as mpl
mpl.font_manager.fontManager.addfont('Sarabun-Regular.ttf')
mpl.rc('font', family='Sarabun')
```

```
import matplotlib.pyplot as plt
import numpy as np

names = ['กองทัพ', 'นาโน', 'ภูผา', 'Arm', 'Mark']
heights = [165, 190, 170, 170, 180]
fig, ax = plt.subplots(figsize=(8, 6))
x = np.arange(len(names))
ax.bar(x, heights)
ax.set_xticks(x)
ax.set_xticklabels(names)

ax.set_xlabel('Members')
ax.set_ylabel('Height (cm)')
ax.set_title('Members Heights')

plt.tight_layout()
plt.show()
```



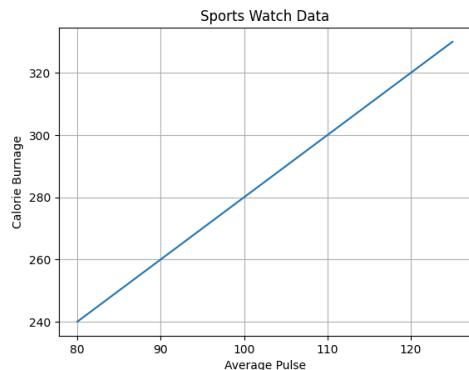
- Adding Grid Lines

- Add Grid

เราสามารถใช้ฟังก์ชัน `grid()` เพื่อเพิ่มเส้นตารางสำหรับการ plot

```
plt.title("Sports Watch Data")
plt.xlabel("Average Pulse")
plt.ylabel("Calorie Burnage")

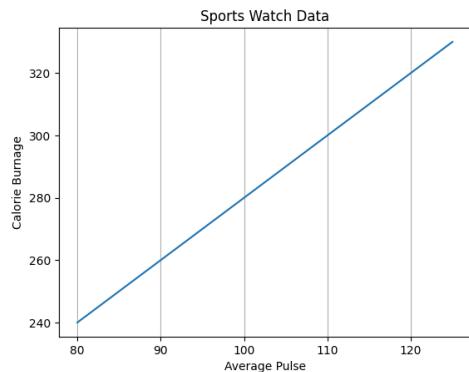
plt.plot(x, y)
plt.grid()
plt.show()
```



- Specify Grid Lines

เราสามารถเลือกให้แสดง grid เฉพาะแกนได้

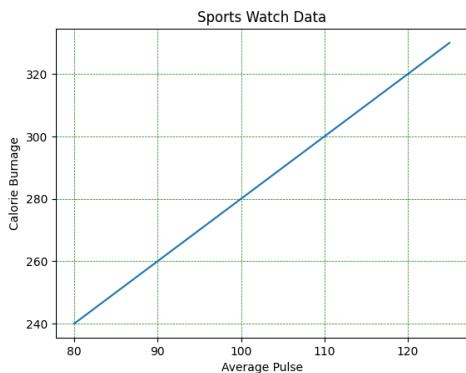
```
plt.grid(axis = 'x')
```



- Line Properties

เรายังสามารถตั้งค่าคุณสมบัติเส้นของเส้นตารางได้ ดังนี้

```
plt.grid(color = 'green', linestyle = '--', linewidth = 0.5)
```

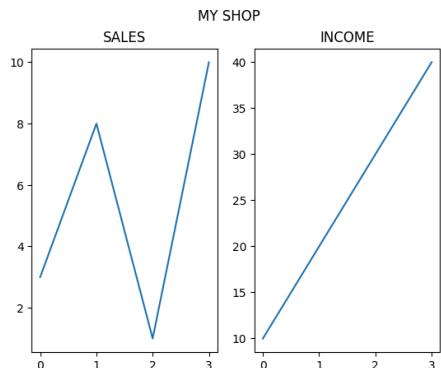


- Subplot

- Display Multiple Plot

ฟังก์ชัน `subplot`(จำนวนແຕວ, จำนวนຄອລິນ໌, ຕຳແໜ່ງຮູບ) ເຮັດວຽກສ້າງ Figure ລາຍ Figure ໄດ້ໄຍ້ໃນຮູບແທນ ໂດຍສາມາດເພີ່ມ Title ຂອງແຕ່ລະ Figure ໄດ້ດ້ວຍຝັ້ນ `title()` ແລະເພີ່ມ ດຳບຽນຂອງທັງຮູບດ້ວຍຝັ້ນ `suptitle()`

```
#plot 1:  
x = np.array([0, 1, 2, 3])  
y = np.array([3, 8, 1, 10])  
plt.subplot(1, 2, 1)  
plt.plot(x,y)  
plt.title("SALES")  
  
#plot 2:  
x = np.array([0, 1, 2, 3])  
y = np.array([10, 20, 30, 40])  
plt.subplot(1, 2, 2)  
plt.plot(x,y)  
plt.title("INCOME")  
plt.suptitle("MY SHOP")  
plt.show()
```



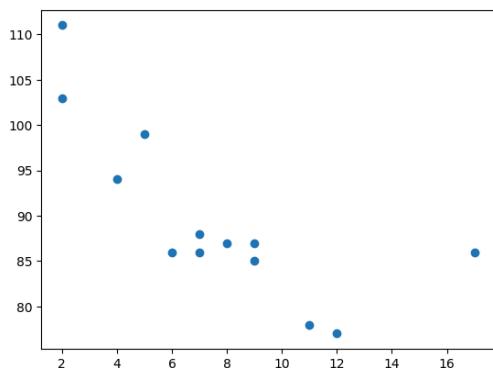
- Scatter Plot

- Creating Scatter

ใช้ฟังก์ชัน `scatter()` เพื่อสร้างจุดหนึ่งจุดสำหรับการ `plot` แต่ละครั้ง ซึ่งต้องการ 2 Array ที่มีขนาดเท่ากัน โดย array อันแรกเอาไว้สำหรับเก็บค่าแกน x และ array อันที่สองเอาไว้สำหรับเก็บค่าแกน y

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
```

```
plt.scatter(x, y)
plt.show()
```



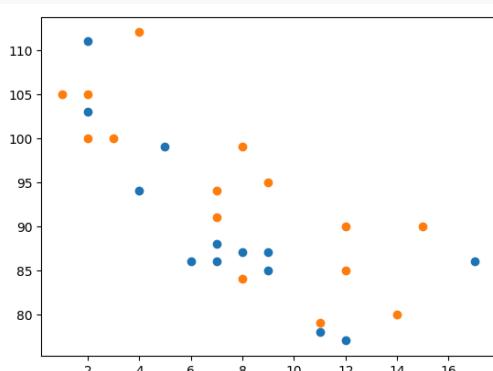
- Compare Plot

สามารถสร้างความแตกต่างของข้อมูลเพื่อแสดงผลเปรียบเทียบในรูปเดียวกันได้โดยการใช้ argument `color` หรือ `c`

```
#day one, the age and speed of 13 cars:
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y)

#day two, the age and speed of 15 cars:
x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y)

plt.show()
```



- Colors

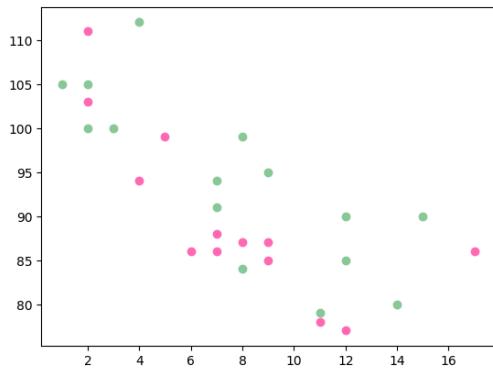
```

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
plt.scatter(x, y, color = 'hotpink')

x = np.array([2,2,8,1,15,8,12,9,7,3,11,4,7,14,12])
y = np.array([100,105,84,105,90,99,90,95,94,100,79,112,91,80,85])
plt.scatter(x, y, color = '#88c999')

plt.show()

```

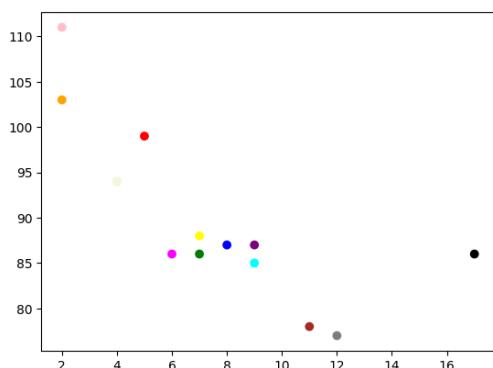


○ Color Each Dot

```

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors =
np.array(["red","green","blue","yellow","pink","black","orange","purple"
,"beige","brown","gray","cyan","magenta"])
plt.scatter(x, y, c=colors)
plt.show()

```



○ ColorMap

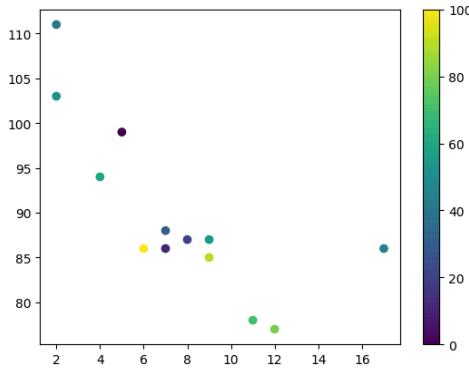
```

x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
colors = np.array([0, 10, 20, 30, 40, 45, 50, 55, 60, 70, 80, 90, 100])

plt.scatter(x, y, c=colors, cmap='viridis')
plt.colorbar()

```

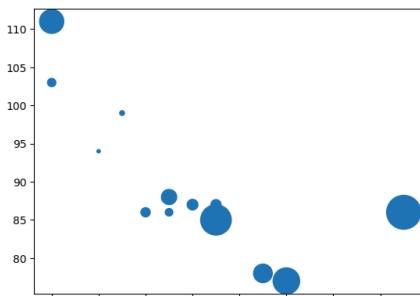
```
plt.show()
```



○ Size

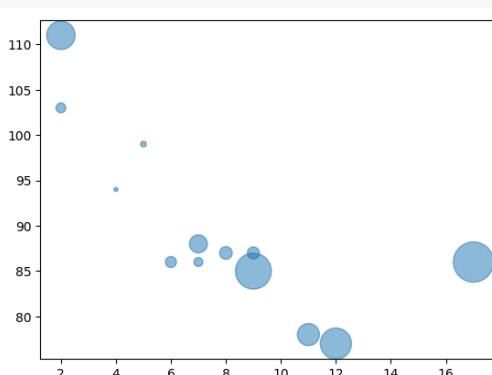
เราสามารถเปลี่ยนขนาดของจุดด้วย argument s และต้องตรวจสอบให้แน่ใจว่า array ของ size และความโปร่งใสมีขนาดเท่ากับ array ที่เก็บค่าของแกน x และ ปุ๊ง เราสามารถปรับความโปร่งใสของจุดด้วย argument alpha โดยที่ เริ่มต้นอ่อน $0 \leq \alpha \leq 1$ เริ่มต้นเข้ม

```
x = np.array([5,7,8,7,2,17,2,9,4,11,12,9,6])
y = np.array([99,86,87,88,111,86,103,87,94,78,77,85,86])
sizes = np.array([20,50,100,200,500,1000,60,90,10,300,600,800,75])
plt.scatter(x, y, s=sizes)
plt.show()
```



○ Alpha

```
plt.scatter(x, y, s=sizes, alpha=0.5)
```



○ Combine Color Size and Alpha

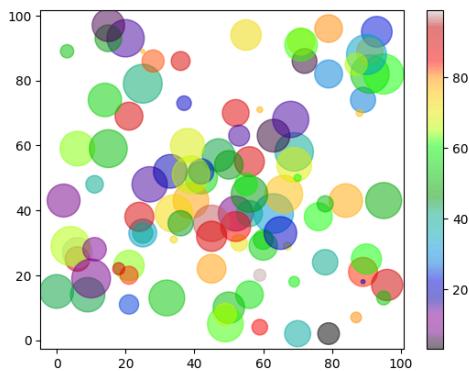
```
x = np.random.randint(100, size=(100))
y = np.random.randint(100, size=(100))
```

```

colors = np.random.randint(100, size=(100))
sizes = 10 * np.random.randint(100, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.5, cmap='nipy_spectral')
plt.colorbar()
plt.show()

```



- *Bars Plot*

- *Creating Bars*

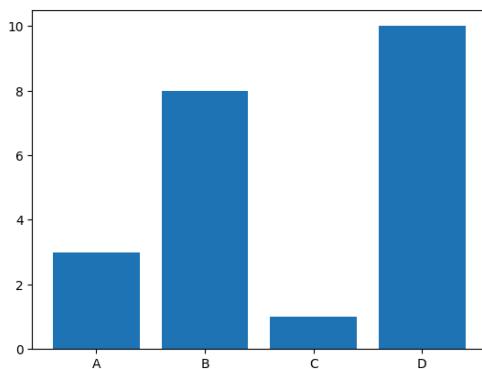
เรารสามารถใช้ฟังก์ชัน `bar()` เพื่อสร้าง bar graphs

```

x = np.array(["A", "B", "C", "D"])
y = np.array([3, 8, 1, 10])

plt.bar(x,y)
plt.show()

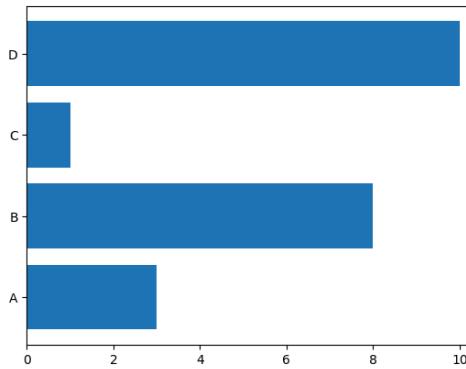
```



- *Horizontal Bars*

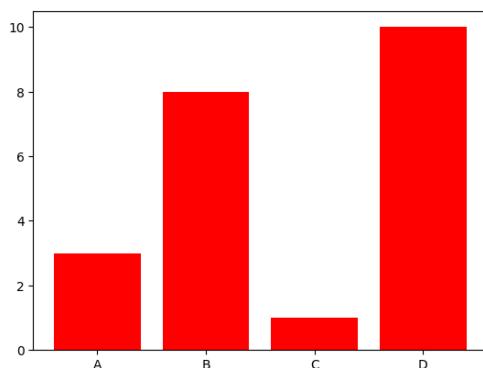
หากเราต้องการให้เดบแสดงแนวโนนแทนที่จะเป็นแนวตั้ง ให้ใช้ฟังก์ชัน `barh()` แทนได้ โดยทั้ง ฟังก์ชัน `bar()` และ `barh()` สามารถใช้ argument `color` เพื่อตั้งค่าสีของแผนภูมิเท่งได้ และใช้ argument `width` และ `height` เพื่อกำหนดความกว้างของแต่ละแท่ง

```
plt.barh(x, y)
```



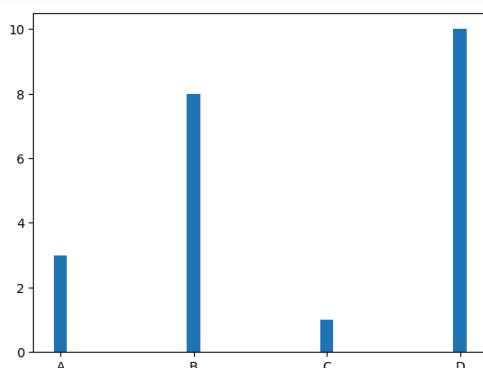
○ Bar Color

```
plt.bar(x, y, color = "red")
```



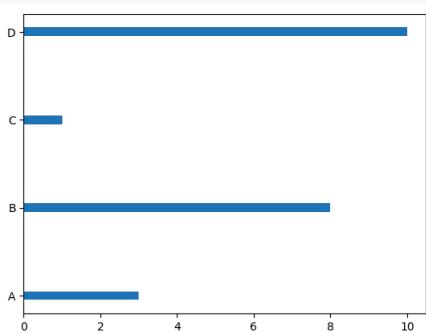
○ Bar Width

```
plt.bar(x, y, width = 0.1)
```



○ Bars Height

```
plt.barh(x, y, height = 0.1)
```



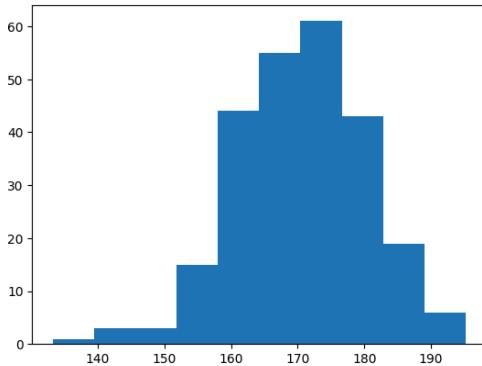
- *Histogram*

- Creating Histogram

เราสามารถใช้ฟังก์ชัน `hist()` เพื่อสร้าง Histogram

```
x = np.random.normal(170, 10, 250)
```

```
plt.hist(x)  
plt.show()
```



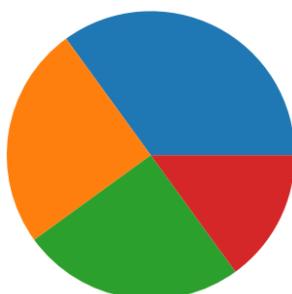
- *Pie Chart*

- Creating Pie

เราสามารถใช้ฟังก์ชัน `pie()` เพื่อวาดแผนภูมิวงกลม

```
y = np.array([35, 25, 25, 15])
```

```
plt.pie(y)  
plt.show()
```



เพิ่มป้ายกำกับให้กับแผนภูมิวงกลมด้วย parameter `label` ต้องเป็น Array ที่มี 1 label สำหรับแต่ละส่วน ซึ่งบางที่เรารายการให้ส่วนใดส่วนหนึ่งโดดเด่น

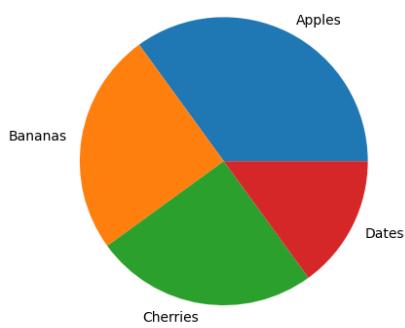
parameter `explode` หากจะบุรุ่ว ไม่ใช่ None ต้องเป็น Array ที่มีค่าหนึ่งค่าสำหรับแต่ละส่วนซึ่งแต่ละค่าแสดงถึงระยะห่างจากศูนย์กลางแต่ละส่วน

หากเราต้องการให้รูปของเรามีมิติมากขึ้นเราราสามารถกำหนด parameter `shadow` เป็น True

- Labels

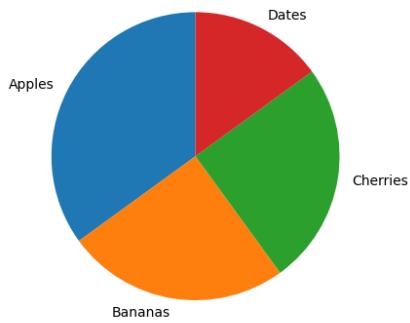
```
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
plt.pie(y, labels = mylabels)
```

```
plt.show()
```



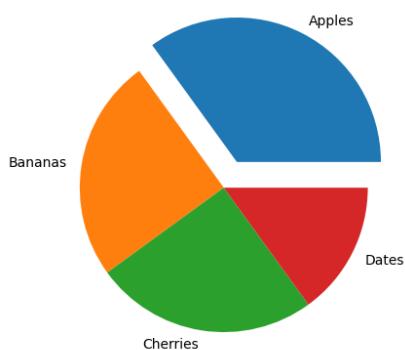
○ Start Angle

```
plt.pie(y, labels = mylabels, startangle = 90)  
plt.show()
```



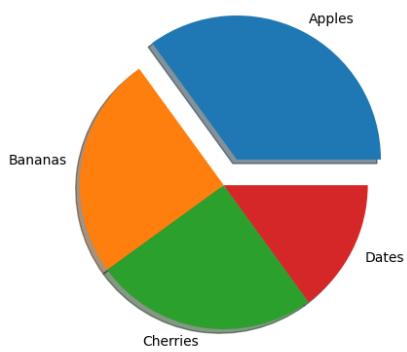
○ Explode

```
y = np.array([35, 25, 25, 15])  
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]  
myexplode = [0.2, 0, 0, 0]  
  
plt.pie(y, labels = mylabels, explode = myexplode)  
plt.show()
```



○ Shadow

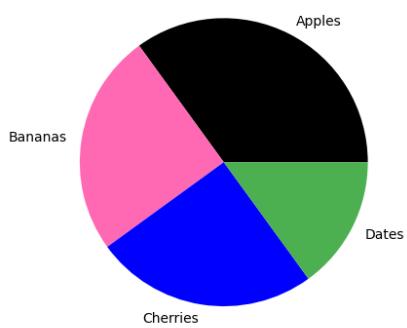
```
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
```



○ Colors

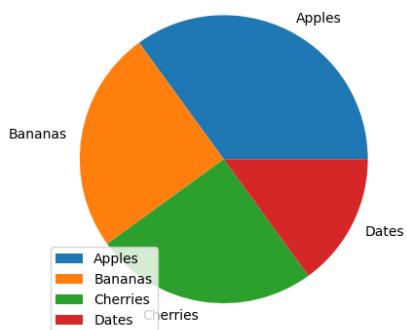
```
y = np.array([35, 25, 25, 15])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]
mycolors = ["black", "hotpink", "b", "#4CAF50"]
```

```
plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```



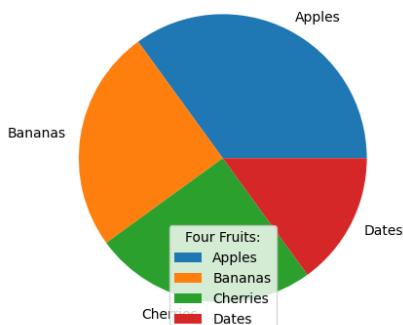
○ Legend

```
plt.pie(y, labels = mylabels)
plt.legend()
```



■ Legend with Header

```
plt.legend(title = "Four Fruits:")
```

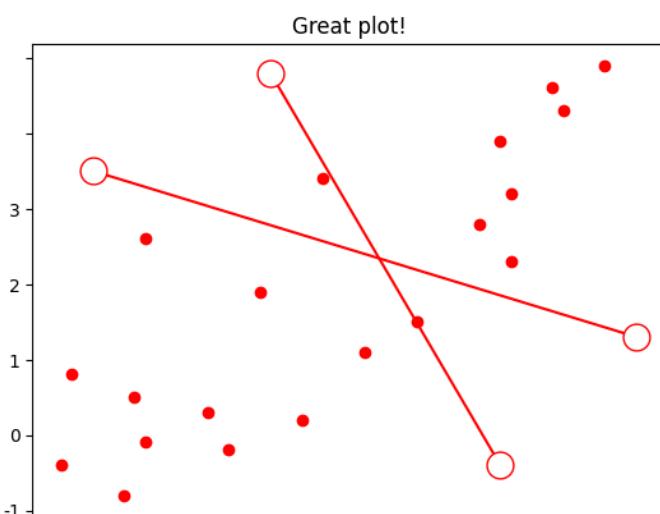


- *Cheatsheets*

สามารถดูข้อมูลเพิ่มเติมได้ที่ <https://matplotlib.org/cheatsheets/>

Good Data Visualization?

- เห็นอะไรจาก (กราฟ?) นี้



เรามาเริ่มดูกันเลย เริ่มจากด้านบน คือ หัวข้อที่เขียนบนอกไว้ว่า 'Great plot!' ซึ่งไม่ได้ทำให้เรารู้ความหมายของกราฟนี้เลย ถ้ามาดีอแกน x, y คืออะไรทำไม่ข้อมูลบางส่วนหายไป ทำไม่แกน x ไม่มีค่า แล้วทำไม่เลือกจุดที่เป็นสีเดียวกันทั้งหมด ทำไม่มีเส้นมีจุด เส้นบอกความสัมพันธ์อะไรหรือไม่

ดังนั้น กราฟนี้เป็นตัวอย่างที่ไม่ดีนัก เพราะไม่ทำให้เรารู้สึกความสัมพันธ์ของข้อมูล หรือแม้แต่ยังไม่รู้ด้วยซ้ำว่าเป็นข้อมูลอะไร

- แนวทางการเลือก chart คนดูให้เข้าใจง่าย

ก่อนที่จะเลือกราฟ เราต้องรู้จุดมุ่งหมายของเราก่อนว่าจะพลอตไปเพื่ออะไร เพราะแต่ละกราฟก็มีความแตกต่างกัน และสื่อความหมายออกมาต่างกันด้วย เราต้องเริ่มคิดจากว่า เราอยากจะบอกอะไรกับผู้ชม หรือคนที่ดูกราฟของเรา

การเลือกราฟให้คนดูเข้าใจได้ง่าย เริ่มต้นง่าย ๆ ได้โดย ถามคำถามว่า “เราต้องการพลอตกราฟเพื่ออะไร”

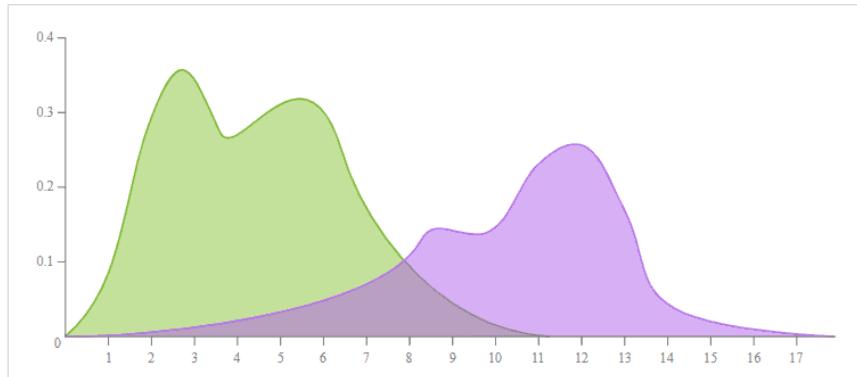
โดยคำตอบของเราจะช่วยในกลุ่มใดกลุ่มหนึ่ง ตามหัวข้อด้านล่างนี้

- plot เพื่อเข้าใจการแจกแจงของ data

เมื่อเรารู้แล้วว่า เราอยากรู้การแจกแจง (Distribution) ของดาต้า ซึ่งสามารถดูได้ว่า Trend ของข้อมูลเป็นอย่างไร range อยู่ในช่วงไหน มีข้อมูลที่หน้าตาผิดเปลกจากข้อมูลส่วนใหญ่ (outlier) ใหม่ เราสามารถใช้กราฟได้ดังต่อไปนี้

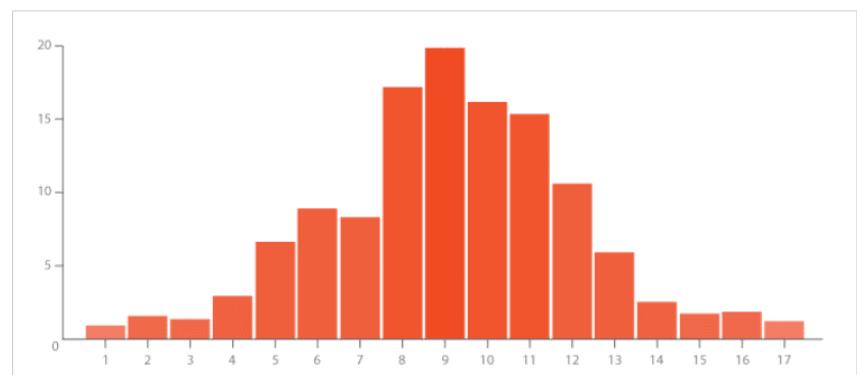
★ 1 variable

- Frequency Plot/Density Plot



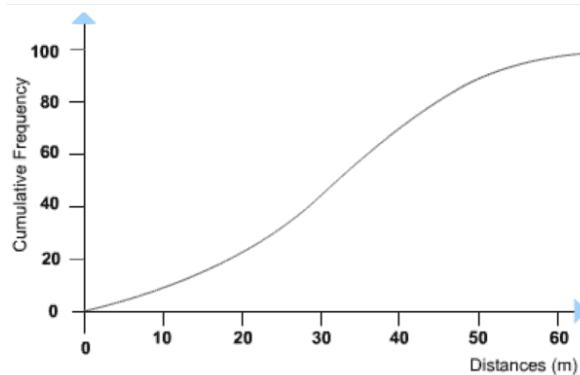
- เรียกอีกชื่อว่า Kernel Density Plot หรือ Density Trace Graph
- เป็นผลลัพธ์ที่แสดงการแจกแจงของข้อมูลเป็นแบบช่วงค่าที่ต่อเนื่อง (Continuous) หรือ ตามช่วงเวลา
- จุดสูงสุดของผลลัพธ์ คือจุดที่มีความถี่ (Frequency) หรือจำนวนของดาต้าสูงสุด
- ข้อดีคือสามารถบอกรูปร่างของการแจกแจงของข้อมูลได้เลย โดยไม่ต้องแบ่งเป็นช่วง แบบ Histogram
- ใช้ในการณ์ที่มีดาต้าเยอะ

- Histogram



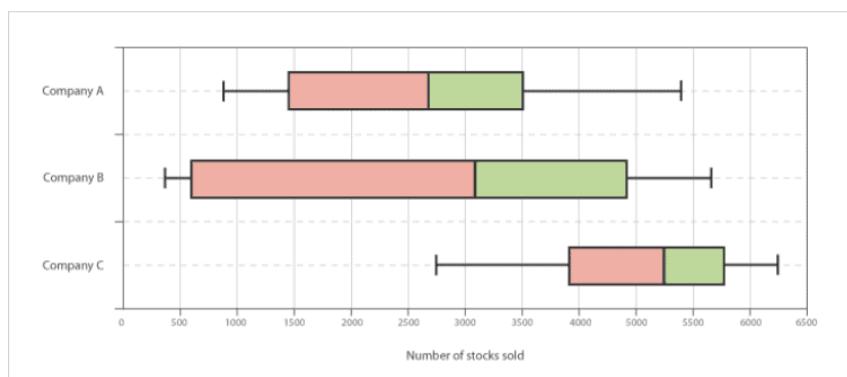
- เป็นผลลัพธ์ที่มีหน้าตาคล้าย Bar Chart ทำหน้าที่เหมือนกับ Density Plot ตรงที่แสดง การแจกแจงของข้อมูลเป็นแบบช่วงค่าที่ต่อเนื่อง (Continuous) หรือตามช่วงเวลา
- แต่ละบาร์แสดงสถิติความถี่ (Frequency) ของดาต้าในช่วงนั้นๆ (Bin/Interval)
- สามารถบอกได้ว่าจุดไหนมีค่ากระฉูดตัวมาก หรือมีค่าที่ต่างไปจากพวกใหม่
- ใช้ในการณ์ที่มีดาต้าน้อย

- Cumulative Frequency Diagram



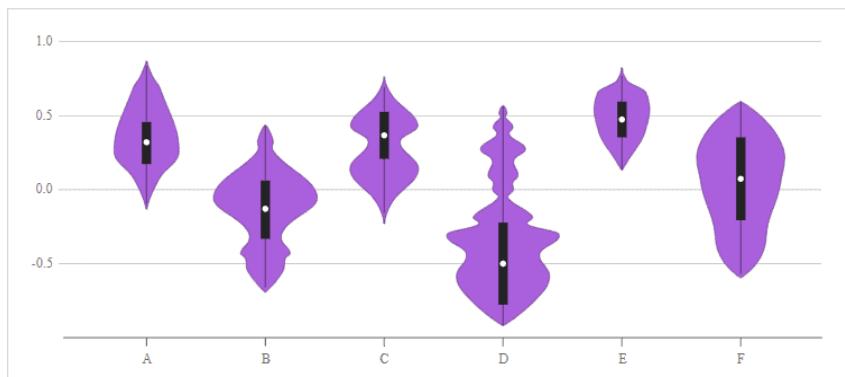
- เป็นพLOTที่ต่อมาจาก Frequency หรือ Density Plot ต่างกันที่plotตนี้แสดงความถี่สะสม
- สามารถหาค่ามัธยฐาน (Median) และ ค่า Quartile โดยประมาณได้ โดยใช้ 0.5 0.25 0.75 คูณกับจำนวนทั้งหมดเพื่อหาค่า median Q1 และ Q3 ตามลำดับ

- Box and Whisker Plot



- เรียกอีกชื่อว่า Box & Whisker Plot
- แสดงภาพรวมของการแจกแจงข้อมูลผ่าน Quartiles
- เส้นที่ยื่นออกมาระหว่างตัวกล่องเรียกว่า Whisker
- ข้อดีคือใช้พื้นที่ได้แบบจำกัด ในขณะที่สามารถเปรียบเทียบหลายกลุ่ม
- ลิงที่บอกได้จากplotตนี้คือ
 - Range ค่าต่ำสุด (Min) ค่าสูงสุด (Max) ค่ามัธยฐาน (Median) ค่า Quartile ที่1และ3
 - ดูว่ามี Outlier ใหมและมีค่าเท่าไหร่
 - แจกแจงของข้อมูล สมมาตร (Symmetrical) หรือไม่
 - ดาต้ากระฉูดตัวกันมากขนาดไหน
 - ดูว่ามีการ skew หรือไม่ไปในทิศทางไหน

- Violin Plot



- เป็นการรวมกันของ Box Plot และ Kernel Density Plot สามารถเห็นรูปร่างการแจกแจงของข้อมูล (Probability Density) ได้
- จุดสีขาวตรงกลางคือค่ามัธยฐาน (Median)
- ช่วงด้านบนและด้านล่างคือ Interquartile Range (Q3 – Q1)
- เส้นที่ยื่นออกไปจากบนลงล่างคือ ค่าสูงสุด (Max) และค่าต่ำสุด (Min)

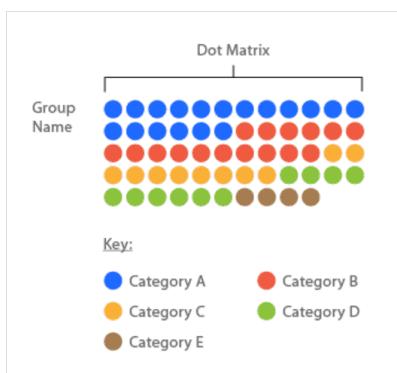
- Stem and Leaf Plot

STEMS	LEAVES
0	5 8
1	2 3 5 7
2	0 0 5 8 8 9
3	0 0 1 3 3 6 6 7 7 7 7 7 8 8 8 8 9 9
4	1 3 5 5 6 7 7 8 8 8 9 9
5	0 0 0 1 1 1 2 6 8
6	0 0 1 1 2 4 4 4 4 8 8 9
7	0 5 5 5 7
8	3 4 4 5 6 6 6 7 8 9
9	0 1 2 2 2 5 5 6 8 9 9
10	2 2 2 5 7

Raw Data in Row:
102, 102, 102, 105, 107

- เรียกอีกชื่อว่า Stem Plot หรือ Stem & Leaf Display
- แสดงการแจกแจงของข้อมูลโดยเรียงลำดับจากน้อยไปมาก โดยให้ตัวเลขแรกเป็นหลัก ใส่ไว้ใน stem ตัวเลขที่เหลือใส่ด้านซ้ายใน leaf
- สามารถเห็นภาพรวม หา Outlier หรือหาค่าฐานนิยม
- ข้อเสียของพلوตตนี้คือ แสดงได้ดีในขนาดของข้อมูลจำกัดเท่านั้น

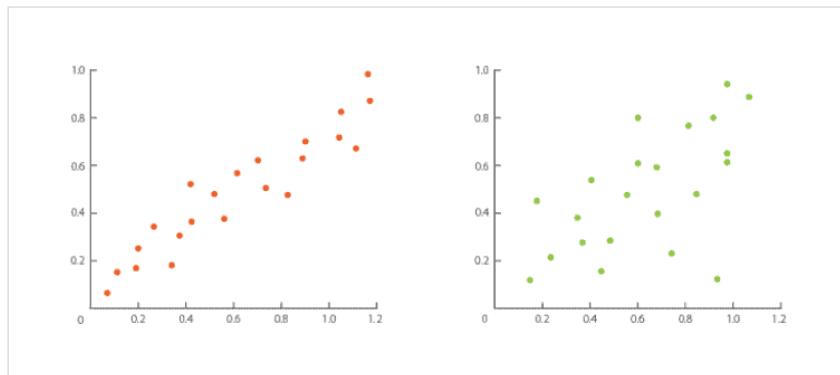
- Dot Matrix Chart



- แสดงข้อมูลแบบ discrete เป็นจำนวนจุด
- สีแต่ละสีแสดงกลุ่มที่แตกต่างกัน
- แสดงภาพรวมของการแจกแจง และสัดส่วนในแต่ละกลุ่มได้ในพلوตเดียว

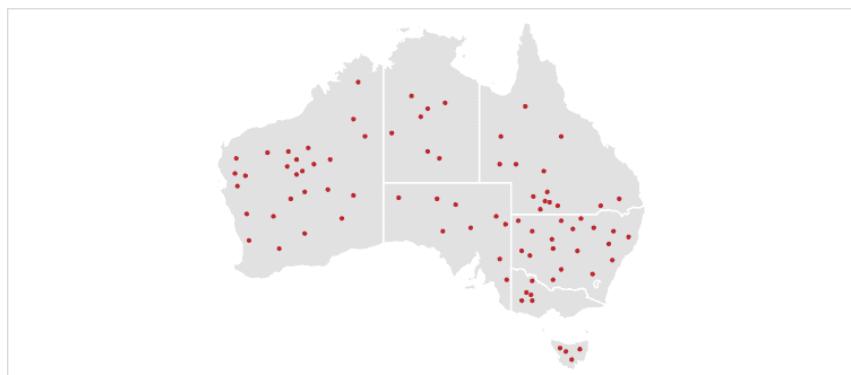
★ 2 variable

- Scatter Plot



- เรียกอีกชื่อว่า Stem Graph, Point Graph, Scatter Chart หรือ Scattergram
- เป็นพล๊อตที่ใช้บ่อยมากที่สุด
- แสดงรูปร่างของการแจกแจง (Distribution) ให้เห็นได้
- สามารถเปรียบเทียบการแจกแจงของสอง variable ได้

- Dot Map



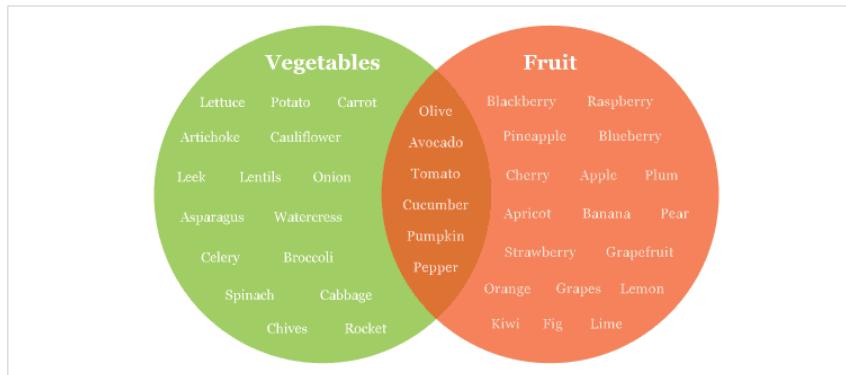
- เรียกอีกชื่อว่า Point Map, Dot Distribution Map หรือ Dot Density Map
- เป็นพล๊อตที่เป็นแผนที่ที่สามารถเห็นการแจกแจงของข้อมูลทั่วทุกเขตได้โดยให้จุดขนาดเท่า ๆ กันแทนตัวเลข
- สามารถมองเห็นแบบแผนได้ง่าย

● plot เพื่อดูความสัมพันธ์ระหว่าง variable

ถ้าเราอยากรู้ว่า variable แต่ละอันมีความสัมพันธ์กันอย่างไร จะเป็นแบบว่า อันหนึ่งเพิ่ม อีกอันหนึ่งก็เพิ่ม หรือ อันหนึ่งเพิ่ม อีกอันหนึ่งลด หรือ ดูความสัมพันธ์ที่ซับซ้อนขึ้น แบบมีมากกว่าสอง variable ขึ้นไป ก็ทำได้ดังนี้

★ 1 variable

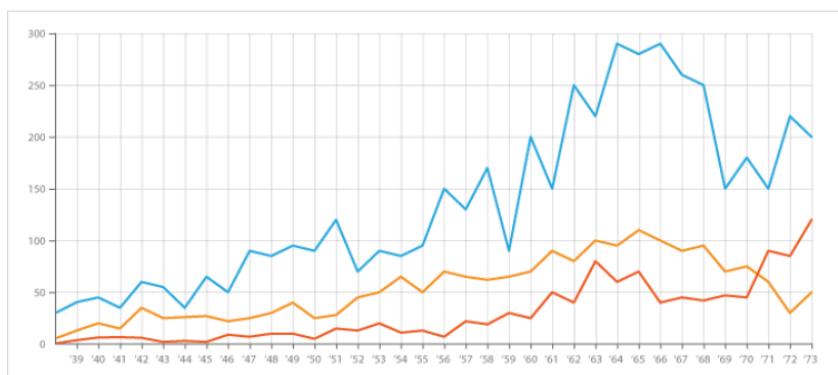
- Venn Diagram



- เรียกอีกชื่อว่า Set Diagram
- แสดงความสัมพันธ์ระหว่างกลุ่ม
- ส่วนที่ซ้อนกัน (Intersection) คือส่วนที่มี\dataต้าเหมือนกัน
- สามารถมี\dataต้าเชทได้ตั้งแต่ 2, 3, 4, 5, 6 และ 7 โดยที่ความซับซ้อนจะเพิ่มขึ้นตามลำดับ

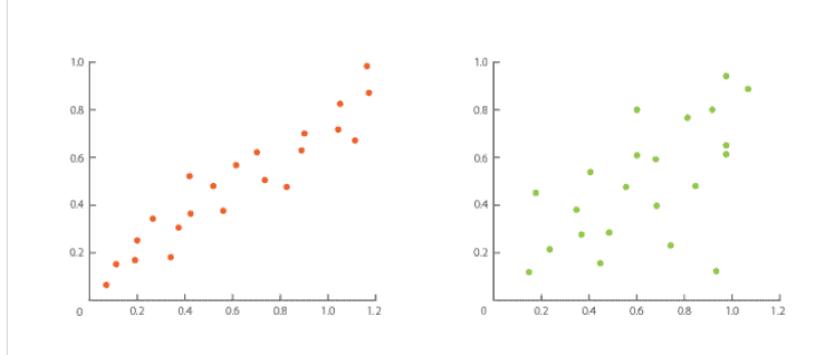
★ 2 variable

- Line Chart



- เป็นพLOTที่โชว์ค่าเชิงปริมาณของค่าที่เป็นลำดับ หรือถูกเรียงไว้อยู่แล้ว
- แสดงแนวโน้ม หรือ ความคืบหน้าของ variable ตามเวลา
- แสดง\dataต้าได้หลายกลุ่มในกราฟเดียว
- ควรใช้เมื่odataต้าที่เป็น continuous

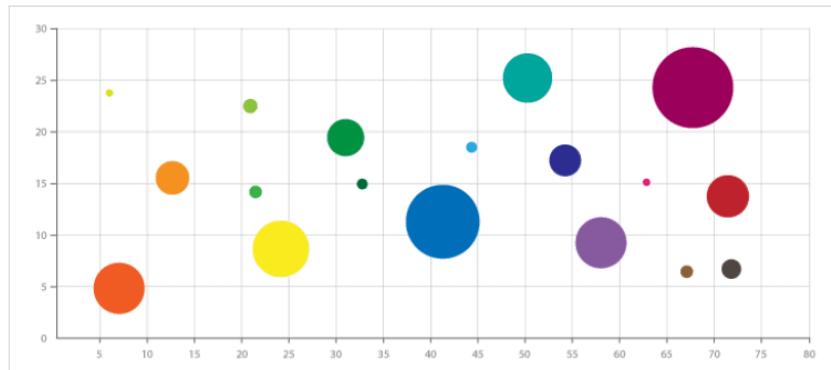
- Scatter Plot



- แสดงความสัมพันธ์ระหว่างตัวแปรสองตัวแปร ว่ามีลักษณะเป็นเส้นตรง (Linear) หรือว่าเป็นเส้นโค้ง (Curvilinear)
- แสดงสหสัมพันธ์ (Correlation) ระหว่างตัวแปร ว่าเป็นบวก ลบ หรือ ไม่มีสหสัมพันธ์
- แสดงค่า extreme หรือ ค่าที่มีโอกาสเป็น outlier

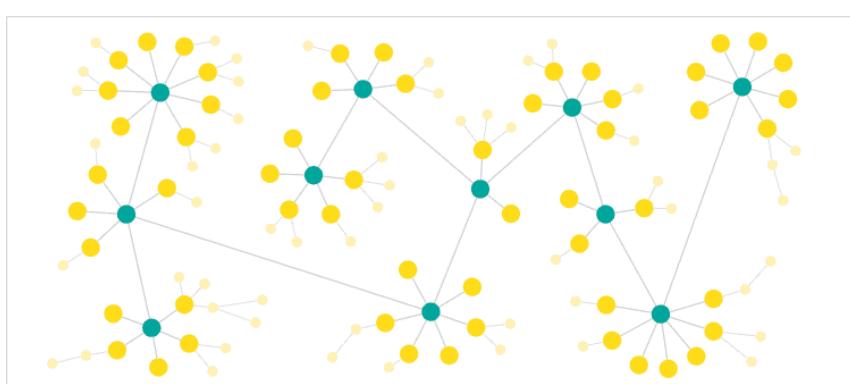
★ > 3 variable

- Bubble Chart



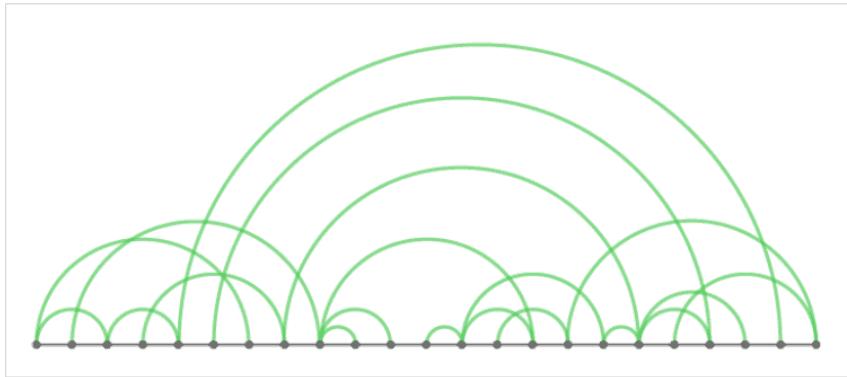
- เป็นผลลัพธ์ที่มีหน้าตาคล้าย Scatter Plot
- แสดงความสัมพันธ์ระหว่าง variable โดยเปรียบเทียบความสัมพันธ์ สัดส่วน และ correlation
- มีแกน x แกน y สำหรับสอง variable แรก มีขนาดของ bubble สำหรับ variable ที่สาม อีกทั้งยังใช้สีบ่งบอกประเภทของvariable ที่สี่ และก็สามารถทำให้ bubble chart เคลื่อนไหวตามเวลาได้

- Network Diagram



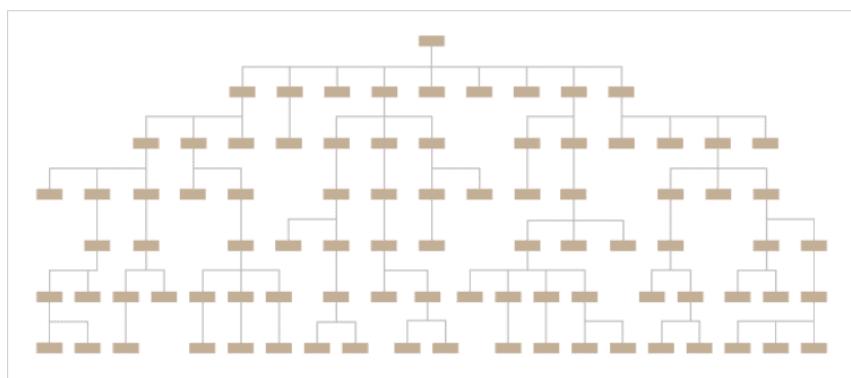
- เรียกอีกชื่อว่า Network Graph, Network Map หรือ Node-Link Diagram
- แสดงความสัมพันธ์ว่าแต่ละส่วนเชื่อมต่อกันอย่างไร ผ่าน node หรือ vertices และ link
- มีจุดหรือวงกลมเล็กๆเป็น node และมีเส้นเชื่อมต่อกัน (Link) แสดงความสัมพันธ์ระหว่าง node
- ขนาดและความยาวของเส้นเชื่อม อาจไม่เท่ากันเสมอไปขึ้นอยู่กับน้ำหนักที่ถูกตั้งไว้

- Arc Diagram



- เป็นผลลัพธ์ที่ทำหน้าที่คล้าย Network Diagrams 2 มิติ
- จุดที่อยู่ระหว่างเส้นเรียกว่า node เส้นที่อยู่ระหว่าง node เรียกว่า arc โดยที่ความหนาของ arc เส้นบ่งบอกความถี่ระหว่าง node แรกกับ node ที่สอง

- Tree Diagram



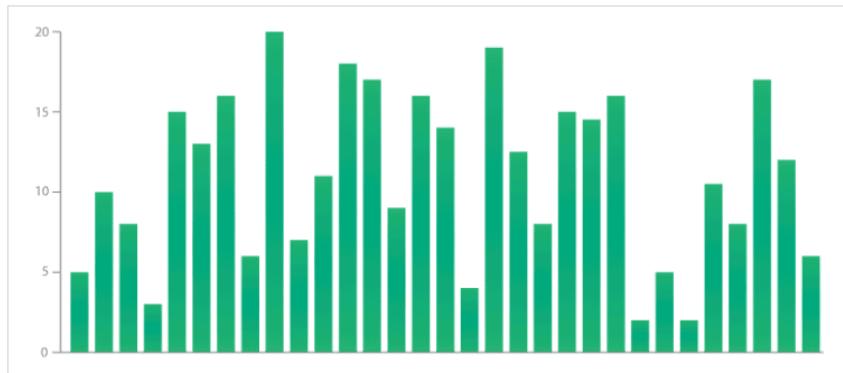
- เรียกอีกชื่อว่า Organisational Chart หรือ Linkage Tree
- เป็นผลลัพธ์แบบลำดับชั้น (Hierarchy) ที่เห็นง่าย และเข้าใจได้ง่ายที่สุดผลลัพธ์หนึ่ง
- มีส่วนบนสุดเป็น root node แต่ตัวเป็น node ໄล่ลงมาจนถึง leaf node (ไม่มีการแตกตัวต่อ) เส้นที่เชื่อมกันแสดงความสัมพันธ์เรียกว่า branch
- มากใช้แสดงความล้มเหลวในครอบครัว แผนผังองค์กร หรือแผนผังสปีชีส์ต่างๆ

- plot เพื่อเปรียบเทียบค่า

ในกรณีที่เราอยากรู้ว่าค่าใดๆ trend ความแตกต่าง หรือการเปลี่ยนแปลง ว่ามีค่ามากน้อย เพิ่มลดตรงไหน สามารถทำได้โดย

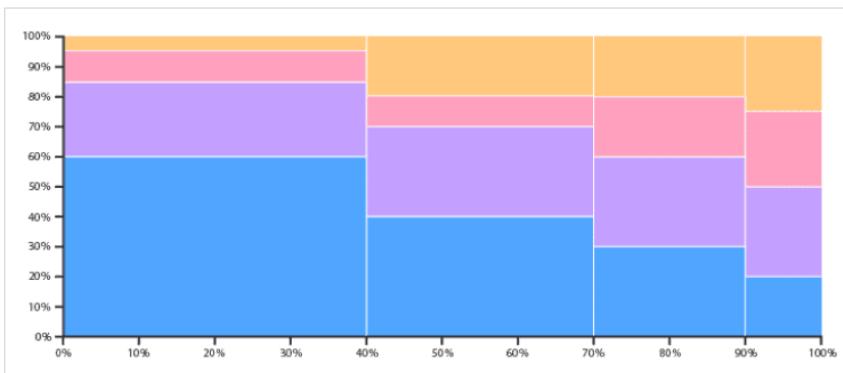
★ ดูความแตกต่างระหว่างกลุ่ม

- Bar Chart/Column Chart



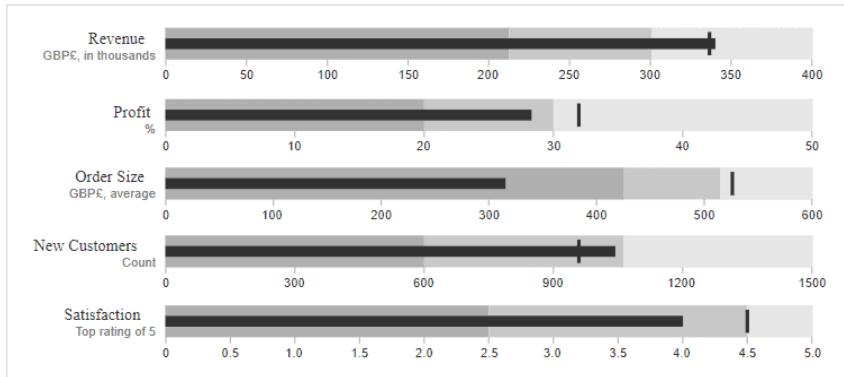
- เป็นพล๊อตที่ง่ายต่อการอ่าน และการเปรียบเทียบให้เห็นภาพมาก
- แสดงค่าเชิงปริมาณของกลุ่มนั้นๆ
- ไม่ควรใช้เมื่อมีอิฐเมืองในการเปรียบเทียบยอดเกินไป เพราะจะยากในการอ่านชื่อ (Label)
- สามารถแสดงค่าติดลบได้

- Mekko Chart/Marimekko Chart



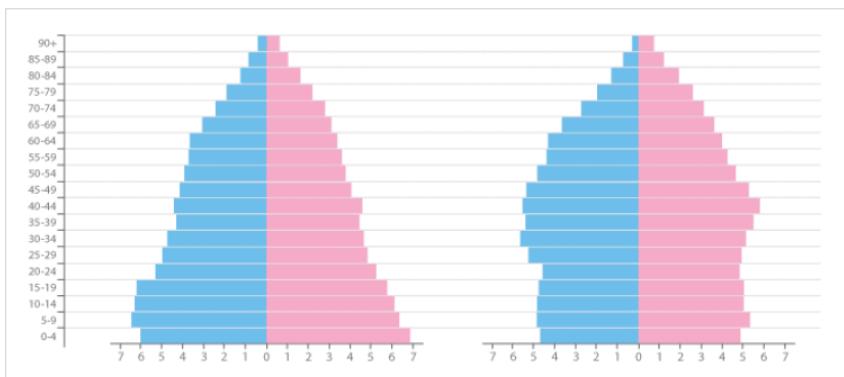
- เรียกอีกชื่อว่า Mosaic Plot
- เป็นพล๊อตที่เปรียบเทียบค่า แสดง composition และสัดส่วนของdata แต่ละกลุ่ม (Categorical Data)
- แกนแต่ละแกนแสดงค่าในรูปแบบเบอร์เช็นต์ รวมถึงความกว้างก็แสดงอุณหภูมตามเบอร์เช็นต์ นั้นด้วย
- ข้อเสียอย่างหนึ่งคือ อาจจะยากที่จะดูกราฟเมื่อมีกลุ่มหลายกลุ่ม

- Bullet Graph



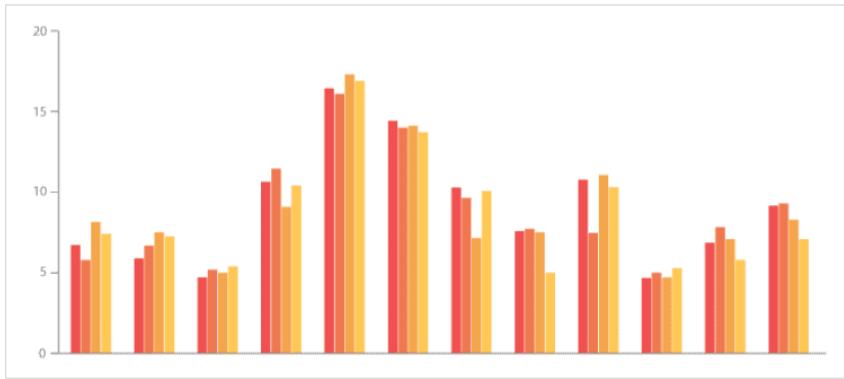
- เป็นผลลัพธ์ที่มีตัวชี้วัดได้
- มักจะใช้แสดง rating หรือ performance
- เส้นหนาที่อยู่ต่ำลงมาเรียกว่า Feature Measure
- เส้นที่อยู่ในแนวตั้งเรียกว่า Comparative Measure เป็นจุดที่บอกว่าเราถึงเป้าหมายหรือไม่
- ถ้าค่า Feature Measure เลยจุดที่ตั้งไว้ก็แปลว่าเราถึงเป้าหมายนั้นเอง

- Population Pyramid



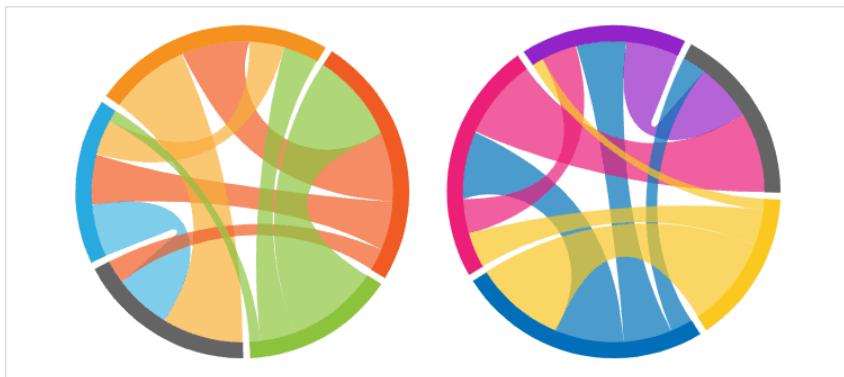
- เรียกอีกชื่อว่า Age & Sex Pyramid
- แสดงการแยกแยะของประชากรโดยแบ่งตามเพศ และอายุ
- แกน x เป็นจำนวนประชากร ส่วนแกน y เป็นกลุ่มอายุ
- สามารถเห็นการเปลี่ยนแปลง หรือแบบแผนของค่าต่างๆ ของประชากรจากหลายประเทศทั่วโลกได้
- มักจะใช้ในนิเวศวิทยา สังคมวิทยา และเศรษฐศาสตร์

- *Multi-set Bar Chart*



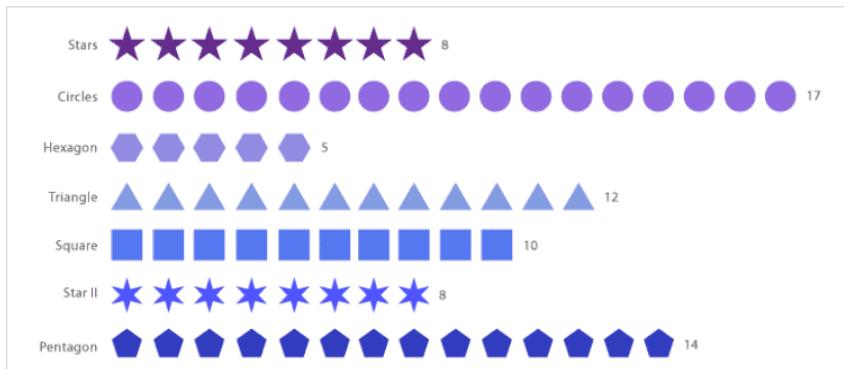
- เรียกอีกชื่อว่า Grouped Bar Chart หรือ Clustered Bar Chart
- เป็นพล็อตแบบหนึ่งของ Bar Chart ใช้มีเดาต้าสองชุดขึ้นไป
- แสดงค่าเชิงปริมาณของแต่ละกลุ่ม สามารถเปรียบเทียบค่าของกลุ่มนั้นๆได้ และยังแยกกลุ่มได้โดยใช้สีโคนต่างๆ

- *Chord Diagram*



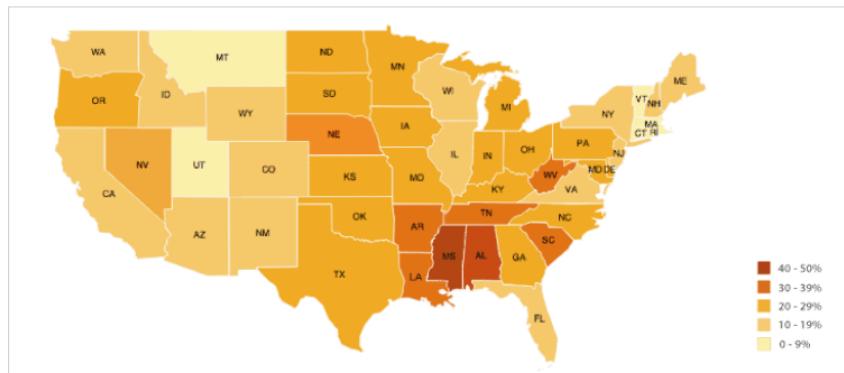
- เป็นพล็อตที่แสดงความสัมพันธ์ระหว่างกัน (Inter-Relationship)
- เปรียบเทียบความเหมือนในดาต้าเซท หรือระหว่างกลุ่มในดาต้าเซท
- ความหนาของ เส้นที่เชื่อม (Arc) จะขึ้นอยู่กับน้ำหนักที่ให้มา โดยมีสีที่ต่างกันในการบอกกลุ่ม ขอบของวงกลมแสดงกลุ่มแต่ละกลุ่มนั้นเรียกว่า node

- *Pictogram Chart*



- เรียกอีกชื่อว่า Pictograph Chart, Pictorial Chart, Pictorial Unit Chart หรือ Picture Graph
- แสดงจำนวนโดยใช้ไอคอนแทนปริมาณ เช่น ให้หนึ่งไอคอนแทนคน 10 คน
- ง่ายต่อการเข้าใจสำหรับทุกคน

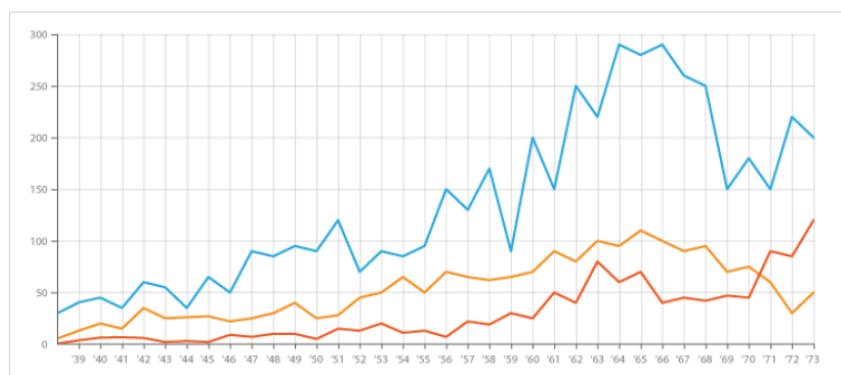
- *Choropleth Map*



- เป็นพلوตในแบบแผนที่ที่แบ่งแต่ละพื้นที่ตามเขต
- ความเข้มของสีขึ้นอยู่กับค่าเชิงปริมาณของ variable ที่เราสนใจ
- สามารถมองภาพรวมของแต่ละพื้นที่ได้ในพلوตเดียว
- มากใช้กับ raw ดาต้า เช่น จำนวนประชากร มากกว่าที่จะใช้กับดาต้าที่แปลงค่ามาแล้ว เช่น จำนวนประชากรต่อพื้นที่หนึ่งกิโลเมตร

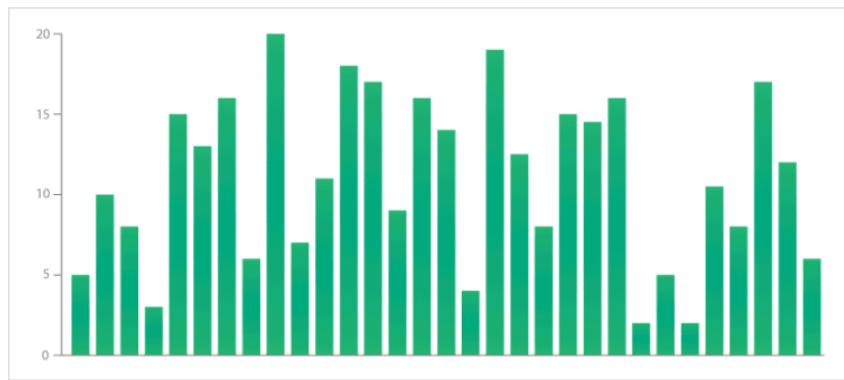
★ ดูการเปลี่ยนแปลงตามเวลา

- *Line Chart*



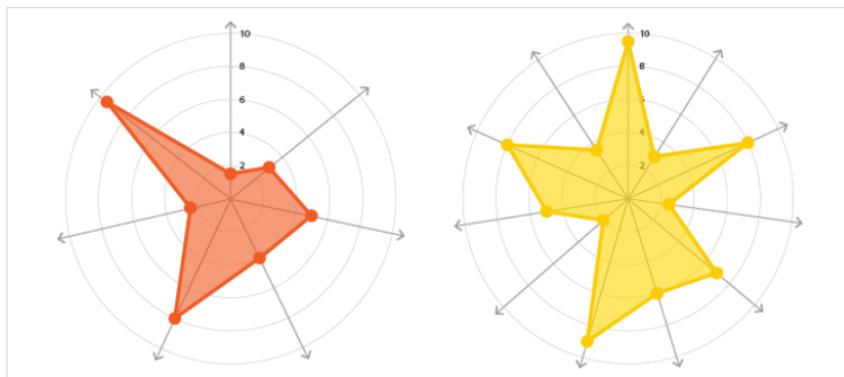
- เป็นพلوตที่โชว์ค่าเชิงปริมาณของค่าที่เป็นลำดับ หรือถูกเรียงไว้อยู่แล้ว
- แสดงแนวโน้ม หรือ ความคืบหน้าของ variable ตามเวลา
- แสดงดาต้าได้หลายกลุ่มในกราฟเดียว
- ควรใช้เมื่อดาต้าที่เป็น continuous

- Column Chart



- Bar Chart แบบแนวตั้ง
- เป็นผลลัพธ์หนึ่งที่ใช้ดูตามเวลาได้ โดยให้แกน x เป็นช่วงเวลา แกน y เป็นค่าที่เราสนใจเชิงปริมาณ

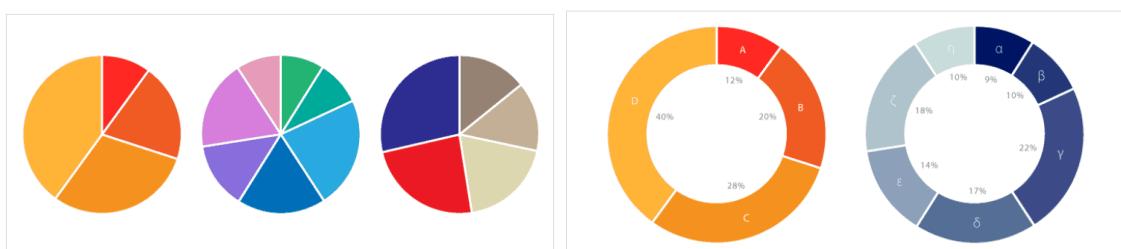
- Radar Chart



- เรียกอีกชื่อว่า Circular Area Chart, Spider Chart, Web Chart, Polar Chart หรือ Star Chart
- เปรียบเทียบค่าเชิงปริมาณของหลาย variable ได้ โดยให้มุมแต่ละมุมแทน variable และมุม
- สามารถเห็น outlier ได้
- มากใช้ในการแสดง performance

- plot เพื่อดูส่วนประกอบของ data นั้น ๆ
★ ไม่เปลี่ยนแปลงตามเวลา

- Pie Chart/Doughnuts Chart



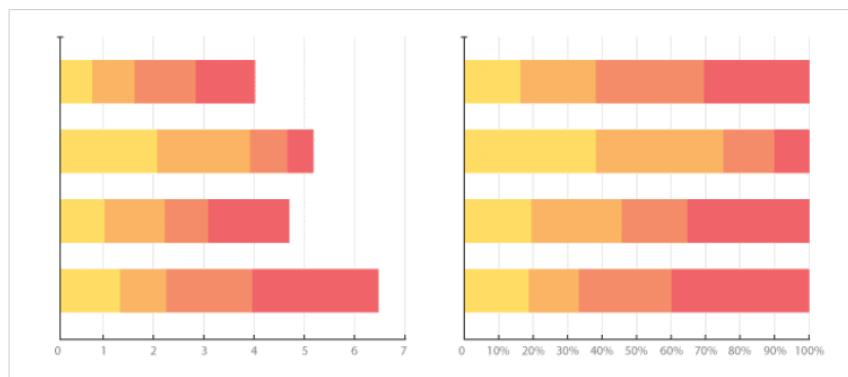
- เป็นผลลัพธ์ที่เป็นที่นิยมของไดรฟ์หลายๆ คน
- แสดงสัดส่วนของแต่ละไอเทมในกลุ่ม โดยที่แต่ละ slice แสดงสัดส่วนเป็นเปอร์เซนต์
- มักถูกใช้แสดงรายได้ของแต่ละคนเดียว แต่ตัวของคนเราปกติจะแยกขนาดของ slice ขนาดใกล้เคียงกันได้ยาก
- ในกรณีที่มีส่วนเล็กๆ จะยากต่อการจับคู่ label

- *Waterfall Chart*



- เป็นผลลัพธ์ที่แสดง composition
- แสดงค่าตั้งต้น การเปลี่ยนแปลงไม่ว่าจะเป็นบวก หรือลบจากค่าตั้งต้น และค่าสุดท้าย (ผลลัพธ์)
- ยกตัวอย่าง การแสดงรายได้ของบริษัท และค่าใช้จ่ายในส่วนต่างๆ รวมกันเป็นกำไร หรือขาดทุน

- *Stacked Bar Chart*



- แสดงสัดส่วนที่อยู่ในกลุ่มของแต่ละแท่ง
- มีสองแบบ Simple Stacked Bar Graphs ซึ่งอยู่ทางซ้าย และ 100% Stack Bar Graphs ซึ่งอยู่ทางขวาในรูป
- Simple Stacked Bar Graphs เป็นการบวกค่าเพิ่มขึ้นไปเรื่อยๆ หมายความว่าการเปรียบเทียบค่าทั้งหมดของแต่ละแท่ง
- 100% Stack Bar Graphs แสดงออกโดยคิดสัดส่วนออกมาเป็นเปอร์เซนต์ของแต่ละกลุ่ม

- Tree Map



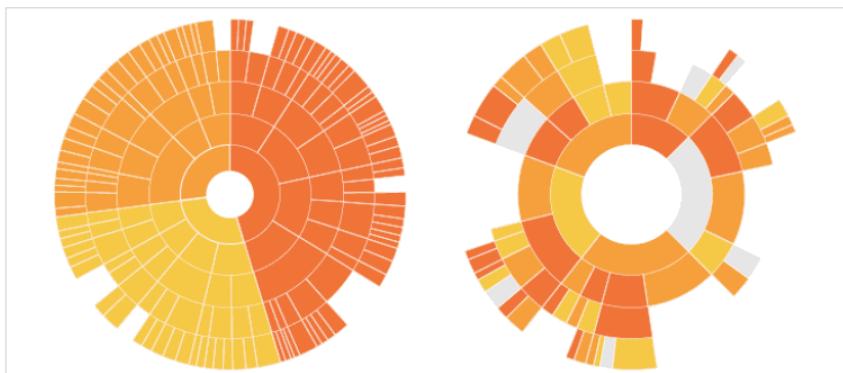
- เป็นผลลัพธ์ที่ทำหน้าที่คล้าย Tree Diagram แต่แสดงค่าเชิงปริมาณเข้ามาด้วยโดยใช้ขนาดเป็นตัวกำหนด
 - กลุ่มแทรลักษณะจะบอกปริมาณได้ด้วยขนาด เช่น เติมวัสดุกลุ่มอยู่ที่มีขนาดเล็กลงไปในกลุ่มนั้นๆ
 - ในการณ์ที่ไม่รู้ขนาด พื้นที่ที่เหลือจะถูกแบ่งให้เท่ากันในกลุ่มอยู่

- Word Cloud



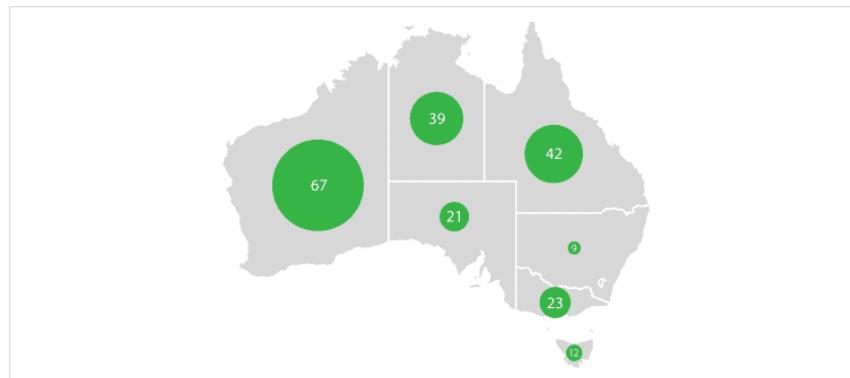
- เรียกอีกชื่อว่า Tag Cloud
 - เป็นการแสดง key word หรือความถี่ของคำที่ปรากฏ
 - ยิ่งมีจำนวนความถี่มากเท่าไร ขนาดของคำนั้นก็จะยิ่งใหญ่ขึ้นเท่านั้น
 - สามารถทำเป็นรูปร่างต่างๆได้ เช่น คอลัมน์ หรือแบบวง

- *Sunburst Diagram*



- เรียกอีกชื่อว่า Sunburst Chart, Ring Chart, Multi-level Pie Chart, Belt Chart หรือ Radial Treemap
- เป็นพLOTที่ทำหน้าที่คล้าย Tree Diagram
- วงแหวนจะเรียงลำดับ โดยนับจากชั้นในสุดเป็นกลุ่มใหญ่สุด และย่ออยลงมาเป็นวงแหวนชั้นต่อๆ ไปในกลุ่มนั้นๆ จนถึงชั้นสุดท้ายที่ไม่แตกตัวต่อไป

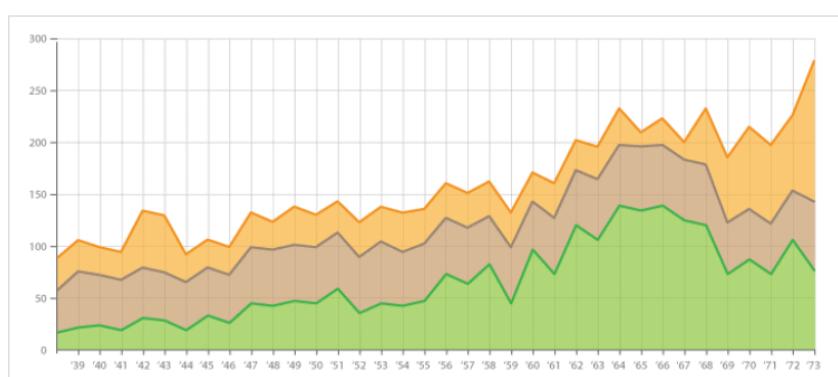
- *Bubble Map*



- เป็นพLOTในแบบแผนที่ที่แสดงจำนวน หรือค่าเชิงปริมาณเป็นตัวเลขใน bubble
- ขนาดของ bubble จะขึ้นอยู่กับค่านั้นๆ
- ใช้ได้ดีในการเปรียบเทียบสัดส่วนในแต่ละเขตคล้ายกับ Choropleth Map
- ข้อเสียคือขนาดของ bubble อาจใหญ่เกินพื้นที่ได้

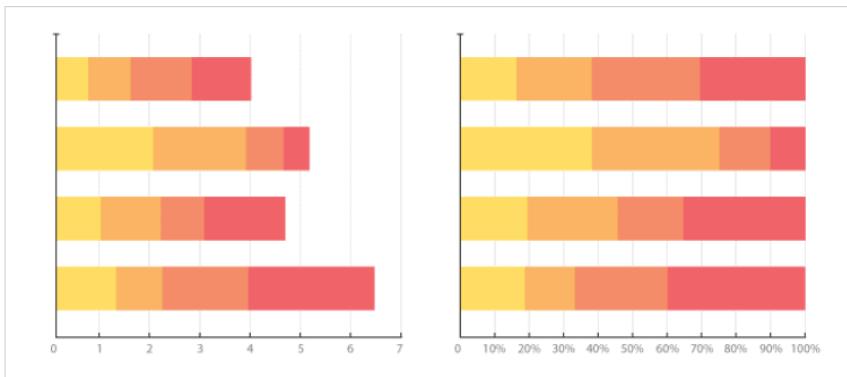
★ เปลี่ยนแปลงตามเวลา

- *Stacked Area Chart*

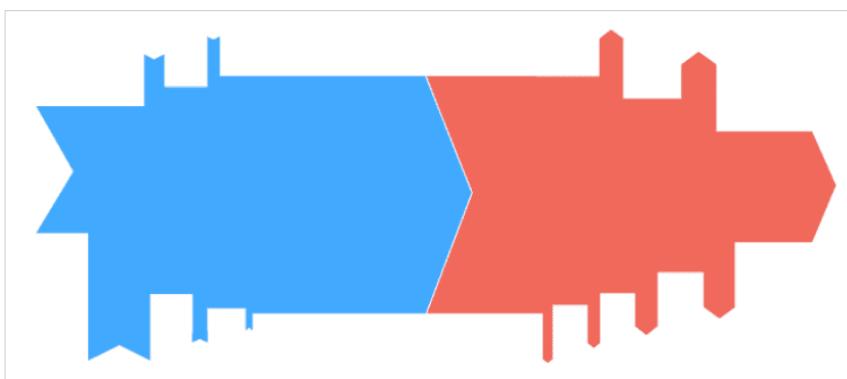


- พLOTที่คล้ายกับ Line Chart โดยใช้สีให้พื้นที่ใต้เส้น
- แสดง DATAได้หลายกลุ่มในกราฟเดียว
- แสดงค่าเชิงปริมาณทั้งหมดในแกน x ตามเวลาในแกน y โดยที่ไม่สามารถแสดงค่าที่ติดลบได้

- *Stacked Bar Chart*



- เป็นพلوตที่แสดงสัดส่วนของไอเทมหลายไอเทมไว้ในบาร์เดียว หรือ แสดง composition เพื่อเปรียบเทียบไอเทมในกลุ่มนั้นๆ
 - แสดงความสัมพันธ์แบบ part-to-whole
- plot เพื่อดูการเคลื่อนไหว
 - *Sankey Diagram*



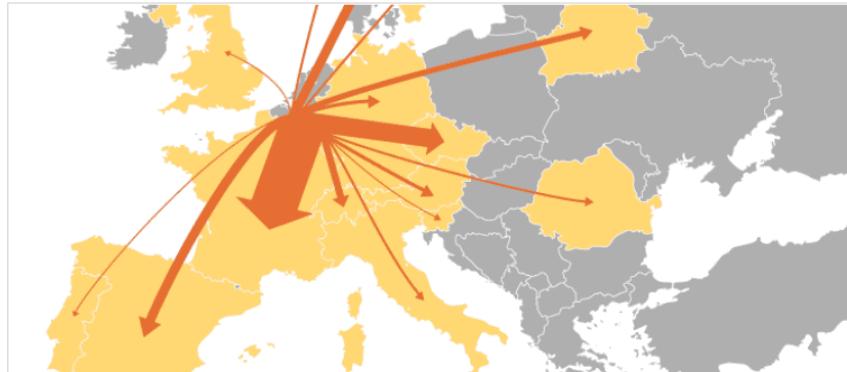
- แสดงปริมาณของลิ่งๆหนึ่งในสัดส่วนที่กำหนดไว้
- ยิ่งสัดส่วนใหญ่ ขนาดของແບກີຈະໃໝ່ໄປດ້ວຍ
- สีสามารถແປ່ງອາກນາເປັນກລຸ່ມຕ່າງໆໄດ້
- ມັກໃຊ້ໃນ ການໄລຂອງພລັງງານ ເງິນ ແລະ ລຶ່ງຕ່າງໆ

- *Connection Map*



- เรียกอีกชื่อว่า Link Map หรือ Ray Map
- การวัดผลต้นสามารถทำได้โดยใช้ทั้งเลนส์โค้ง หรือเลนส์ตรงเชื่อมระหว่างจุดก็ได้
- แสดงการเชื่อมกัน หรือความล้มเหลวระหว่างแต่ละพื้นที่
- สามารถใช้แสดงเส้นทางการบิน หรือแสดงแบบแผนต่างๆบนแผนที่

- Flow Map



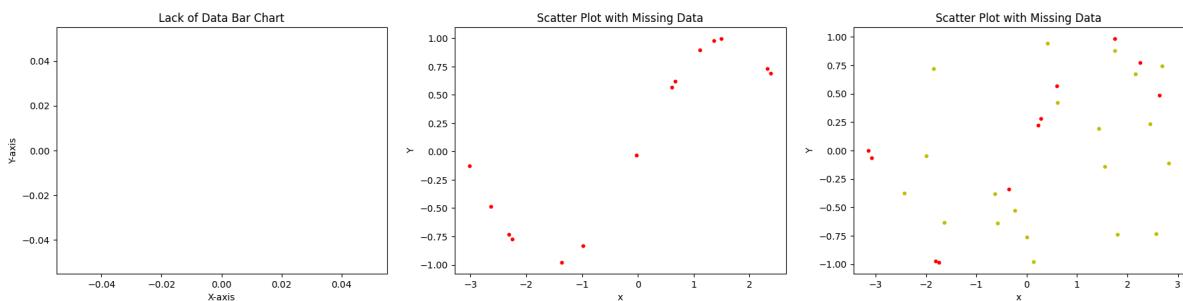
- เป็นแผนที่ที่แสดงการเคลื่อนไหวของข้อมูลจากจุดหนึ่งไปยังอีกจุดหนึ่งพร้อมค่าเชิงปริมาณ
- มากจะใช้กับการย้ายถิ่นฐานของผู้อพยพ ให้รู้จะเป็นมนุษย์ หรือสัตว์ และก็ใช้ได้กับการขยายลิงของด้วย

Precautions

ข้อควรระวังที่อาจจะเกิดขึ้นได้ในการทำ Data Visualization ในหัวข้อนี้เราจะยกตัวอย่างมา 4 หัวข้อ ได้แก่ Lack of Data/Missing Data, Color/Contrast, Opacity, Misleading Scaling

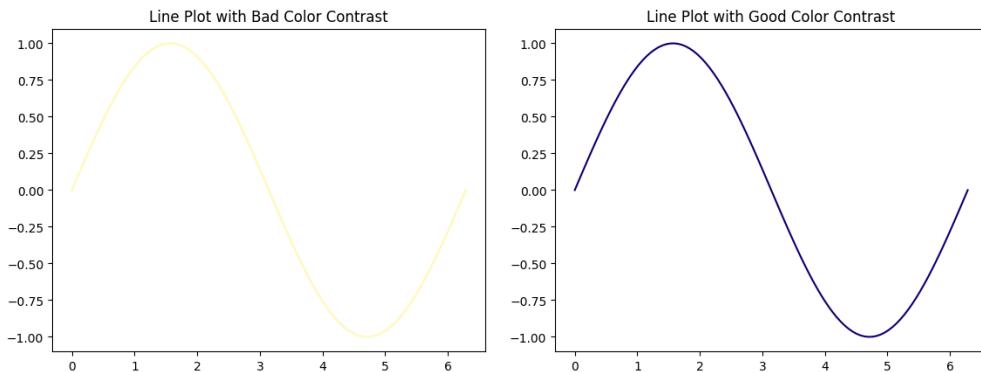
- Lack of Data/Missing Data

ข้อนี้คือการที่ข้อมูลของเรามากไป อาจจะหายไปทั้งหมด หรือเพียงบางส่วนก็ได้ อาจเกิดจากการที่เราจัดการทำความสะอาดข้อมูลไม่ดี หรือบางชุดข้อมูลอาจจะเก็บมาไม่ครบแล้วเราอาจจะลองทำนายดูได้แต่ก็ต้องระวังว่าข้อมูลจริง ๆ ไม่ได้เป็นอย่างที่เราทำนาย



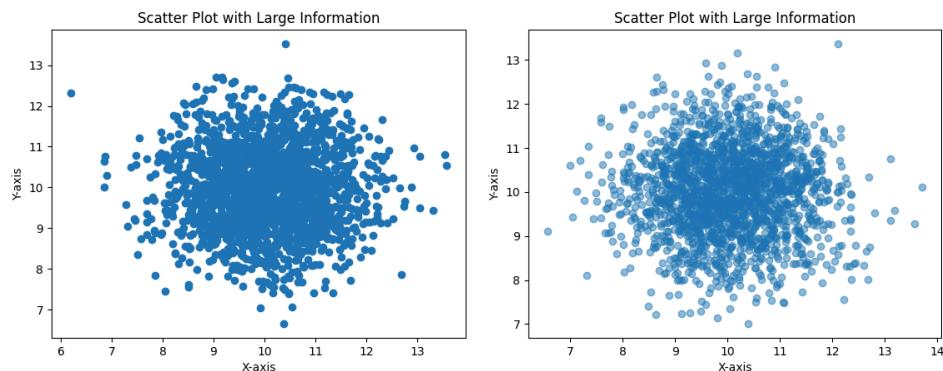
○ Color/Contrast

ข้อนี้เกี่ยวกับการใช้สีในการนำเสนอ เรายังต้องพยายามทำให้กราฟของเรา มีความโดดเด่นของมาเพื่อให้เข้าใจกันได้ระหว่างผู้ส่งสารและผู้รับสาร



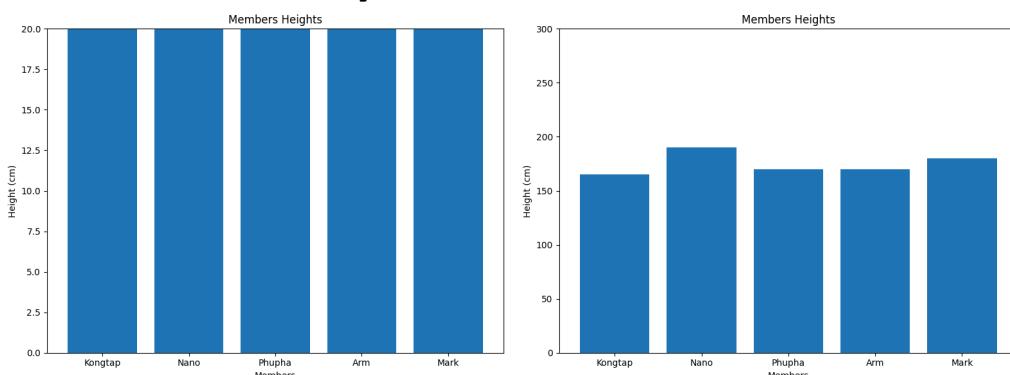
○ Opacity

ข้อนี้เราราจจะมีปัญหาเมื่อนำข้อมูลที่มีจำนวนมาก ๆ มา plot ซึ่งอาจจะทำให้เกิดหาร plot ในจุดที่ซ้ำกัน เราจำเป็นต้องกำหนด opacity ของจุดเพื่อแสดงให้เห็นถึงความหนาแน่นของข้อมูล



○ Misleading Scaling

ข้อนี้เราระวังการกำหนดช่วงของแกน y เพราะหากเรากำหนดช่วงแกนต่ำหรือสูงเกินไปจะทำให้เราไม่สามารถรู้ถึงความแตกต่างของแต่ละ component ได้เลย ปัญหานี้แก้ได้ง่าย ๆ ด้วยการกำหนดช่วงของแกน y ให้เหมาะสมกับช่วงของข้อมูล



Case Study

○ Dataset

Dataset ชุดนี้ (<https://kmutt.me/pkmds>) ประกอบไปด้วยข้อมูลของ Pokemon 802 ตัว จากทั้งหมด 7 Generation และถูก scrape มาจาก (<http://serebii.net/>)

Content

- **name:** ชื่อภาษาอังกฤษของ Pokemon
- **japanese_name:** ชื่อภาษาญี่ปุ่นของ Pokemon
- **pokedex_number:** entry number ของ Pokemon ใน National Pokédex
- **percentage_male:** เปอร์เซ็นต์ของสายพันธุ์ที่เป็นเพศชาย
จะว่างเปล่าหาก Pokemon ไม่มีเพศ
- **type1:** The Primary Type of the Pokemon
- **type2:** The Secondary Type of the Pokemon
- **classification:** การจำแนกประเภทของ Pokemon ตาม the Sun and Moon Pokédex
- **height_m:** ความสูงของ Pokemon หน่วยเป็นเมตร
- **weight_kg:** น้ำหนักของ Pokemon เป็นกิโลกรัม
- **capture_rate:** อัตราการจับ Pokemon
- **base_egg_steps:** จำนวนขั้นตอนที่ต้องใช้ในการฟักไข่ Pokemon
- **abilities:** ความสามารถที่ Pokemon สามารถมีได้
- **experience_growth:** การเติบโตของประสบการณ์ของ Pokemon
- **base_happiness:** ความสุขพื้นฐานของ Pokemon
- **against_?:** คุณสมบัติ 18 ประการที่แสดงถึงจำนวนความเสียหายที่ได้รับจากการโจมตีประเภทใดประเภทหนึ่ง
- **hp:** HP พื้นฐานของ Pokemon
- **attack:** การโจมตีพื้นฐานของ Pokemon
- **defense:** การป้องกันพื้นฐานของ Pokemon
- **sp_attack:** การโจมตีพิเศษพื้นฐานของ Pokemon
- **sp_defense:** การป้องกันพิเศษพื้นฐานของ Pokemon
- **speed:** ความเร็วพื้นฐานของ Pokemon
- **generation:** รุ่นหมายเลขอารบิกที่ Pokemon เปิดตัวครั้งแรก
- **isLegendary:** ระบุว่า Pokemon เป็นตำนานหรือไม่

○ แนวทางการทำ Data Visualization

- ★ ความเร็วของ Pokemon สัมพันธ์กับปัจจัยพื้นฐานต่าง ๆ อย่างไร?
- ★ แต่ละ Generation มี Pokemon อยู่เท่าไหร่?
- ★ Pokemon แต่ละ Generation มีกี่ชนิด?
- ★ Pokemon ประเภทใดที่แพร่หลายมากที่สุดทั้งประเภท primary และ secondary?
- ★ ส่วนสูงและน้ำหนักของ Pokemon สัมพันธ์กับค่าพลังพื้นฐานต่าง ๆ อย่างไร?
- ★ เราจะหา Pokemon ที่แข็งแกร่งที่สุดได้หรือไม่?

เข้าลิ้งก์นี้ <https://kmutt.me/DVLab> เพื่อทำ Lab

Books

DATA + DESIGN

หนังสือ Data + Design เขียนโดยผู้เชี่ยวชาญกว่า 50 ท่านใน 14 ประเทศทั่วโลก โดยสอนทำ Data Visualization ตั้งแต่ต้นจนจบ นั่นคือ :

1. Collection – วิธีการเก็บข้อมูล, เอาข้อมูลมาจากไหนได้บ้าง, ต้องทำ Survey เก็บข้อมูลต้องทำยังไง, แจกแหล่งรวมข้อมูลฟรีที่เรานำไปใช้ได้ (Public Data)
2. Preparing – การเตรียมข้อมูลก่อนนำไป Visualize มีขั้นตอนอย่างไรบ้าง, การทำ Data Cleaning, การทำ Data Transformation
3. Visualize – นำข้อมูลที่เตรียมไว้แล้วมาแสดงผล, เราจะแสดงผลอย่างไร, การเลือก Font/สี/ไอคอน มีผลอย่างไร

เนื้อหาในเล่มนี้อ่านเข้าใจง่าย มีตัวอย่างประกอบ และใช้คำที่ไม่ยากจนเกินไป เข้าไปอ่านกันได้ที่นี่เลย : <https://kmutt.me/DVbook1>

Fundamental of Data Visualization

หนังสือเล่มนี้ทั้งเล่มเขียนด้วย R Markdown แปลว่าทุก plot ที่เราเห็นในเล่ม เราสามารถไปเอาโค้ด R (ggplot) เด้ามาใช้ได้ฟรีเลย จาก Github ของหนังสือเล่มนี้ (<https://kmutt.me/DVref2>) แต่คนเขียนเดาเนินย้ำว่าทุกภาระในนี้เขาไปปรับใช้กับการนำเสนอข้อมูลด้วยเครื่องมือของ R ก็ได้

เนื้อหาในหนังสือเล่มนี้แบ่งเป็น 3 ส่วนหลัก ๆ :

1. From Data to Visualization – อธิบายว่าการนำเสนอแบบต่าง ๆ มีแบบไหนบ้าง และประเภทไหนเหมาะสมกับอะไร เช่น กราฟแบบบาร์, กราฟแบบจุด, อันใหญ่เหมาๆ กับข้อมูลที่เป็นตัวแปรต่อเนื่อง ฯลฯ เป็นต้น ซึ่งอันนี้ถือเป็นเรื่องสำคัญในศาสตร์ Data Visualization
2. Principles of Figure Design – ส่วนนี้จะแนะนำเกี่ยวกับการเลือกสี เลือกฟ้อนต์ เลือกเครื่องหมาย ในงานนำเสนอข้อมูล โดยจะเน้นเรื่องปัญหาต่าง ๆ ที่เจอบ่อย ๆ เวลาเราทำงานด้านนี้
3. Miscellaneous Topics – ส่วนนี้จะเป็นเรื่องอื่น ๆ ที่ควรรู้ แต่ไม่ได้อยู่ใน 2 หัวข้อข้างบน เช่น การเลือกโปรแกรมในการ Visualization เราต้องคำนึงถึงอะไรบ้าง?, หรือการเล่าเรื่องด้วยเนื้อหาต้องทำอย่างไร และสามารถเข้าไปอ่านกันได้ที่ <https://kmutt.me/DVbook2>

Reference

กิตติภณ พละการ, กิตติภพ พละการ, สมชาย ประลิทธิ์จูตระกูล และ สุรี สินธุกิจโภุ. ๒๕๑๐. **Python ๑๐๑.** [<https://www.cp.eng.chula.ac.th/books/python101/>].

Brain Code Camp. **Data Visualization.** [<https://course-braincodecamp.web.app/Fundamentals/Prereqs/BBB2BCC/DataVisualization.html>].

datath. **Cheatsheet Pandas.** [<https://blog.datath.com/cheatsheet-pandas>].

datath. **วิธีเลือก Chart ให้คนดูเข้าใจง่าย.** [<https://blog.datath.com/data-visualization-mindmap/>].

w3schools. **Matplotlib Tutorial.** [https://www.w3schools.com/python/matplotlib_intro.asp].

w3schools. **Python Data Types.** [https://www.w3schools.com/python/python_datatypes.asp].