
PYTHON
101
ALL YOU NEED IS CODE.
JO SRIYAPAN

PYTHON 101

ALL YOU NEED IS CODE.

JO SRIYAPAN

เผยแพร่ครั้งแรกประกอบการเรียนการสอนสำหรับ CoderDojo Thailand พ.ศ. 2562

จัดพิมพ์ออนไลน์ครั้งที่ 1 พ.ศ. 2564

ข้อมูลทางบรรณานุกรมของสำนักหอสมุดแห่งชาติ

จุมพฏ ศรียะพันธ์
PYTHON 101
เรียบเรียงครั้งที่ 1
CoderDojo Thailand
ISBN ––––––-

Jo Sriyapan
CoderDojo Thailand
www.coderdojo.in.th

แด่ ครอบครัวอันเป็นที่รักยิ่ง

PROLOGUE

ภาษา Python เป็นภาษาที่ได้รับความนิยมอย่างสูงทั่วไป โดยเฉพาะงานคำนวณทางคณิตศาสตร์ วิทยาศาสตร์ข้อมูล และปัญญาประดิษฐ์ รวมทั้งเป็นภาษาที่โปรแกรมเมอร์ หรือผู้สนใจการเขียนโปรแกรมคอมพิวเตอร์ใหม่ๆ นิยมเรียนรู้เป็นภาษาแรกๆ เพราะเรียนรู้ได้ง่าย เร็ว และสนุก ครอบคลุมรูปแบบของการเขียนภาษาคอมพิวเตอร์ระดับสูงมากมาย

หนังสือเล่มนี้ เขียนขึ้นเพื่อมุ่งหวังให้ผู้เรียนข้ามพรมแดนของความ รู้ และ ไม่รู้ ไม่ได้เน้นไปที่ตรรกะวิธีคิดคำนวณ การเรียนรู้จึงทำได้ง่ายมาก โดยเขียนโปรแกรมตามตัวอย่าง ในหนังสือไปตามลำดับ แม้จะเข้าใจบ้าง ไม่เข้าใจบ้าง แต่หากผู้เรียนได้ทดลองเขียนทดลอง ให้คอมพิวเตอร์ทำงานตามโปรแกรมในตัวอย่างที่ให้ไว้ได้โดยการพิมพ์ด้วยตนเองไปทีละตัวอย่าง ทีละแบบฝึกหัด เมื่อผ่านไประยะหนึ่ง ก็จะพบว่าตนเองมีความรู้เพิ่มขึ้น และเริ่มจะเข้าใจการเขียนโปรแกรมและการทำงานของภาษา Python ได้เอง

และสำหรับผู้ที่เคยเรียนรู้ภาษาอื่นมาแล้ว ยิ่งจะเป็นเรื่องที่ง่ายมาก เพราะตัวอย่างโปรแกรม ในหนังสือเล่มนี้รวบรวมไวยากรณ์ (Syntax) พื้นฐานของภาษา Python ในเบื้องต้นไว้แล้ว ทำให้สามารถต่อยอดการเขียนโปรแกรมภาษาใหม่ได้อย่างรวดเร็ว

ขอให้สนุกสนานกับการเขียนโปรแกรมครับ

Chapter 00

ติดตั้ง

การติดตั้งตัวแปรภาษา Python ในคอมพิวเตอร์

1. ไปที่เว็บ http://www.python.org

2. กดตรงที่เขียนว่า Download https://www.python.org/downloads/

3. เลือก Python เวอร์ชันที่ต้องการ ตามตัวอย่างนี้อ้างอิงกับ Python เวอร์ชัน 3.9.5 แต่คำสั่งส่วนมากสามารถใช้งานได้กับเวอร์ชัน 3.xx ทั่วไป

4. เลือกดาวน์โหลดตัวแปรภาษา Python ตามระบบปฏิบัติการที่ใช้อยู่ เช่น Windows, OSX, Linux

5. กดติดตั้งโปรแกรม จะมีช่องให้ติ๊ก Install launcher for all users (recommended) กับ Add Python 3.xx to PATH ให้ติ๊กเลือกทั้งหมด

6. พร้อมใช้งาน

7. การเรียกใช้งานผ่าน Windows ให้เรียกโปรแกรมชื่อ IDLE จะมีหน้าจอ Python Shell ขึ้นมาให้ใช้ สามารถเขียนโปรแกรมบนหน้านี้ได้เลย หรือเลือกเปิด editor โดยกดไปที่ File->New File จะมีหน้าจอ editor ขึ้นมาให้เขียนโปรแกรม

8. เมื่อเขียนโปรแกรมเสร็จ กด File->Save จากนั้นจะสามารถกด Run จากหน้าจอ editor ในเมนู Run->Run Module หรือกด F5 ได้เช่นกัน

9. หรือสามารถเลือกใช้ editor ตัวอื่นๆ ได้ ที่นิยมกันมากคือ vscode ของ Microsoft ซึ่งมีให้ download ได้ที่ https://code.visualstudio.com/download จากนั้นก็ติดตั้ง Python extension ก็จะสามารถใช้งานกับภาษา Python ได้

10. การติดตั้งซอฟต์แวร์ และการใช้ editor มีเขียนในหนังสือหลายเล่ม หรือสามารถหาในเน็ตหรือถามผู้รู้ได้ทั่วไป จึงไม่ลงรายละเอียดไว้ในนี้นะครับ

11. หรือสามารถใช้ Python editor สำเร็จรูป เช่น Mu Editor https://codewith.mu/ ก็ใช้งานง่ายดีสำหรับผู้เริ่มต้น และสามารถใช้กับตัวอย่างในนี้ได้ทั้งหมด

หนังสือเล่มนี้ รวบรวมคำสั่งเบื้องต้นเพื่อให้เข้าใจพื้นฐานการเขียนโปรแกรมด้วยภาษา Python เท่าที่จำเป็นไว้ ซึ่งเมื่อได้ทดลองเขียนโปรแกรมตามตัวอย่างจนหมดเล่ม เราจะมีคลังคำสั่งมากพอให้สามารถเขียนโปรแกรมประยุกต์ได้ประมาณหนึ่งแล้ว

ภาษา Python เป็นภาษาคอมพิวเตอร์ระดับสูง หรือเรียกว่า High-level Computer Language ซึ่งบางคนจัดกลุ่มให้ละเอียดลงไปอีกเป็น Scripting Language เพราะสามารถทำงานทีละคำสั่งได้โดยตรงผ่านตัวแปลภาษาที่เรียกว่า Interpreter โดยไม่จำเป็นต้องใช้ตัวแปลภาษาแบบที่ต้องแปลทั้งโปรแกรมในทีเดียว (Compiler) แต่โดยทั่วๆ ไปเราก็เรียกภาษา Python นี้เป็น Programming Langauge ชนิดหนึ่ง

ในปัจจุบัน ภาษา Python มีการใช้งานหลากหลายรูปแบบ ทั้งในงานของโปรแกรมเมอร์ รวมไปถึงสายงานอื่นๆ เช่นวิทยาศาสตร์ คณิตศาสตร์ วิศวกร นักการเงิน และอีกหลายๆ นัก ก็ถือว่ามีการนำไปใช้งานอย่างกว้างขวาง รวมทั้งเป็นภาษาที่เรียนรู้ไม่ยากและสนุก เหมาะกับการเป็นภาษาคอมพิวเตอร์สำหรับผู้เริ่มต้นได้ด้วย

Chapter11

HELLO, WORLD!

เริ่มเขียนโปแกรมภาษา Python กับ Hello,World!

1.1.1 hello, world! คำสั่ง print และเครื่องหมาย #
1.1.2 สวัสดี ชาวโลก
1.1.3 เขียนโปรแกรมให้เป็นระเบียบด้วยฟังก์ชัน
1.1.4 docstring

```
#Example 1.1.1
#Python 3.9.5

print ("hello, world!”)
```

```
hello, world!
```

ตัวอย่าง 1.1.1

การเขียนโปรแกรมคอมพิวเตอร์ หรือเรียนรู้ภาษาใหม่ๆ โปรแกรมที่นิยมเขียนกันเป็นโปรแกรมแรก หรือเรียกว่าเป็นท่าไหว้ครูสำหรับคนที่หัดเขียนโปรแกรม ไม่ว่าภาษาอะไร ก็มักจะเริ่มจากโปรแกรม hello, world ซึ่งเป็นการสั่งให้คอมพิวเตอร์แสดงคำว่า hello, world บนหน้าจอ

สำหรับภาษา Python เราสามารถเขียนโค้ดง่ายๆ คือ

```
print(“hello, world!”)
```

สังเกตหัวโปรแกรม สองบรรทัดแรก ที่เขียนไว้ว่า

```
#Python 3.7.3
```

และ

```
#Example 1-1-1
```

เครื่องหมาย # เรียกว่า hash (แฮช) หรือบางคนเรียกว่าเครื่องหมาย sharp (ชาร์ป) หรือบนแป้นกดของโทรศัพท์ บางคนก็เรียกว่าเครื่องหมายสี่เหลี่ยม ในภาษา Python ถ้าเจอเครื่องหมาย hash นี้อยู่หน้าบรรทัดไหน จะเรียกบรรทัดนั้นว่า comment (คอมเมนต์) คือเขียนไว้เพื่อบันทึกเฉยๆ ตัวแปรภาษา Python จะอ่านข้ามบรรทัดนั้นไปไม่ทำงาน มีประโยชน์คือใช้เขียนบันทึกแทรกไว้กับ code โปรแกรม หรือบางทีเวลาทดสอบโปรแกรมแล้วเราไม่อยากให้คำสั่งไหนหรือบรรทัดไหนทำงาน วิธีง่ายๆ ก็คือเอา # ไปวางไว้หน้าบรรทัดนั้น

ส่วนคำสั่งที่คอมพิวเตอร์เริ่มทำงานจริงๆ คือ

```
print (“hello, world!”)
```

ซึ่งจะเป็นการสั่งให้คอมพิวเตอร์แสดงคำว่า hello, world! บนหน้าจอ

```
#Example 1.1.2
#Python 3.9.5

print ("สวัสดี ชาวโลก")
```

```
สวัสดีชาวโลก
```

ตัวอย่าง 1.1.2

hello,world! ภาษาอังกฤษไปแล้ว ให้แสดงผลเป็นภาษาไทยบ้างก็ได้

TIP
เท่าที่สืบค้นกันมาได้ เล่ากันว่าบุคคลแรกที่ใช้ *hello, world* เป็นตัวอย่างในการเขียนโปรแกรมอย่างเป็นทางการคือคุณไบรอัน เคอร์นิแกน *(Brian Kernighan)* โดยเป็นส่วนหนึ่งของ เอกสารประกอบภาษา *BCPL (Basic Combined Programming Language)*

จากนั้นก็มาอยู่ในตำราภาษา *C* ฉบับคลาสสิคและใช้กันแพร่หลายที่สุดซึ่งแต่งขึ้นโดยคุณเดนนิส ริชชี่ *(Dennis ritchie)* ร่วมกับไบรอัน เคอร์นิแกนเอง

แล้วก็มาเป็น ตัวอย่างในเอกสารประกอบภาษา *C++* และได้รับความนิยมกันมาเรื่อยๆ จนกลายเป็นโปรแกรมพื้นฐานที่สุดที่นิยมใช้กันสำหรับทดสอบการใช้ ภาษาโปรแกรมมิ่งต่างๆ ที่นิยมเริ่มเขียนโปรแกรมแรกด้วย

```
print (“hello, world”)
```

```
#Python 3.9.5
#Example 1-1-3

def MyFirstFunction():
   print ("มาเขียนโปรแกรมภาษา Python อย่างสนุกสนานกันเถอะ")

if __name__ == "__main__":
   MyFirstFunction()
```

```
มาเขียนโปรแกรมภาษา Python อย่างสนุกสนานกันเถอะ
```

ตัวอย่าง 1-1-3

เพื่อความเป็นระเบียบเรียบร้อยของโปรแกรม และสะดวกในการเก็บไว้ใช้หรืออ้างอิงภายหลัง เรานิยมเขียนโปรแกรมไว้ใน function (ฟังก์ชัน) ซึ่งกำหนดด้วยคำสั่ง def ดังตัวอย่างคือ

```
def MyFirstFunction():
```

แล้วค่อยทำการเรียกใช้งานฟังก์ชันนั้นในโปรแกรมหลัก ก่อนที่จะเรียกใช้งานฟังก์ชัน ก็จะมีการตรวจสอบว่าไฟล์นี้เป็นไฟล์โปรแกรมหลักหรือเปล่า ด้วยคำสั่ง

```
if __name__=="__main__"
  MyFirstFunction()
```

MyFirstFunction() คือการเรียกใช้งานฟังก์ชันที่เขียนไว้ก่อนแล้วข้างบน

เรื่องนี้เขียนไปก็ยาว ถือว่ายังไม่ต้องสนใจมาก พิมพ์ตามแบบนี้ไปก่อนได้ แล้วค่อยๆ เรียนรู้กันไป

มีข้อควรระวังอย่างหนึ่งคือ การย่อหน้าหรือ indent เป็นเรื่องสำคัญมากๆ ในภาษา Python ครับ เพราะเป็นการกำหนดขอบเขตของโปรแกรม เดี๋ยวมาคุยเรื่องนี้ละเอียดๆ กันอีกที ตอนนี้พยายามเขียนตามตัวอย่างไม่ให้ error ก่อน

```
#Python 3.9.5
#Example 1-1-4

'บันทึกสนุกๆ'
def MySecondFunction():
   'เขียนเล่น'
   print ("ขีดๆ","เขียนๆ","เรียนๆ","เล่นๆ")

if __name__ == "__main__":
   MySecondFunction()
   print ("Program Name is",__name__)
   print ("Program Document is",__doc__)
   print ("Function Name is", MySecondFunction.__name__)
   print ("Function Document is", MySecondFunction.__doc__)
```

ขีดๆ เขียนๆ เรียนๆ เล่นๆ
Program Name is __main__
Program Document is **บันทึกสนุกๆ**
Function Name is MySecondFunction
Function Document is **เขียนเล่น**

ตัวอย่าง 1-1-4

ตัวอย่างนี้เพิ่มบรรทัดเขียนว่า

```
'บันทึกสนุกๆ'
```

ไว้ถัดจากบรรทัดที่เป็น comment และ

```
'เขียนเล่น'
```

ในบรรทัดแรกของ function

ซึ่งการเขียนข้อความในตำแหน่งนี้ เรียกว่า docstring ความหมายของ docstring ในเอกสาร PEP หรือ Python Enhancement Proposal เขียนไว้ว่า

> *A docstring is a string literal that occurs as the first statement in a module, function, class, or method definition. Such a docstring becomes the __doc__ special attribute of that object.*

แปลคือ docstring คือ string หรือสายอักษรที่ปรากฏอยู่ในบรรทัดแรกของ module (โมดูล) function (ฟังก์ชัน) class (คลาส) หรือ method (เมธอด) ซึ่งจะกลายเป็น attribute (แอททริบิวต์) พิเศษชื่อ __doc__ ของ object (ออปเจ็ค) นั้นๆ

อยู่ๆ เพิ่งเริ่มเขียนโปรแกรม มาอ่านเจอแบบนี้จะรู้สึกชีวิตรันทด แทบจะอยากหลบไปร้องไห้ ไม่เป็นไรครับ ฟังผ่านๆ หูไว้พอ

สรุปคือ docstring มีไว้สำหรับบันทึกหรือทำเอกสารให้กับฟังก์ชัน ซึ่งเราสามารถเรียกดูบันทึกนี้ได้ด้วยการอ่านค่าจากตัวแปร __doc__ ดังตัวอย่าง

Tip
ถ้าเขียนโปรแกรมแล้ว *error* ไม่ผ่านสักที ก็ควรมาอ่านตรงนี้ให้ดีๆ ก่อน แล้วค่อยกลับไปเขียนใหม่ น่าจะแก้ปัญหาได้

ในภาษา *Python* มีเทคนิคพิเศษอันหนึ่ง เรียกว่า *indentation* (อินเดนเทชั่น) หรือ บล๊อคย่อหน้า

> คือภาษา *Python* สร้างบล๊อคหรือกลุ่มของคำสั่งด้วยการ ใช้ *indent* (อินเดนท์) หรือย่อหน้าเป็นตัวกำหนด

ซึ่งปกติๆ เราจะกำหนดย่อหน้านี้ ด้วยการ กดปุ่ม *tab* บนคีย์บอร์ด (ปุ่มซ้ายมือ มักจะอยู่เหนือปุ่ม *caps lock*) หรือถ้าใช้การเคาะ *space bar* (แป้นยาวๆ ล่างสุดของคีย์บอร์ด) ก็ต้องเคาะให้มันเท่าๆ กันจึงจะนับเป็นบล๊อคเดียวกัน ไม่เช่นนั้นก็ถือเป็น คนละบล๊อค

ภาษา *Python* เวอร์ชันใหม่ๆ กำหนดให้การใช้ *tab* เท่ากับการเคาะ *space bar* แล้ว ดังนั้นจะเคาะ *space* หรือกด *tab* ก็ถือว่าไม่ต่างกัน ขอให้ลำดับคั่นหน้ามันตรงกันก็ใช้ได้แล้ว แต่ถ้ามันไม่เท่ากันก็จะ *error*

ตรงนี้คนเขียนโปรแกรม *Python* มือใหม่ๆ พลาดกันเยอะ แต่ถ้าไปเจอ *Python* เวอร์ชันเก่าหน่อยซึ่ง *space bar* ไม่เหมือนกับ *tab* ก็ต้องดูดีๆ บางที *error* เอาง่ายๆ เพราะเราดูว่ามันย่อหน้าเท่ากัน แต่อัน หนึ่งเป็นย่อหน้าจากการเคาะ *space* หลายทีแต่อีกอันเป็นย่อหน้าจาก การกด *tab* ทีเดียว โปรแกรมจะตีความออกมาต่างกัน

ส่วนโปรแกรมที่อยู่ใน *indentation block* หรือในย่อหน้าเดียวกันก็จะถือว่า อยู่ในลำดับชั้นเดียวกัน และเป็นบล็อคลูกของคำสั่งในลำดับชั้นก่อนหน้า นี้ เช่นตัวอย่าง

```
def MyFirstFunction():
  print ("มาเขียนโปรแกรมภาษา Python อย่างสนุกสนานกันเถอะ")

if __name__ == "__main__":
  MyFirstFunction()
```

จากตัวอย่างนี้ บล๊อคลูกของ*def MyFirstFunction():* ก็คือคำสั่ง

```
print (“มาเขียนโปรแกรมภาษา Python อย่างสนุกสนานกันเถอะ”)
```

หรืออาจมีคำสั่งถัดๆ ลงมาอีกก็ได้ ถ้าย่อหน้าตรงกันก็ยังถือว่าอยู่ในบล็อคเดียวกัน ซึ่งขอบเขตของบล๊อคลูกก็จะถูกกำหนดโดยตัวย่อหน้าหรือ *indent* นั่นเอง เท่ากับว่า *def MyFirstFunction():* มีขอบเขตแค่นี้ ส่วนคำสั่งถัดมาคือ

```
if __name__==”__main__”
```

ไม่ถือว่าอยู่ในบล็อคลูกของฟังก์ชัน *def MyFirstFunction():* แล้ว

แต่เป็นตัวโปรแกรมหลักที่อยู่นอกฟังก์ชัน อ่านแล้วงงไม่เป็นไร เขียนตามไปเรื่อยๆ เดี๋ยวก็หายงงเอง

ข้อควรระวังคือ ในการกำหนดบล๊อคลูก คำสั่งก่อนหน้านั้นมักจะมีเครื่องหมายจุดสองจุด **:** อยู่ ท้ายคำสั่งด้วย สังเกตให้ดีๆ เพราะเป็นอีกจุดที่ผิดกันบ่อย เครื่องหมาย **:** หรือจุดสองจุดบนล่างนี้เรียกว่า โคล่อน *(colon)* คำนี้ถ้าเปิดพจนานุกรม จะแปลว่าลำไส้ใหญ่ ไม่รู้เกี่ยวกันอย่างไร แต่ถ้าอยากหาให้เจอด้วยภาษา อังกฤษ ต้องหาด้วยคำว่า *colon punctuation* ภาษาไทยเรียก ทวิภาค แปลว่าสองภาค ทวิแปลว่า *2* ภาษอังกฤษเขียนว่า *two* คงมาจากราก เดียวกัน แต่คนอังกฤษขี้เกียจอ่าน “ทวู” ก็เลยอ่านสั้นๆ เป็น “ทู”

ที่สำคัญคืออย่าไปสับสนกับ ; อันนี้เรียก *semiconlon* มีจุดอยู่ข้างบน และลูกน้ำอยู่ข้างล่าง หรือภาษาไทยเรียกว่าอัฒภาค อัฒตัวนี้เขียนเหมือนกับ อัฒจันทร์ แปลว่าครึ่งก็ได้ ซีกก็ได้ อัฒภาคก็แปลว่าครึ่งภาค ไหนๆ ก็พูดเรื่องจุดกับลูกน้ำแล้ว . จุดนี้ ภาษาไทยเรียกมหัพภาค และลูก น้ำ , เรียกว่าจุลภาค เรียกแบบนี้ได้จะดูเท่มาก ครูจะชมเพื่อนจะร้องหู

Chapter12

VARIABLE, MATH OPERATION

ทำความรู้จักกับตัวแปร และตัวดำเนินการทางคณิตศาสตร์

1.2.1 variable แปลว่าตัวแปร
1.2.2 variable type ชนิดของตัวแปร
1.2.3 การดำเนินการทางคณิตศาสตร์
1.2.4 การดำเนินการทางตรรกะ
1.2.5 กำหนดค่าตัวแปรเป็นชุด

```python
#Python 3.9.5
#Example 1-2-1

def func_1_2_1():
    'รู้จักกับตัวแปร'

    a = 2
    b = 3.1416
    c = "ABC"
    d = True
    e = False
    f = 4+3j
    g = None

    print ("a =",a)
    print ("b =",b)
    print ("c =",c)
    print ("d =",d)
    print ("e =",e)
    print ("f =",f)
    print ("g =",g)

if __name__ == "__main__":
    func_1_2_1()
```

```
a = 2
b = 3.1416
c = ABC
d = True
e = False
f = (4+3j)
g = None
```

ตัวอย่าง 1-2-1

การใช้ตัวแปร หรือเรียกกันว่า variable (วาริเอเบิล) หลายตำราจะเปรียบตัวแปรเป็นถังสำหรับเก็บข้อมูล ส่วนชื่อตัวแปร a,b,c หรือสุดแท้แต่จะตั้ง ก็คือชื่อถัง เวลาจะเอาข้อมูลมาใช้จะได้เรียกถูกว่าไปเอาถัง a หรือถัง b จะไม่สับสน

เราสามารถกำหนดค่าให้กับตัวแปรภาษา Python ได้ง่ายๆ แค่ตั้งชื่อตัวแปร แล้วก็ใส่เครื่องหมายเท่ากับค่าที่ต้องการได้เลย เช่น

```
a=2
b=3.14
```

ก็คือการเอาค่า 2 ไปเก็บไว้ในถังชื่อ a และเอาค่า 3.14 ไปใส่ถังชื่อ b เข้าใจง่ายมาก

การกำหนดค่าเป็นตัวเลขนี้นิยมทำเพื่อเอาไว้ใช้ในการคำนวณ หรือสามารถกำหนดค่าที่เป็นสายตัวอักษรหรือ string (สตริง) ก็ทำได้ โดยใส่ค่าที่ต้องการไว้ในเครื่องหมาย “” เช่น

```
c=”ABC”
```

หรือใช้เครื่องหมาย ‘ ก็ได้ เช่น

```
c=’ABC’
```

หรือแม้แต่กำหนดเป็นค่าทางตรรกศาสตร์ คือ True หรือ False ก็ได้ด้วย เช่น

```
d=True
e=False
```

และพิเศษสุดๆ สำหรับคนที่เรียนคณิตศาสตร์จะได้เจอได้ใช้คือการกำหนดค่าเป็นจำนวนเชิงซ้อน

```
f=4+3j
```

ทั้งนี้การตั้งชื่อตัวแปรด้วยตัวอักษรอังกฤษตัวพิมพ์ใหญ่หรือเล็กถือว่าเป็นคนละตัวกัน และไม่สามารถตั้งชื่อตัวแปรที่ขึ้นต้นด้วยตัวเลขหรือสัญลักษณ์ได้ ยกเว้นสัญลักษณ์ _ underscore สามารถนำมาใช้นำหน้าชื่อตัวแปรได้

```
#Python 3.9.5
#Example 1-2-2

def func_1_2_2():
    'ชนิดของตัวแปร'

    a = 2
    b = 3.1416
    c = "ABC"
    d = True
    e = False
    f = 4+3j
    g = None

    print ("type of a =",type(a))
    print ("type of b =",type(b))
    print ("type of c =",type(c))
    print ("type of d =",type(d))
    print ("type of e =",type(e))
    print ("type of f =",type(f))
    print ("type of g =",type(g))

if __name__ == "__main__":
    func_1_2_2()
```

```
type of a = <class 'int'>
type of b = <class 'float'>
type of c = <class 'str'>
type of d = <class 'bool'>
type of e = <class 'bool'>
type of f = <class 'complex'>
type of g = <class 'NoneType'>
```

ตัวอย่าง 1-2-2

ได้กล่าวไปแล้วว่าตัวแปรมีหลายชนิด ซึ่งปกติแล้วตัวแปรในภาษา Python จะถูกกำหนดชนิดโดยอัตโนมัติเมื่อมีการกำหนดค่า แต่เราสามารถขอดูชนิดของตัวแปรด้วยคำสั่ง

```
type(variable_name)
```

<class 'int'>	ตัวแปรชนิด integer หรือเลขจำนวนเต็ม
<class 'float'>	ตัวแปรชนิด floating point หรือเลขทศนิยม
<class 'str'>	ตัวแปรชนิด string หรือสายอักษร
<class 'bool'>	ตัวแปรชนิด boolean หรือตรรกะ
<class 'complex'>	ตัวแปร complex number จำนวนเชิงซ้อน
<class 'NoneType'>	ตัวแปร NoneType ไม่ระบุชนิดด้วย

```python
#Python 3.9.5
#Example 1-2-3

def func_1_2_3():
    'การคำนวณทางคณิตศาสตร์'
    a = 10
    b = 3
    c = 0+1j

    print ("a+b =",a+b)
    print ("a-b =",a-b)
    print ("a*b =",a*b)
    print ("a/b =",a/b)
    print ("a%b =",a%b)
    print ("a//b = ",a//b)
    print ("a**b =",a**b)
    print ("(a+b)*(a-b)/13 =",(a+b)*(a-b)/13)
    print ("c+c = ",c+c)
    print ("c*c =",c*c)

if __name__ == "__main__":
    func_1_2_3()
```

```
a+b = 13
a-b = 7
a*b = 30
a/b = 3.3333333333333335
a%b = 1
a//b =  3
a**b = 1000
(a+b)*(a-b)/13 = 7.0
c+c =  2j
c*c = (-1+0j)
```

ตัวอย่าง 1-2-3

เราค่อนข้างเชื่อว่าคอมพิวเตอร์จะเก่งคณิตศาสตร์ ดังนั้นภาษา Python ย่อมสามารถใช้คิดเลขได้ เอาตั้งแต่ง่ายๆ ก่อน เช่น การบวกลบคูณหารก็ต้องมีครบ โดย

+	ใช้แทนเครื่องหมายบวก
-	ใช้แทนเครื่องหมายลบ
*	ใช้แทนเครื่องหมายคูณ
/	ใช้แทนเครื่องหมายหาร
%	ใช้ในการหารเอาเศษ เช่น 8%3 จะได้ 2 หรือ 10%5 ก็จะได้เศษ 0 เป็นการหารลงตัว
//	หารเอาจำนวนเต็ม เช่น 10//3 จะได้ 3
**	แทนเครื่องหมายยกกำลัง เช่น 2**3 ได้ 8

ซึ่งนอกจากภาษา Python จะบวกลบคูณหารเลขธรรมดาได้แล้ว ยังทำกับจำนวนเชิงซ้อนได้ด้วย

```
#Python 3.9.5
#Example 1-2-4

def func_1_2_4():
   'การดำเนินการทางตรรกะ True=จริง False=เท็จ'

   a=True; b=False

   print ("True and True =",a&a)
   print ("True and False =",a&b)
   print ("False and False =",b&b)
   print ("True or True =",a|True)
   print ("True or False =",a|False)
   print ("False or False =",b|b)
   print ("Not True =",not(a))
   print ("a==a is",a==a)
   print ("9==8 is",a==b)
   print ("9>8 is",9>8)
   print ("8>=8 is",8>=8)
   print ("8!=9 is",8!=9)

if __name__ == "__main__":
   func_1_2_4()
```

```
True and True = True
True and False = False
False and False = False
True or True = True
True or False = True
False or False = False
Not True = False
a==a is True
9==8 is False
9>8 is True
8>=8 is True
8!=9 is True
```

ตัวอย่าง 1-2-4

ว่ากันว่าคอมพิวเตอร์นี่มักจะเกี่ยวข้องกับ logic หรือตรรกะ ดังนั้น ภาษา Python ย่อมจะมีตรรกะดี

มาดูสัญลักษณ์ที่ใช้เป็นตัวดำเนินการทางตรรกกันก่อน

&	คือ “และ” (and)
|	คือ “หรือ” (or)
==	คือ “เท่าหรือไม่” (is equal?)
!=	คือ “ไม่เท่าหรือไม่” (is not equal?)

การดำเนินการทางตรรก จะได้คำตอบเป็น True หรือ False แทนความหมาย “จริง” และ “เท็จ” โดย

```
True & True = True
True & False = False
False & True = False
False & False = False

True | True = True
True | False = True
False | True = True
False | False = False

not(True) = False
not(False)=True
```

ลองทดลองตามตัวอย่างดู

```python
#Python 3.9.5
#Example 1-2-5

def func_1_2_5():
   'การทำงานกับตัวแปร'

   a,b,c,d = 1,2,"hello", "world"

   print (a,b,c,d)
   print (a+b, c+d)
   print (a*3, c*3)
   a = a+4
   b+=7
   print (a,b)
   c+="!"; d*=3
   print (c,d)

if __name__ == "__main__":
   func_1_2_5()
```

```
1 2 hello world
3 helloworld
3 hellohellohello
5 9
hello! worldworldworld
```

ตัวอย่าง 1-2-5

ตัวอย่างนี้เริ่มจากการกำหนดค่าให้กับตัวแปรทีเดียวเป็นชุดคือ

```
a,b,c,d = 1,2,"hello","world"
```

ก็คือกำหนดให้

```
a=1
b=2
c="hello"
d="world"
```

เพียงแต่เขียนรวบมาเป็นชุด จบในบรรทัดเดียว

ทีนี้ดูตัวอย่างการทำงาน คือถ้าเอาตัวแปรที่เป็นชนิดตัวเลขบวกกัน ผลก็จะได้เป็นการบวกทางคณิตศาสตร์

a+b จะเท่ากับ 1+2 ได้ 3

แต่ c+d จะเท่ากับ "hello" + "world" ก็จะได้ "helloworld" คือกลายเป็นเอาสองข้อความหรือสองสายอักษรมาต่อกัน

และพิเศษ d*3 หรือ "world" *3 ผลคือจะแสดงผล "worldworldworld" เป็นข้อความนี่ต่อกันสามครั้ง

ในตัวอย่างยังมีคำสั่งแปลกๆ อีกคือ

```
a=a+4
```

บรรทัดนี้หมายถึง เอาค่าใน a บวกด้วย 4 แล้วเอากลับไปใส่ไว้ในตัวแปร a

ในกรณีนี้ a ตั้งต้นคือ 1 ดังนั้นคำสั่งนี้ก็เหมือนกับเขียนว่า a=1+4 หรือ a=5 นั่นเอง

มีอีกคำสั่งซึ่งเป็นการรวบคำสั่งคือ

```
b+=7
```

มีความหมายเท่ากับ b=b+7 เพียงแต่เขียนรวบให้สั้น

และบรรทัดก่อนสุดท้าย

```
c+=”!”; d*=3
```

เป็นการเขียนสองคำสั่งให้อยู่ในบรรทัดเดียวกันโดยคั่นด้วยเครื่องหมาย semi-colon ;
ความหมายคือ

```
c+=”!”
d*=3
```

หรือ

```
c=c+”!”
d=d*3
```

Chapter13

GUESS NUMBER

รู้จักการรับข้อมูลจากแป้นพิมพ์ และการดำเนินการตามเงื่อนไข

1-3-1 รับข้อมูลด้วย input
1-3-2 ตรวจสอบเงื่อนไขด้วย if
1-3-3 สร้างเกมทายตัวเลขแบบง่าย
1-3-4 ตรวจสอบหลายเงื่อนไขด้วย if elif else
1-3-5 ทำงานวนรอบด้วย while
1-3-6 การ import และ module random
1-3-7 เกมทายตัวเลขฉบับสมบูรณ์

```
#Python 3.9.5
#Example 1-3-1

def func_1_3_1():
   'คำสั่ง input รับข้อมูลจากคีย์บอร์ด'

   x = input ("Input x ")
   print ("x=",x)
   print ("x=",x,type(x))
   print ("x+3 =",x+"3")
   x = int(x)
   print ("int(x) =",x,type(x))
   print ("x+3 =",x+3)
   x = float(x)
   print ("float(x) =",x,type(x))
   print ("x+3 =",x+3)

if __name__ == "__main__":
   func_1_3_1()
```

```
Input x 10
x= 10
x= 10 <class 'str'>
x+3 = 103
int(x) = 10 <class 'int'>
x+3 = 13
float(x) = 10.0 <class 'float'>
x+3 = 13.0
```

ตัวอย่าง 1-3-1

เราสามารถรับข้อมูลจากคีย์บอร์ดด้วยคำสั่ง input โดยเขียนในรูป

```
variable_name=input(“xxx”)
```

จะแสดงผล xxx บนหน้าจอ และเมื่อเรากดป้อนอะไรเข้าไป ค่าที่ป้อนเข้าไปจะไปเก็บอยู่ในตัวแปร variable_name โดยมี type เป็น string

สามารถตรวจสอบ type ด้วยคำสั่ง type()

ซึ่งถ้าเราต้องการนำค่าตัวแปรนี้ไปใช้ในการคำนวณทางคณิตศาสตร์ต่อ ก็ต้องแปลงค่าให้เป็นชนิดตัวเลข แบบ integer ด้วยคำสั่ง int()

หรือ floating point ด้วยคำสั่ง float()

```
#Python 3.9.5
#Example 1-3-2

def func_1_3_2():
  'if ถ้ามาแบบนี้ แล้วจะไปแบบไหน'
  x=10; y=11

  if x==10:
    print ("Yes, x =",x)

  if y==10:
    print ("Yes, y =",y)

if __name__ == "__main__":
  func_1_3_2()
```

```
Yes, x = 10
```

ตัวอย่าง 1-3-2

การตรวจสอบเงื่อนไขด้วยคำสั่ง if

if x==10: คือตรวจสอบว่า x เท่ากับ 10 หรือไม่

ถ้าใช่ ก็ทำคำสั่งบรรทัดต่อไปในบล๊อคคือ print(“Yes, x=”,x)

if y==10: ตรวจสอบค่าของ y ว่าเท่ากับ 10 หรือไม่

ซึ่งในกรณีนี้ y ไม่เท่ากับ 10 แหงๆ อยู่แล้ว เพราะเรากำหนดค่าตั้งต้นให้ y=11

โปรแกรมก็จะข้ามคำสั่ง print(“Yes, y=”,y) ไม่แสดงผลออกมา

แต่ถ้าอยากให้โปรแกรมพิมพ์บรรทัด print(“Yes, y=”,y) ก็อาจจะลองเปลี่ยนค่าตรงบรรทัด y=11 ให้เป็น y=10 แล้วลองเรียกโปรแกรมดูอีกทีก็ได้

ทบทวนเรื่อง indentation หรือย่อหน้ากันอีกที ในตัวอย่างนี้ จะมีบล๊อคลูกคือ

```
print(“Yes, x=”,x) เป็นบล็อคลูกของ if x==10:
```

และ

```
print(“Yes, y=”,y) เป็นบล๊อคลูกของ if y==10:
```

ซึ่งบล๊อคลูกนี้จะทำงานก็ต่อเมื่อเงื่อนไขของคำสั่ง if ก่อนหน้านั้นเป็นจริงเท่านั้น

```
#Python 3.9.5
#Example 1-3-3

def func_1_3_3():
  'สร้างเกมทายตัวเลข'

  x=input("Input x (guess 1-10) ")

  if x=="10":
    print ("Yes, you win x=10!!!")
  else:
    print ("Nooo, try again (try 10)")

if __name__ == "__main__":
  func_1_3_3()
```

```
Input x (guess 1-10) 5
Nooo, try again (try 10)

Input x (guess 1-10) 10
Yes, you win x=10!!!
```

ตัวอย่าง 1-3-3

ใช้คำสั่ง input เพื่อรับค่าจากคีย์บอร์ดมาสร้างเกมทายตัวเลขแบบง่ายๆ

เพิ่มเติมเรื่องคำสั่ง if else

```
if x==”10”:
  <คำสั่งที่ 1>
else:
  <คำสั่งที่ 2>
```

If คู่กับ else จะใช้ในกรณีตรวจสอบเงื่อนไขตาม if

ถ้าเป็นจริง ก็จะทำงานตาม <คำสั่งที่ 1>
แต่ถ้าไม่ตรงเงื่อนไข ก็จะทำ <คำสั่งที่ 2>

```
#Python 3.9.5
#Example 1-3-4

def func_1_3_4():
   'พัฒนาเกมทายตัวเลข'

   x=input("Input x (guess 1-10) ")

   if x=="10":
      print ("Yes, you win x=10!!!")
   elif x=="9":
      print ("Too low, try again (try 11)")
   elif x=="11":
      print ("Too high, try again (try 10)")
   else:
      print ("Nooo, try again (try 11)")

if __name__ == "__main__":
   func_1_3_4()
```

```
Input x (guess 1-10) 5
Nooo, try again (try 11)

Input x (guess 1-10) 11
Too high, try again (try 10)

Input x (guess 1-10) 10
Yes, you win x=10!!!
```

ตัวอย่าง 1-3-4

เพิ่มเติมจาก if else เป็น if elif else

เราสามารถใช้คำสั่ง elif ตรวจสอบมากกว่าหนึ่งหรือสองเงื่อนไขได้ เช่น

```
if a:
  <คำสั่งที่ 1>
elif b:
  <คำสั่งที่ 2>
elif c:
  <คำสั่งที่ 3>
else:
  <คำสั่งที่ 4>
```

รูปแบบนี้ โปรแกรมจะตรวจสอบทีละเงื่อนไข

ถ้าตรงกับเงื่อนไข a ก็ทำ
    <คำสั่งที่ 1>
แต่ถ้าไม่ตรง ก็ตรวจเงื่อนไข b ต่อ ถ้าใช่ ก็ทำ
    <คำสั่งที่ 2>
ถ้าไม่ใช่อีก ก็ตรวจสอบว่าตรงกับเงื่อนไข c หรือเปล่า

... ไปเรื่อยๆ และถ้าไม่เข้าเงื่อนไขใดๆ เลย ก็ทำ
    <คำสั่งที่ 4>

ลองกดทำงานดูหลายๆ ครั้ง แล้วป้อนค่าที่แตกต่างกันดู

```
#Python 3.9.5
#Example 1-3-5

def func_1_3_5():
   'while ทำงานเป็นวงรอบ ตราบที่ยังเป็นจริง'

   x=0
   while x!=10:
      x = input ("Input x ")
      x = int(x)

      if x==10:
         print ("Yes, you win x=10!!!")
      elif x>10:
         print ("Too high, try again.")
      elif x<10:
         print ("Too low, try again.")

   print ("END GAME!!!")

if __name__ == "__main__":
   func_1_3_5()
```

```
Input x 6
Too low, try again.
Input x 8
Too low, try again.
Input x 9
Too low, try again.
Input x 10
Yes, you win x=10!!!
END GAME!!!
```

ตัวอย่าง 1-3-5

คำสั่งทำงานวนรอบ

```
while <condition>:
```

เป็นคำสั่งให้คอมพิวเตอร์ทำงานในบล๊อค while
ซ้ำไปเรื่อยๆ ตราบที่ <condition> ยังเป็นจริงอยู่

เช่นในตัวอย่างนี้คือ โปรแกรมจะทำงานไปเรื่อยๆ
ตราบที่ x!=10 หรือ x ไม่เท่ากับ 10
และจะหยุดทำงานเมื่อ x=10

แต่ถ้าต้องการหยุดการทำงานของโปรแกรมเอง
สามารถกดปุ่ม Ctrl กับ c พร้อมกัน โปรแกรมก็จะหยุดทำงาน

```
#Python 3.9.5
#Example 1-3-6

import random

def func_1_3_6():
   'การสร้างตัวเลขสุ่ม คำสั่ง random.seed() จะใช้ค่าตั้งต้นจากนาฬิกาของ
ระบบ'

   random.seed()
   x = random.randint(1,10)
   print (x)

if __name__ == "__main__":
   func_1_3_6()
```

```
1

6

4
```

ตัวอย่าง 1-3-6

คำสั่ง import

ใช้สำหรับการนำ module (คนไทยทั่วๆ ออกเสียงโมดูล แต่บางคนออกเสียงว่า โม้ด-หยู่ล ก็ไม่ต้องตกใจ ตัวเดียวกันแหละ) ภายนอกมาใช้งาน ซึ่งจะมีทั้ง module มาตรฐานที่ติดมากับตัวภาษา Python เอง หรือสามารถโหลดมาเพิ่มเติม หรือเขียนเองก็ได้

```
import random
```

คือการโหลด module random เข้ามาในโปรแกรม ทำให้เรามีคำสั่ง random หรือการสร้างตัวเลขสุ่มไว้ใช้งาน ซึ่งคำสั่งนี้ถ้าไม่ import เข้ามาก่อนก็จะเรียกใช้ไม่ได้

คำสั่ง random จะสร้างตัวเลขสุ่มโดยคำนวณจากเลขก่อนหน้า แต่เมื่อเราเรียกใช้ครั้งแรก ไม่มีเลขก่อนหน้าให้ใช้ จึงต้องสร้างเลขก่อนหน้าขึ้นมาก่อนด้วยคำสั่ง random.seed() ซึ่งสามารถกำหนดค่าใน random.seed() นี้ก็ได้ หรือเรียกเฉยๆ เลยก็ได้ ซึ่งโปรแกรมจะไปสร้างค่าตั้งต้นมาจากนาฬิกาของระบบ

```
#Python 3.9.5
#Example 1-3-7

import random

def func_1_3_7():
  'นำ random ไปทำเป็นเกมทายตัวเลข'

  x=0
  random.seed()
  y=random.randint(1,10)

  while x!=y:
    x = int(input("Input x (1-10)"))
    if x==y:
      print ("Yes, you win x=",y,"!!!")
    elif x>y:
      print ("Too high, try again.")
    elif x<y:
      print ("Too low, try again.")

  print ("END GAME!!!")

if __name__ == "__main__":
  func_1_3_7()
```

```
Input x (1-10)6
Too high, try again.
Input x (1-10)2
Too low, try again.
Input x (1-10)4
Yes, you win x= 4 !!!
END GAME!!!
```

ตัวอย่าง 1-3-7

นำคำสั่ง random มาสร้างเป็นเกมทายตัวเลขแบบสุ่ม สร้างเกมง่ายๆ ได้แล้ว

Chapter14

LOOP

รู้จักกับคำสั่งควบคุมทำงานเป็นวงรอบ

1-4-1 ทำงานเป็นวงรอบด้วย for และ range
1-4-2 กำหนดค่าเริ่มต้นให้ range
1-4-3 เงื่อนไขจำนวนรอบสำหรับ range
1-4-4 ทำงานเป็นวงรอบด้วย while (อีกครั้ง)
1-4-5 ทำงานเป็นรอบอนันต์และคำสั่ง brake

```
#Python 3.7.3
#Example 1-4-1

def func_1_4_1():
   'การทำงานเป็นวงรอบด้วยคำสั่ง for'

   for i in range(10):
      print (i)
if __name__ == "__main__":
   func_1_4_1()
```

```
0
1
2
3
4
5
6
7
8
9
```

ตัวอย่าง 1-4-1

ตัวอย่างนี้สั้นๆ ง่ายๆ ว่าด้วยเรื่องของ loop หรือการทำงานแบบวนรอบ โดยกำหนดรอบด้วยคำสั่ง

```
for <variable_name> in <range>:
    <do_something>
```

ในตัวอย่างนี้ใช้ตัวแปร i เป็นตัวนับหรือเรียกทับศัพท์ว่า counter (เคาน์เตอร์) โดยกำหนดขอบเขตของการนับด้วยคำสั่ง range(10) ซึ่งเป็นคำสั่งสร้างค่าแบบช่วง ตั้งแต่ 0-9

```
for i in range(10):
    print(i)
```

ในแต่ละรอบ โปรแกรมจะทำงานในบล๊อคลูกของคำสั่ง for คือให้ print(i) ไปจนกว่าจะหลุดจากวงรอบของ for ซึ่งในกรณีนี้คือเมื่อนับครบตาม range(10)

ก็คือโปรแกรมจะ print(i) จำนวนสิบรอบ แต่ละรอบค่าของ i จะเปลี่ยนไปตามค่าที่อ่านได้จากคำสั่ง range(10) คือเริ่มตั้งแต่ 0,1,2,3 ไปจนถึง 9 คือจะนับไปไม่เกินค่าที่กำหนดไว้ใน range ซึ่งเรากำหนดไว้เป็น 10 ก็จะนับตั้งแต่ 0-9

ถ้าจะไม่ให้ง่ง ก็ทดลองเขียนตามตัวอย่างครับ

```
#Python 3.9.5
#Example 1-4-2

def func_1_4_2():
   'กำหนดค่าใน range() สำหรับคำสั่ง for'

   for i in range(2,10):
      print (i,i+1)
if __name__ == "__main__":
   func_1_4_2()
```

```
2 3
3 4
4 5
5 6
6 7
7 8
8 9
9 10
```

ตัวอย่าง 1-4-2

จากตัวอย่างเรื่องการใช้ for i in range(10): ซึงจะได้ค่าใน range(10) เป็น 0-9

เรายังสามารถกำหนดค่าตั้งต้นใน range ได้ด้วย
เช่น ในตัวอย่าง

```
for i in range(2,10):
```

หมายถึงเริ่มต้นนับที่ 2 เอาค่าไปใส่ใน i แล้วทำซ้ำไปเรื่อยๆ จนกว่าจะถึง 10

หยุดการทำงานเมื่อค่าใน range เป็น 10 คือไม่ทำ ไม่เอาค่า 10 ไปให้ i และไม่ทำงานในบล๊อคลูก

ดังนั้น คำสั่ง print(i,i+1) จะ print ไปเรื่อยๆ ตั้งแต่ค่า 2-9 เท่านั้น ไม่พิมพ์ 10 ออกมาด้วย

ในบล๊อคลูก คำสั่ง print(i,i+1) เห็นได้ว่านอกจากจะแสดงค่า i ออกมาดูเล่นเฉยๆ แล้ว เรายังเอา i ไปทำอย่างอื่นได้ด้วย เช่นเอาไปบวกกับ 1 ก่อนแล้วค่อยแสดงผลออกมา ดังตัวอย่าง

```
#Python 3.9.5
#Example 1-4-3

def func_1_4_3():
   'กำหนดค่าใน range ให้ข้ามทีละ 2'

   for i in range(1,20,2):
      print (i)
      if i>10:
         break
if __name__ == "__main__":
   func_1_4_3()
```

```
1
3
5
7
9
11
```

ตัวอย่าง 1-4-3

ตัวอย่างนี้แสดงการใช้คำสั่ง range(x,y,z) โดย

x เป็นค่าเริ่มต้น
y บอกว่านับไม่เกินนี้
z แต่ละรอบให้เพิ่มค่าไปทีละ +z ซึ่งถ้าไม่ระบุปกติจะเพิ่มทีละ 1

ดังนั้นในตัวอย่างนี้ range(1,20,2) คือจะเริ่มนับจาก 1 นับไม่เกิน 20 และเพิ่มทีละ 2 ก็จะได้ผลเป็น

รอบแรก		1
รอบสอง	1+2 =	3
รอบสาม	3+2 =	5
รอบสี่	5+2 =	7
รอบห้า	7+2 =	9
รอบหก	9+2 =	11
หยุด		

range(1,20,2) ทำไมหยุด
เพราะในโปรแกรมเราแอบใส่คำสั่ง

```
If i> 10:
        break
```

คือให้หยุดทำงานเมื่อ i มากกว่า 10 ดังนั้นเมื่อ i=11 มากกว่า 10 แล้ว ทำให้โปรแกรมเข้าไปทำงานในบล๊อคลูกของ if i>10: คือสั่งให้โปรแกรม break หรือหยุดการทำงาน

```
#Python 3.9.5
#Example 1-4-4

def func_1_4_4():
   'ตรวจสอบเงื่อนไขของวงรอบในคำสั่ง while'
   y = 2

   while y<10000:
      print ("y=",y)
      y*=2

if __name__ == "__main__":
   func_1_4_4()
```

```
y= 2
y= 4
y= 8
y= 16
y= 32
y= 64
y= 128
y= 256
y= 512
y= 1024
y= 2048
y= 4096
y= 8192
```

ตัวอย่าง 1-4-4

การทำงานด้วยคำสั่งวนรอบในภาษา Python นอกจากคำสั่ง for ซึ่งใช้นับค่าใน range หรือใน list แล้ว ยังมีอีกคำสั่งหนึ่ง คือคำสั่ง

```
while
```

คำสั่ง while ไม่ได้ใช้การนับค่า แต่จะทำงานในบล๊อคลูก วนซ้ำไปเรื่อยๆ ตราบที่เงื่อนไขที่ตั้งไว้ยังเป็นจริง หรือยังมีค่าเป็น True

ตัวอย่างนี้กำหนดค่าเริ่มต้นของ y=2 จากนั้นก็ทำงานในบล๊อคลูกไปเรื่อยๆ ตราบที่ y<10000

ซึ่งในบล๊อคลูก เราเพิ่มค่า y แต่ละรอบด้วยคำสั่ง y*=2 หรือ y=y*2 เพิ่มค่าโดยการคูณสองไปเรื่อยๆ แล้วพิมพ์ออกมาดูกัน

```
#Python 3.9.5
#Example 1-4-5

def func_1_4_5():
   'while True วงรอบอนันต์ ทำงานไปเรื่อยๆ จนกว่าจะเจอ break'

   while True:
      x = input ("Input 1-10 (10 to Break) ")
      if x=="10":
         print ("END LOOP!!!")
         break
      else:
         print ("x =",x)
if __name__ == "__main__":
   func_1_4_5()
```

```
Input 1-10 (10 to Break) 6
x = 6
Input 1-10 (10 to Break) 7
x = 7
Input 1-10 (10 to Break) 10
END LOOP!!!
```

ตัวอย่าง 1-4-5

ตัวอย่างนี้เป็นการทำงานซ้ำปเรื่อยๆ ด้วยคำสั่ง

```
while True
```

โปรแกรมจะทำงานวนรอบไปเรื่อยๆ เพราะไม่ได้ใส่เงื่อนไขหยุดการทำงานไว้

แต่เรายังมีท่าไม้ตายคือสามารถหยุดการทำงานด้วยคำสั่ง break โดยป้อนค่าให้ตรงตามเงื่อนไข ซึ่งก็เขียนบอกอยู่โต้งๆ ว่า ป้อนเลข 10 เข้าไปแล้วโปรแกรมจะหยุด

หรือถ้ากดปุ่ม Ctrl กับ c พร้อมๆ กัน ก็สามารถหยุดการทำงานของโปรแกรมได้เช่นกัน

ปุ่ม Ctrl นี้เรียกชื่อว่าปุ่ม control แต่ส่วนมากจะเขียนไว้แค่ Ctrl มักจะเป็นปุ่มซ้ายมือล่างสุดบนแป้นพิมพ์ หาไม่ยาก

Tip
ทำไมหลายคนชอบใช้ตัวแปร *i* กับคำสั่ง *for* เช่น

```
for i in range(n):
```

โปรแกรมเมอร์บางคนอาจใช้ตัวแปรอื่น เช่น

```
for counter in range(n):
```

หรือ

```
for index in range(n)
```

ก็มี แต่หลายๆ ตัวอย่างก็จะเจอ *for i* โผล่ ออกมาเรื่อยๆ แถมมาในหลายๆ ภาษาด้วย

ถ้าลองค้นดูใน *google* ก็จะเจอคำตอบที่หลากหลาย เช่น *i* แทนคำว่า *index* บ้าง หรือ *i, j* ใช้ในภาษาทางคณิตศาสตร์บ้าง

แต่คำตอบที่เชื่อว่าน่าจะเป็นจุดเริ่มต้นจริงๆ เพราะพอดีผู้เขียนเกิดทันยุค ที่โปรแกรมเมอร์ทุกคนใช้คำสั่ง *for i* ตั้งแต่หลายสิบปีก่อน

คือในยุคโบร่ำโบราณนู้น สมัยที่คนจะเขียนโปรแกรม ต้องเขียนลงใน สมุด แล้วเอาคำสั่งไปเจาะรูในกระดาษแข็งๆ เรียกกันว่าการ์ดเจาะรู การ์ดหนึ่งแผ่นแทนหนึ่งคำสั่ง โปรแกรมหนึ่งก็ใช้การ์ดปึ๊งหนึ่งแล้วเอา ไปป้อนลงคอมพิวเตอร์ทีละใบๆ ให้คอมพิวเตอร์อ่านแล้วทำงาน ยุคนั้น มีภาษาคอมพิวเตอร์ที่เป็นที่นิยมอยู่ภาษาหนึ่งคือ *FORTRAN IV*

ซึ่งก่อน *FORTRAN IV* ก็คงมีเวอร์ชันก่อนหน้านั้น

เพราะคงไม่มีใครเกิดมาก็ตั้ง ชื่อเป็นเวอร์ชัน *IV* หรือ *4* เลย แต่ไม่เคยเห็นเวอร์ชันเก่ากว่านี้ เกิดมาก็ เจอเวอร์ชัน *IV* นี้แล้ว เป็นที่นิยมมาก

ภาษานี้กำหนดให้ตัวแปรที่ตั้งชื่อ นำหน้าด้วยอักษร *I* ถึง *N* เป็นตัวแปร *integer* หรือจำนวนเต็มโดยอัตโนมัต ขณะที่ตัวแปรที่ขึ้นต้นด้วยอักษรอื่น เป็น *real* (หรือ *float* ในภาษา *Python)*

ทีนี้ตัวแปรที่เราจะใช้มาเป็นตัวนับใน *loop* ได้ก็ต้องเป็นตัวแปรชนิดจำนวนเต็มหรือ *integer* นี่เอง ทำให้โปรแกรมเมอร์สมัยนั้นนิยมใช้ตัวแปร *i* แทนความหมายตัวแปรชนิด *integer* ทีนี้ภาษายุคที่ใกล้ๆ กับ *FORTRAN* แล้วเริ่มมีการใช้ *for i* เท่าที่ได้ทันเห็นคือภาษา *BASIC* ซึ่ง ใช้รูปแบบคำสั่ง

```
FOR NN = XX TO YY
...DO SOMETHING
NEXT NN
```

โดย *NN* เป็นตัว *counter* เหมือนใน *Python* และด้วยความนิยมในการ ใช้ตัวแปร *i* จากภาษา *FORTRAN* ก็ติดมาถึงภาษา *BASIC* นี้ด้วย จาก ตัวอย่างโปรแกรมสมัยนั้นเป็น *for i* และถ้ามีการใช้คำสั่ง *for* ซ้อนกัน ก็จะใช้ตัวแปรชื่อ *i* และ *j* ตามตัวอย่างนี้

```
FOR I = 1 TO 10
        FOR J = 1 TO 9
....                DO SOMETHING
        NEXT J
NEXT I
```

...หลังจากนั้นไม่รู้ยังไงคำสั่ง *for i* นี้ก็สืบทอดต่อๆ เป็นที่นิยมมาเรื่อยๆ จนปัจจุบัน

ข้อสังเกตนิดหนึ่งคือ สมัยนั้นเขียนโปรแกรมกันด้วยตัวพิมพ์ใหญ่เท่านั้นครับ ไม่มีการใช้ตัวพิมพ์เล็ก

Chapter15

LIST, TUPLE

รู้จักกับตัวแปรชนิด list และ tupple

1-5-1 ตัวแปรชนิด list
1-5-2 ตำแหน่งของข้อมูลใน list
1-5-3 การกำหนดค่าให้กับ list
1-5-4 list ซ้อน list
1-5-5 เพิ่มข้อมูลใน list ด้วยคำสั่ง append, extend, insert,index
1-5-6 ลบข้อมูลออกจาก list ด้วยคำสั่ง remove, del
1-5-7 เรียงข้อมูลใน list ด้วยคำสั่ง sort, reverse
1-5-8 การ copy หรือคัดลอกข้อมูลใน list
1-5-9 tuple

```
#Python 3.9.5
#Example 1-5-1

def func_1_5_1():
   'ตัวแปรชนิด list'

   a = [9,2,5,3,8,7]
   b = ["abc",2,3,False]

   print ("a =",a)
   print ("b =",b)
   print ("type of a =",type(a))
   print ("type of b =",type(b))
   print ("a+b =",a+b)

if __name__ == "__main__":k
   func_1_5_1()
```

```
a = [9, 2, 5, 3, 8, 7]
b = ['abc', 2, 3, False]
type of a = <class 'list'>
type of b = <class 'list'>
a+b = [9, 2, 5, 3, 8, 7, 'abc', 2, 3, False]
```

ตัวอย่าง 1-5-1

บทนี้มาพูดกันถึงตัวแปรชนิด list (ลิสต์) ซึ่งไม่นิยมแปลเป็นไทย ปกติก็จะทับศัพท์ว่าลิสต์ไปเลย

ตัวแปรลิสต์สร้างง่ายมาก คือกำหนดค่าเป็นชุดๆ ไว้ในวงเล็บก้ามปู หรือสัญลักษณ์แบบนี้ [ ] โดยคั่นข้อมูลแต่ละตัวด้วยเครื่องหมายจุลภาค , หรือเรียกว่าลูกน้ำก็ได้

จากตัวอย่าง

```
a=[9,2,5,3,8,7]
```

และ

```
b = [“abc”,2,3,False]
```

จะเห็นว่าข้อมูลในลิสต์เป็นอะไรก็ได้ จะเป็นตัวเลข สายอักษร หรือค่าตรรกะก็ได้ หรือหลายๆ ชนิดปนกันอยู่ก็ได้ เมื่อใช้คำสั่ง type(a) ก็จะได้ออกมาว่าเป็น <class 'list'> หรือ ตัวแปรชนิดลิสต์

TIP
ภาษาอังกฤษเรียกเครื่องหมาย *[ ]* นี้ว่า *brackets* แต่ภาษาไทยเรียกวงเล็บทุกแบบว่า นขลิขิตเหมือนกันหมด แล้วค่อยกำหนดชนิดย่อยของ วงเล็บแบบนี้ว่าเป็นวงเล็บก้ามปู

```
#Python 3.9.5
#Example 1-5-2

def func_1_5_2():
   'การเรียกข้อมูลใน list'

   a = [9,2,5,3,8,7]
   b = ["abc",2,3,False]
   c = "HELLO"

   print ("a[1] =",a[1])
   print ("b[0] =",b[0])
   print ("a[-2] =",a[-2])
   print ("a[1:4]",a[1:4])
   print ("c[2] =",c[2])
   print ("c[2:] =",c[2:])

if __name__ == "__main__":
   func_1_5_2()
```

```
a[1] = 2
b[0] = abc
a[-2] = 8
a[1:4] [2, 5, 3]
c[2] = L
c[2:] = LLO
```

ตัวอย่าง 1-5-2

ตัวอย่างนี้เข้าใจง่ายมาก เนื่องจากลิสต์จะมีข้อมูลเป็นชุดๆ ดังนั้นการจะอ้างอิงหรือเรียกข้อมูลตัวใดตัวหนึ่งในลิตส์ออกมาก็จะอ้างอิงด้วยตำแหน่งของข้อมูลในลิสต์นั้น โดยเริ่มต้นนับตัวแตกเป็นตัวที่ 0

การอ้างตำแหน่ง ทำได้ด้วยการเขียนชื่อตัวแปร ตามด้วยวงเล็บก้ามปูบอกตำแหน่งข้อมูลนั้น เช่น

```
a=[9,2,5,3,8,7]
```

a[0] คือ 9
a[1] คือ 2
a[2] คือ 5

ถ้าใส่เลขลบ เช่น

```
a[-2]
```

จะได้ 8 คือนับจากข้างหลังมาตัวที่ 2 โดยตัวท้ายสุดจะเป็นตัวที่ -1

หรือเรียกมาเป็นช่วงก็ได้ เช่น

a[1:4] ได้ [2,5,3]

ได้ผลลัพธ์ออกมาเป็นลิสต์ ตั้งแต่ตัวที่ 1 และสิ้นสุดก่อนตัวที่ 4 อันนี้ทำความคุ้นเคยไว้ ดีๆ เพราะภาษา Python มีวิธีนับแปลกๆ คือค่าสุดท้ายมักจะเป็นค่าที่ไม่เอามาด้วย

ในกรณีนี้คือจะได้ลิสต์ที่ประกอบด้วยข้อมูลตัวที่ 1,2,3 มา

แต่ถ้าอ้างข้อมูลที่ไม่มีอยู่ในลิสต์ เช่น a[10] โปรแกรมก็จะ error สามารถทดลองดูก็ได้ คอมพิวเตอร์ไม่ระเบิด แค่ error เฉยๆ

และแถมให้นิดหนึ่งคือการเรียกข้อมูลด้วยการระบุชื่อตัวแปรแล้วตามด้วยก้ามปูนี้ ยังสามารถประยุกต์ใช้กับตัวแปร string หรือสายอักษรได้ด้วย ดูตามตัวอย่างได้เลย

```
#Python 3.9.5
#Example 1-5-3

def func_1_5_3():
   'list a[start:end] not include end'

   a = [9,2,5,3,8,7]
   print ('a =',a)
   a[0] = 7
   a[2] =['a','Hello',9+2]
   print ('a =',a)
   a[1:4] = 1,3
   print ('a =',a)

if __name__ == "__main__":
   func_1_5_3()
```

```
a = [9, 2, 5, 3, 8, 7]
a = [7, 2, ['a', 'Hello', 11], 3, 8, 7]
a = [7, 1, 3, 8, 7]
```

ตัวอย่าง 1-5-3

ตัวอย่างการใช้งานลิสต์เพิ่มเติม ตัวอย่างที่แล้วเป็นการเรียกดูค่าในลิสต์ ส่วนตัวอย่างนี้จะเป็นการกำหนดค่าให้กับลิสต์ ก็สามารถกำหนดทีละตัว หรือกำหนดเป็นชุดก็ได้ ไม่อธิบายละ ลองศึกษาจากตัวอย่าง

```
#Python 3.9.5
#Example 1-5-4

def func_1_5_4():
   'list ซ้อน list'

   a = [[3,4,5],
      [6,7,8],
      ['a','b','hello']]

   print (a)
   print (a[1])
   print (a[2][1])
   print (a[2][2][2])

if __name__ == "__main__":
   func_1_5_4()
```

```
[[3, 4, 5], [6, 7, 8], ['a', 'b', 'hello']]
[6, 7, 8]
b
l
```

ตัวอย่าง 1-5-4

ตัวอย่างนี้แสดงการสร้างลิสต์สองชั้นหรือลิสต์ซ้อนลิสต์ ซึ่งจริงๆ จะซ้อนกันกี่ชั้นก็ได้ เวลาอ้างถึงก็จะเรียงไปทีละชั้นเช่น

a[1] หมายถึงข้อมูลตัวที่ 1 (ข้อมูลตัวที่ 1 คือตำแหน่งที่ 2 เพราะเริ่มนับมาจาก 0,1) ของลิสต์ a

a[2][1] หมายถึง ข้อมูลตัวที่ 1 จากลิสต์ ซึ่งเป็นข้อมูลตัวที่ 2 ของลิสต์ a

ในตัวอย่างมีที่น่าสนใจคือเราสามารถอ้างถึงข้อมูล string ในรูปแบบของลิสต์ได้ด้วย

เช่น

```
a="hello"
```

a[2] คือ “l”

หรือข้อมูลตัวที่ 2 จาก “hello” (นับ h เป็นตัวที่ 0, e เป็นตัวที่ 1, l เป็นตัวที่ 2)

```
#Python 3.9.5
#Example 1-5-5

def func_1_5_5():
   'การเพิ่มข้อมูลเข้าใน list'

   a =[]

   print ("1. a =",a)
   a.append("hello")
   print ("2. a =",a)
   a.append("world")
   print ("3. a =",a)
   a.extend(["!",9,9])
   print ("4. a =",a)
   a.insert (2,"สวัสดี")
   print ("5. a =",a)
   print ("6. find position of '!'=",a.index('!'))
   a.append([0,0,1])
   print ("7. a =",a)

if __name__ == "__main__":
   func_1_5_5()
```

```
[1. a = []
2. a = ['hello']
3. a = ['hello', 'world']
4. a = ['hello', 'world', '!', 9, 9]
5. a = ['hello', 'world', 'สวัสดี', '!', 9, 9]
6. find position of '!'= 3
7. a = ['hello', 'world', 'สวัสดี', '!', 9, 9, [0, 0, 1]]
```

ตัวอย่าง 1-5-5

คำสั่งที่ใช้กับลิสต์

append	ใช้สำหรับเพิ่มข้อมูลใหม่เข้าไปในลิสต์ต่อท้ายข้อมูลเดิม
extend	เอาลิสต์ใหม่ไปต่อกับลิสต์เดิม
insert	แทรกข้อมูลเข้าไปในลิสต์ในตำแหน่งที่ต้องการ
index	หาตำแหน่งของข้อมูลตัวใดตัวหนึ่ง โดยนับตำแหน่งตัวแรกเป็น 0

```
#Python 3.9.5
#Example 1-5-6

def func_1_5_6():
   'ลบข้อมูลออกจาก list'

   a = [1,2,3,4,5,6,3,4,3,6]
   b = "HELLO"

   print ("a =",a)
   print ("count 3 in a =",a.count(3))
   print ("count L in b =",b.count("L"))
   a.remove(3)
   print ("a =",a)
   a.remove(3)
   print ("a =",a)
   del a[0]
   print ("a =",a)

if __name__ == "__main__":
   func_1_5_6()
```

```
a = [1, 2, 3, 4, 5, 6, 3, 4, 3, 6]
count 3 in a = 3
count L in b = 2
a = [1, 2, 4, 5, 6, 3, 4, 3, 6]
a = [1, 2, 4, 5, 6, 4, 3, 6]
a = [2, 4, 5, 6, 4, 3, 6]
```

ตัวอย่าง 1-5-6

การลบข้อมูลออกจากลิสต์

remove ใช้ลบข้อมูลตัวนั้นออกจากลิสต์ ซึ่งถ้ามีข้อมูลซ้ำกันหลายตัว ก็จะลบออกไปทีละตัว โดยลบจากตำแหน่งน้อยที่สุดออกไปก่อน ดังตัวอย่าง

```
a=[1,2,3,4,5,6,3,4,3,6]
a.remove(3)
print(a)
```

จะได้ [1,2,4,5,6,3,4,3,6]

del ใช้ลบข้อมูลจากลิสต์ในตำแหน่งที่ต้องการ เช่น

```
a = [1,2,4,5,6,4,3,6]
del a[0]
print(a)
```

จะได้ [2,4,5,6,4,3,6]

ข้อสังเกต

remove ใช้ลบตัวข้อมูลที่ต้องการ
del ใช้ลบข้อมูลในตำแหน่งที่ต้องการ

```python
#Python 3.9.5
#Example 1-5-7

def func_1_5_7():
    'เรียงข้อมูลใน list ด้วย sort และ reverse'

    a = [2,3,6,1,4,8,1,5,3]
    b = a

    print ("a =",a)
    print ("b =",b)
    a.sort()
    print ("a =",a)
    print ("b =",b)
    b.reverse()
    print ("a =",a)
    print ("b =",b)


if __name__ == "__main__":
    func_1_5_7()
```

```
a = [2, 3, 6, 1, 4, 8, 1, 5, 3]
b = [2, 3, 6, 1, 4, 8, 1, 5, 3]
a = [1, 1, 2, 3, 3, 4, 5, 6, 8]
b = [1, 1, 2, 3, 3, 4, 5, 6, 8]
a = [8, 6, 5, 4, 3, 3, 2, 1, 1]
b = [8, 6, 5, 4, 3, 3, 2, 1, 1]
```

ตัวอย่าง 1-5-7

การเรียงข้อมูลในลิสต์

คำสั่ง sort() ใช้ในการเรียงข้อมูลในลิสต์จากน้อยไปมาก
คำสั่ง reverse() ใช้ในการสลับตำแหน่งข้อมูลจากหน้าไปหลัง ลองดูจากตัวอย่างก็จะเข้าใจได้

บทนี้มีข้อสังเกตนิดหนึ่ง คือ
เมื่อกำหนดค่าตัวแปร a เป็นลิสต์
จากนั้นกำหนดให้ b=a
เมื่อสั่ง a.sort() จะพบว่า b ก็ถูก sort ไปด้วย

และที่สนุกสนานกว่านั้นคือเมื่อสั่ง
b.reverse() หรือกลับตำแหน่งของค่า b
ก็จะพบว่า a ก็มีการเปลี่ยนแปลงไปด้วย
นั่นคือ a และ b เสมือนว่าเป็นลิสต์เดียวกันไปแล้ว ไม่ใช่สองลิสต์ที่มีค่าเท่ากัน เมื่อกระทำอะไรกับลิสต์หนึ่งก็จะส่งผลไปถึงอีกลิสต์หนึ่งด้วย ซึ่งเรื่องนี้จะอธิบายเพิ่มเติมในบทต่อไป

```
#Python 3.9.5
#Example 1-5-8

def func_1_5_8():
   'การ copy list'

   a = [1,3,5,7,9]
   b = a
   c = a[:]

   print ("a =",a)
   print ("b =",b)
   print ("c =",c,"\n")
   a[2] = 11
   print ("a =",a)
   print ("b =",b)
   print ("c =",c,"\n")
   b.append(13)
   print ("a =",a)
   print ("b =",b)
   print ("c =",c)

if __name__ == "__main__":
   func_1_5_8()
```

```
a = [1, 3, 5, 7, 9]
b = [1, 3, 5, 7, 9]
c = [1, 3, 5, 7, 9]

a = [1, 3, 11, 7, 9]
b = [1, 3, 11, 7, 9]
c = [1, 3, 5, 7, 9]

a = [1, 3, 11, 7, 9, 13]
b = [1, 3, 11, 7, 9, 13]
c = [1, 3, 5, 7, 9]
```

ตัวอย่าง 1-5-8

การ copy หรือคัดลอกข้อมูลในลิสต์
จากตัวอย่าง

```
a=[1,3,5,7,9]
b=a
c=a[:]
```

ทั้งคำสั่ง b=a และ c=a[:] นี้ให้ผลคล้ายๆ กันคือเป็นการคัดลอกข้อมูลในลิสต์

ซึ่งเมื่อสั่ง

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

จะได้ค่าเหมือนๆ กันคือ [1,3,5,7,9]

แต่ความแตกต่างคือ การสั่ง b=a นั้นเป็นการกำหนดให้ตัวแปร a และ b ชี้ไปที่ข้อมูลชุดเดียวกัน ดังนั้นไม่ว่าจะเปลี่ยนแปลงอะไรที่ตัวข้อมูลก็จะกระทบทั้ง a และ b

เช่น

```
a[2]=11
```

ผลคือ a และ b จะกลายเป็น [1,3,11,7,9]
ขณะที่ c ยังเป็นค่าเดิมคือ [1,3,5,7,9]

และเมื่อสั่ง

```
b.append(13)
```

เพิ่มค่า 13 เข้าไปใน b ก็มีผลทั้ง a และ b เช่นกัน

สรุปคือ การสั่ง

```
b=a
```

หรือ

```
list2=list1
```

จะเป็นการกำหนดชื่อใหม่เพิ่มให้กับลิสต์เดิม ซึ่งไม่ว่าจะเรียกด้วยชื่อเดิมหรือชื่อใหม่ ก็จะหมายถึงลิสต์ตัวเดียวกัน

ขณะที่

```
c=a[:]
```

หรือ

```
list2=list1[:]
```

หมายถึงการลอกข้อมูลทั้งหมดของ list1 มาใส่ใน list2 ซึ่ง list2 ก็จะกลายเป็นคนละลิสต์กับ list1 ไม่เกี่ยวข้องกัน

```
#Python 3.9.5
#Example 1-5-9

def func_1_5_9():
   'ตัวแปรชนิด tuple'

   a = (7,8,9)
   b = 3,4,5
   c = a+b

   print ("a= ",a)
   print ("type of a = ",type(a))
   print ("a[1] =",a[1])
   print ("b =",b)
   print ("type of b =",type(b))
   print ("c =",c)
   print ("type of c =",type(c))

if __name__ == "__main__":
   func_1_5_9()
```

```
a=  (7, 8, 9)
type of a =  <class 'tuple'>
a[1] = 8
b = (3, 4, 5)
type of b = <class 'tuple'>
c = (7, 8, 9, 3, 4, 5)
type of c = <class 'tuple'>
```

ตัวอย่าง 1-5-9

มีตัวแปรอีกชนิดคล้ายๆ list แต่ชื่อของเค้าจริงๆ คือ tuple อ่านว่า ทูเปิ้ล ที่ไม่ได้เกี่ยวอะไรกับแอปเปิลสองลูก หรือน้องเปิ้ลคนไหนๆ ทั้งสิ้น

ตัวแปรชนิดทูเปิ้ลนี้คล้ายๆ ลิสต์ วิธีเรียกใช้ก็คล้ายๆ กัน ความแตกต่างหนึ่งเดียวคือเมื่อกำหนดค่าหรือสร้างตัวแปรทูเปิ้ลขึ้นมาแล้ว จะไม่สามารถแก้ไขเปลี่ยนแปลงค่าได้ เรียกใช้เรียกดูได้เท่านั้น

สร้างง่ายมาก ตามตัวอย่างเลย คือกำหนดตัวแปรด้วยชุดข้อมูลในวงเล็บ หรือไม่ต้องวงเล็บก็ได้อยู่ แต่ถ้าใส่วงเล็บไว้จะป้องกันอาการงงได้มากกว่า

Chapter16

การประยุกต์ FOR

เพิ่มเติมกับคำสั่ง for

1-6-1 for กับ tuple
1-6-2 for กับ list
1-6-3 for กับ tuple ที่มีข้อมูลหลายๆ ชนิด
1-6-4 for กับ brake
1-6-5 เพิ่มจำนวนรอบให้ for
1-6-6 for กับ string
1-6-7 for กับ tuple ของ string
1-6-8 try except finally

```
#Python 3.9.5
#Example 1-6-1

def func_1_6_1():
   'การใช้คำสั่ง for กับ tuple'

   for i in (1,1,2,3,5,8,13):
      print (i)

if __name__ == "__main__":
   func_1_6_1()
```

```
1
1
2
3
5
8
13
```

ตัวอย่าง 1-6-1

เราเคยใช้คำสั่งวนรอบ for กับ range() มาแล้ว

ทีนี้ลองดูการประยุกต์ใช้คำสั่ง for กับตัวแปรชนิด tuple บ้าง

รูปแบบง่ายมากโดยใช้ tuple มาใส่แทน range() แล้ว for ก็จะอ่านค่าจาก tuple ทีละตัวมาใส่ในตัวแปรของ for

```
#Python 3.9.5
#Example 1-6-2

def func_1_6_2():
   'ใช้คำสั่ง for กับ list'

   for i in [1,1,2,3,5,8,13]:
      print (i)

if __name__ == "__main__":
   func_1_6_2()
```

```
1
1
2
3
5
8
13
```

ตัวอย่าง 1-6-2

ก็ในเมื่อใช้ for กับ tuple ได้ ก็ย่อมใช้กับ list ได้

```
#Python 3.9.5
#Example 1-6-3

def func_1_6_3():
   'ใช้ for กับ tuple ที่มีข้อมูลต่างชนิด'

   for i in ("abc",True,2,(3,4,5),"def"): print (i)

if __name__ == "__main__":
   func_1_6_3()
```

```
abc
True
2
(3, 4, 5)
def
```

ตัวอย่าง 1-6-3

ตัวอย่างการใช้ for กับ tuple ที่มีข้อมูลต่างชนิดหลากหลาย ไม่รู้ทำไปทำไมเหมือนกัน แต่ทำให้ดูเป็นตัวอย่างว่าทำแบบนี้ก็ได้อยู่ เผื่อใครมีกรณีไหนอยากเอาไปใช้งาน

```
#Python 3.9.5
#Example 1-6-4

def func_1_6_4():
  'ใช้ break เพื่อหยุดการทำงานของ for'

  for i in (1,3,5,7,8,9,10):
    print (i)
    if i==5:
      break
    else:
      pass
  print ("end at",i)

if __name__ == "__main__":
  func_1_6_4()
```

```
1
3
5
end at 5
```

ตัวอย่าง 1-6-4

ตัวอย่างนี้เป็นตัวอย่างการใช้คำสั่ง break เพื่อหยุดการทำงานของ for ในขณะที่ยังทำงานไม่ครบรอบที่กำหนดไว้ตอนต้น

จากตัวอย่าง

```
for i in (1,3,5,7,8,9,10):
```

โปรแกรมจะต้องทำงาน 7 รอบ โดยอ่านค่าใน tuple มาใส่ใน i จนหมดข้อมูลตัวสุดท้ายใน tuple

แต่เราไปหยุดการทำงานของโปรแกรมกลางทาง

โดยตรวจสอบค่าใน i ด้วยคำสั่ง if

นั่นคือเมื่อค่าใน i==5 เราก็สั่ง

```
break
```

หยุดการทำงานเองดื้อๆ ไม่รอให้จบ

```
#Python 3.9.5
#Example 1-6-5

def func_1_6_5():
   'เพิ่มจำนวนรอบตั้งต้นของ for'

   x = [1,3,5,7,9]
   for i in x:
      print (i)
      if i==5:
         x.extend([10,11,12,13])
   print (x)

if __name__ == "__main__":
   func_1_6_5()
```

```
1
3
5
7
9
10
11
12
13
[1, 3, 5, 7, 9, 10, 11, 12, 13]
```

ตัวอย่าง 1-6-5

ตัวอย่างก่อนหน้านี้ เราหยุดการทำงานของคำสั่ง for กลางคัน ตัวอย่างนี้เราลองเพิ่มรอบการทำงานของคำสั่ง for ให้เพิ่มขึ้นกลางโปรแกรมกัน

ดูจากตัวอย่างเลยครับ

เริ่มจากกำหนดค่าให้ลิสต์

```
x=[1,3,5,7,9]
```

จากนั้นเอาลิสต์มาเป็นตัวกำหนดจำนวนรอบให้กับคำสั่ง for ด้วย

```
for i in x:
```

แล้วกลางๆ รอบ อยู่ดีๆ เราก็ไปเพิ่มค่าให้กับ x ด้วย

```
x.extend([10,11,12,13])
```

ซึ่งทำให้ค่าใน x กลายเป็น [1,3,5,7,9,10,11,12,13]

ผลก็คือการทำงานวนรอบของ for ที่น่าจะจบแค่ 5 รอบ วิ่งต่อไปเป็น 9 รอบหน้าตาเฉย

```
#Python 3.9.5
#Example 1-6-6

def func_1_6_6():
  'การใช้ for กับตัวแปรชนิด string'

  for i in "Hello World!":
    print (i)

if __name__ == "__main__":
  func_1_6_6()
```

```
H
e
l
l
o

W
o
r
l
d
!
```

ตัวอย่าง 1-6-6

นอกเราให้โปรแกรมทำงานเป็นวงรอบจาก range() จาก tuple และจาก list แล้ว เรายังสามารถใช้ตัวแปรชนิด string เป็นตัวกำหนดจำนวนรอบให้กับ for ได้ด้วย ดังตัวอย่าง

```
#Python 3.9.5
#Example 1-6-7

def func_1_6_7():
   'การใช้ for กับ tuple ของ string'

   for i in ("Superman", "Batman", "Aquaman"):
     print ("Hello",i)

if __name__ == "__main__":
   func_1_6_7()
```

```
Hello Superman
Hello Batman
Hello Aquaman
```

ตัวอย่าง 1-6-7

ตัวอย่างนี้ไม่อธิบายก็เข้าใจได้มั้ง

```
#Python 3.9.5
#Example 1-6-8

def func_1_6_8():
  'try กับ except เพื่อดักการเกิด error'

  x = [1,2,3,4]
  try:
    print (x[2])
  except:
    print ("Ha Ha Ha")
  finally:
    print ("Ho Ho Ho")

  try:
    print (x[5])
  except:
    print ("Error x is out of range")
  finally:
    print ("Ho Ho Ho")

if __name__ == "__main__":
  func_1_6_8()
```

```
3
Ho Ho Ho
Error x is out of range
Ho Ho Ho
```

ตัวอย่าง 1-6-8

การใช้งานลิสต์หรือทูเปิลนั้น เสี่ยงต่อการ error เพราะเราอาจเผลอไปเรียกข้อมูลที่ไม่มีอยู่จริง

เช่น

```
x = [1,2,3]
```

แล้วเราไปเรียก print(x[6]) ซึ่งมันไม่มี เพราะเรากำหนด x ไว้แค่สามค่า

ปกติก็จะเกิด error และโปรแกรมจะหยุดทำงาน ค้างแน่ๆ แต่เราสามารถดักจับ error ไว้ก่อนล่วงหน้า เพื่อป้องกันโปรแกรมหยุดทำงาน โดยคำสั่งชุด

```
try:
except:
finally:
```

โดยนำ try มาครอบโปรแกรมในส่วนที่อาจเกิด error ไว้ ซึ่งถ้าโปรแกรมของเราไม่มี error อะไร โปรแกรมก็จะทำงานไปจนจบตามปกติ และจะข้ามส่วน except ไป

แต่ถ้าหากโปรแกรมในส่วน try เกิด error กลางทางขณะทำงานอยู่ โปรแกรมจะกระโดดไปยังคำสั่งในบล๊อค except: โดยอัตโนมัติ ทำให้โปรแกรมเราทำงานจนจบได้ ไม่ค้าง ไม่หลุด

คำสั่ง try except นี้ยังเหมาะแก่การเอาไว้ครอบโปรแกรมส่วนที่คาดว่าจะ error หรือ error แล้วหาสาเหตุไม่ค่อยเจอ เพื่อจะป้องกันหรือดักจับการ error ด้วยก็ได้

```
Error x is out of range
```

ในบล็อคคำสั่ง try เมื่อเกิด error โปรแกรมจะข้ามมาทำงานในบล็อค except แต่ถ้าไม่มี error ก็จะทำงานในบล๊อค try จนจบแล้วก็หลุดจากบล๊อคไป สุดท้ายเลยมีอีกบล๊อคหนึ่งไว้ให้ใช้ ซึ่งจะใส่หรือไม่ใส่ไว้ก็ได้ คือ

```
finally:
```

บล็อค finally นี้จะทำงานเสมอ ไม่ว่าจะ error หรือไม่

คือถ้าไม่มี error โปรแกรมก็จะทำงานในบล็อค try ต่อด้วย finally แต่ถ้ามี error โปรแกรมจะข้ามมาทำงานในบล๊อค except แล้วต่อด้วย finally เช่นกัน เป็นการรับรองว่าโปรแกรมจะต้องมาทำงานในบล็อค finally แน่ๆ ไม่ว่าจะ error หรือไม่ แต่คำสั่ง finally: นี้ถ้าไม่ใช้ก็ตัดออกได้ ไม่มีผลอะไรกับโปรแกรม

Chapter17

FUNCTION

ฟังก์ชัน และการส่งค่าเข้าไปในฟังก์ชัน

1-7-1 การส่งค่าเข้าไปในฟังก์ชัน
1-7-2 ส่งสองค่าเข้าไปในฟังก์ชัน
1-7-3 ขอบเขตของตัวแปรในฟังก์ชัน
1-7-4 การส่งตัวแปรลิสต์เข้าไปในฟังก์ชัน
1-7-5 กำหนดค่าเริ่มต้นให้กับ parameter ของฟังก์ชัน
1-7-6 return คำสั่งคืนค่าออกมาจากฟังก์ชัน
1-7-7 return ค่าออกมาหลายๆ ตัว
1-7-8 return ได้หลายตำแหน่ง

```
#Python 3.9.5
#Example 1-7-1

def func_1_7_1(x):
   'การส่งค่าเข้าไปในฟังก์ชัน'

   print("x =",x)
if __name__ == "__main__":
   func_1_7_1(10)
   func_1_7_1("hello")
   func_1_7_1("สวัสดี")
   func_1_7_1(3+4)
   func_1_7_1([3,4,5])
   func_1_7_1("")
```

```
x = 10
x = hello
x = สวัสดี
x = 7
x = [3, 4, 5]
x =
```

ตัวอย่าง 1-7-1

มีคำสั่งหนึ่งที่ใช้กันมาตั้งแต่บทแรก แต่ไม่มีคำอธิบายคือ

```
def func_1_2_1():
```

จนถึง

```
def_1_6_8():
```

คำสั่ง def ตามด้วยชื่อ function_name ตามด้วย () และ : นี้ เป็นคำสั่งสำหรับการสร้าง function (ฟังก์ชัน) ด้วยภาษา Python

ซึ่งเมื่อเราเขียนคำสั่งอะไรก็ตามไว้ในบล็อคของฟังก์ชันแล้ว เราจะสามารถเรียกใช้งานฟังก์ชันนั้นกี่ครั้งก็ได้ โดยเรียกด้วย function_name() ซึ่งจะมีข้อดีคือ ทำให้เราไม่ต้องไปเขียนโปรแกรมเดิมซ้ำๆ

เช่น

```
def func():
        print(“hahaha”)
```

ทุกครั้งที่เราเรียก func() คอมพิวเตอร์ก็จะแสดงผล hahaha มาบนหน้าจอ โดยไม่ต้องสั่ง print ซ้ำอีก

บทนี้เราจะมาศึกษาการใช้งานฟังก์ชันให้ละเอียดขึ้น ว่าฟังก์ชันทำอะไรได้บ้าง

เริ่มจากตัวอย่างนี้ คือการส่งข้อมูลเข้าไปในฟังก์ชัน ซึ่งฟังก์ชันก็จะสามารถนำข้อมูลที่ส่งเข้าไป ไปใช้งานได้

จากตัวอย่างเมื่อเรากำหนดค่า หรือเรียกว่า parameter (พารามิเตอร์) ให้กับฟังก์ชันด้วย

```
def func_1_7_1(x):
```

x ที่อยู่ในวงเล็บนี้ อาจใช้เป็นตัวแปรอื่นก็ได้ จะเป็น a หรือ x หรือ xxx ก็ได้ ซึ่งในฟังก์ชันเรากำหนดให้

```
print("x =",x)
```

คือให้โปรแกรมพิมพ์หรือแสดงค่าของ x ออกมา

ดังนั้นใน main program หรือส่วนโปรแกรมหลัก เมื่อเราเรียก

```
func_1_7_1(10)
```

โปรแกรมก็จะแสดงค่า 10 บนหน้าจอ

```
func_1_7_1("hello")
```

โปรแกรมก็จะแสดงค่า hello บนหน้าจอ
..เป็นต้นใบดอกผล

เราอาจส่งค่าแบบอื่นๆ เข้าไปในฟังก์ชันได้อีก ดังตัวอย่าง

```python
#Python 3.9.5
#Example 1-7-2

def func_1_7_2(x,y):
    'ส่งค่าเข้าไปในฟังก์ชันสองตัว'

    print("x =",x)
    print("y =",y)
    print()

if __name__ == "__main__":
    func_1_7_2(10,20)
    func_1_7_2(10,"hello")
    func_1_7_2([123],(10,20,30))
    a = 10; b = 20
    func_1_7_2(a,b)
```

```
x = 10
y = 20

x = 10
y = hello

x = [123]
y = (10, 20, 30)

x = 10
y = 20
```

ตัวอย่าง 1-7-2

นอกจากจะส่งค่าหรือพารามิเตอร์เข้าไปฟังก์ชันตัวเดียวแล้ว สามารถส่งมากกว่าหนึ่งตัวก็ได้ สองตัวหรือหลายๆ ตัวก็ไม่ได้ผิดกติกา ดูตามตัวอย่าง

```
#Python 3.9.5
#Example 1-7-3

def func_1_7_3(x,y):
   'การเปลี่ยนแปลงค่าตัวแปรในฟังก์ชันไม่ส่งผลต่อตัวแปรนอกฟังก์ชัน'

   x+=1
   y+=2
   print("func: x=",x," y=",y)

if __name__ == "__main__":
   x=10; y=20
   print("x=",x," y=",y)
   func_1_7_3(x,y)
   print("x=",x," y=",y)
```

```
x= 10  y= 20
func: x= 11  y= 22
x= 10  y= 20
```

ตัวอย่าง 1-7-3

ตัวอย่างนี้ มีลูกเล่นนิดหน่อย คือในโปรแกรมหลัก หลังคำสั่ง if _name_==”_main_”: มีการเรียกใช้งานฟังก์ชัน โดยใช้ตัวแปร x และ y เป็นพารามิเตอร์

ตอนแรกเขียนบรรยายเรื่องนี้ยาวยืด แต่ตัดสินใจตัดออกหมด เพราะยิ่งอธิบายเยอะยิ่งงง เอาว่าลองทำความเข้าใจจากตัวอย่างครับ ลองเปลี่ยนชื่อตัวแปร parameter ของ func_1_7_3(x,y): เป็น func_1_7_3(a,b): ดูก็ได้ แล้วดูว่าเกิดอะไรขึ้น บทนี้ตัวอย่างสั้นๆ แต่ลึกซึ้ง เพราะว่าด้วยขอบเขตของตัวแปร ถ้าจะสรุปให้ง่ายๆ สั้นๆ แบบเข้าใจก็ดีไม่เข้าใจก็ช่าง คือ

ตัวแปรที่กำหนดในฟังชัน มีขอบเขตแค่ในฟังก์ชันเท่านั้น

...จริงๆ มีทริกสำหรับทำให้ตัวแปรในฟังก์ชันออกไปวอแวนอกฟังชันได้ด้วย แต่จะยิ่งทำให้ชีวิตลำบากมากขึ้นอีก ลองทำความเข้าใจเพียงนี้ดูก่อน

```
#Python 3.9.5
#Example 1-7-4

def func_1_7_4(x):
   'การส่งค่าตัวแปร list เข้าไปในฟังก์ชัน'

   x[0] +=10
   print ("Now x=",x)

if __name__ == "__main__":
   x=[10]
   print ("x=",x)
   func_1_7_4(x)
   print ("x=",x)
   func_1_7_4(x)
```

```
x= [10]
Now x= [20]
x= [20]
Now x= [30]
```

ตัวอย่าง 1-7-4

ในเมื่อเราสามารถส่งค่าตัวแปรเข้าไปเป็นพารามิเตอร์ของฟังก์ชันได้ ก็แปลว่าส่งลิสต์เข้าไปได้ด้วย

แต่การส่งลิสต์เข้าไปนี้มีเรื่องพิสดารนิดหนึ่ง คือ ปกติแล้วเมื่อเราส่งค่าตัวแปรทั่วไปจากโปรแกรมหลักเข้าไปในฟังก์ชัน แล้วมีการเปลี่ยนแปลงค่าตัวแปรนั้นในฟังก์ชัน ตัวแปรที่อยู่ในโปรแกรมหลักจะไม่ถูกเปลี่ยนแปลงค่าตามไปด้วย

แต่กรณีส่งลิสต์เข้าไปในฟังก์ชันนี้ เรื่องจะคล้ายๆ กับที่เคยเรียนรู้มาแล้วเรื่องการคัดลอกลิสต์ นั่นคือสิ่งที่ส่งเข้าไปในฟังก์ชันมีแต่ “ชื่อ” ของลิสต์เท่านั้น ไม่ได้ส่งค่าทั้งหมดในลิสต์เข้าไป ดังนั้นเมื่อมีการเปลี่ยนแปลง “ค่า” ในลิสต์จากในฟังก์ชัน จึงมีผลกระทบไปถึง “ค่า” ในลิสต์ที่อยู่ในโปรแกรมหลักด้วย ลองศึกษาจากตัวอย่างครับ

```
#Python 3.9.5
#Example 1-7-5

def func_1_7_5(x=10,y=20):
   'การกำหนดค่าเริ่มต้นให้กับตัวแปร parameter ของฟังก์ชัน'

   print(x,y)

if __name__ == "__main__":
   func_1_7_5()
   func_1_7_5(100)
   func_1_7_5(y=100)
   func_1_7_5(y="hello",x=100)
```

```
10 20
100 20
10 100
100 hello
```

ตัวอย่าง 1-7-5

เราสามารถกำหนดค่าเริ่มต้นให้กับพารามิเตอร์ของฟังก์ชันได้ ในตัวอย่างนี้

```
def func_1_7_5(x=10,y=20):
```

กำหนดให้พารามิเตอร์ x และ y มีค่าเริ่มต้นเป็น 10 และ 20 ไว้ก่อนเลย จากนั้นในโปรแกรมหลักถ้าเราเรียกชื่อฟังก์ชันโดยไม่ระบุค่าในพารามิเตอร์ ฟังก์ชันจะใช้ค่าเริ่มต้นที่ตั้งไว้ให้นี้เองโดยอัตโนมัติ

ฟังก์ชัน func_1_7_5(x=10,y=20) ของเรามีพารามิเตอร์สองตัว ซึ่งกำหนดค่าเริ่มต้นไว้ จากตัวอย่างจะเห็นว่าเราสามารถเรียกฟังก์ชันนี้โดยไม่ใส่ค่าพารามิเตอร์ หรือใส่ค่าเพียงตัวเดียว หรือระบุพารามิเตอร์ที่ต้องการส่งค่าเข้าไปก็ได้ ซึ่งตัวที่ไม่ระบุ โปรแกรมก็จะไปเอาค่าเริ่มต้นที่ตั้งไว้มาใช้เอง

```
#Python 3.9.5
#Example 1-7-6

def func_1_7_6(x):
   'คืนค่าออกจากฟังก์ชันด้วย return'

   x+=10
   return(x)

if __name__ == "__main__":
   y=0
   y=func_1_7_6(10)
   print(y)
   print (func_1_7_6(20))
```

```
20
30
```

ตัวอย่าง 1-7-6

เมื่อส่งค่าเข้าไปในฟังก์ชันได้ ก็มีวิธีส่งค่าออกมาจากฟังก์ชันได้ด้วยคำสั่ง

```
return
```

จากตัวอย่างจะเห็นวิธีการใช้ค่าที่ส่งออกมาจากฟังก์ชัน โดยฟังก์ชันส่งค่าตัวแปร x คืนออกมา

เอาตัวแปรมารับค่าก็ได้เช่น

```
y=func_1_7_6(10)
```

ฟังก์ชัน func_1_7_6(10) จะนำค่า 10 ไปคำนวณ แล้วส่งค่ากลับเข้ามาไว้ในตัวแปร y

หรือ

```
print(func_1_7_6(20))
```

ตัวอย่างนี้จะ print หรือแสดงผลค่าที่ส่งออกมาตรงๆ เลย โดยไม่ต้องเอาตัวแปรไปรองรับ

```
#Python 3.9.5
#Example 1-7-7

def func_1_7_7(x,y,z):
   'return ค่าหลายๆ ตัวเป็น tuple'

   return(x,y,z)

if __name__ == "__main__":
   print(func_1_7_7(1,10,20))
   print(func_1_7_7((1,2),[3,4],"hello"))
```

```
(1, 10, 20)
((1, 2), [3, 4], 'hello')
```

ตัวอย่าง 1-7-7

นอกจากจะ return ค่าออกมาจากฟังก์ชันเป็นค่าเดียวเดี่ยวๆ แล้ว เรายังสามารถส่งค่าออกมาด้วยคำสั่ง return ออกมาเป็นชุดได้ โดยชุดข้อมูลที่ออกมาจะกลายเป็น tuple

```
#Python 3.9.5
#Example 1-7-8

def func_1_7_8(x,y):
   'ฟังชันหนึ่ง มี return จากหลายจุดได้'

   if x>y:
      return x
   elif x<y:
      return y
   else:
      return "x=y!!"

if __name__ == "__main__":
   print(func_1_7_8(10,20))
   print(func_1_7_8(9,1))
   print(func_1_7_8(100,100))
```

```
20
9
x=y!!
```

ตัวอย่าง 1-7-8

ไม่มีอะไรพิเศษ แสดงให้เห็นเฉยๆ ว่าเราสามารถใส่คำว่า return ไว้ในโปรแกรมได้หลายจุด โดยเมื่อโปรแกรมทำงานไปเจอคำสั่ง return ก็จะหลุดออกจากฟังก์ชัน กลับสู่โปรแกรมหลักที่เรียกฟังก์ชันนี้มา

Chapter18

DICTIONARY, SET

ตัวแปรชนิด dictionary และ set

1-8-1 ตัวแปรชนิด dictionary
1-8-2 การลบข้อมูลใน dictonary
1-8-3 การเพิ่มข้อมูลใน dictonary
1-8-4 การอ่านและนับค่าใน dictionary
1-8-5 ตัวแปรชนิด set
1-8-6 ยูเนี่ยนและอินเตอร์เซ็คชัน สำหรับ set
1-8-7 ซับเซ็ต และซูเปอร์เซ็ต

```
#Python 3.9.5
#Example 1-8-1

def func_1_8_1():
   'ตัวแปรชนิด dictionary'

   x={ 'a': 'Hello',
      'b': 'World',
      'c': '!!!'
      }

   print (x)
   print (type(x))

if __name__ == "__main__":
   func_1_8_1()
```

```
{'a': 'Hello', 'b': 'World', 'c': '!!!'}
<class 'dict'>
```

ตัวอย่าง 1-8-1

รู้จักกับตัวแปรชนิด dictionary (ดิคชันนารี)

ตัวแปรชนิดนี้ว่าไปก็คล้ายๆ กับลิสต์ แต่ต่างตรงที่ว่าในลิสต์เราใส่ค่าเรียงๆ ไปแล้วเรียกใช้ค่าในลิสต์จากตำแหน่งของมัน

แต่ dictionary นี้จะคล้ายๆ กับหนังสือพจนานุกรมซึ่งจะมีคำศัพท์และคำแปลควบคู่กัน ซึ่งในภาษา Python นี้เราจะเรียกว่า

key (คีย์) และ value

(คำนี้เวลาเขียนเป็นไทยจะเขียนว่าวาลู หรือแวลู แต่ฝรั่งส่วนมากอ่านว่าวายยู่)

เปรียบเทียบง่ายๆ dictionary ก็เหมือนกับกล่องเยอะๆ หน้ากล่องมีป้ายแปะชื่อกล่องไว้ ซึ่งเรียกว่า key และข้อมูลในกล่องเรียกว่า value

เริ่มจากทดลองสร้างตัวแปรชนิด dictionary กันก่อน

```
#Python 3.9.5
#Example 1-8-2

def func_1_8_2():
  'การลบข้อมูลใน dictionary'

  x={ 'Jo': '123 Bangkok',
    'Mo': '456 Nonthaburi',
    'Yo': '789 Chiangmai'
    }

  for name,address in x.items():
    print (name,address)

  del x['Jo']
  print ('\n--- Delete Key Jo ---\n')

  for name,address in x.items():
    print (name,address)

if __name__ == "__main__":
  func_1_8_2()
```

```
Jo 123 Bangkok
Mo 456 Nonthaburi
Yo 789 Chiangmai

--- Delete Key Jo ---

Mo 456 Nonthaburi
Yo 789 Chiangmai
```

ตัวอย่าง 1-8-2

การลบข้อมูลใน dictionary ดูเอาจากตัวอย่างได้เลย

เราใช้คำสั่ง for เพื่ออ่านค่าจาก dictionary

ใช้ตัวแปรสองตัวมาอ่านค่าจาก key และ value คือตัวแปรชื่อ name กับ address โดยอ้างถึงข้อมูลใน dictionary ด้วยคำสั่ง

```
var_name.items()
```

แล้วนำมาแสดงผลด้วยคำสั่ง print()

การลบข้อมูลใน dictionary ทำได้ง่ายๆ ด้วยคำสั่ง

```
del var_name['Key']
```

ตามตัวอย่างคือ

```
del x['Jo']
```

ข้อมูลชุดนั้นก็จะถูกลบออกไปจาก dictionary ทั้ง key และ value

ส่วนการเพิ่มค่ายิ่งง่ายใหญ่ แค่กำหนดค่าใหม่เข้าไปเฉยๆ เลย

รูปแบบคือ

```
var_name['Key'] = 'Value'
```

ดังตัวอย่าง

```
x['Go'] = '200 Chiangrai'
```

```
#Python 3.9.5
#Example 1-8-3

def func_1_8_3():
   'การเพิ่มข้อมูลใน dictionary'

   x={ 'Jo': '123 Bangkok',
      'Mo': '456 Nonthaburi',
      'Yo': '789 Chiangmai'
      }

   for name,address in x.items():
      print (name,address)

   print ('\n--- Add Key Go and Do ---\n')
   x['Go'] = '200 Chiangrai'
   x['Do'] = '321 Lumpoon'

   for name,address in x.items():
      print (name,address)

if __name__ == "__main__":
   func_1_8_3()
```

```
Jo 123 Bangkok
Mo 456 Nonthaburi
Yo 789 Chiangmai

--- Add Key Go and Do ---

Jo 123 Bangkok
Mo 456 Nonthaburi
Yo 789 Chiangmai
Go 200 Chiangrai
Do 321 Lumpoon
```

ตัวอย่าง 1-8-3

ส่วนการเพิ่มค่ายิ่งง่ายใหญ่ แค่กำหนดค่าใหม่เข้าไปเฉยๆ

รูปแบบคือ

```
var_name[‘Key’] = ‘Value’
```

ดังตัวอย่าง

```
x[‘Go’] = ‘200 Chiangrai’
x['Do'] = '321 Lumpoon'
```

```
#Python 3.9.5
#Example 1-8-4

def func_1_8_4():
   'การอ่านและนับค่าใน dictionary'
   x={}

   x['A'] = 'Ada'
   x['B'] = 'Basic'
   x['C'] = 'Cobol'

   for first, second in x.items():
      print (first,second)

   print ("***************")
   if 'A' in x:
      print ("A for",x['A'])

   print ("***************")
   print(x)
   print (x.items())
   print ("number of item in x=",len(x.items()))

if __name__ == "__main__":
   func_1_8_4()
```

```
A Ada
B Basic
C Cobol
***************
A for Ada
***************
{'A': 'Ada', 'B': 'Basic', 'C': 'Cobol'}
dict_items([('A', 'Ada'), ('B', 'Basic'), ('C', 'Cobol')])
number of item in x= 3
```

ตัวอย่าง 1-8-4

ทำตัวอย่างเรื่องการเพิ่ม-ลบข้อมูลใน dictionary ไปแล้ว ถึงนึกได้ว่าก่อนจะเพิ่มลดอะไรควรเริ่มจากการอ่านและนับค่าใน dictionary ให้ได้ก่อนสินะ แต่ไม่เป็นไร เรียนอะไรก่อนอะไรหลังก็ได้เหมือนกัน

สำหรับการอ่านค่าใน dictionary นั้นง่ายมาก คือเขียนคล้ายๆ กับ list เพียงแต่แทนที่จะอ้างถึงค่าใน list ด้วยตำแหน่ง เช่น x[0], x[2] อะไรแบบนี้ เราก็อ้างด้วย key แทน เช่น x[‘A’] หรือ x[‘cnx’]

การนับข้อมูลใน dictionary ใช้คำสั่ง

```
len(var_name.items())
```

แล้วก็มาดูกันว่าถ้าสั่ง

```
print (x.items())
```

จะได้อะไรออกมา

```
#Python 3.9.5
#Example 1-8-5

def func_1_8_5():
   'เซ็ต'

   x = set((3,5,7,8,3,8,3))
   y = x.copy()

   print ("1)",x,y)
   x.add(22)
   print ("2)",x,y)
   y.add(3)
   print ("3)",x,y)
   x.remove(3)
   print ("4)",x,y)

if __name__ == "__main__":
   func_1_8_5()
```

```
1) {8, 3, 5, 7} {8, 3, 5, 7}
2) {3, 5, 7, 8, 22} {8, 3, 5, 7}
3) {3, 5, 7, 8, 22} {8, 3, 5, 7}
4) {5, 7, 8, 22} {8, 3, 5, 7}
```

ตัวอย่าง 1-8-5

ตัวแปรอีกชนิดที่มีประโยชน์มากสำหรับคนที่เรียนคณิตศาสตร์คือตัวแปรชนิด set (เซ็ต) หน้าตาเซ็ตก็คล้ายๆ ลิสต์ครับ ความแตกต่างคือเซ็ตจะเป็นการแจกแจงข้อมูล ข้อมูลในเซ็ตจึงไม่มีข้อมูลซ้ำกัน อย่างเช่นถ้าเจอ 3 สองตัวในเซ็ต ก็จะถือว่ามีตัวเดียว และมีตัวดำเนินการเฉพาะซึ่งสามารถนำไปใช้ประโยชน์ทางคณิตศาสตร์หรือใช้ทำการบ้านได้

มาดูตัวอย่างกันก่อน

```
x = set([3,5,7,8,3,8,3])
```

ข้อมูลใน set() จะเป็นลิสต์หรือทูเปิลก็ได้ ดังนั้นตรงนี้จะเขียนเป็น

```
x = set((3,5,7,8,3,8,3))
```

ก็ได้

```
y= x.copy()
```
คัดลอกข้อมูลใน x ไปใส่ y

```
x.add(22)
```
เพิ่ม 22 เข้าไปในเซ็ต x

```
x.add(3)
```
จะเห็นว่าไม่มีผลอะไร เพราะ x มี 3 เป็นสมาชิกอยู่แล้ว

```
x.remove(3)
```
ลบ 3 ออกจากเซ็ต เข้าใจง่ายมาก

```
#Python 3.9.5
#Example 1-8-7

def func_1_8_7():
   'ตัวดำเนินการตรวจสอบการเป็นซับเซ็ตหรือซูเปอร์เซ็ต'

   x= set([3,5,7,9])
   y= set([1,3,5,7,9,11])

   print ("y is superset of x is",y.issuperset(x))
   print ("x is subset of y is",x.issubset(y))
   print ("set(3,7) is subset of y is",set((3,7)).issubset(y))
   if x < y: print ("x<y is",x<y)

if __name__ == "__main__":
   func_1_8_7()
```

```
{8, 5, 22, 7} {8, 3, 5, 7}
**************************
{8, 5, 7}
{8, 5, 7}
{3, 5, 7, 8, 22}
{3, 5, 7, 8, 22}
```

ตัวอย่าง 1-8-6

สำหรับคนที่เรียนเรื่องเซ็ตมาแล้ว จะคุ้นเคยกับตัวดำเนินการสำหรับเซ็ตสองแบบคือ

intersection (อินเตอร์เซ็คชัน) คือการเลือกค่าที่ซ้ำกันออกมาจากเซ็ต ใช้คำสั่ง

```
x.intersection(y)
```

หรือ

```
x & y
```

union (ยูเนี่ยน) คือการรวมเซ็ตสองเซ็ตเข้าด้วยกัน

```
x.union(y)
```

หรือ

```
x | y
```

ดังตัวอย่าง

```
#Python 3.9.5
#Example 1-8-6

def func_1_8_6():
   'ตัวดำเนินการอินเตอร์เซ็คชั่น และยูเนี่ยน สำหรับเซ็ต'

   x = set([5, 7, 8, 22])
   y = set([8, 3, 5, 7])
   print (x,y)
   print ("**************************")
   print (x.intersection(y))
   print (x & y)

   print (x.union(y))
   print (x | y)

if __name__ == "__main__":
   func_1_8_6()
```

```
y is superset of x is True
x is subset of y is True
set(3,7) is subset of y is True
x<y is True
```

ตัวอย่าง 1-8-7

มีคุณสมบัติอีกสองอย่างสำหรับเซ็ตคือ ซับเซ็ต (subset) และ ซูเปอร์เซ็ต (superset)

ซับเซ็ต คือเซ็ตที่มีสมาชิกทุกตัวเป็นสมาชิกของอีกเซ็ตหนึ่ง

และในมุมกลับ อีกเซ็ตก็ถือเป็นซูเปอร์เซ็ตของเซ็ตแรก

ซึ่งเราสามารถตรวจสอบการเป็นซับเซ็ตและซูเปอร์เซ็ตได้ ดังตัวอย่าง

Chapter19

OOP

การเขียนโปรแกรมเชิงวัตถุ Object Oriented Programming

1-9-1 การสร้าง class
1-9-2 initializer หรือ constructor _init_
1-9-3 การคืนค่า string ออกจาก object ด้วย method _str_
1-9-4 destructor _del_
1-9-5 1 object และ 2 instances
1-9-6 2 objects และ 2 instances
1-9-7 parameter และ attribute
1-9-8 กำหนดค่าเริ่มต้นให้ parameter ของ class
1-9-9 inheritance การสืบทอด class
1-9-10 method overriding
1-9-11 การสืบทอดจากหลาย class
1-9-12 ตัวอย่างการใช้งาน object
1-9-13 object docstring

```
#Python 3.9.5
#Example 1-9-1

class Cat:
   'Class name Cat'

   def say(self):
      'Method name say'
      print ("Meow")

if __name__ == "__main__":
   tom = Cat()
   tom.say()
   print ("*****************")
   print ("Type of tom is",type(tom))
   print ("Type of tom.say is",type(tom.say))
   print ("Class's doc is",tom.__doc__)
   print ("Method's doc is",tom.say.__doc__)
```

```
Meow
*****************
Type of tom is <class '__main__.Cat'>
Type of tom.say is <class 'method'>
Class's doc is Class name Cat
Method's doc is Method name say
```

ตัวอย่าง 1-9-1

Class (คลาส) เป็นการเขียนโปรแกรมแบบ Object Oriented Programming (ออบเจ็คโอเรียนเต็ด) หรือการเขียนโปรแกรมเชิงวัตถุ ภาคทฤษฎีค่อยว่ากัน ดูตัวอย่างก่อน

```
Class Cat:
        def say(self):
                print (“Meow”)
```

เป็นการสร้าง class ชื่อ Cat ซึ่งมีสมาชิกเป็นฟังก์ชัน หรือเวลาอยู่ในคลาสเราจะเรียกฟังก์ชันว่า method (เมธอด) ชื่อ say(self)

ทีนี้มาดูในส่วนโปรแกรมหลัก

```
tom = Cat()
```

เป็นการกำหนดให้ tom เป็น object (ออบเจ็ค) หรือเป็นตัววัตถุ ที่สร้างขึ้นจากแม่พิมพ์ class Cat

ซึ่งเมื่อสร้าง tom ขึ้นมาจาก class Cat แล้ว tom ก็จะใช้ความสามารถทุกอย่างหรือใช้สมาชิกต่างๆ ที่อยู่ใน class Cat ได้ ซึ่งตอนนี้ class Cat มีสมาชิกให้เรียกใช้งานได้อย่างเดียวคือ say(self) ส่วน self มีไว้ทำไม ยังไม่ต้องสนใจตอนนี้

```
tom.say()
```

เป็นการเรียกใช้งานเมธอดชื่อ say() ของ class Cat

ซึ่งก็จะแสดงผลออกมาเป็น

```
Meow
```

ส่วนคำสั่งถัดๆ มาคือ print ชนิดของตัวแปร tom และ tom.say รวมถึงการ print document ของคลาสออกมาดูเล่น

```
#Python 3.9.5
#Example 1-9-2

class Cat:
  'Initializer หรือ Constructor'

  def __init__(self):
    print ("Aow")

  def say(self):
    print ("Meow")

if __name__ == "__main__":
  tom = Cat()
  tom.say()
  tom.say()
  print()

  Cat()
  print()

  Cat().say()
```

```
Aow
Meow
Meow

Aow

Aow
Meow
```

ตัวอย่าง 1-9-2

ตัวอย่างนี้เราเพิ่มสมาชิกหรือเมธอดของ class Cat เป็นสองตัว

```
__init__(self)
```

เรียกว่า initializer (อินิเชียลไลเซอร์) จะเป็นเมธอด ที่ทำงานโดยอัตโนมัติเมื่อออบเจ็คของคลาสนั้นถูกสร้างขึ้น หรือถูกเรียกใช้ ตรงนี้บางตำราอาจเรียกเป็น constructor (คอนสตรัคเตอร์) ก็ไม่ต้องตกใจ เพราะ _init_ ใน Python ถูกใช้ในฐานะของ constructor เมื่อเทียบกับภาษาที่เป็น OOP ภาษาอื่นๆ

ส่วน constructor จริงๆ ใน Python คือ

```
__new__(self)
```

แต่เราจะยังไม่ใช้ตัวนี้

ย้ำอีกที ฟังก์ชันใน class เราจะเรียกว่า method (เมธอด) นะ

ส่วนคำสั่งถัดๆ มาคือ print ชนิดของตัวแปร tom และ tom.say รวมถึงการ print document ของคลาสออกมาดูเล่น

```
#Python 3.9.5
#Example 1-9-3

class Cat:
  'คืนค่า string ออกมาจาก object'

  def __init__(self):
    print ("Aow")
  def say(self):
    print ("Meow")
  def __str__(self):
    return ("I am a Cat")

if __name__ == "__main__":
  tom = Cat()
  tom.say()
  print (tom)
```

```
Aow
Meow
I am a Cat
```

ตัวอย่าง 1-9-3

```
__str__(self)
```

เป็น method พิเศษตัวหนึ่ง เอาไว้ return ค่าเป็นสายอักษรหรือ string ออกมาจากออบเจ็ค

ค่อยๆ เติมทฤษฎีกันทีละเล็กละน้อยนะ

```
tom=Cat()
```

คือการสร้าง object (ออบเจ็ค) โดยมี instance (อินสแตนซ์) ชื่อ tom จาก class Cat

ดังนั้นเราจะเรียก tom ว่า instance และ tom เป็น object

เราอาจคิดในแง่ที่ว่า class คือแม่พิมพ์แมว ทีนี้เราปั๊มแมวออกมาตัวนึง แมวก็เป็น object ที่เกิดจากพิมพ์เขียว class นะ

จากนั้นพอปั๊ม object แมวออกมาแล้วเราก็ต้องตั้งชื่อ ซึ่งจากตัวอย่างคือเราตั้งชื่อ object แมวนี้ว่า tom และเพื่อไม่ให้สับสน (หรือยิ่งสับสนมากขึ้น) เราก็เรียก tom ว่าเป็น instance ของ object ที่มาจาก class Cat

ซึ่ง tom ก็มีความสามารถตามที่ระบุไว้ในพิมพ์เขียวแมวคือ

```
say(self)
```
ได้ Meow

บางคนหรือบางตำราก็เรียก tom เป็น object บ้าง หรือเป็น instance บ้าง ก็ไม่ได้ผิดกติกาครับ เพราะก็ใกล้ๆ กันอยู่ ใช้แทนกันพอได้

และเรียก function ใน class ว่า method (เมธอด) –- ห้ามลืม

```
#Python 3.9.5
#Example 1-9-4

class Cat:
  'destructor'

  def __init__(self):
    print ("Aow")
  def say(self):
    print ("Meow")
  def __str__(self):
    return ("I am a Cat")
  def __del__(self):
    print ("I am killed")

if __name__ == "__main__":
  tom = Cat()
  tom.say()
  del tom
```

```
Aow
Meow
I am killed
```

ตัวอย่าง 1-9-4

เราเรียกเมธอด

```
__del__(self)
```

ว่า destructor (เดสตรัคเตอร์) คือเป็นเมธอดที่จะทำงานโดยอัตโนมัติเมื่อมีการลบ หรือทำลายออบเจ็คนั้นๆ

ในกรณีนี้ _del_(self) ทำงานเมื่อ

```
del tom
```

ซึ่งเป็นการสั่งลบ object tom ก็จะได้

```
I am killed
```

```
#Python 3.9.5
#Example 1-9-5

class Cat:
  '1 object, 2 instances'

  def __init__(self):
    print ("Aow")
  def say(self):
    print ("Meow")
  def __str__(self):
    return ("I am a Cat")
  def __del__(self):
    print ("AAAAAAHHH")

if __name__ == "__main__":
  tom = Cat()
  garfield = tom
  print ("--Object Created--")

  tom.say()
  garfield.say()

  print ("---Kill tom---")
  del tom

  #tom.say()
  garfield.say()

  print ("---Kill garfield---")
  del garfield
```

```
Aow
--Object Created--
Meow
Meow
---Kill tom---
Meow
---Kill garfield---
AAAAAAHHH
```

ตัวอย่าง 1-9-5

ตัวอย่างนี้ค่อยๆ ทำความเข้าใจ ค่อยๆ สังเกตไป ลองเล่นหลายแบบ

```
tom=Cat()
```

สังเกตว่า initializer ทำงาน print (“Aow”)

```
garfield = tom
```

ตรงนี้ initializer ไม่ทำงานครับ เพราะเป็นการสร้าง instance ชื่อ garfield บน object เดิม

ดังนั้น object นี้มี 2 ชื่อ หรือ 2 instance คือ tom และ garfield

```
tom.say()
```
ก็
```
Meow
```

```
garfield.say()
```
ก็
```
Meow
```

ก็เหมือนแมวมีสองชื่อ เรียกยังไงก็หันเหมือนกัน

```
del tom
```

ไม่มีอะไรเกิดขึ้น destructor ไม่ทำงาน เพราะ object นี้ยังไม่ตาย ยังมี garfield อยู่ แค่บอกว่าจะขอลบชื่อออกไปชื่อนึงนะ ยังเหลืออีกชื่อ

```
#tom.say()
```

เอา #แปะไว้เพื่อไม่ให้คำสั่งนี้ทำงาน เพราะจริงๆ เราลบ tom ไปแล้ว ถ้าสั่ง tom.say ก็จะ error อันนี้จะลองดูให้ error เล่นๆ ก็ได้โดยเอา # ออก ให้เหลือ tom.say()

```
garfield.say()
```
ได้
```
Meow
```
ตามปกติ

```
Del garfield
```

คราวนี้ลบจริงตายจริงเพราะเหลือชื่อเดียวแล้ว object โดนลบ destructor _del_(self) ทำงาน ก็จะได้

```
AAAAAAHHH
```

```
#Python 3.9.5
#Example 1-9-6

class Cat:
  '2 objects, 2 instances'

  def __init__(self):
    print ("Aow")
  def say(self):
    print ("Meow")
  def __str__(self):
    return ("I am a Cat")
  def __del__(self):
    print ("AAAAAAHHH")

if __name__ == "__main__":
  tom = Cat()
  garfield = Cat()

  tom.say()
  garfield.say()

  print ("---Kill tom---")
  del tom

  #tom.say()
  garfield.say()

  print ("---Kill garfield---")
  del garfield
```

```
Aow
Aow
Meow
Meow
---Kill tom---
AAAAAAHHH
Meow
---Kill garfield---
AAAAAAHHH
```

ตัวอย่าง 1-9-6

คราวนี้ลองอีกแบบ

```
tom = Cat()
_init_ ทำงาน ได้
Aow

garfield = Cat()
_init_ ทำงาน ได้
Aow
```

กรณีนี้เราสร้าง 2 object

โดย instance garfield เป็นคนละ object กับ instance tom ไม่เกี่ยวกัน

เพียงแต่ว่าสร้างขึ้นจากพิมพ์เขียว class Cat() ด้วยกัน

```
tom.say()
ก็
Meow

garfield.say()
ก็
Meow
```

ยังไงก็ปั๊มออกมาจากพิมพ์เขียว class Cat ด้วยกันนี่นะ

```
del tom
_del_ ทำงาน ได้
AAAAAAHHH
```

เพราะ tom มีชีวิตเดียว instance เดียว โดนฆ่าแล้วตายเลย

ดังนั้นถ้าสั่ง tom.say() อีกหลังจากนี้ก็ error แน่ๆ นอนๆ

แต่ garfield.say() ยังได้ Meow ยังมีชีวิตปกติดีอยู่

และสุดท้าย

```
del garfield
```

```
__del__(self)
```

ทำงาน ได้
AAAAAAHHH

```
#Python 3.9.5
#Example 1-9-7

class Cat:
   'parameter และ attribute'

   def __init__(self,n,c):
      self.name = n
      self.color = c
   def say(self):
      print ("My name is",self.name)

if __name__ == "__main__":
   tom = Cat("tom","B&W")
   tom.say()
   print (tom.color)
   tom.name = "TOM"
   tom.color = "Black & White"
   tom.say()
   print (tom.name,"is",tom.color)
```

```
My name is tom
B&W
My name is TOM
TOM is Black & White
```

ตัวอย่าง 1-9-7

ตัวอย่างนี้แสดงการส่ง parameter หรือส่งค่าตอนสร้าง object ด้วย โดยกำหนดค่าที่จะส่งไว้ที่

```
__init__(self,n,c):
```

n กับ c นี่ตั้งเป็นชื่ออะไรก็ได้นะ อันนี้ตั้งแบบมักง่ายเฉยๆ

```
self.name = n
self.color = c
```

สองบรรทัดนี้เป็นการสร้างตัวแปรที่เรียกว่า attribute (แอททริบิวต์) ของ class หรือเป็นตัวแปรสำหรับเอาไว้ใช้งานกันใน class นั้นๆ

```
tom= Cat(“tom”,”B&W”)
```

ส่งค่า tom และ B&W เข้าไปในตัวแปร n และ c
ซึ่งใน _init_ จะสั่งให้ self.name = n และ self.color = c

ได้ใช้ self เสียที หลังจากใส่ไว้ทำไมไม่รู้ตั้งนาน

เราใช้ self แทนตัวออบเจ็คนี้เอง หรือเรียกว่าแทนตัวเองครับ เพื่อให้รู้ว่า self.name และ self.color เป็น attribute ในคลาสนี้นะ สามารถเรียกใช้งานเมื่อไรก็ได้ในขอบเขตของคลาสหรือออบเจ็คที่สร้างขึ้น

ซึ่งก็จะเห็นว่าในเมธอด say เราเอา class attribute self.name ไปใช้งานได้เลย

นอกจากนี้เรายังอ่านค่าหรือเปลี่ยนแปลงค่าใน class attribute ได้โดยเรียกผ่านตัว object ก็ได้ ดังตัวอย่าง

```
#Python 3.9.5
#Example 1-9-8

class Animal:
  'กำหนดค่าตั้งต้นสำหรับพารามิเตอร์ของคลาส'

  def __init__(self,common_name='Cat'):
    self.common_name = common_name
  def say(self):
    print ("I am",self.common_name)

if __name__ == "__main__":
  tom = Animal()
  tom.say()
  tweety = Animal("Bird")
  tweety.say()
```

```
I am Cat
I am Bird
```

ตัวอย่าง 1-9-8

คล้ายๆ กับฟังก์ชันครับ คือเราสามารถกำหนดค่าตั้งต้นให้กับ parameter ของ class ได้ ถ้าไม่มีการส่งค่าเข้ามา เวลาสร้าง objext ก็จะใช้ค่าจาก parameter ตั้งต้นนี้ได้เลย

ดูจากตัวอย่างได้เลย เข้าใจไม่ยาก

```
#Python 3.9.5
#Example 1-9-9

'inheritance'

class Animal:
  def __init__(self,common_name='cat'):
    self.common_name = common_name
  def say(self,say='...'):
    print ("I am",self.common_name,say)

class Bird(Animal):
  def __init__(self):
    self.common_name = 'bird'
  def sing(self):
    print ("tweet tweet tweety")

if __name__ == "__main__":
  tom = Animal()
  tom.say()
  tom.say("Meow")
  print ("***********")

  tweety = Bird()
  tweety.say()
  tweety.say("Tweet")
  tweety.sing()
```

```
I am cat ...
I am cat Meow
***********
I am bird ...
I am bird Tweet
tweet tweet tweety
```

ตัวอย่าง 1-9-9

Class สามารถออกลูกออกหลานสืบทอดต่อๆ กันไปได้ เรียกว่าการ inherit (อินเฮอริท) หรือ inheritance (อินเฮอริแทนซ์) ในตัวอย่าง class Bird(Animal) เป็นลูกของ class Animal ทำให้ object ที่สร้างจาก class Bird สืบทอดคุณสมบัติ สามารถใช้เมธอดของ class Animal ได้ด้วย

และพิเศษกว่านั้นคือ class ลูก สามารถสร้าง method ของตัวเองเพิ่มขึ้นมาที่แตกต่างจาก class แม่ได้ด้วย

```
#Python 3.9.5
#Example 1-9-10

'method overriding'
class Animal:
  def __init__(self,common_name='animal'):
    self.common_name = common_name
  def say(self,words='...'):
    print ("I am",self.common_name,words)

class Bird(Animal):
  def __init__(self):
    self.common_name = 'bird'
  def sing(self):
    print ("tweet tweet tweety")

class Cat(Animal):
  def __init__(self):
    self.common_name = 'cat'
  def say(self):
    print ("Meow")

if __name__ == "__main__":
  tom = Animal()
  tom.say()
  tom.say("ha ha ha")
  print ("************\n")

  tweety = Bird()
  tweety.say()
  tweety.sing()
  print ("************\n")

  garfield = Cat()
  garfield.say()
  print (garfield.common_name)
```

I am animal ...
I am animal ha ha ha
****

I am bird ...
tweet tweet tweety
****

Meow
cat

ตัวอย่าง 1-9-10

ตัวอย่างนี้เราสร้าง class Bird และ class Cat ซึ่ง inherit หรือสืบทอดมาจาก class Animal

class Animal มี method say()
class Bird เพิ่ม method sing()

ดังนั้น object ที่สร้างจาก class Bird สามารถ say() (inherit จาก class แม่) และ sing() ได้

นอกจากนี้ class Cat สร้าง method say() ใหม่ขึ้นมาเองที่ทำงานไม่เหมือน say() เดิมของ class Animal เราเรียกกระบวนการนี้ว่า method overriding คือการเขียน method ใหม่ขึ้นมาในชื่อเดิมที่เคยมีมาแล้วในคลาสแม่

ดังนั้น object ที่สร้างจาก class Cat ก็จะ say() ได้ แต่คนละ say() กับ class แม่

```
#Python 3.9.5
#Example 1-9-11

'multiple inheritance'
class Animal:
  def __init__(self,common_name='animal'):
    self.common_name = common_name
  def say (self,words='...'):
    print ("I am",self.common_name,words)

class Action:
  def walk(self):
    print ("walk")
  def run(self):
    print ("run")

class Cat(Animal,Action):
  def run(self):
    print ("run run run")
  def dance(self):
    self.walk()
    self.run()
    self.walk()


if __name__ == "__main__":
  tom = Cat()
  print (tom.common_name)
  tom.dance()
  print ("**********\n")
  tom.say("ha ha ha")
  tom.walk()
  tom.run()
```

```
animal
walk
run run run
walk
**********

I am animal ha ha ha
walk
run run run
```

ตัวอย่าง 1-9-11

ตัวอย่างการสร้าง class ที่สืบทอดจาก สอง class โดย class Cat สืบทอดจาก class Animal และ Action ทำให้สามารถใช้ method หรือความสามารถของ class พ่อแม่ได้ทั้งสอง class เลย เรียกว่า multiple inheritance (มัลติเพิล อินเฮริแทนซ์)

สังเกต method dance() ของ class Cat จะเห็นรูปแบบการเรียกใช้ method ที่สืบทอดมาด้วย

```
#Python 3.9.5
#Example 1-9-12

'intention no description'

class Animal:
  name = 'no name'
  action = 'no action'
  common_name = 'no common name'

  def say(self):
    print ("I am",self.name,",",self.action)
  def animal_say(self):
    print ("Animal say",Animal.name,",",Animal.action)


if __name__ == "__main__":
  print ("1)")
  tom = Animal()
  print (tom.name,tom.action)
  print (Animal.name,Animal.action)
  tom.say()
  tom.animal_say()
  print ("**********\n")

  print ("2)")
  tom.name = "Tom"
  tom.action = "running"
  print (tom.name,tom.action)
  print (Animal.name,Animal.action)
  tom.say()
  tom.animal_say()
  print (“**********\n")

```     ต่อ     ```

```
print ("3)")
Animal.name = "..."
Animal.action = "..."
jerry = Animal()
tom.say()
jerry.say()
print (Animal.name,Animal.action)
print ("************\n")

print ("4)")
tom.animal_say()
jerry.animal_say()
```

```
1)
no name no action
no name no action
I am no name , no action
Animal say no name , no action
************

2)
Tom running
no name no action
I am Tom , running
Animal say no name , no action
************

3)
I am Tom , running
I am ... , ...
... ...
************

4)
Animal say ... , ...
Animal say ... , ...
```

ตัวอย่าง 1-9-12

ไม่อธิบาย ลองศึกษาจากตัวอย่างดู

```
#Python 3.9.5
#Example 1-9-13

'Program Document'

class Animal():
  'Class Animal'

  def say(self):
    'Method Say'
    print ("hi")

if __name__ == "__main__":
  tom = Animal()
  tom.say()
  print ("************\n")

  print(__doc__)
  print (Animal.__doc__)
  print (Animal.say.__doc__)
  print (__name__)
  print (Animal.__name__)
  print (Animal.say.__name__)
```

```
hi
************

Program Document
Class Animal
Method Say
__main__
Animal
say
```

ตัวอย่าง 1-9-13

ตัวอย่างง่ายๆ ก่อนจบเรื่องนี้ ทบทวนเรื่อง docstring กันอีกที

Chapter110

FILE

การทำงานกับไฟล์

1-10-1 การสร้างไฟล์เก็บข้อมูล
1-10-2 การ append หรือเขียนข้อมูลต่อท้ายไฟล์เดิม
1-10-3 การเปิดไฟล์สำหรับอ่านอย่างเดียว read only และคำสั่ง seek
1-10-4 การเขียนภาษาไทยลงในไฟล์

```
#Python 3.9.5
#Example 1-10-1

def func_1_10_1():
  'สร้าง และเขียนข้อมูลลงในไฟล์ ถ้ามีไฟล์ชื่อเดิมอยู่จะสร้างใหม่ทับไป'
  try:
    with open("test.txt","w") as f:
      f.write ("hello, world!")
      f.write ("123 456 789\n")
      print ("HELLO", file=f)
      print ("WORLD", file=f)
    print ("File Created!!!")
  except:
    print ("Cannot open file!!!")

if __name__ == "__main__":
  func_1_10_1()
```

ถ้าเปิดไฟล์ไม่สำเร็จ

```
Cannot open file!!!
```

ถ้าเปิดไฟล์สำเร็จ

```
File Created!!!
```

และในโพลเดอร์ที่เก็บโปรแกรมจะมีไฟล์ชื่อ test.txt โผล่ขึ้นมา ถ้าเปิดไฟล์ดูข้างในจะมีข้อความแบบนี้

```
hello, world!123 456 789
HELLO
WORLD
```

ตัวอย่าง 1-10-1

ตัวอย่างเรื่องการใช้ไฟล์นี่ดูจะเชยๆ ไปหน่อย เพราะในการใช้งานจริงเรามักจะไม่ค่อยใช้กลุ่มคำสั่งชุดนี้ จนบางคนลืมไปแล้วด้วยซ้ำว่าภาษา Python มีคำสั่งที่ทำงานกับไฟล์ได้โดยตรง แต่ยังไงก็เรียนรู้ไว้สักหน่อยก็ดี ระหว่างที่ยังไม่รู้จักวิธีติดต่อกับหน่วยความจำสำรองหรือระบบจัดเก็บข้อมูลวิธีอื่นๆ การใช้ชุดคำสั่งเกี่ยวกับไฟล์ก็เป็นวิธีการที่ใช้งานได้อยู่

ตัวอย่างนี้แสดงการใช้คำสั่งสร้างไฟล์แบบง่ายๆ ด้วยคำสั่ง

```
with open(<file_name>, “w”) as variable_name:
```

เราสามารถเปิดไฟล์โดยไม่ต้องใช้ with นำหน้าก็ได้ แต่คำแนะนำทั่วไปคือเขียนในลักษณะนี้ดีกว่า เพราะจะได้ไม่ลืมปิดไฟล์ ลดปัญหาหลายๆ อย่างได้ เพราะพอจบบล็อค with ไฟล์จะถูกปิดเองด้วยเลย

และเรานิยมเอาบล็อค try except มาครอบไว้เพื่อป้องกัน error เพราะคำสั่งชุดเกี่ยวกับไฟล์นี้มีโอกาสเกิด error สูงมาก และเกิดกันตลอดเวลา เช่นเขียนไฟล์ไม่ได้บ้างอะไรบ้าง ดังนั้นจึงมักจะต้องป้องกันการเกิด error หรือดักจับ error ไว้ด้วยเสมอ

```
with open (“test.txt”,”w”) as f:
```

จะสร้างไฟล์ชื่อ test.txt ในโพลเดอร์เดียวกับที่เก็บโปรแกรมนี้

“w” คือ mode (โหมด) ของไฟล์ ปกติก็มี r, w, a, r+, w+, a+, rb, wb, ab, rb+, wb+, ab+ โดยมีความหมายคือ

r ถ้าไม่ระบุโหมดจะเป็น r คือ read only เปิดไฟล์เพื่ออ่านข้อมูลเท่านั้น

w เปิดไฟล์เพื่อการเขียน write only ถ้ามีไฟล์นี้อยู่ก็จะเขียนใหม่ทับ ถ้าไม่มีอยู่ก็จะสร้างใหม่

a	เปิดไฟล์เพื่อ append ถ้ามีไฟล์เดิมอยู่ จะเขียนไฟล์ต่อจากเดิม ถ้าไม่มีอยู่ก็จะสร้างใหม่
r+	เปิดเพื่ออ่านและเขียน pointer (ตัวชี้) จะอยู่ต้นไฟล์
w+	เปิดไฟล์เพื่ออ่านและเขียน ถ้ามีไฟล์อยู่ก่อนจะเขียนใหม่ทับ ถ้าไม่มีก็จะสร้างใหม่
a+	เปิดไฟล์เพื่ออ่านและเขียน ถ้ามีไฟล์อยู่ก่อน จะเขียนต่อจากเดิม ถ้าไม่มีจะสร้างใหม่

rb, wb, ab, rb+, wb+, ab+ ความหมายเหมือนเดิม เพิ่ม b เพื่อจะบอกว่าจะเขียนอ่านข้อมูลในแบบ binary

```
#Python 3.9.5
#Example 1-10-2

def func_1_10_2():
  'เปิดไฟล์ และเขียนข้อมูลเพิ่ม ถ้าไม่มีไฟล์เดิม จะสร้างใหม่ให้'
  try:
    with open("test.txt","a") as f:
      f.write ("Append")
      f.write ("!!!\n")
      print ("HELLO", file=f)
      print ("AGAIN", file=f)
    print ("File Appended!!!")
  except:
    print ("Cannot open file")

if __name__ == "__main__":
  func_1_10_2()
```

ถ้าเปิดไฟล์สำเร็จ

```
File Appended!!!
```

และในไฟล์ test.txt จะมีข้อความแบบนี้

```
hello, world!123 456 789
HELLO
WORLD
Append!!!
HELLO
AGAIN
```

ตัวอย่าง 1-10-2

เปิดไฟล์ด้วยโหมด a หรือ append

ถ้าไม่มีไฟล์ชื่อ test.txt อยู่ก่อน ก็จะสร้างไฟล์นี้ขึ้นมา และ เขียนข้อมูลลงไป

แต่ถ้ามีไฟล์ชื่อ test.txt อยู่แล้ว ก็จะเป็นการเปิดไฟล์เดิมและเขียน

```
Append!!!
HELLO
AGAIN
```

ต่อท้ายข้อมูลเดิม

```
#Python 3.9.5
#Example 1-10-3

def func_1_10_3():
  'อ่านไฟล์'
  try:
    with open("test.txt","r") as f:
      a = f.readline()
      print (a)
      print (f.readlines())
      print ("***Current file position is",f.tell())
      f.seek(0)
      print ("***Set current file position to",f.tell())
      print (f.readlines())
      f.seek(3)
      print ("***Set current file position to",f.tell())
      print (f.readline())
      print ("***Current file position is",f.tell())
    print ("File Read!!!")
  except:
    print ("Cannot open file")

if __name__ == "__main__":
  func_1_10_3()
```

```
hello, world!123 456 789

['HELLO\n', 'WORLD\n', 'Append!!!\n', 'HELLO\n', 'AGAIN\n']
***Current file position is 59
***Set current file position to 0
['hello, world!123 456 789\n', 'HELLO\n', 'WORLD\n',
'Append!!!\n', 'HELLO\n', 'AGAIN\n']
***Set current file position to 3
lo, world!123 456 789

***Current file position is 25
File Read!!!
```

ตัวอย่าง 1-10-3

เปิดไฟล์ด้วยโหมด r หรือ read only ทดลองดูเองเลย
ถ้าเปิดสำเร็จ จะแสดงผล

```
hello, world!123 456 789

['HELLO\n', 'WORLD\n', 'Append!!!\n', 'HELLO\n', 'AGAIN\n']
***Current file position is 65
***Set current file position to 0
['hello, world!123 456 789\n', 'HELLO\n', 'WORLD\n', 'Append!!!
\n', 'HELLO\n', 'AGAIN\n']
***Set current file position to 3
lo, world!123 456 789

***Current file position is 26
File Read!!!
```

คำสั่ง seek() เป็นการสั่งให้เลื่อน file pointer หรือตัวชี้ตำแหน่งข้อมูล ไปอยู่ตำแหน่งที่ต้องการ

ส่วน tell() จะให้ค่าออกมาเป็นตำแหน่งปัจจุบันของ file pointer
คำสั่งที่เหลือไม่อธิบายละ ลองทดลองจากตัวอย่างเอา

```
#Python 3.9.5
#Example 1-10-4

def func_1_10_4():
  'เขียนภาษาไทยลงในไฟล์'
  try:
    with open("test.txt",mode="r+",encoding="utf-8") as f:
      f.writelines("สวัสดี")
      print ("ชาวโลก",file=f)
      print ("***Current file position is",f.tell())
      f.seek(0)
      print ("***Set file position to",f.tell())
      print (f.readlines())
    print ("File Opened!!!")
  except:
    print ("Cannot open file")

if __name__ == "__main__":
  func_1_10_4()
```

```
***Current file position is 37
***Set file position to 0
['สวัสดีชาวโลก\n', 'Append!!!\n', 'HELLO\n', 'AGAIN\n']
File Opened!!!
```

ตัวอย่าง 1-10-4

ที่ผ่านมาเราทดลองเก็บข้อมูลเป็นภาษาอังกฤษ แต่ถ้าจะใช้ภาษาไทย เราต้องเล่นใหญ่เต็มยศแบบนี้

คำสั่งพื้นฐานของภาษา Python ยังมีอีกมาก รวมถึงยังมีลูกเล่นในภาษาอีกหลายอย่างซึ่งเราคงค่อยๆ ศึกษา ค่อยๆ หัดใช้งานไป แต่โดยเบื้องต้น ถ้าศึกษาคำสั่งพื้นฐานมาถึงระดับนี้ ก็จะรู้จักชุดคำสั่งมากพอที่จะนำมาทำโครงงานต่างๆ หรือมาศึกษาอัลกอริธึม หรือวิธีคิด วิธีแก้ปัญหาด้วยคอมพิวเตอร์ได้แล้ว ซึ่งจะแสดงตัวอย่างปัญหาในบทต่อไป

Chapter21

HELLO MONDAY

โปรแกรมสวัสดีวันจันทร์

2-1-1 if elif else
2-1-2 dictionary

ตัวอย่าง 2-1-1

ให้โปรแกรมรับค่าตัวเลขตั้งแต่ 1-7 แล้วแสดงค่าออกมาตามเงื่อนไข และป้อน 0 เมื่อต้องการออกจากโปรแกรม

เงื่อนไขคือ

1 แสดงผล “สวัสดีวันอาทิตย์”
2 แสดงผล “สวัสดีวันจันทร์”
3 แสดงผล “สวัสดีวันอังคาร”
4 แสดงผล “สวัสดีวันพุธ”
5 แสดงผล “สวัสดีวันพฤหัสบดี”
6 แสดงผล “สวัสดีวันศุกร์”
7 แสดงผล “สวัสดีวันเสาร์”
0 แสดงผล “ลาก่อน” และ ออกจากโปรแกรม

```
#Python 3.9.5
#Example 2-1-1

def hello():
  'ป้อนเลข 1-7 แล้วพิมพ์ค่าออกมาตามเงื่อนไข'

  loop = True

  while loop:
    x = input("ป้อนเลข 1-7 (ป้อน 0 เพื่อออกจากโปรแกรม) :")
    if x=="1":
      print ("สวัสดีวันอาทิตย์")
    elif x=="2":
      print ("สวัสดีวันจันทร์")
    elif x=="3":
      print ("สวัสดีวันอังคาร")
    elif x=="4":
      print ("สวัสดีวันพุธ")
    elif x=="5":
      print ("สวัสดีวันพฤหัสบดี")
    elif x=="6":
      print ("สวัสดีวันศุกร์")
    elif x=="7":
      print ("สวัสดีวันเสาร์")
    elif x=="0":
      print ("ลาก่อน")
      loop = False

if __name__=='__main__':
  hello()
```

ตัวอย่าง 2-1-2

ปรับปรุงโปรแกรม 2-1-1 ให้ดูง่ายขึ้น แยกเอาส่วนของข้อมูล (data) ออกจากตัวโปรแกรม ทำให้การแก้ไข หรือเปลี่ยนแปลงค่า ที่ต้องการให้แสดงทำได้ง่ายขึ้นและลดโอกาสผิดพลาดได้

```
#Python 3.9.5
#Example 2-1-2

def hello():
  'ป้อนเลข 1-7 แล้วพิมพ์ค่าออกมาตามเงื่อนไข'

  word = {
    "1":"สวัสดีวันอาทิตย์",
    "2":"สวัสดีวันจันทร์",
    "3":"สวัสดีวันอังคาร",
    "4":"สวัสดีวันพุธ",
    "5":"สวัสดีวันพฤหัสบดี",
    "6":"สวัสดีวันศุกร์",
    "7":"สวัสดีวันเสาร์",
    "0":"ลาก่อน" }

  loop = True
  while loop:
    x = input("ป้อนเลข 1-7 (ป้อน 0 เพื่อออกจากโปรแกรม) :")
    if x == "0":
      print (word["0"])
      loop = False
    elif x in word:
      print (word[x])

if __name__=='__main__':
  hello()
```

Chapter22

TRIANGLE

สร้างสามเหลี่ยมด้วย *

2-2-1 print print print
2-2-2 for loop
2-2-3 วาดสามเหลี่ยมกลับหัว
2-2-4 วาดสามเหลี่ยมกลับด้าน
2-2-5 สามเหลี่ยมกลับด้านกลับหัว
2-2-6 ประกอบสามเหลี่ยมกลายเป็น..เพชร
2-2-7 ปรับปรุงโปรแกรมวาดเพชร

ตัวอย่าง 2-2-1

ตัวอย่างนี้เป็นตัวอย่างคลาสสิค ใครเขียนโปรแกรมใหม่ๆ เขาก็มักจะให้เขียนโปรแกรมทำรูป

```
*
**
***
****
*****
******
*******
********
*********
**********
```

ซึ่งเขียนเป็นโปรแกรมได้ง่ายมาก แค่สั่ง print ทีละบรรทัดๆ ก็ได้แล้ว

```
#Python 3.9.5
#Example 2-2-1

def draw_triangle():
   'ใช้ * วาดเป็นสามเหลี่ยม'

   print ("*")
   print ("**")
   print ("***")
   print ("****")
   print ("*****")
   print ("******")
   print ("*******")
   print ("********")
   print ("*********")
   print ("**********")

if __name__=='__main__':
   draw_triangle()
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

ตัวอย่าง 2-2-2

แต่การเขียนโปรแกรมวาดรูปสามเหลี่ยมตามตัวอย่าง 2-2-1 มันดูไม่ค่อยเท่เท่าที่ควร คือสำหรับคนที่เรียนเขียนโปรแกรมมาดีๆ เราควรทำออกมาดังตัวอย่างนี้ ซึ่งจากตัวอย่างนี้เราสามารถกำหนดได้ด้วยว่าให้วาดออกมากี่แถว จากตัวอย่าง 2-2-1 เราต้องมาสั่ง print ทีละแถวๆ ทีนี้เราใช้คำสั่ง for เข้ามาช่วย จะให้ print ออกมากี่แถวก็ได้ง่ายๆ

```python
#Python 3.9.5
#Example 2-2-2

def draw_triangle(max=10):
    'ใช้ * วาดเป็นสามเหลี่ยม โดยใช้ loop'

    for i in range (max):
        counter = i+1
        star = "*" * counter
        print (star)

if __name__=='__main__':
    draw_triangle()
```

```
*
**
***
****
*****
******
*******
********
*********
**********
```

ตัวอย่าง 2-2-3

ลองวาดสามเหลี่ยมกลับหัวบ้าง ปรับแก้โปรแกรมนิดเดียว คือแทนที่จะ loop ตั้งแต่ 1 ถึงค่า max ก็เปลี่ยนเป็นให้เริ่มจาก max แล้วค่อยๆ ลดลงมาถึง 1

```python
#Python 3.9.5
#Example 2-2-3

def draw_upside_triangle(max=10):
    'ใช้ * วาดเป็นสามเหลี่ยมกลับหัว'

    for i in range(max,0,-1):
        counter = i
        star = "*" * counter
        print (star)

if __name__=='__main__':
    draw_upside_triangle()
```

```
**********
*********
********
*******
******
*****
****
***
**
*
```

ตัวอย่าง 2-2-4

จากการเอา * มาเรียงเป็นสามเหลี่ยมแบบง่ายๆ ลองเล่นท่าพิสดารดูบ้าง ถ้าอยากวาดให้กลับด้านเป็นแบบนี้

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

ต้องทำอย่างไร
อย่าเพิ่งรีบไปดูคำตอบครับ ลองคิดดูก่อนว่า จะทำอย่างไร

```python
#Python 3.9.5
#Example 2-2-4

def draw_mirror_triangle(max=10):
    'ใช้ * วาดเป็นสามเหลี่ยมกลับด้าน'

    for i in range (max,0,-1):
        counter = i-1
        space = " " * counter
        star = "*" * (max-counter)
        print (space+star)

if __name__=='__main__':
    draw_mirror_triangle()
```

```
         *
        **
       ***
      ****
     *****
    ******
   *******
  ********
 *********
**********
```

ตัวอย่าง 2-2-5

เอาโปรแกรมจากตัวอย่าง 2-2-4 มาวาดให้กลับหัว
โปรแกรม 2-2-4 และ 2-2-5 นี้ หลายๆ คนคิดไม่ออกว่าจะวาดสามเหลี่ยมให้มันกลับด้าน ได้อย่างไร แต่พอดูเฉลยแล้วก็อ๋อกัน เพราะหลักการก็คือการวาด “ “ ช่องว่าง ให้เป็นสามเหลี่ยม แล้วค่อยเอา * ไปเติมข้างหลัง พอรู้แล้วก็ไม่ยาก แต่ถ้าใครยังงง ก็ค่อยๆ คิดตามดีๆ ครับ ลองเริ่มจากจำนวนแถวน้อยๆ ก่อนก็ได้

```python
#Python 3.9.5
#Example 2-2-5

def draw_mirror_upside_triangle(max=10):
    'ใช้ * วาดเป็นสามเหลี่ยมกลับด้านและกลับหัว'

    for i in range (0,max):
        counter = i
        space = " " * counter
        star = "*" * (max-counter)
        print (space+star)

if __name__=='__main__':
    draw_mirror_upside_triangle()
```

```
**********
 *********
  ********
   *******
    ******
     *****
      ****
       ***
        **
         *
```

ตัวอย่าง 2-2-6

เอาตัวอย่างตั้งแต่ 2-2-2 ถึง 2-2-5 มาประกอบรวมเข้าด้วยกันกลายเป็น ....

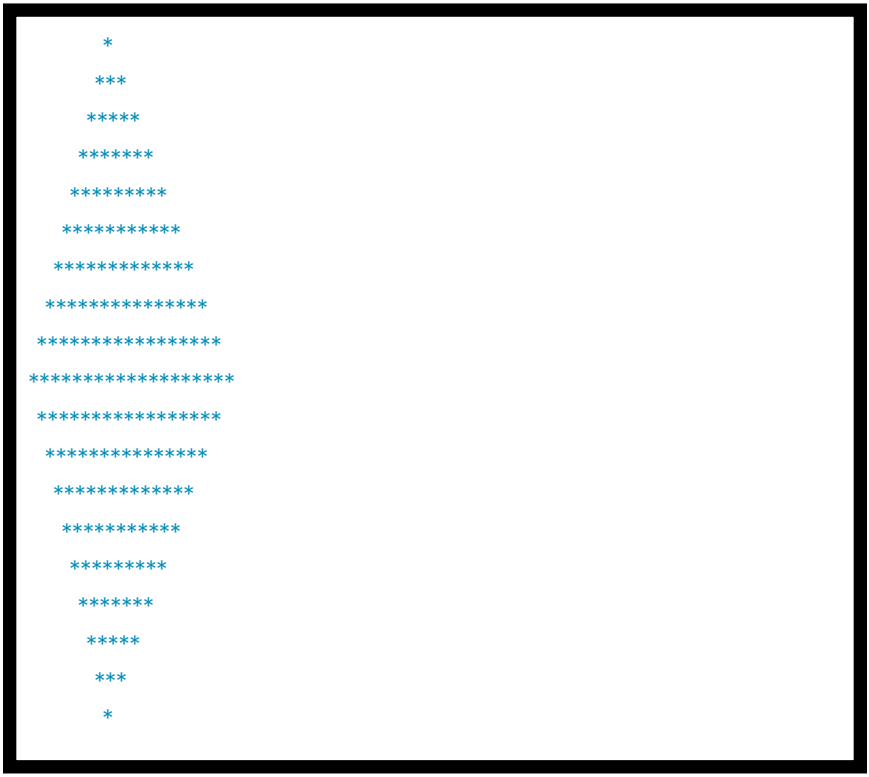
```python
#Python 3.9.5
#Example 2-2-6

def draw_diamond(max=10):
    'ใช้ * วาดเป็นรูปเพชร'

    for i in range (max):
        counter = i+1
        space = " " * (max-counter)
        star = "*" * counter + "*" * (counter-1)
        print (space+star)

    for i in range (max,1,-1):
        counter = i-1
        space = " " * (max-counter)
        star = "*" * counter + "*" * (counter-1)
        print (space+star)

if __name__=='__main__':
    draw_diamond()
```



ตัวอย่าง 2-2-7

จากตัวอย่าง 2-2-6 จะเห็นว่ากลางๆ คำสั่ง for สองครั้ง มันมีช่วงที่เขียนซ้ำๆ กันอยู่ เราเลยดึงออกมาสร้างเป็นฟังก์ชันใหม่ซะเลย ปรับปรุงโปรแกรมเดิมออกมาได้แบบนี้

```
#Python 3.9.5
#Example 2-2-7

def print_line(counter,max):
   'แสดงผล * แต่ละแถว'

   space = " " * (max-counter)
   star = "*" * counter + "*" * (counter-1)
   print (space+star)

def draw_diamond2(max=10):
   'ใช้ * วาดเป็นรูปเพชร'

   for i in range (max):
      counter = i+1
      print_line(counter,max)

   for i in range (max,1,-1):
      counter = i-1
      print_line(counter,max)

if __name__=='__main__':
   draw_diamond2()
```

Chapter23

1TON

หาผลบวก 1 - n

2-3-1 for loop
2-3-2 หาผลบวก 1-n ด้วยสูตร n*(n+1)/2
2-3-3 ฟังชันบรรทัดเดียว
2-3-4 lambda function

ตัวอย่าง 2-3-1

ตัวอย่างโปรแกรมหาผลบวกของเลข 1+2+3+...+n คือบวกไปเรียอยๆ จนถึงค่าที่ n

เช่น n=5 ก็คือ 1+2+3+4+5 ก็ได้ผลลัพธ์คือ 15

โปรแกรมก็ไม่ยากนัก สร้างลูปขึ้นมาแล้วบวกค่ากันไปเรื่อยๆ ตามจำนวนรอบที่ต้องการก็ได้คำตอบแล้ว

บรรทัด x+=i เป็นการเขียนแบบย่อๆ ของ x=x+i หรือเอาค่าของ x เดิม บวกกับค่าใน i ซึ่งเป็นตัวนับรอบ แล้วเอาไปเก็บคืนไว้ในตัวแปร x แล้วสุดท้ายก็คืนค่าออกมาด้วยคำสั่ง return

```
#Python 3.9.5
#Example 2-3-1

def sum_to_n(n):
   'หาผลบวกของ 1+2+3+...+n ด้วยคำสั่ง for loop'
   x=0
   n=n+1
   for i in range(1,n):
      x+=i
   return x

if __name__ == "__main__":
   print (sum_to_n.__doc__)
   print ("n=5 ผลบวก=",sum_to_n(5))
   print ("n=10 ผลบวก=",sum_to_n(10))
   print ("n=2 ผลบวก=",sum_to_n(2))
   print ("n=100 ผลบวก=",sum_to_n(100))
```

```
หาผลบวกของ 1+2+3+...+n ด้วยคำสั่ง for loop
n=5 ผลบวก= 15
n=10 ผลบวก= 55
n=2 ผลบวก= 3
n=100 ผลบวก= 5050
```

ตัวอย่าง 2-3-2

จากตัวอย่าง 2-3-1 จริงๆ แล้วเราสามารถหาค่า 1+2+3+...+n ได้จากสูตร n*(n+1)/2 ซึ่งพิสูจน์สูตรได้ไม่ยากนัก เลยเอามาเปรียบเทียบกัน ให้ดูว่าโจทย์แต่ละข้อ อาจใช้ได้หลายวิธีในการหาคำตอบ

บางคนถนัดคิดทางคณิตศาสตร์ก็อาจจะคิดออกมาเป็นสูตร แต่บางคนนึกสูตรไม่ทันหรือไม่ถนัด ก็สามารถหาคำตอบได้โดยให้คอมพิวเตอร์บวกกันไปเรื่อยๆ ก็ได้เช่นกัน

สังเกตว่าการใช้สูตรนี้ คำตอบที่ออกมาจะมีจุดทศนิยมติดมาด้วย เพราะในสูตรเรามีการนำค่าไปหาร ซึ่งทำให้ชนิด (type) ของตัวแปรถูกเปลี่ยนจาก interger ไปเป็น floting point

```
#Python 3.9.5
#Example 2-3-2

def sum_to_n2(n):
   'หาผลบวกของ 1+2+3+...+n จากสูตร n*(n+1)/2'
   return n*(n+1)/2

if __name__ == "__main__":
   print (sum_to_n2.__doc__)
   print ("n=5 ผลบวก=",sum_to_n2(5))
   print ("n=10 ผลบวก=",sum_to_n2(10))
   print ("n=2 ผลบวก=",sum_to_n2(2))
   print ("n=100 ผลบวก=",sum_to_n2(100))
```

```
หาผลบวกของ 1+2+3+...+n จากสูตร n*(n+1)/2
n=5 ผลบวก= 15.0
n=10 ผลบวก= 55.0
n=2 ผลบวก= 3.0
n=100 ผลบวก= 5050.0
```

ตัวอย่าง 2-3-3

จากตัวอย่าง 2-2-2 จะเห็นว่าโปรแกรม return ค่าเป็นสูตรคำนวณได้ในบรรทัดเดียวเลย เราสามารถปรับปรุงโปรแกรมให้สั้นลงโดยเอา return ไปต่อกับชื่อฟังก์ชันได้เลย

แต่วิธีนี้ทำให้เราไม่สามารถเขียน function document ได้ จะเห็นว่าเมื่อสั่ง print (sum_to_n3.__doc__) ในบรรทัดสุดท้ายจะได้ค่าออกมาเป็น None

และถ้าเราไปเคาะ indent ให้กับ บรรทัด

```
'หาผลบวก 1-n จากสูตร n*(n+1)/2 ในหนึ่งบรรทัด'
```

ใต้ชื่อฟังก์ชัน sum_to_n3
เพื่อให้มันอยู่ในบล็อกลูกของโปรแกรม sum_to_n3 ก็จะ error ทดลองดูได้ครับ

```
#Python 3.9.5
#Example 2-3-3

def sum_to_n3(n):return n*(n+1)/2
'หาผลบวกของ 1+2+3+...+n จากสูตร n*(n+1)/2 ในหนึ่งบรรทัด'

if __name__ == "__main__":
   print("หาผลบวก 1-n จากสูตร n*(n+1)/2 ในหนึ่งบรรทัด")
   print ("n=5 ผลบวก=",sum_to_n3(5))
   print ("n=10 ผลบวก=",sum_to_n3(10))
   print ("n=2 ผลบวก=",sum_to_n3(2))
   print ("n=100 ผลบวก=",sum_to_n3(100))
   print (sum_to_n3.__doc__)
```

```
หาผลบวกของ 1+2+3+...+n จากสูตร n*(n+1)/2
n=5 ผลบวก= 15.0
n=10 ผลบวก= 55.0
n=2 ผลบวก= 3.0
n=100 ผลบวก= 5050.0
```

ตัวอย่าง 2-3-4

ตัวอย่างนี้ให้ดูเทคนิคการสร้างฟังก์ชันอีกแบบหนึ่ง โดยสร้างไว้ในโปรแกรมหลัก หรือสร้างไว้ในฟังก์ชันอื่นๆ ไม่ได้แยกออกมา เหมาะกับการสร้างฟังก์ชันที่ใช้ซ้ำในขอบเขตที่ไม่กว้างนัก เราเรียกว่า lambda function หรือ แลมบ์ด้าฟังก์ชัน

ซึ่งเมื่อสั่ง print(type(sum_to_n4)) จะได้ออกมาเป็น <class ‘function'>

แลมบ์ด้าฟังก์ชันนี้สามารถใช้งานได้จนถึงระดับพิสดาร ค่อยๆ ศึกษาไปครับ

```
#Python 3.9.5
#Example 2-3-4
'ใช้ lambda function หาผลบวกของ 1+2+3+...+n'

if __name__ == "__main__":
   sum_to_n4=lambda n:n*(n+1)/2

   print (__doc__)
   print ("n=5 ผลบวก=",sum_to_n4(5))
   print ("n=10 ผลบวก=",sum_to_n4(10))
   print ("n=2 ผลบวก=",sum_to_n4(2))
   print ("n=100 ผลบวก=",sum_to_n4(100))
   print (type(sum_to_n4))
```

```
ใช้ lambda function หาผลบวกของ 1+2+3+...+n
n=5 ผลบวก= 15.0
n=10 ผลบวก= 55.0
n=2 ผลบวก= 3.0
n=100 ผลบวก= 5050.0
<class 'function'>
```

Chapter24

INTEREST

หาผลรวมดอกเบี้ยทบต้น

2-4-1 หาผลรวมดอกเบี้ยด้วย for loop
2-4-2 หาผลรวมดอกเบี้ยด้วยสูตร deposit*(1+interest)**n

ตัวอย่าง 2-4-1

ตัวอย่างนี้เป็นโปรแกรมง่ายๆ คือคำนวณมูลค่าของดอกเบี้ยทบต้น โดย

deposit	คือเงินต้น
interest	คืออัตราดอกเบี้ย
n	คือจำนวนรอบของการเกิดดอกเบี้ย

แต่ละรอบเอาดอกเบี้ยมารวมกับเงินต้น แล้วคำนวณใหม่ซ้ำเรื่อยๆ จนครบ n รอบ

เพื่อให้เห็นอัตราเพิ่มหรืออัตราลด ก็เอาค่าแต่ละรอบไปใส่ไว้ในลิสต์ชื่อ amount แล้ว return ลิสต์ออกมา

ฟังก์ชัน compound_interest ก็มีแค่นี้ ไม่ได้มีอะไรพิเศษ

แต่ที่พิเศษคือในโปรแกรมหลักที่เรียกใช้ฟังก์ชัน compound_interest ครับ

```
print(compound_interest(100,0.01,10))
```

แสดงผลออกมาเป็นลิสต์

```
print(*compound_interest(100,0.01,10))
```

มี * เพิ่มมาตัวหนึ่ง ผลออกมาไม่เป็นลิสต์แล้ว แต่เป็นชุดตัวเลขต่อๆ กันดูแทบไม่รู้เรื่อง

```
print(*compound_interest(100,0.01,10),sep="\")
```

มีคำว่า sep="\n" เพิ่มมาในคำสั่ง print คราวนี้เป็นการบอกว่าคำสั่ง print นี้แยกค่าแต่ละค่า (separator) ด้วย "\n" ซึ่งเป็นการสั่งให้ขึ้นบรรทัดใหม่ ทีนี้ก็จะเห็นแต่ละค่าง่ายขึ้น

```
print(round(compound_interest(100,0.01,365)[-1],2))
```

ดูงงๆ แต่ไม่ต้องงง

ฟังก์ชัน round(<value>,x) คือการบอกว่าจะเอาทศนิยมกี่ตำแหน่ง ซึ่งในกรณีนี้ก็คือทศนิยม 2 ตำแหน่ง ปัดขึ้นปัดลงให้เรียบร้อย

```
compound_interest(100,0.01,365)[-1]
```

เนื่องจาก compound_interest คืนค่าออกมาเป็นลิสต์ ซึ่งปกตินี่ค่าในลิสต์จะนับเริ่มจากตัวที่ 0,1,2,... ไปเรื่อยๆ แต่ถ้านับจากท้ายเราสามารถนับเป็น -1, -2, -3,... ได้ด้วย โดยนับจากตัวสุดท้ายในลิสต์มา ทีนี้เราเอาลิสต์ตัวที่ -1 ก็คือตัวสุดท้าย หรือผลรวมดอกเบี้ยทบต้นรอบเมื่อนับไปถึงรอบสุดท้ายนั่นเอง จะเห็นว่าถ้าเราเริ่มเก็บเงินจาก 100 บาท แล้วได้ดอกเบี้ยแค่ 0.01 หรือเท่ากับ 1% ทุกวัน ปีหนึ่งเราจะมีเงินรวมถึง 3778.34 บาท

```
print(round(compound_interest(100,-0.01,365)[-1],2))
```

บรรทัดนี้คล้ายๆ บรรทัดก่อนหน้า ต่างแค่ว่าจาก 0.01 เป็น -0.01 หรือความหมายคือ ถ้าเรามีเงินตั้งต้น 100 บาท แล้วใช้ไปหรือหายไปแค่วันละ 0.01 หรือ 1% ของเงินต้น ปีหนึ่งเราจะเหลือเงินแค่ 2.55 บาท

หรือมีคนเอาตรงนี้มาเปรียบเทียบประมาณว่าถ้าตัวเราสมมติว่ามีความฉลาดตั้งต้นที่ 100 หน่วย แล้วเราฉลาดขึ้นวันละ 1% สิ้นปีเราจะมีความฉลาดระดับ 3,778 หน่วย ขณะที่ถ้าเขาโง่ลงสักวันละ 1% ทุกๆ วัน ผ่านไปปีหนึ่งเราจะมีความเก่งเหลือแค่ 2.55 หน่วย

```
#Python 3.9.5
#Example 2-4-1

def compound_interest(deposit,interest,n):
    'หาผลรวมเงินต้นกับดอกเบี้ยทบต้น n รอบ'

    amount = []
    for i in range(n):
        deposit+=deposit*interest
        amount.append(deposit)
    return amount

if __name__ == "__main__":
    print (compound_interest(100,0.01,10))
    print ("\n")
    print (*compound_interest(100,0.01,10))
    print ("\n")
    print (*compound_interest(100,0.01,10),sep="\n")
    print ("\n")
    print ("เงินต้น 100 เพิ่มวันละ 1% เป็นเวลา 1ปี =")
    print (round(compound_interest(100,0.01,365)[-1],2))
    print ("เงินต้น 100 บาท ลดวันละ 1% เป็นเวลา 1ปี =")
    print (round(compound_interest(100,-0.01,365)[-1],2))
```

[101.0, 102.01, 103.0301, 104.060401, 105.10100501,
106.1520150601, 107.213535210701,
108.28567056280801, 109.36852726843608,
110.46221254112044]

101.0 102.01 103.0301 104.060401 105.10100501
106.1520150601 107.213535210701 108.28567056280801
109.36852726843608 110.46221254112044

101.0
102.01
103.0301
104.060401
105.10100501
106.1520150601
107.213535210701
108.28567056280801
109.36852726843608
110.46221254112044

เงินต้น 100 เพิ่มวันละ 1% เป็นเวลา 1ปี =
3778.34
เงินต้น 100 บาท ลดวันละ 1% เป็นเวลา 1ปี =
2.55

ตัวอย่าง 2-4-2

จากโปรแกรมคำนวณดอกเบี้ยทบต้นในตัวอย่าง 2-4-1 ปกติเราก็ไม่ถึงกับต้องดูค่ากันทุกรอบ มักจะดูกันเฉพาะรอบสุดท้ายว่าค่าสุดท้ายเป็นอะไร

โปรแกรมนี้เลยแสดงตัวอย่างให้ดูว่าเราสามารถคิดดอกเบี้ยทบต้นนี้ด้วยสูตร

ผลรวม= deposit*(1+interest)**n

ได้ด้วย หรือสามารถเขียนสูตรในรูปของ lambda function ก็ได้

เอาทุกแบบมาเปรียบเทียบกันให้ดู ไม่ว่าจะใช้ลูปบวกซ้ำไปเรื่อยๆ หรือคำนวณจากสูตรทีเดียวเลย ก็ได้ค่าเท่ากัน
บรรทัด

```
compound_int3=lambda deposit,interest,n: \
round(deposit*(1+interest)**n,3)
```

สามารถเขียนติดยาวเป็นบรรทัดเดียวกัน

```
compound_int3=lambda deposit,interest,n:
round(deposit*(1+interest)**n,3)
```

แบบนี้ได้เลย แต่พอดีมันยาว เราเลยใช้เครื่องหมาย \ คั่น ทำให้สามารถแบ่งคำสั่งยาวๆ คำสั่งเดียวเป็นหลายบรรทัดได้

```
#Python 3.9.5
#Example 2-4-2

def compound_int1(deposit,interest,n):
   'หาผลรวมเงินต้นกับดอกเบี้ยทบต้น n รอบ'
   for i in range(n):
      deposit+=deposit*interest
   return round(deposit,3)

def compound_int2(deposit,interest,n):
   'หาผลรวมเงินต้นกับดอกเบี้ยทบต้นจากสูตร PV*(1+i)**n'
   return round(deposit *(1+interest)**n,3)

if __name__ == "__main__":
   print ("เงินต้น 100 บาท เพิ่มวันละ 1% เป็นเวลา 1ปี")
   print ("คำนวณด้วย loop =",compound_int1(100,0.01,365))
   print ("คำนวณด้วยสูตร=",compound_int2(100,0.01,365))
   print("\n")
   print ("เงินต้น 100 บาท ลดวันละ 1% เป็นเวลา 1ปี")

   print ("คำนวณด้วย loop=",compound_int1(100,-0.01,365))
   print ("คำนวณด้วยสูตร=",compound_int2(100,-0.01,365))

   compound_int3=lambda deposit,interest,n: \
   round(deposit*(1+interest)**n,3)

   print ("ใช้ lambda function=", \
      compound_int3(100,-0.01,365))
```

เงินต้น 100 บาท เพิ่มวันละ 1% เป็นเวลา 1ปี
คำนวณด้วย loop = 3778.343
คำนวณด้วยสูตร= 3778.343

เงินต้น 100 บาท ลดวันละ 1% เป็นเวลา 1ปี
คำนวณด้วย loop= 2.552
คำนวณด้วยสูตร= 2.552
ใช้ lambda function= 2.552

Chapter25

FACTORIAL

หาเลขแฟคตอเรียล (Factorial Number)

2-5-1 for loop
2-5-2 recursion
2-5-3 นับถอยหลัง
2-5-4 lambda function
2-5-5 math module

ตัวอย่าง 2-5-1

ตัวอย่างโปรแกรมหาค่าของเลขแฟกทอเรียล หรือผลคูณของเลข 1*2*3*...*n ตั้งแต่ 1 ถึง n โปรแกรมรูปแบบเดิมๆ น่าเบื่อมาก ดูปุ๊บเข้าใจทันที ลงให้ดูขำๆ แล้วข้ามไปตัวอย่างถัดไปได้

```
#Python 3.9.5
#Example 2-5-1

def factorial1(n):
   'หา factorial number'
   n = n+1
   ans=1
   for i in range(1,n):
      ans=ans*i
   return ans

if __name__ == "__main__":
   print ("3! =",factorial1(3))
   print ("5! =",factorial1(5))
   print ("1! =",factorial1(1))
   print ("0! =",factorial1(0))
```

```
3! = 6
5! = 120
1! = 1
0! = 1
```

ตัวอย่าง 2-5-2

โปรแกรมหาเลขแฟกตอเรียลเหมือนตัวอย่าง 2-5-1 แต่เขียนไม่เหมือนเดิม อันนี้ละที่น่าศึกษา

การเขียนโปรแกรมลักษณะนี้เราเรียกว่าการเขียนโปรแกรมแบบ recursion คือ ในฟังชัน factorial2(n) มีการเรียกใช้งานฟังก์ชันตัวเองด้วย

> ในฟังก์ชัน factorial2(n) บรรทัดแรกจะตรวจสอบว่าถ้า n เป็น 0 หรือ 1 ก็ให้ return 1 เลย
>
> แต่ถ้าไม่ใช่ จะ return n*factorial2(n-1)
>
> แปลว่า ถ้า n = 2 บรรทัดนี้ก็จะ return 2*factorial(2-1) หรือ return 2*factorial(1)
>
> ส่วน factorial(1) นี่มัน return 1 แน่ๆ อยู่แล้ว เพราะเขียนบอกอยู่บรรทัดแรกว่าถ้า n เป็น 0 หรือ 1 ให้ return 1
>
> บรรทัด return n*factorial(n-1) ในกรณี n=2 ก็จะ return 2*1 หรือได้คำตอบคือ 2

ค่อยๆ คิดตาม ใจเย็นๆ ลองหลายๆ ค่า n=3, n=4 แล้วค่อยๆ คิดว่าโปรแกรมทำงานอย่างไร

ถ้างง ทำใจไม่ได้จริงๆ ช่างมันครับ ไม่ต้องเขียนโปรแกรมสไตล์นี้ก็ไม่เป็นไร โปรแกรมที่เขียนเป็น recursion ทุกโปรแกรมสามารถเขียนแทนด้วย loop แบบปกติๆ ได้ เพียงแต่ถ้าเขียนแบบนี้เป็นมันก็เท่ดี แล้วโปรแกรมมันก็สั้นดี

```
#Python 3.9.5
#Example 2-5-2

def factorial2(n):
   'หา factorial number ด้วย recursion'
   if n==1 or n==0:
      return 1
   else:
      return n*factorial2(n-1)

if __name__ == "__main__":
   print ("3! =",factorial2(3))
   print ("5! =",factorial2(5))
   print ("1! =",factorial2(1))
   print ("0! =",factorial2(0))
```

```
3! = 6
5! = 120
1! = 1
0! = 1
```

ตัวอย่าง 2-5-3

อย่างที่เขียนไปแล้วว่าโปรแกรมหนึ่งๆ สามารถเขียนได้หลายสไตล์

โจทย์ข้อนี้เคยให้เด็กเขียนออกมาแล้วเค้าเขียนออกมาแบบนี้ แถมนับถอยหลัง คือเค้าเอาค่า n มาใช้เป็นตัวนับ แล้วก็เลยนับถอยลงมาจนถึง 0 โดยใช้คำสั่ง while เป็นตัวสร้างวงรอบ

ซึ่งก็ได้อยู่

> บรรทัด while n!=0: หมายถึงให้ทำไปเรื่อยๆ ตราบที่ n ไม่เท่ากับ 0
>
> บรรทัด n-=1 หมายถึง n=n-1 ยังไม่ลืมนะ คือลดค่า n ไปเรื่อยๆ จน n=0 ก็จะหลุดจาก loop ของคำสั่ง while

```
#Python 3.9.5
#Example 2-5-3

def factorial3(n):
   'หา factorial number โดยนับถอยหลัง'
   ans=1
   while n!=0:
      ans*=n
      n-=1
   return ans

if __name__ == "__main__":
   print ("3! =",factorial3(3))
   print ("5! =",factorial3(5))
   print ("1! =",factorial3(1))
   print ("0! =",factorial3(0))
```

```
3! = 6
5! = 120
1! = 1
0! = 1
```

ตัวอย่าง 2-5-4

ไหนๆ ก็ไหนๆ แล้ว อันนี้เป็นโปรแกรมหาเลขแฟกทอเรียล โดยใช้ lambda function

สังเกตคือ lambda function ก็เขียนแบบ recursion หรือเรียกตัวเองได้ด้วยนะ เขียนสั้นนิดเดียวเอง

```
#Python 3.9.5
#Example 2-5-4
'หา factorial number ด้วย lambda function'

if __name__ == "__main__":
   factorial4 = lambda n: 1 if n==0 else n*factorial4(n-1)

   print ("3! =",factorial4(3))
   print ("5! =",factorial4(5))
   print ("1! =",factorial4(1))
   print ("0! =",factorial4(0))
```

```
3! = 6
5! = 120
1! = 1
0! = 1
```

ตัวอย่าง 2-5-5

ในความเป็นจริงแล้ว ภาษา Python เค้าฝังโปรแกรมหาเลขแฟกทอเรียลไว้ให้เราแล้ว ใน math module

ซึ่งเราสามารถเรียกใช้ได้โดยเริ่มจาก

```
import math as m
```

บรรทัดนี้ import as m อธิบายแบบง่ายๆ คือ import math module แล้วให้อ้างถึงโดยใช้ชื่อ m จะได้ไม่ต้องเขียนยาวๆ เท่านั้นแหละ

ซึ่งพอเรามี math module ให้ใช้แล้ว ทีนี้อยากได้เลขแฟกทอเรียลค่าไหน ก็สั่ง print เอาดื้อๆ ดังตัวอย่างเลย เช่น

```
print("3! =",m.factorial(3))
```

จะได้เลขแฟกทอเรียลของ 3! จากคำสั่ง m.factorial(3) ใช้ m แทน math

ชีวิตง่ายขึ้นมาก

```
#Python 3.9.5
#Example 2-5-5
import math as m
'หา factorial number ด้วย math module'

if __name__ == "__main__":
   print ("3! =",m.factorial(3))
   print ("5! =",m.factorial(5))
   print ("1! =",m.factorial(1))
   print ("0! =",m.factorial(0))
```

```
3! = 6
5! = 120
1! = 1
0! = 1
```

Chapter26

FIBONACCI

หาเลขฟิโบนัคคี (Fibonacci Number)

2-6-1 for loop
2-6-2 recursion
2-6-3 ปรับปรุงการเขียนแบบ recursion โดยส่งลิสต์ไปด้วย
2-6-4 lambda function

ตัวอย่าง 2-6-1

ยังอยู่กับการใช้ลูปหรือวงรอบในการหาคำตอบทางคณิตศาสตร์นี่ละ คราวนี้มารู้จักกับเลขฟิโบนักซี (Fibonacci numbers) หรือลำดับฟิโบนักซี (Fibonacci sequence) คือลำดับที่ค่าตัวถัดไปเกิดจากค่าก่อนหน้านั้นสองตัวบวกกัน หรือ 0,1,1,2,3,5,8,13,21,...

ซึ่งจะเห็นว่า

2 เกิดจาก 1+1
3 เกิดจาก 1+2
5 เกิดจาก 2+3
8 เกิดจาก 3+5

เป็นลำดับต่อเนื่องไปเรื่อยๆ

เขียนเป็นโปรแกรมให้หาค่าแบบนี้ออกมา n ค่า โดย return ออกมาเป็นลิสต์ที่เก็บค่าทั้งหมดไว้

โปรแกรมเขียนไม่ยาก มีบรรทัดหนึ่งที่อาจงงคือ

```
n0,n1 = n1,n0+n1
```

บรรทัดนี้เป็นการกำหนดค่าทีละสองตัว โดยนำ n1 มาใส่ไว้ใน n0 และ เอาค่า n0(เดิม)+n1 มาใส่ไว้ใน n1

```
#Python 3.9.5
#Example 2-6-1

def fibo1(n=2):
   'Fibonacci number'

   aList = [0,1]
   n0 = 0
   n1 = 1
   if n<3: return aList

   for i in range(2,n):
      n0,n1 = n1,n0+n1
      aList.append(n1)
   return aList

if __name__ == "__main__":
   print ("fib() =",fibo1())
   print ("fib =0",fibo1(0))
   print ("fib =1",fibo1(1))
   print ("fib =2",fibo1(2))
   print ("fib =3",fibo1(3))
   print ("fib =10",fibo1(10))
   print ("fib =13",fibo1(13))
```

```
fib = [0, 1]
fib 0= [0, 1]
fib 1= [0, 1]
fib 2= [0, 1]
fib 3= [0, 1, 1]
fib 10= [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
fib 13= [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

ตัวอย่าง 2-6-2

ตัวอย่างการหาลำดับฟิโบนักซีโดยใช้วิธี recursion แต่ถ้าลองเรียกเฉพาะโปรแกรม fibo2(n) จะ return เฉพาะเลขตัวสุดท้ายออกมา แต่เราต้องการเอาค่ามาใส่ลิสต์ เลยต้องมีอีกฟังก์ชั่นคือ fibo_list(n) มาเก็บค่าไปใส่ลิสต์

โปรแกรมนี้ไม่ค่อยดีนัก เพราะว่าแต่ละค่าที่ใส่ไว้ในลิสต์นี่ เราต้องหาตั้งแต่ต้นใหม่ทุกครั้งเลย เพราะโปรแกรม fibo2 จะคำนวณไปถึงค่าตั้งต้นคือเมื่อ n<=1 เสมอ

แปลว่า ถ้า n = 4 โปรแกรมจะทำสี่รอบเพื่อเอาค่าแต่ละรอบมาใส่ลิสต์

รอบแรก fibo2(1)
รอบสอง fibo2(2)
รอบสาม fibo2(3)
รอบสี่ fibo2(4)

โดยแต่ละรอบก็ต้องบวกตั้งแต่ต้นใหม่หมดเสมอ
ค่อยๆ คิดตามนะครับ

```
#Python 3.9.5
#Example 2-6-2
'Fibonacci number in recursion'

def fibo2(n):
  if n<=1:
    return n
  else:
    return(fibo2(n-1)+fibo2(n-2))

def fibo_list(n=2):
  aList=[]
  for i in range(n):
    aList.append(fibo2(i))
  return aList

if __name__ == "__main__":
  print("fib 0=",fibo_list(0))
  print("fib 1=",fibo_list(1))
  print("fib 2=",fibo_list(2))
  print("fib 3=",fibo_list(3))
  print("fib 10=",fibo_list(10))
  print("fib 13=",fibo_list(13))
```

```
fib 0= []
fib 1= [0]
fib 2= [0, 1]
fib 3= [0, 1, 1]
fib 10= [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
fib 13= [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

ตัวอย่าง 2-6-3

จากโปรแกรม 2-6-2 ซึ่งดูเวิ่นเว้อวุ่นวาย เพราะต้องแยกเป็นสองโปรแกรม แถมจะหาค่าถึงไหนก็ต้องมาตั้งต้นหากันใหม่หมดเพื่อเอาค่าสุดท้ายค่าเดียว โปรแกรมช่างไม่สวยงาม เราจึงดัดแปลงออกมาเป็นแบบนี้

คือมีการส่งลิสต์และค่า n0,n1 ไปในฟังก์ชันด้วย

อันนี้ถ้าอธิบายจะงง ต้องค่อยๆ ทำความเข้าใจดูเองจะดีกว่า ถ้าพยายามแล้วไม่เข้าใจ ก็ผ่านไปก่อนได้ หัวโล่งๆ แล้วค่อยมาดูใหม่จะเข้าใจไปเอง

```
#Python 3.9.5
#Example 2-6-3

def fibo3(n=0,aList=[],n0=0,n1=1):
   'Modified Fibonacci number in recursion'

   if len(aList)<n:
      aList.append(n0)
      n0,n1=n1,n0+n1
   else:
      return aList
   return fibo3(n,aList,n0,n1)

if __name__ == "__main__":
   print ("fib() =",fibo3())
   print ("fib 0 =",fibo3(0,aList=[]))
   print ("fib 1 =",fibo3(1,aList=[]))
   print ("fib 2 =",fibo3(2,aList=[]))
   print ("fib 3 =",fibo3(3,aList=[]))
   print ("fib 10 =",fibo3(10,aList=[]))
   print ("fib 13 =",fibo3(13,aList=[]))
   print ("fib 15 =",fibo3(15,aList=[]))
```

```
fib() = []
fib 0 = []
fib 1 = [0]
fib 2 = [0, 1]
fib 3 = [0, 1, 1]
fib 10 = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
fib 13 = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
fib 15 = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

ตัวอย่าง 2-6-4

ตัวอย่างการหาลำดับฟิโบนักซีโดยใช้ lambda function เอามาให้ดูเป็นตัวอย่างเฉยๆ ไม่อยากอธิบาย เอาไว้เรียนเรื่อง lambda function ดีๆ แล้วค่อยว่ากัน

```
#Python 3.9.5
#Example 2-6-4
'Modified Fibonacci number in lambda function'

if __name__ == "__main__":

   fibo4 = lambda n=0, o=[]: [o.append(i) or \
   i if i<=1 else o.append(o[-1]+o[-2]) or \
   o[-1] for i in range(n)]

   print ("fib () =",fibo4())
   print ("fib 0 =",fibo4(0))
   print ("fib 1 =",fibo4(1))
   print ("fib 2 =",fibo4(2))
   print ("fib 3 =",fibo4(3))
   print ("fib 10 =",fibo4(10))
   print ("fib 13 =",fibo4(13))
```

```
fib() = []
fib 0 = []
fib 1 = [0]
fib 2 = [0, 1]
fib 3 = [0, 1, 1]
fib 10 = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
fib 13 = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

Chapter27

MORSE CODE

โปรแกรมแปลงรหัสมอร์ส

2-7-1 แปลงรหัสด้วย dictionary

ตัวอย่าง 2-7-1

ตัวอย่างโปรแกรมแปลงตัวอักษรภาษาอังกฤษให้เป็นรหัสมอร์ส และแปลงกลับ โดยนำรหัสมอร์สสำหรับการแปลงค่ามาทำเป็น dictionary

มีคำสั่งไม่คุ้นเคยในฟังก์ชัน text_morse(txt) คือ

```
for letter in txt.lower():
```

เราส่งค่าเข้ามาในฟังก์ชันผ่านตัวแปร txt อันนี้รู้อยู่แล้ว

ส่วน txt.lower() คือการแปลงค่าใน txt ให้กลายเป็นอักษรตัวพิมพ์เล็กให้หมด เนื่องจาก dictionary ของเราทำไว้เฉพาะสำหรับอักษรภาษาอังกฤษตัวพิมพ์เล็กเท่านั้น ถ้าใส่ตัวพิมพ์ใหญ่เข้ามาจะอ่านไม่รู้เรื่อง ดังนั้นเราเลยต้องแปลงซะ

ส่วนการหาข้อมูลใน dictionary ก็ไม่ยาก คือวนรอบอ่านค่าอักษรมาทีละตัว แล้วเอาไปใช้เป็น key ก็จะได้ค่าที่แปลงแล้วเป็น value ใน dictionary ออกมา ถ้าเป็นตัวที่ไม่รู้จักก็ให้เป็น ‘ ‘ ช่องว่างแทนไป

ทีนี้ถ้าจะแปลกลับ แปลรหัสมอร์สให้กลายเป็นตัวอักษร ถ้าคิดแบบง่ายๆ ก็ทำ dictionary อีกชุดหนึ่ง แล้วเขียนกลับกับชุดนี้คือ เอารหัสมอร์สมาเป็น key แล้วให้ value เป็นตัวอักษร แล้วเขียนโปรแกรมแบบ text_morese(txt) ก็จะได้

แต่เราอยากทำอะไรให้มันพิสดารๆ มาลองดูกัน คือเอาค่า value มาหา key ซึ่งมันทำตรงไปตรงมาไม่ได้ ต้องใช้เทคนิคพิสดารสักหน่อย ลองศึกษาจากฟังก์ชัน morse_text(txt) ดูครับ
บรรทัดนี้

```
t_code+=list(morse_dict.keys()) \
[list(morse_dict.values()).index(letter)]
```

ซึ่งจริงๆ มันเป็นคำสั่งบรรทัดเดียวเขียนต่อกันเลย คือ

```
t_code+=list(morse_dict.keys())
[list(morse_dict.values()).index(letter)]
```

ส่วนบรรทัด

```
list(morse_dict.keys())
[list(morse_dict.values()).index(letter)]
```

นี่ละคือวิธีนำตัวแปร letter มาย้อนหา key ใน dictionary ออกมา ตรงนี้จะไม่อธิบายนะครับ ค่อยๆ แกะดูว่ามันทำงานอย่างไร หรือไม่ก็เอารูปแบบนี้ไปใช้งานได้เลย

ในโปรแกรมหลัก ลองสังเกตคำสั่งชุดนี้

```
print ("'S' ",*text_morse("'S'"))
print ("\"0\"",*text_morse('\"0\"'))
print(morse_text([‘...','---','abcd','...',]))
```

มีทริคเกี่ยวกับการใช้เครื่องหมาย “ “ และ ‘ ‘ ลองพิจารณากันดู

```
#Python 3.9.5
#Example 2-7-1

'MORSE Code encoder/decoder'
morse_dict= {
   'a':'.-', 'b':'-...', 'c':'-.-.', 'd':'-..',
   'e':'.', 'f':'..-.', 'g':'--.', 'h':'....',
   'i':'..', 'j':'.---', 'k':'-.-', 'l':'.-..',
   'm':'--', 'n':'-.', 'o':'---', 'p':'.--.',
   'q':'--.-', 'r':'.-.', 's':'...', 't':'-',
   'u':'..-', 'v':'...-', 'w':'.--', 'x':'-..-',
   'y':'-.--', 'z':'--..',
   '1':'.----', '2':'..---', '3':'...--', '4':'....-',
   '5':'.....', '6':'-....', '7':'--...', '8':'---..',
   '9':'----.', '0':'-----', '.':'.-.-.-', ',':'--..--',
   ':':'---...', '?':'..--..', '\'':'.----.', '-':'-....-',
   '/':'-..-.', '(':'-.--.', ')':'-.--.-', '"':'.-..-.',
   '=':'-...-', '+':'.-.-.', '*':'-..-', '@':'.--.-.',
   }


def text_morse(txt):
   m_code = []
   for letter in txt.lower():
      if letter in morse_dict:
         m_code.append(morse_dict[letter])
      else:
         m_code.append('')
   return m_code

def morse_text(txt):
   t_code = ''
```

```
    for letter in txt:
        try:
            t_code+=list(morse_dict.keys()) \
            [list(morse_dict.values()).index(letter)]
        except:
            t_code+=' '
    return t_code

if __name__ == "__main__":
    print ("hello,world! ",*text_morse("hello,world!"))
    print ("sos SOS!! ",text_morse("sos SOS!!"))
    print ("'S' ",*text_morse("'S'"))
    print ("\"O\"",*text_morse('\"O\"'))
    print(morse_text(['...','---','abcd','...',]))
```

```
hello,world!  .... . .-.. .-.. --- --..-- .-- --- .-. .-.. -..
sos SOS!!  ['...', '---', '...', '', '...', '---', '...', '', '']
'S'  .----. ... .----.
"O" .-..-. --- .-..-.
```

Chapter28

SQUARE ROOT

โปรแกรมหารากที่สอง (square root)

2-8-1 หาค่าด้วย for loop 100 รอบ
2-8-2 หาคำตอบตามจำนวนหลักทศนิยมที่ต้องการ
2-8-3 หาจากสูตร
2-8-4 math module

ตัวอย่าง 2-8-1

ตัวอย่างนี้เป็นตัวอย่างโปรแกรมหาค่ารากที่สอง หรือ square root ของตัวเลขใดๆ

เช่น รากที่สองของ 4 ก็คือ 2 หรือเขียนเป็นสัญลักษณ์คือ $\sqrt{4} = 2$ หรือ 4 = 2*2

อีกตัวอย่าง $\sqrt{9} = 3$ หรือ 9 = 3*3

ทีนี้ถ้าเลขที่มันไม่ลงตัวแบบนี้ละ เช่น $\sqrt{2}$ คือต้องหาเลขสองตัวมาคูณกันให้ได้ 2

1*1 = 1

แต่

2*2 =4

แปลว่าค่าที่ว่านี้ต้องอยู่ระหว่าง 1 กับ 2

ลอง 1.5*1.5 = 2.25

ยังเกิน 2 อยู่ แปลว่าเลขที่หาต้องอยู่ระหว่าง 1 กับ 1.5 เพราะถ้ามากกว่า 1.5 มันก็ยิ่งเกิน 2 ไปกันใหญ่

ลอง 1.25*1.25 = 1.5625

ทีนี้น้อยกว่า 2 ไปละ ก็แปลว่าค่าที่เราหาอยู่ระหว่าง 1.25 แต่ไม่เกิน 1.5

แล้วเราก็หาแบบนี้ไปเรื่อยๆ ในที่สุดก็จะได้ค่าที่ต้องการแถวๆ 1.414 ได้ว่า

1.414 * 1.414 = 1.999396

ยังไม่เท่ากับ 2 หรอก แต่ก็ใกล้เคียงมากแล้ว

และถ้าเรายังหาไปเรื่อยๆ ก็จะได้เลขที่ละเอียดเป็นทศนิยมไม่รู้จบไปเรื่อยๆ เพราะเลขตัวนี้ภาษาคณิตศาสตร์เขาเรียกว่าจำนวนอตรรกยะ ภาษาอังกฤษเรียกว่า irrational number แปลไทยตรงๆ คือจำนวนที่ไม่มีเหตุผล ซึ่งคงฟังแล้วไม่เสนาะหู คนที่เก่งๆ ภาษาไทยเค้าเลยเรียกว่า จำนวนอตรรกยะ คือมันไม่มีตรรกะ ไม่มีเหตุผลนั่นแหละ เพราะมันหาไปได้เรื่อยๆ ไม่รู้จบ

ทีนี้เรารู้วิธีหาด้วยมือแล้ว จะเขียนเป็นโปรแกรมอย่างไรดี

โดยหลักการก็คล้ายๆ กับเกมทายตัวเลข คือทายเลขตัวใหม่เรื่อยๆ แล้วก็เอาตัวเลขที่ทายไปเทียบกับคำตอบว่ามากกว่าหรือน้อยกว่า แล้วก็ทายค่าใหม่ไปเรื่อยๆ ให้มันอยู่ในช่วงมากกว่าหรือน้อยกว่าที่เราหาไว้แต่แรก

ลองคิดหรือลองเขียนเป็นโปรแกรมดูก่อนได้ แต่ถ้าใจร้อนใจเร็วอยากดูตัวอย่างเลย ก็ตามนี้

```
#Python 3.9.5
#Example 2-8-1

def square_root(n):
   'หาค่ารากที่สองของ n'
   n0 = 1
   n1 = n
   for i in range(100):
      n_mid = (n0+n1)/2
      n2 = n_mid*n_mid
      if n2 == n:
         return n_mid
      elif n2 < n:
         n0 = n_mid
      elif n2 > n:
         n1 = n_mid
   return n_mid

if __name__ == "__main__":
   print ("Square Root 2 =",square_root(2))
   print ("Square Root 4 =",square_root(4))
   print ("Square Root 5 =",square_root(5))
```

```
Square Root 2 = 1.414213562373095
Square Root 4 = 2.0
Square Root 5 = 2.23606797749979
```

ตัวอย่าง 2-8-2

จากตัวอย่าง 2-8-1 ซึ่งเราหาค่ารากที่สองของตัวเลขใดๆ โดยหาซ้ำๆ ไป 100 รอบ เราก็จะได้ความละเอียดของค่าที่หามาประมาณหนึ่ง ซึ่งยิ่งหาหลายรอบมากขึ้นก็จะยิ่งได้ค่าใกล้เคียงความจริงมากขึ้น เพราะยังไงมันก็เป็นค่าไม่รู้จบ

ทีนี้ลองมาปรับปรุงโปรแกรมสักหน่อย

อันดับแรกคือมีการตรวจสอบว่าค่าที่ส่งเข้ามาเท่ากับหรือน้อยกว่าศูนย์ก่อนหรือเปล่า ถ้า n เป็น 0 ก็ return 0 เลย ไม่ต้องหาต่อ แต่ถ้า n น้อยกว่า 0 ก็ให้ return “Cannot Calculate” คำนวณไม่ได้

จริงๆ ถ้าคิดแบบคณิตศาสตร์ยังคำนวณต่อได้ แต่โปรแกรมนี้ไม่ได้ทำเผื่อไว้ ก็เลยดักไว้ก่อน

จุดต่อไป

```
while abs(n2-n)>0.0001:
```

บรรทัดนี้คือให้ตรวจสอบว่าถ้าค่า n2-n หรือ n-n2 มากกว่า 0.0001 ก็ยังให้ทำซ้ำไปเรื่อยๆ

ทีนี้เราใส่ฟังก์ชัน abs(n2-n) คือเพื่อป้องกันกรณีที่ n>n2 ซึ่งจะทำให้ค่าที่ได้ติดลบ

abs(2) เท่ากับ 2 และ abs(-2) ก็เท่ากับ 2 เราเรียกฟังก์ชัน abs() นี้ว่า absolute หรือการหาค่าสัมบูรณ์

มีอีกจุดที่ปรับปรุงเพิ่มจากตัวอย่าง 2-8-1 คือในโปรแกรมที่แล้ว เราตรวจสอบค่า n2==n ก่อน แล้วค่อยตรวจสอบมากกว่าน้อยกว่า แต่ในความเป็นจริง โอกาสที่ n2 จะเท่ากับ n แทบจะไม่มีเลย มีแต่มากกว่ากับน้อยกว่าอยู่ตลอด ดังนั้นเราเอาไว้ตรวจสอบเป็นลำดับสุดท้ายจะดีกว่าไม่ต้องมาตรวจสอบกันบ่อยๆ

นอกจากนั้นก็ปรับปรุงอีกจุดหนึ่งคือแทนที่จะหาแค่รากที่สอง ก็ให้หารากลำดับใดๆ ได้ด้วย เช่น รากที่ 3 ของ 27 =3
หรือเขียนได้เป็น $\sqrt[3]{27} = 3$ หรือ 27 = 3*3*3 เป็นต้น

สุดท้ายคือ คำสั่ง round(n_mid,3) อันนี้คือให้แสดงค่าทศนิยมของตัวแปร n_mid แค่สามตำแหน่ง

```
#Python 3.9.5
#Example 2-8-2

def root(n,order=2):
  'หาค่ารากลำดับใดๆ ของ n'
  if n==0:
    return 0
  elif n<0:
    return "Cannot Calculate"

  n0,n1,n2 = 1,n,0
  while abs(n2-n)>0.0001:
    n_mid = (n0+n1)/2
    n2 = n_mid**order
    if n2 < n:
      n0 = n_mid
    elif n2 > n:
      n1 = n_mid
    elif n2 ==n:
      return round(n_mid,3)
  return round(n_mid,3)

if __name__ == "__main__":
  print ("Root 2 of -2 =",root(-2))
  print ("Root 2 of 0 =",root(0))
  print ("Root 2 of 2 =",root(2))
  print ("Root 2 of 3 =",root(3))
  print ("Root 2 of 4 =",root(4))
  print ("Root 3 of 8 =",root(8,3))
  print ("Root 4 of 81 =",root(81,4))
  print ("Root 2.5 of 3 =“,root(3,2.5))
```

```
Root 2 of -2 = Cannot Calculate
Root 2 of 0 = 0
Root 2 of 2 = 1.414
Root 2 of 3 = 1.732
Root 2 of 4 = 2.0
Root 3 of 8 = 2.0
Root 4 of 81 = 3.0
Root 2.5 of 3 = 1.552
```

ตัวอย่าง 2-8-3

ภาษา Python นั้นสามารถคำนวณเลขยกกำลังได้ เช่น

> 2 ยกกำลัง 3 ก็เขียนว่า 2**3 ได้เท่ากับ 8
> 3 ยกกำลัง 2 ก็เขียนว่า 3**2 ได้เท่ากับ 9

ทีนี้ถ้าเราเรียนคณิตศาสตร์กันมาดีๆ และไม่ได้หลับตอนครูสอนเรื่องเลขชี้กำลัง เราก็จะรู้ว่า

> รากที่สองของ 2 หรือ √2 นั้น เขียนให้อยู่ในรูปของ 2 ยกกำลัง ½ หรือ 2**(½) หรือ 2**0.5 ได้ด้วยละ

ตามตัวอย่างนี้เลย

มีค่าที่แปลกๆ และได้คำตอบไม่ตรงคือ root 2 of -2 นะครับ เข้าใจว่าคอมพิวเตอร์ยังคำนวณได้ผลไม่ตรงเสียทีเดียว แต่ก็ใกล้ความจริงมาก

คือถ้าเราทดสอบด้วยการสั่ง

```
print((8.659560562354934e-17+1.4142135623730951j)**2)
```

จะได้ผลลัพธ์คือ

```
-2.0000000000000004+2.449293598294707e-16j
```

หรือ -2 กว่าๆ นิดๆ บวกด้วยพจน์จินตภาพน้อยๆๆๆๆ มากๆๆๆๆ หรือถ้าปัดๆ เศษ รวมๆ แล้วก็ได้ -2 นี่แหละ

```
#Python 3.9.5
#Example 2-8-3
'หา root ด้วยสูตร root n of x = x**(1/n)'

if __name__ == "__main__":
   print ("Root 2 of -2 =",(-2)**(1/2)) #Wrong Answer
   print ("Root 2 of 0 =",0**(1/2))
   print ("Root 2 of 2 =",2**(1/2))
   print ("Root 2 of 3 =",3**(1/2))
   print ("Root 2 of 4 =",4**(1/2))
   print ("Root 3 of 8 =",8**(1/3))
   print ("Root 4 of 81 =",81**(1/4))
   print ("Root 2.5 of 3 =",3**(1/2.5))
```

```
Root 2 of -2 =
(8.659560562354934e-17+1.4142135623730951j)
Root 2 of 0 = 0.0
Root 2 of 2 = 1.4142135623730951
Root 2 of 3 = 1.7320508075688772
Root 2 of 4 = 2.0
Root 3 of 8 = 2.0
Root 4 of 81 = 3.0
Root 2.5 of 3 = 1.551845573915
```

ตัวอย่าง 2-8-4

อันนี้เป็นตัวอย่างแถม คือใน math module ของภาษา Python มีคำสั่งใช้หารากที่สองได้โดยตรงครับ ตามตัวอย่างนี้เลย

คำสั่ง math.sqrt() นี้ใช้หารากของเลขจำนวนลบไม่ได้นะครับ เช่น ถ้าสั่ง

```
print (math.sqrt(-2))
```

ก็จะขึ้นว่า math domain error

```
#Python 3.9.5
#Example 2-8-4
import math

'หา root ด้วย math module'

if __name__ == "__main__":
   print ("Root 2 of 0 =",math.sqrt(0))
   print ("Root 2 of 2 =",math.sqrt(2))
   print ("Root 2 of 3 =",math.sqrt(3))
   print ("Root 2 of 4 =",math.sqrt(4))
```

```
Root 2 of 0 = 0.0
Root 2 of 2 = 1.4142135623730951
Root 2 of 3 = 1.7320508075688772
Root 2 of 4 = 2.0
Root 2.5 of 3 = 1.5518455739153598
```

Chapter29

PRIME & FACTOR

จำนวนเฉพาะและตัวประกอบ

2-9-1 หาจำนวนเฉพาะ (Prime Number)
2-9-2 หาตัวประกอบ (Factorize)

ตัวอย่าง 2-9-1

โปรแกรมหาค่าจำนวนเฉพาะ ตัวอย่างนี้เป็นโปรแกรมที่มักใช้เป็นรอบคัดเลือก จนถึงรอบแข่งขันของการแข่งขันเขียนโปรแกรมเสมอๆ และเป็นโจทย์ที่เหมาะแก่การไว้ฝึกฝน ให้ผู้เรียนได้หัดคิดแก้ปัญหา

ซึ่งวิธีคิดก็ง่ายมาก แค่หาว่าเลขอะไรที่ไม่มีค่าอื่นๆ นอกจาก 1 และตัวมันเองที่หารมันลงตัว จำนวนนั้นก็เป็นจำนวนเฉพาะ แล้วเอาค่าที่ได้มาเก็บใส่ลิสต์ไว้ แล้ว return ลิสต์ออกมา

```
#Python 3.9.5
#Example 2-9-1

def prime(n):
   'หาจำนวนเฉพาะตั้งแต่ 2 ถึง n'

   aList=[2]
   for num in range(n):
      if num>1:
         for i in range(2,num):
            if num%i ==0:
               break
            elif i==num-1:
               aList.append(num)
   return aList

if __name__ == "__main__":
   print ("จำนวนเฉพาะที่น้อยกว่า 97 คือ",prime(97))
   print ("\n")
   print ("จำนวนเฉพาะที่น้อยกว่า 200 คือ",prime(200))
```

จำนวนเฉพาะที่น้อยกว่า 97 คือ [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89]

จำนวนเฉพาะที่น้อยกว่า 200 คือ [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199]

ตัวอย่าง 2-9-2

โปรแกรมหาตัวประกอบของจำนวนใดๆ ตัวประกอบก็คือจำนวนเฉพาะทุกตัวที่หารจำนวนนั้นลงตัว เช่น

ตัวประกอบของ 8 คือ 2
ตัวประกอบของ 15 คือ 3,5
ตัวประกอบของ 45 คือ 3,5

โปรแกรมเหมือนจะยาก แต่ถ้าคิดดีๆ ค่อยๆ คิดจะง่ายมาก

โปรแกรมนี้จะหาตัวประกอบซ้ำออกมาด้วย เช่น factorize(8) จะได้ 2, 2, 2 ซึ่งเป็นค่าซ้ำ

เราสามารถกำจัดค่าซ้ำนี้ได้โดยการจับมันใส่เซ็ต ด้วยคำสั่ง set() เช่น

print (set([2,2,2])) จะได้คำตอบเป็น {2}

หรือ

print (set([2,2,2,3,5])) จะได้คำตอบคือ {2, 3, 5}

มีคำสั่ง join และ map ไม่อธิบายละ ให้หาความหมายกันเอง

```
#Python 3.9.5
#Example 2-9-2

def factorize(num):
  'แยกตัวประกอบ'

  aList = []
  i=2
  while i<=num:
    if num%i==0:
      num=num/i
      aList.append(i)
      i=2
    else:
      i+=1
  return aList

if __name__ == "__main__":
  print (factorize(0))
  print (factorize(10))
  print (factorize(8))
  print (*factorize(16))
  print (*factorize(30))
  print ("factor of", str(2*3*7*2*5*11*11), \
  "is",*set(factorize(2*3*7*2*5*11*11)))
  print (' * '.join(map(str,factorize(99020))), \
    " = 99020")
  print (' * '.join(map(str,factorize(990200))), \
    " = 990200")
  print ("factor of 990200 is",set(factorize(990200)))
```

```
[]
[2, 5]
[2, 2, 2]
2 2 2 2
2 3 5
factor of 50820 is 2 3 5 7 11
2 * 2 * 5 * 4951  = 99020
2 * 2 * 2 * 5 * 5 * 4951  = 990200
factor of 990200 is {2, 5, 4951}
```

Chapter210

GCD, LCM

หรม. (หารร่วมมาก, GCD Greatest Common Divisor)
และ ครน. (คูณร่วมน้อย, LCM Least Common Multiple)

2-10-1 หาค่าหรม. ของเลขสองจำนวน
2-10-2 หาค่าหรม. ของเลขหลายจำนวน
2-10-3 ดัดแปลงตัวอย่าง 2-10-2 ให้สั้นลง
2-10-4 หาค่าหรม. ด้วยโมดูล math
2-10-1 หาค่าครน. ของเลขสองจำนวน
2-10-2 หาค่าครน. ของเลขหลายจำนวน

ตัวอย่าง 2-10-1

โปรแกรมหาค่าหารร่วมมาก หรม. หรือ Greastest Common Diversor

> หรม. คือตัวเลขที่มีค่ามากที่สุดที่สามารถหารตัวเลขทุกตัวในลิสต์ได้ลงตัว

ซึ่งการจะหาหรม.ของตัวเลขหลายๆ ตัว วิธีง่ายที่สุดคือหาหรม.ของตัวเลขหนึ่งคู่ใดๆ ในลิสต์ให้ได้เสียก่อน

วิธีการคือ

> นำตัวเลขคู่ใดๆ มาหารกัน ถ้าหารกันได้ลงตัวเลยแปลว่าตัวหารนั้นเป็น หรม.
>
> แต่ถ้าไม่ลงตัว เอาตัวหารมาเป็นตัวตั้ง แล้วเอาเศษที่ได้จากการหารรอบก่อน มาเป็นตัวหาร
>
> ทำซ้ำไปเรื่อยๆ จนกว่าจะหารลงตัวหรือได้เศษเป็น 0 ตัวเลขสุดท้ายที่ได้ก็จะเป็นค่าหรม. ที่ต้องการ

อธิบายแล้วดูเหมือนยาก แต่โปรแกรมจริงๆ ง่ายมาก เขียนแค่ไม่กี่บรรทัด

ในฟังก์ชัน gcd มีคำสั่งที่ปิดไว้คือ

```
#print (n1,n2)
```

ตรงนี้ถ้าอยากรู้ว่าโปรแกรมทำงานอย่างไร ก็ลองเอา # ออก ก็จะเห็นการทำงานในแต่ละวงรอบว่าหารกันอย่างไร ได้คำตอบอย่างไร

```python
#Python 3.9.5
#Example 2-10-1

def gcd(n1,n2):
    'หารร่วมมาก(หรม.) greatest common divisor(gcd) สองจำนวน'

    while n2!=0:
        #print (n1,n2)
        n1,n2 = n2, n1%n2
    return n1

if __name__ == "__main__":
    print ("หรม. 5,3 =",gcd(5,3))
    print ("หรม. 8,2 =",gcd(8,2))
    print ("หรม. 2,8 =",gcd(2,8))
    print ("หรม. 6,8 =",gcd(6,8))
    print ("หรม. 35,77 =",gcd(35,77))
    print ("หรม. 625,50 =",gcd(625,50))
    print ("หรม. 72,21 =",gcd(72,21))
```

```
หรม. 5,3 = 1
หรม. 8,2 = 2
หรม. 2,8 = 2
หรม. 6,8 = 2
หรม. 35,77 = 7
หรม. 625,50 = 25
หรม. 72,21 = 3
```

ตัวอย่าง 2-10-2

โปรแกรมหาค่าหรม. แบบหลายๆ ค่า ดัดแปลงมาจากโปรแกรมที่หาหรม.ทีละคู่ในตัวอย่าง 2-10-1

โดยนำค่าที่ต้องหารหา ใส่ไว้ในลิสต์ แล้วหาไปทีละคู่ โดยลบตัวเลขที่หาเสร็จแล้วออกจากลิสต์ไปเรื่อยๆ จนเหลือตัวสุดท้ายก็จะเป็นหรม. ของทั้งลิสต์ที่เราต้องการ

```
#Python 3.9.5
#Example 2-10-2

def gcd(aList):
   'หารร่วมมาก(หรม.) greatest common divisor (gcd) หลายจำนวน'
   while len(aList)>1:
      n1 = aList[0]
      n2 = aList[1]
      while n2!=0:
         n1,n2 = n2,n1%n2
      aList[1] = n1
      del aList[0]
   return n1

if __name__ == "__main__":

   print ("หรม 10,15,20,30 =",gcd([10,15,20,30]))
   print ("หรม 45,30,60,120 =",gcd([45,30,60,120]))
   print ("หรม 49,35,77,70 =",gcd([49,35,77,70]))
   print ("หรม 26,39 =“,gcd([26,39]))
```

```
หรม 10,15,20,30 = 5
หรม 45,30,60,120 = 15
หรม 49,35,77,70 = 7
หรม 26,39 = 13
```

ตัวอย่าง 2-10-3

โปรแกรมนี้ดัดแปลงจากโปรแกรมในตัวอย่าง 2-10-2 ให้สั้นลงเฉยๆ แต่การทำงานเหมือนกันทุกอย่าง เพียงแต่ว่าเอาตัวเลขจากในลิสต์มาใช้โดยตรง ไม่ได้เอาค่าตัวแปรอื่นมารับค่า

```
#Python 3.9.5
#Example 2-10-3

def gcd(aList):
   'หารร่วมมาก(หรม.) greatest common divisor (gcd) หลายจำนวน'

   while len(aList)>1:
      while aList[1]!=0:
         aList[0],aList[1] = aList[1],aList[0]%aList[1]
      del aList[1]
   return aList[0]

if __name__ == "__main__":
   print ("หรม 10,15,20,30 =",gcd([10,15,20,30]))
   print ("หรม 45,30,120,60 =",gcd([45,30,120,60]))
   print ("หรม 49,35,77,70 =",gcd([49,35,77,70]))
   print ("หรม 26,39 =“,gcd([26,39]))
```

```
หรม 10,15,20,30 = 5
หรม 45,30,120,60 = 15
หรม 49,35,77,70 = 7
หรม 26,39 = 13
```

ตัวอย่าง 2-10-4

และจริงๆ แล้วเราก็ไม่ต้องเขียนโปรแกรมหรม.เองก็ได้ เพราะภาษา Python เตรียมไว้ให้เราแล้วในโมดูล math ซึ่งนำเข้ามาใช้ด้วยคำสั่ง

```
import math
```

คำสั่ง math.gcd() จะหาค่าหรม.ได้จากตัวแปรสองตัวเท่านั้น แต่เราสามารถหาหรม.ของเลขหลายจำนวนได้โดยทำซ้ำทีละสองจำนวนไปเรื่อยๆ

```
#Python 3.9.5
#Example 2-10-4

'หารร่วมมาก(หรม.) ด้วย math library'
import math

if __name__ == "__main__":
   print ("หรม. 5,3 =",math.gcd(5,3))
   print ("หรม. 2,8 =",math.gcd(2,8))
   print ("หรม. 6,8 =",math.gcd(6,8))
   print ("หรม. 35,77 =",math.gcd(35,77))
   print ("หรม. 625,50 =",math. gcd(625,50))
   print ("หรม. 49,35,77 =", \
      math.gcd(math.gcd(math.gcd(49,35),77),70))
```

```
หรม. 5,3 = 1
หรม. 2,8 = 2
หรม. 6,8 = 2
หรม. 35,77 = 7
หรม. 625,50 = 25
หรม. 49,35,77 = 7
```

ตัวอย่าง 2-10-5

หาหรม. กันแล้ว ก็ต้องต่อด้วยครน. (คูณร่วมน้อย Least Common Multiply LCM) **กันต่อ**

ครน. คือจำนวนที่น้อยที่สุดที่ทุกตัวในลิสต์สามารถหารลงตัว ซึ่งครน. ของเลขสองจำนวนใดๆ ก็หาได้จากสูตร

(n1*n2)/gcd(n1,n2)

ดูจากตัวอย่างได้เลย

```
#Python 3.9.5
#Example 2-10-5

def gcd(n1,n2):
   'หารร่วมมาก(หรม.) สองจำนวน'
   while n2!=0:
      n1,n2 = n2, n1%n2
   return n1

def lcm(n1,n2):
   'คูณร่วมน้อย(ครม.) least common multiple (lcm) สองจำนวน'
   return int(n1*n2/gcd(n1,n2))

if __name__ == "__main__":
   print ("ครน. 20,30 =",lcm(20,30))
   print ("ครน. 3,7 =",lcm(3,7))
   print("ครน. 15,30 =“,lcm(15,30))
```

```
ครน. 20,30 = 60
ครน. 3,7 = 21
ครน. 15,30 = 30
```

ตัวอย่าง 2-10-6

หาครน. สองจำนวนได้ ก็หาหลายจำนวนได้ โดยหาทีละสองจำนวนซ้ำไปเรื่อยๆ เหมือนเดิม

```
#Python 3.9.5
#Example 2-10-6

def gcd(n1,n2):
   'หารร่วมมาก(หรม.) สองจำนวน'
   while n2!=0:
      n1,n2 = n2, n1%n2
   return n1

def lcm(aList):
   'คูณร่วมน้อย(ครน.) หลายจำนวน'
   ans = aList[0]
   for i in aList[1:]:
      ans = int(ans*i/gcd(ans,i))
   return ans

if __name__ == "__main__":
   print ("ครน.20,30,40,5=",lcm([20,30,40,5]))
   print ("ครน.3,7,3=",lcm([3,7,2]))
   print("ครน.15,30,45,60=",lcm([15,30,45,60]))
   print("ครน.2,5,10,12=",lcm([2,5,10,12]))
```

```
ครน.20,30,40,5= 120
ครน.3,7,3= 42
ครน.15,30,45,60= 180
ครน.2,5,10,12= 60
```

Chapter211

SORT

Sort คือการจัดเรียงข้อมูล

2-11-1 Selection Sort
2-11-2 Bubble Sort
2-11-3 Python Sort

ตัวอย่าง 2-11-1

โจทย์โปรแกรม Sort หรือโปรแกรมเรียงข้อมูล เป็นตัวอย่างยอดนิยมที่คนเรียนเขียนโปรแกรมจะต้องเรียนรู้เป็นตัวอย่างแรกๆ

Selection Sort เป็นโปรแกรมเรียงข้อมูลแบบพื้นฐาน เขียนง่าย เข้าใจง่าย แต่ประสิทธิภาพต่ำ

การทำงานคือไล่ตัวเลขไปทีละตัว โดยเริ่มจากตัวแรกในลิสต์ แล้วเปรียบเทียบกับตัวที่เหลือ เพื่อหาตัวที่มีค่าน้อยที่สุด เมื่อได้ค่าน้อยที่สุดก็เอาสลับที่กับตัวแรกในลิสต์ แล้วจากนั้นก็ทำจากตัวที่สองไปเรื่อยๆ จนหมด

ถ้าอยากรู้ว่าในแต่ละรอบโปรแกรมทำงานอย่างไร ก็เอาเครื่องหมาย # หน้าบรรทัด

```
#print (aList)
```

ในบรรทัดที่ 12 ออกไปให้เหลือแค่

```
print (aList)
```

ก็จะเห็นการทำงาน หรือการเปลี่ยนแปลวค่าในลิสต์ในแต่ละรอบ

```
#Python 3.9.5
#Example 2-11-1

def SelectionSort(aList):
   'Selection Sort'

   for i in range(len(aList)):
      min = i
      for j in range(i+1,len(aList)):
         if aList[min]>aList[j]:
            min =j
      #print (aList)
      aList[i],aList[min]= aList[min],aList[i]
   return aList

if __name__ == "__main__":
   aList=[20,12,3,18,7,8,25,14]
   print ("Unsorted List = ",aList)
   print ("Sorted List = “,SelectionSort(aList))
```

```
Unsorted List =  [20, 12, 3, 18, 7, 8, 25, 14]
Sorted List =  [3, 7, 8, 12, 14, 18, 20, 25]
```

ตัวอย่าง 2-11-2

Bubble Sort เป็นโปรแกรม Sort ยอดนิยมอีกแบบหนึ่ง ซึ่งใช้การตรวจสอบค่าในลิสต์ไปทีละคู่และสลับค่ามาก-น้อยทุกคู่ที่เจอในรอบเดียวไปเลยจนจบ แล้วค่อยกลับมาตั้งต้นใหม่อีกรอบ ทำให้จำนวนรอบที่ทำงานน้อยกว่า Selection Sort

อธิบายแล้วงง สามารถดูการทำงานแต่ละรอบได้โดยเอาเครื่องหมาย # หน้าบรรทัด

```
#print (aList)
```

ในบรรทัดที่ 10 ออกไปให้เหลือแค่

```
print (aList)
```

```
#Python 3.9.5
#Example 2-11-2

def BubbleSort(aList):
   'Bubble Sort'
   n= len(aList)
   for i in range(len(aList)):
      for j in range(0,n-i-1):
         if aList[j]>aList[j+1]:
            #print (aList)
            aList[j],aList[j+1]=aList[j+1],aList[j]
   return aList

if __name__ == "__main__":
   aList=[20,12,3,18,7,8,25,14]
   print ("Unsorted List = ",aList)
   print ("Sorted List = ",BubbleSort(aList))
```

```
Unsorted List =  [20, 12, 3, 18, 7, 8, 25, 14]
Sorted List =  [3, 7, 8, 12, 14, 18, 20, 25]
```

ตัวอย่าง 2-11-3

หัดเขียนโปรแกรม Sort กันมาสองแบบ แต่ในความเป็นจริงผู้สร้างภาษา Python เล็งเห็นแล้วว่าการ sort หรือการเรียงข้อมูลนี่เป็นคำสั่งยอดนิยมที่ได้ใช้งานกันเรื่อยๆ เขาก็เลยใส่คำสั่ง sort ไว้เป็นคำสั่งพื้นฐานสำหรับจัดการกับลิสต์ไว้ให้เลย โดยมีให้ใช้สองรูปแบบ คือ

```
sorted(List)
```

และ

```
List.sort()
```

```
#Python 3.9.5
#Example 2-11-3

if __name__ == "__main__":
   aList=[20,12,3,18,7,8,25,14]
   print ("Unsorted List = ",aList)
   print ("Sorted List 1 = ",sorted(aList))
   print ()
   aList=[20,12,3,18,7,8,25,14]
   print ("Unsorted List = ",aList)
   aList.sort()
   print ("Sorted List 2 = “,aList)
```

```
Unsorted List =  [20, 12, 3, 18, 7, 8, 25, 14]
Sorted List 1 =  [3, 7, 8, 12, 14, 18, 20, 25]

Unsorted List =  [20, 12, 3, 18, 7, 8, 25, 14]
Sorted List 2 =  [3, 7, 8, 12, 14, 18, 20, 25]
```

Chapter212

SEARCH

การค้นหาข้อมูลในลิสต์

2-12-1 Linear Search
2-12-2 Binary Search

ตัวอย่าง 2-12-1

Linear Search ทำงานง่ายมากก็คือเอาค่าที่ต้องการไล่หาไปเรื่อยๆ ในลิสต์จนกว่าจะเจอ แล้วคืนค่ามาเป็นตำแหน่งในลิสต์

```
#Python 3.9.5
#Example 2-12-1

def LinearSearch(aList,key):
   'LinearSearch return index'

   for i in range(len(aList)):
      if aList[i]==key:
         return i
   return "Not Found!!!"

if __name__ == "__main__":
   aList=[20,12,3,18,7,8,25,14]
   print (aList)
   print ("Search 20 = ",LinearSearch(aList,20))
   print ("Search 18 = ",LinearSearch(aList,18))
   print ("Search 14 = ",LinearSearch(aList,14))
   print ("Search 30 = ",LinearSearch(aList,30))
```

```
[20, 12, 3, 18, 7, 8, 25, 14]
Search 20 =  0
Search 18 =  3
Search 14 =  7
Search 30 =  Not Found!!!
```

ตัวอย่าง 2-12-2

Binary Search จะ search ได้เร็วกว่า Linear Search เพราะไม่ต้องเอาค่าไปวิ่งเปรียบเทียบกับทุกตัวในลิสต์ แต่ใช้วิธีเรียงหรือ Sort ข้อมูลก่อน ซึ่งเมื่อข้อมูลถูกจัดเรียงแล้ว

การหาค่าที่ต้องการก็จะทำได้ง่ายมาก เหมือนเวลาเราเล่นทายตัวเลข คือเอาค่าที่ต้องการจะหาไปตรวจสอบกับค่าที่อยู่กึ่งกลางของ list ก่อน

ถ้าค่าที่ต้องการหามีค่ามากกว่าค่ากึ่งกลาง ก็ไปตรวจสอบเฉพาะครึ่งบนของลิสต์ โดยเทียบทีละกึ่งกลางไปเรื่อยๆ

เป็นตัวอย่างการ Search ข้อมูลอีกวิธีหนึ่ง ซึ่งยังมีอีกหลายวิธี จะอยู่ในบทเรียนเกี่ยวกับ algorithm โดยเฉพาะ

```
#Python 3.9.5
#Example 2-12-2

def BinarySearch(aList,key):
   'Binary Search return location of key in list'

   first = 0
   last = len(aList)-1
   i = 0
   found = False

   while (first<=last) and found==False:
      mid = (first+last)//2
      if aList[mid] == key:
         i=mid
         found = True
      else:
         if key<aList[mid]:
            last = mid-1
         else:
            first = mid+1

   if found == False:
      return "Not Found!!!"
   else:
      return i

if __name__ == "__main__":
   aList=[20,12,3,18,7,8,25,14]
   print ("Unsorted List = ", aList)
   aList.sort()
   print ("Sorted List = ",aList)
   print ("Search 3 = ",BinarySearch(aList,3))
   print ("Search 18 = ",BinarySearch(aList,18))
   print ("Search 25 = ",BinarySearch(aList,25))
   print ("Search 30 = “,BinarySearch(aList,30))
```

```
Unsorted List =  [20, 12, 3, 18, 7, 8, 25, 14]
Sorted List =  [3, 7, 8, 12, 14, 18, 20, 25]
Search 3 =  0
Search 18 =  5
Search 25 =  7
Search 30 =  Not Found!!!
```

ถ้าเราได้ทดลองพิมพ์โปรแกรมต่างๆ ตามตัวอย่างในหนังสือเล่มนี้มาจนถึงตอนนี้ เราจะเริ่มมีความเข้าใจรูปแบบของภาษา Python ได้ประมาณหนึ่ง และเริ่มอ่านโปรแกรมออก

ถึงตรงนี้ บางคนอาจจะรู้สึกเหมือนกับว่ายังไม่สามารถเขียนโปรแกรมได้ หรือเมื่อเจอโจทย์ยากๆ ก็เหมือนกับยังเขียนไม่ได้ ไม่รู้จะเริ่มต้นอย่างไร ก็ยังไม่ต้องตกใจครับ การเขียนโปรแกรมคอมพิวเตอร์ก็เปรียบได้กับการเขียนตัวอักษร แรกเริ่มก็ต้องรู้จัก ก-กา, ข-ขา อาพาตามานา ป้ากับปู่กู้อีจู้ อ่านตัวอักษรให้ออก มีความคุ้นเคยกับตัวอักษร กับวิธีการสะกดประมาณหนึ่งก่อน จากนั้นจึงค่อยๆ นำตัวอักษรเหล่านี้มาเรียบเรียงเป็นคำ เป็นประโยค จนถึงขั้นแต่งเป็นเรื่องสั้นหรือนิยายต่อไป

คำแนะนำที่อยากให้ทำกันต่อก็คือ ค่อยๆ ศึกษาจากตัวอย่างโปรแกรมของผู้อื่น หรือจากแหล่งอื่น เช่น github http://www.github.com หรือ stack overflow https://stackoverflow.com/ ซึ่งเป็นแหล่งรวบรวมโปรแกรมตัวอย่างจำนวนมาก ค่อยๆ อ่าน ค่อยๆ ศึกษา หรือนำมาลองฝึกเขียนตาม เราก็จะค่อยๆ เก่งขึ้นไปตามลำดับได้เอง

ขอให้สนุกสนานกับการเขียนโปรแกรมครับ

Jo Sriyapan
CoderDojo Thailand