ตัวอย่าง **1.1.1**

หลังจากติดตั้งตัวแปรภาษา Python เสร็จแล้ว หรืออาจจะเขียนผ่านเครื่องมือ online เราก็สามารถเริ่มเขียนโปรแกรม Python ได้เลยครับ

โปรแกรมแรกที่นิยมเขียนกัน หรือเรียกว่าเป็นท่าไหว้ครูเวลาใครจะหัดเขียนโปรแกรมใหม่ๆ ภาษาอะไรก็มักจะเริ่มจากโปรแกรม hello, world หรือสั่งให้คอมพิวเตอร์แสดงคำว่า hello, world บนหน้าจอ

สำหรับภาษา Python เราก็สามารถเขียนได้ง่ายๆ คือ

print (“hello, world”)

ตรงที่เขียนว่า #Example 1.1.1 เพื่อบันทึกเฉยๆ ไม่มีผลอะไร

คำสั่งที่คอมพิวเตอร์จะเริ่มทำงานคือ

print (“hello, world”)

จะแสดงผลคำว่า hello, world บนหน้าจอ

#Example 1.1.1

#Python 3.7.3

print ("hello, world”)

**Example 1.1.1**

hello, world

ตัวอย่าง **1.1.2**

หรือจะแสดงผลเป็นภาษาไทยก็ได้

***TIP***

เท่าที่สืบค้นกันมาได้เล่ากันว่าบุคคลแรกที่ใช้ *hello, world* เป็นตัวอย่างในการเขียนโปรแกรมอย่างเป็นทางการคือคุณไบรอันเคอร์นิแกน *(Brian Kernighan)* โดยเป็นส่วนหนึ่งของเอกสารประกอบภาษา *BCPL ( Basic Combined Programming Language)* จากนั้นก็มาอยู่ในตำราภาษา *C* ฉบับคลาสสิคและใช้กันแพร่หลายที่สุดซึ่งแต่งขึ้นโดยคุณเดนนิสริชชี่ *(Dennis ritchie)* ร่วมกับไบรอันเคอร์นิแกนจากนั้นก็มาเป็นตัวอย่างในเอกสารประกอบภาษา *C++* และแพร่หลายมาเรื่อยๆจนกลายเป็นโปรแกรมพื้นฐานที่สุดที่นิยมใช้กันสำหรับทดสอบการใช้ภาษาโปรแกรมมิ่งต่างๆที่นิยมเริ่มเขียนโปรแกรมแรกด้วย

*print (“hello, world”)*

#Example 1.1.2

#Python 3.7.3

print (“hello, world")

print ("สวัสดี ชาวโลก")

**Example 1.1.2**

hello, world

สวัสดี ชาวโลก

ตัวอย่าง **1.1.3**

เราสามารถเขียนโปรแกรมเพื่อคำนวณทางคณิตศาสตร์ได้ โดย

* ใช้แทนการบวก
* ใช้แทนการลบ
* ใช้แทนการคูณ

/ ใช้แทนการหาร

% ใช้ในการหารเอาเศษ เช่น 8%3 จะได้เศษ 2

// หารเอาจำนวนเต็ม เช่น 10//3 จะได้ 3

\*\* แทนเครื่องหมายยกกำลัง เช่น 2\*\*3 ได้ 8

e แทนการคูณด้วย 10 ยกกำลัง n

เช่น 2e3 = 2\*(10\*\*3) จะได้เท่ากับ 2000

**Example 1.1.3**

7

4

20

2.0

1.5

2.3333333333333335

0

2

3

8

2000.0

2000

4

24

7

#Example 1.1.3

#Python 3.7.3

print (3+4)

print (6-2)

print (4\*5)

print (4/2)

print (3/2)

print (7/3)

print (4%2)

print (8%3)

print (10//3)

print (2\*\*3)

print (2e3)

print (2\*(10\*\*3))

print (10-2\*3)

print ((10-2)\*3)

print (3\*2+1)

ตัวอย่าง **1.1.4**

คำสั่ง print สามารถแสดงผลข้อมูลหลายชุด โดยคั่นแต่ละชุดด้วยเครื่องหมาย ,

เมื่อคั่นด้วยเครื่องหมาย , ข้อความจะถูกเว้นวรรคหนึ่งช่อง

เช่น print (1+1,3\*4,”ABC”) ผลจะได้ 2 12 ABC

เครื่องหมาย , นี้ในภาษา Python เรียกว่าตัว separator ซึ่งปกติจะแสดงผลเป็นการเว้นวรรคหนึ่งช่อง แต่สามารถเปลี่ยนให้แสดงผลแบบอื่นได้อีกด้วย จากตัวอย่าง

print ("Hello","World","!") แสดงผล Hello World !

print ("Hello","World","!",sep="") แสดงผล HelloWorld!

**Example 1.1.4**

2 12 ABC

3x4 = 12

Hello World !

HelloWorld!

Hello..World..!

Hello-World-1-2-3-4-5

print ("Hello","World","!",sep="..") แสดงผล Hello..World..!

print ("Hello","World",1,2,3,4,5,sep="-")

แสดงผล Hello-World-1-2-3-4-5

#Example 1.1.4

#Python 3.7.3

print (1+1,3\*4,"ABC")

print ("3x4 = ",3\*4)

print (“Hello","World","!")

print (“Hello","World","!",sep="")

print (“Hello","World","!",sep="..")

print ("Hello","World",1,2,3,4,5,sep="-")

ตัวอย่าง **1.1.5**

ปกติแล้วเมื่อจบคำสั่ง print จะเป็นการขึ้นบรรทัดใหม่โดยอัตโนมัต แต่เราสามารถเปลี่ยนให้ไม่ขึ้นบรรทัดใหม่ หรือให้เว้นหลายๆ บรรทัดก็ได้ ด้วยคำสั่ง end=

เช่น

print (1,2,3, end = “-“)

Print (4,5,6)

จะได้ผลลัพท์ 1 2 3-4 5 6

Print (1,2,3,end=“\n\n”)

จะเป็นการสั่งให้ขึ้นบรรทัดใหม่สองครั้ง เนื่องจาก “\n” เป็นคำสั่งให้ขึ้นบรรทัดใหม่

**Example 1.1.5**

1 2 3

4 5 6

7 8 910 11 12

1 2 3-4 5 6

1 2 3

4 5 6

7 8 9

10

#Example 1.1.5

#Python 3.7.3

print (1,2,3)

print (4,5,6)

print (7,8,9,end="")

print (10,11,12)

print (1,2,3,end="-")

print (4,5,6)

print ()

print (1,2,3,end="\n")

print (4,5,6,end="\n\n")

print (7,8,9,end="\n\n\n")

print (10)

ตัวอย่าง **1.1.6**

ภาษา Python ใช้เครื่องหมาย \ ตามด้วยตัวอักษร เพื่อแสดงผลพิเศษ ซึ่งเรียกว่า escape character ดังนั้นถ้าต้องการแสดงผลตัวอักษร \ ก็ต้องใช้ \\ แทน หรือกรณีต้องการแสดงผล “Hello” ก็ต้องใช้เป็น \” ดังตัวอย่าง print (“\”Hello\””) จะได้ “Hello”

ตัวอย่าง escape character เพิ่มเติม

\n = ขึ้นบรรทัดใหม่

\n\n = ขึ้นบรรทัดใหม่สองครั้ง

\t = เว้นวรรค tab

\’ = ‘

**Example 1.1.6**

"Hello"

\Hello\

Hello

World!

Hello

World!

Hello World!

'Hello'

Ʃ

\U000001a9 รหัสยูนิโค้ดเครื่องหมาย Ʃ ถ้าเปลี่ยนค่าตรงนี้ก็จะแสดงรหัสยูนิโค้ดของระบบออกมาได้ ตารางรหัสยูนิโค้ด (unicode) หาได้จาก google

#Example 1.1.6

#Python 3.7.3

print (“\"Hello\"")

print ("\\Hello\\")

print (“Hello\n")

print ("World!")

print ("Hello\n\n")

print ("World!")

print ("Hello\tWorld!")

print ("\'Hello\'")

print ("\U000001a9")

ตัวอย่าง **1.1.7**

กรณีคำสั่งยาวๆ เขียนไม่จบในหนึ่งบรรทัดแล้วต้องการจะเขียนต่อในอีกบรรทัดหนึ่งก็ทำได้ง่ายๆ แค่ใน่ \ ไว้หลังคำสั่งที่ยังเขียนไม่จบ แล้วก็มาเขียนต่อบรรทัดถัดไปได้เลย

**Example 1.1.7**

Hello World!

3+5

8

#Example 1.1.7

#Python 3.7.3

print ("Hello \

World!")

print ("3+\

5")

print (3\

+5)

ตัวอย่าง **1.1.8**

ตัวอย่างการใช้คำสั่ง format ร่วมกับ print ไม่อธิบายละ ลองทดลองดูกัน

Print (“Hello {0} World! {1}”.format (3,10))

ตัวอักษรในเครื่องหมาย “” นี้เราเรียกว่าสายอักษรหรือ string ซึ่งเป็นสัญลักษณ์ที่ทำให้ภาษา Python เข้าใจว่าตัวอักษรในเครื่องหมาย “” นี้เป็นชนิดสายอักษรนะ ไม่สามารถนำมาคำนวณทางคณิตศาสตร์ได้

นอกจากนี้กรณีที่ต้องการแสดงผล string ด้วยคำสั่ง print โดยไม่ต้องการให้ติด escape character สามารถทำได้ด้วยการเติม r หน้า string นั้น เรียกว่า raw string สังเกตดูจากตัวอย่าง

ในตัวอย่าง

Print (r”Hel\tlo”,”World\t!”)

ผลออกมาเป็น

**Example 1.1.8**

Hello {0}

Hello 3 World! 10

Hello 10 World! 3

Hello World \n

Hello World \

Hello \t World \\ \n

Hel\tlo World !

Hello\t World\t

Hel\tlo World !

เพราะ r จะมีผลกับคำใน string ตัวแรกเท่านัน้

จะเห็นว่า แม้จะนำคำสองคำมาเชื่อมกันด้วย , หรือ + แต่ raw string จะมีผลเฉพาะคำที่มี r นำหน้าเท่านั้น ส่วนคำที่ไม่มี r นำหน้าก็จะแสดงผลของ escape character เหมือนเดิม

#Example 1.1.8

#Python 3.7.3

print ("Hello {0}")

print ("Hello {0} World! {1}".format(3,10))

print ("Hello {1} World! {0}".format(3,10))

print (r"Hello World \n")

print ("Hello\t World \\ \n")

print (r"Hello \t World \\ \n")

print (r"Hel\tlo","World\t!")

print (r"Hello\t",r"World\t")

ตัวอย่าง **1.1.9**

สำหรับบทนี้ เราได้ทำความรู้จักกับคำสั่ง print กันไปหลายแง่มุมจนน่าจะเบื่อ print กันไปแล้ว ดังนั้นก่อนจบบท ลองมาทำความรู้จักกับคำสั่งอื่นบ้าง

ภาษา Python สามารถสร้างฟังก์ชันเพื่อบรรจุโปรแกรมเอาไว้เรียกใช้ซ้ำๆ ได้ครับ สร้างฟังก์ชันด้วยคำสั่ง def ตามตัวอย่างเลย

คำสั่ง def example\_109(): จะเป็นตัวสร้างฟัง์ชันชื่อ example\_109 ส่วน (): บังคับใส่ไปก่อน เดี๋ยวค่อยว่ากันด้วยวิธีใช้

ภาษา Python ยังสามารถแทรกเอกสารไว้ในตัวโปรแกรมได้ โดยสร้างเป็นค่าคงที่สายอักษรหรือ string ไว้เป็นคำสั่งแรกภายในโปรแกรมหรือฟังก์ชันนั้น แล้วเรียกใช้ได้ด้วย \_\_doc\_\_ หรือ function\_name.\_\_doc\_\_

เรียกเท่ๆ ว่า docstrings หรือ Python Documentation Strings

**Example 1.1.9**

Hello!

สวัสดี

Hello!

สวัสดี

World!

Hello!

สวัสดี

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Function Name= example\_109

Function Docs= คำอธิบายฟังก์ชัน

แบบหลายบรรทัด

\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Program name= \_\_main\_\_

Program Docs= คำอธิบายโปรแกรม

นอกจากนี้ยังสามารถเรียกดูชื่อโปรแกรมได้ด้วย \_\_name\_\_

และดูชื่อของฟังก์ชันได้จาก function\_name.\_\_name\_\_

ซึ่งจะเห็นว่า \_\_name\_\_ ของโปรแกรมหลัก จะเป็น \_\_main\_\_ เสมอ

#Example 1.1.9

#Python 3.7.3

'คำอธิบายโปรแกรม'

def example\_109():

'''คำอธิบายฟังก์ชัน

แบบหลายบรรทัด'''

print (“Hello!")

print (“สวัสดี")

example\_109()

example\_109()

print ("World!")

example\_109()

print ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print ("Function Name=",example\_109.\_\_name\_\_)

print ("Function Docs=",example\_109.\_\_doc\_\_)

print ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print ("Program name=",\_\_name\_\_)

print("Program Docs=",\_\_doc\_\_)

*Tip*

ในตัวอย่าง *1.1.9* ถ้าเขียนแล้ว *error* ไม่ผ่านสักทีก็ควรข้ามมาอ่านตรงนี้ก่อนแล้วค่อยกลับไปเขียนใหม่น่าจะแก้ปัญหาได้

ในภาษา *Python* มีเทคนิคพิเศษอันหนึ่งเรียกว่า *indentation* หรือบล๊อคย่อหน้าคือภาษา *Python* สร้างบล๊อคหรือกลุ่มของคำสั่งด้วยการใช้ *indent* หรือย่อหน้าเป็นตัวกำหนดซึ่งปกติๆเราจะกำหนดย่อหน้านี้ด้วยการกดปุ่ม *tab* บนคีย์บอร์ด *(*ปุ่มซ้ายมือมักจะอยู่เหนือปุ่ม *caps lock)* หรือถ้าใช้การเคาะ *space bar (*แป้นยาวๆล่างสุดของคีย์บอร์ด*)* ก็ต้องเคาะให้มันเท่าๆกันจึงจะนับเป็นบล๊อคเดียวกันไม่เช่นนั้นก็ถือเป็นคนละบล๊อค

ภาษา *Python* เวอร์ชันใหม่ๆกำหนดให้การใช้ *tab* เท่ากับการเคาะ *space bar* แล้วดังนั้นจะเคาะ *space* หรือกด *tab* ก็ถือว่าไม่ต่างกันขอให้ลำดับคั่นหน้ามันตรงกันก็ใช้ได้แล้วแต่ถ้ามันไม่เท่ากันก็จะ *error* ตรงนี้คนเขียนโปรแกรม *Python* มือใหม่ๆพลาดกันเยอะ

แต่ถ้าไปเจอ *Python* เวอร์ชันเก่าหน่อยซึ่ง *space bar* ไม่เหมือนกับ *tab* ก็ต้องดูดีๆบางที *error* เอาง่ายๆเพราะเราดูว่ามันย่อหน้าเท่ากันแต่อันหนึ่งเป็นย่อหน้าจากการเคาะ *space* หลายทีแต่อีกอันเป็นย่อหน้าจากการกด *tab* ทีเดียว

โปรแกรมที่อยู่ใน *indentation block* หรือในย่อหน้าเดียวกันก็จะถือว่าอยู่ในลำดับชั้นเดียวกันและเป็นบล็อคลูกของคำสั่งในลำดับชั้นก่อนหน้านี้เช่น

*def example\_109():*

*print (“Hello!”)*

*print (“*สวัสดี*”)*

*example\_109()*

จากตัวอย่างนี้บล๊อคลูกของ *def example\_109():* ก็คือคำสั่ง

*print (“World!”)*

*print (“*สวัสดี*”)*

ซึ่งขอบเขตของบล๊อคลูกก็จะถูกกำหนดโดยตัวย่อหน้าหรือ *indent* นั่นเองนั่นคือ *def example\_109():* มีขอบเขตแค่นี้

ส่วนคำสั่งถัดมาคือ *example\_109()* ตัวล่างไม่ถือว่าอยู่ในบล็อคลูกแล้วแต่เป็นการเรียกใช้ฟังก์ชันตัวบนที่ถูกกำหนดไว้ก่อนแล้ว

อ่านแล้วงงไม่เป็นไรเขียนตามไปเรื่อยๆเดี๋ยวก็หายงงเอง

ข้อควรระวังคือในการกำหนดบล๊อคลูกคำสั่งก่อนหน้านั้นมักจะมี *:* อยู่ท้ายคำสั่งด้วยสังเกตให้ดีๆเพราะเป็นอีกจุดที่ผิดกันบ่อยเครื่องหมาย *:* หรือจุดสองจุดบนล่างนี้เรียกว่าโคล่อน *(colon)* คำนี้ถ้าเปิดพจนานุกรมจะแปลว่าลำไส้ใหญ่ไม่รู้เกี่ยวกันอย่างไรแต่ถ้าอยากหาให้เจอด้วยภาษาอังกฤษต้องหาด้วยคำว่า *colon punctuation* ภาษาไทยเรียกทวิภาคแปลว่าสองภาคทวิแปลว่า *2* ภาษอังกฤษเขียนว่า *two* คงมาจากรากเดียวกันแต่คนอังกฤษขี้เกียจอ่าน *“*ทวู*”* ก็เลยอ่านสั้นๆเป็น *“*ทู*"*

ที่สำคัญคืออย่าไปสับสนกับ *;* อันนี้เรียก *semiconlon* มีจุดอยู่ข้างบนและลูกน้ำอยู่ข้างล่างหรือภาษาไทยเรียกว่าอัฒภาคอัฒตัวนี้เขียนเหมือนกับอัฒจันทร์แปลว่าครึ่งก็ได้ซีกก็ได้อัฒภาคก็แปลว่าครึ่งภาค

ไหนๆก็พูดเรื่องจุดกับลูกน้ำแล้ว *.* จุดนี้ภาษาไทยเรียกมหัพภาคและลูกน้ำ *,* เรียกว่าจุลภาคเรียกแบบนี้ได้จะดูเท่มากครูจะชมเพื่อนจะร้องหู

ตัวอย่าง **1.2.1**

การใช้ตัวแปร (Varible)

เราสามารถกำหนดค่าให้กับตัวแปรภาษา Python ได้ง่ายๆ แค่ตั้งชื่อตัวแปร แล้วก็ใส่เครื่องหมายเท่ากับค่าที่ต้องการ เช่น

a=2

b=999

c=3.14

ซึ่งเป็นการให้ค่าที่เป็นตัวเลขเพื่อการคำนวณ ส่วนการให้ค่าเป็นตัวอักษรก็ทำได้โดยใส่เครื่องหมาย “” เข้าไป เช่น

d = “ABC”

e = “123”

หรือแม้แต่ให้เป็นค่าทางตรรกศาสตร์ คือ True หรือ False ได้ด้วย เช่น

f = True

g = False

โดยชื่อตัวแปรที่เป็นตัวอักษรใหญ่หรืออักษรตัวเล็กจะมีความหมายต่างกัน และไม่สามารถตั้งชื่อตัวแปรที่ขึ้นต้นด้วยตัวเลขหรือสัญลักษณ์ได้ (ยกเว้นสัญลักษณ์ \_ สามารถนำมาใช้นำหน้าชื่อตัวแปรได้)

**Example 1.2.1**

a= 2

b= 999988887777

c= 3.14159

d= ABC

e= 123 f= True F= False

3.14159 3.14159 3.14159

#Example 1.2.1

#Python 3.7.3

def example\_201():

a = 2

b = 999988887777

c = 3.14159

d = "ABC"

e = "123"

f = True

F = False

print ("a=",a)

print ("b=",b)

print ("c=",c)

print ("d=",d)

print ("e=",e,"f=",f, "F=",F)

a = b = c

print (a,b,c)

example\_201()

ตัวอย่าง **1.2.2**

สลับค่าตัวแปรสองตัวได้ง่ายๆ ด้วยการสั่ง

variable\_1, variable\_2 = variable\_2, variable\_1

หรือตามตัวอย่าง

b,a = a,b

หรือสลับทีละสามตัวก็ได้ ดังตัวอย่าง

a,b,c = c,a,b

#Example 1.2.2

**Example 1.2.2**

2 abc True

abc 2 True

True abc 2

#Python 3.7.3

def example\_202():

a = 2

b = "abc"

c = True

print (a,b,c)

b,a = a,b

print (a,b,c)

a,b,c = c,a,b

print (a,b,c)

example\_202()

ตัวอย่าง **1.2.3**

สามารถกำหนดค่าตัวแปรทีละหลายๆ ค่าได้ เช่น

a = b = c =2 กำหนดค่า a ให้เท่ากับ b และ c และเท่ากับ 2

d,e,f = 4,5,6 กำหนดค่าเป็นชุดให้ d=4, e=5, f=6

หรือผสมระหว่างสองวิธีแรกก็ได้ เช่น

i,j = x,y = 10,20 กำหนด i=x และ j=y และเท่ากับ 10 กับ 20

**Example 1.2.3**

a,b,c = 2 2 2

d,e,f = 4 5 6

i,j,x,y = 10 20 10 20

ดูตัวอย่าง

#Example 1.2.3

#Python 3.7.3

def example\_203():

a = b = c = 2

d, e, f = 4,5,6

i,j = x,y = 10,20

print ("a,b,c =",a,b,c)

print ("d,e,f = ",d,e,f)

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

example\_203()

ตัวอย่าง **1.2.4**

ความแตกต่างระหว่างตัวแปรแบบตัวเลขและสายอักษร (string)

เมื่อกำหนดตัวแปร Hello = 10

print (Hello, Hello+5, Hello\*10) จะได้ 10,15,100 ตามลำดับ

ทีนี้ลองกำหนดตัวแปร Hello = “10” มีเเครื่องหมาย “” มาด้วย ซึ่งทำให้โปรแกรมเข้าใจว่า “10” นี้ไม่ใช่เป็นตัวเลขนะแต่เป็น string

print (Hello, Hello+”5”, Hello\*5) จะได้ 10, 105 และ 1010101010

ผลคือ ได้ “10” ต่อกับ “5” และ Hello\*5 ก็เป็น Hello ต่อกัน 5 ครั้ง

บรรทัดนี้ถ้าใครเขียนผิดเป็น Hello+5 จะทำให้โปรแกรม error นะครับ เพราะจะเท่ากับ “10”+5 เป็น string ต่อกับตัวเลข ก็ error

คราวนี้ลอง Hello = “Hello” อันนี้เข้าใจได้ print (Hello) ก็ได้ Hello

Hello = Hello ไม่มีอะไรเกิดขึ้น print ออกมาก็ได้เหมือนเดิม

Print (Hello+Hello, Hello\*3) ได้ HelloHello HelloHelloHello

**Example 1.2.4**

1. 10 15 100

2. 10 105 1010101010

3. Hello

4. Hello

5. HelloHello HelloHelloHello

6. Hello!!!HelloHelloHello

7. Is Hello=="Hello" False

และลองกำหนดตัวแปรผสมค่าคงที่ string บ้าง

Hello=“Hello!!!”+Hello\*3 ผลออกมาได้ Hello!!!HelloHelloHello

สุดท้ายคือตรวจสอบว่า Hello == “Hello” หรือไม่ คำตอบคือ False ไม่เท่านะครับ

#Example 1.2.4

#Python 3.7.3

def example\_204():

Hello = 10

print ("1.",Hello, Hello+5, Hello\*10)

Hello = "10"

print ("2.",Hello, Hello+"5", Hello\*5)

Hello = "Hello"

print ("3.", Hello)

Hello = Hello

print ("4.", Hello)

print ("5.",Hello+Hello, Hello\*3)

Hello = "Hello!!!" + Hello\*3

print ("6.", Hello)

print ("7. Is Hello==\"Hello\"",Hello=="Hello")

example\_204()

example\_204()

ตัวอย่าง **1.2.5**

ลองเล่นกับตัวแปร

print ((a,b,c)\*2) ไม่ได้หมายถึงเอา 2 คูณเข้าไปในตัวแปรแต่ละตัว

แต่เป็นการแสดงผล (a,b,c) สองครั้ง ได้เป็น (1,3,5,1,3,5) สังเกตว่ามี () ติดมาด้วย

ตรงนี้เพราะภาษา Python มอง (1,3,5) เป็นตัวแปรอีกชนิดหนึ่งเรียกว่า tuple ซึ่งจะพูดถึงในบทถัดๆ ไป

**Example 1.2.5**

1 3 5

(1, 3, 5, 1, 3, 5)

(1, 3, 5, 1, 3, 5, 1, 3, 5)

1 6 15

#Example 1.2.5

#Python 3.7.3

def example\_205():

a,b,c = 1,3,5

print (a,b,c)

print ((a,b,c)\*2)

print ((a,b,c)\*3)

print (a\*1,b\*2,c\*3)

example\_205()

ตัวอย่าง **1.2.6**

ได้กล่าวไปก่อนแล้วว่าตัวแปรมีหลายชนิด ที่เคยใชักันแล้วคือชนิดตัวเลขและตัวอักษร แต่จริงๆ ตัวแปรในภาษา Python มีหลายชนิดมากกว่านั้น ซึ่งจะถูกกำหนดชนิดโดยอัตโนมัตเมื่อมีการกำหนดค่า แต่สามารถขอดูชนิดของตัวแปรได้ด้วยคำสั่ง type(variable\_name)

ดูตัวอย่าง

a=9; b=3.4; c=3e4; สามารถกำหนดค่าหลายๆ ชุดได้ หรือทำหลายๆ คำสั่งในบรรทัดเดียวกันได้ โดยแยกแต่ละคำสั่งด้วยเครื่องหมาย ;

a=9 <class ‘int’> ตัวแปรชนิด int หรือ integer หรือเลขจำนวนเต็ม

b=3.4 <class ‘float’> ตัวแปรชนิด float หรือ floating point หรือเลขทศนิยม

d=3+4j <class ‘complex’> อันนี้คือจำนวนจินตภาพ คนเรียนคณิตศาสตร์รู้จักดี

e=Hello <class ‘str’> ตัวแปรชนิด str คือ string หรือสายอักษร สำหรับเก็บค่าตัวอักษร

f=True <class ‘bool’> คือตัวแปรชนิด boolean หรือตัวแปรตรรกะ

g=None <class ‘NoneType’> มีตัวแปรชนิด NoneType หรือไม่ระบุชนิดด้วย

**Example 1.2.6**

a= 9 <class 'int'>

b= 3.4 <class 'float'>

c= 30000.0 <class 'float'>

d= (3+4j) <class 'complex'>

e= Hello <class 'str'>

f= True <class 'bool'>

g= None <class 'NoneType'>

h= [] <class 'list'>

i= [3, 4, 5, 'Hello'] <class 'list'>

j= () <class 'tuple'>

k= (3, 4, 5, 'Hello') <class 'tuple'>

h = [] <class ‘list’> ตัวแปรชนิด list เดี๋ยวค่อยมาว่ากันละเอียดอีกที i ก็เป็น <class ‘list’>

และ j กับ k เป็น <class ‘tuple’> ได้ยินชื่อนี้มาครั้งหนึ่งแล้ว เดี๋ยวค่อยมาว่ากันละเอียดๆ

#Example 1.2.6

#Python 3.7.3

def example\_206():

a = 9; b=3.4; c=3e4;

d = 3+4j; e = "Hello"; f = True;

g = None

h = []

i = [3,4,5,"Hello"]

j = ()

k = (3,4,5,"Hello")

print ("a=",a,type(a))

print ("b=",b,type(b))

print ("c=",c,type(c))

print ("d=",d,type(d))

print ("e=",e,type(e))

print ("f=",f,type(f))

print ("g=",g,type(g))

print ("h=",h,type(h))

print ("i=",i,type(i))

print ("j=",j,type(j))

print ("k=",k,type(k))

example\_206()

ตัวอย่าง **1.2.7**

การดำเนินการทางตรรกะ

& คือ และ (and)

| คือ หรือ (or)

== คือ เท่ากับหรือไม่ (is equal?)

!= คือ ไม่เท่ากับหรือไม่ (is not equal?)

ตัวดำเนินการทางตรรกะ จะได้คำตอบเป็น True หรือ False

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

ลองดูจากตัวอย่างหรือทดลองเปลี่ยนค่าในตัวอย่างดู

#Example 1.2.7

#Python 3.7.3

**Example 1.2.7**

1. a+b = 17

2. a-b = 1

3. True and False is False

4. True or False is True

5. Not True is False

6. a == a is True

7. a == b is False

8. not (a==b) is True

9. a>b is True

10. a>a is False

11. a>=a is True

12. a!=a is False

def example\_207():

a,b = 9,8

c,d = True, False

print ("1. a+b =",a+b)

print ("2. a-b =",a-b)

print ("3. True and False is ", c&d)

print ("4. True or False is ",c|d)

print ("5. Not True is ", not(c))

print ("6. a == a is",a==a)

print ("7. a == b is",a==b)

print ("8. not (a==b) is",not(a==b))

print ("9. a>b is",a>b)

print ("10. a>a is",a>a)

print ("11. a>=a is",a>=a)

print ("12. a!=a is",a!=a)

example\_207()

ตัวอย่าง **1.2.8**

ศึกษาเรื่องชนิดของตัวแปรต่อ

ตัวอย่างข้อ 7. 8. 9. แสดงให้เห็นว่า

int 3+3 = 6

float 3.0+3.0 = 6.0

และ string “3” + “3” = “33"

#Example 1.2.8

#Python 3.7.3

def example\_208():

print ("1. 3e5 =",3e5, type(3e5))

print ("2. 3\*10\*\*5 =",3\*10\*\*5,type(3\*10\*\*5))

print ("3. 3.12345e5 =",3.12345e5,type(3.12345e5))

**Example 1.2.8**

1. 3e5 = 300000.0 <class 'float'>

2. 3\*10\*\*5 = 300000 <class 'int'>

3. 3.12345e5 = 312345.0 <class 'float'>

4. 3.12345\*10\*\*5 = 312345.0 <class 'float'>

5. interger of 3.5 = 3

6. float of 3 = 3.0

7. 3+3 = 6

8. float of 3+3 = 6.0

9. string of 3 + string of 3 = 33

print ("4. 3.12345\*10\*\*5 =",3.12345\*10\*\*5, \

type(3.12345\*10\*\*5))

print ("5. interger of 3.5 =",int(3.5))

print ("6. float of 3 =",float(3))

print ("7. 3+3 =",3+3)

print ("8. float of 3+3 =",float(3+3))

print ("9. string of 3 + string of 3 =",\

str(3)+str(3))

example\_208()

ตัวอย่าง **1.2.9**

กรณีมีการกระทำทางคณิตศาสตร์กับตัวแปรตัวหนึ่งแล้วนำค่ามาเก็บไว้ในตัวแปรเดิม เช่น

a=a+10

สามารถเขียนสั้นลงเป็น

a+=10

ได้

และใช้ได้กับตัวกระทำทางคณิตศาสตร์อื่นๆ ด้วยเช่น

a-=5 หมายถึง a=a-5

a\*=2 หมายถึง a=a\*2

#Example 1.2.9

**Example 1.2.9**

1. 10

2. 20

3. 30

4. 25

5. 50

6. 10.0

#Python 3.7.3

def example\_209():

a = 10; print (“1. ",a)

a = a+10; print (“2. ",a)

a+=10; print (“3. ",a)

a-=5; print (“4. ",a)

a\*=2; print (“5. ",a); a/=5; print (“6. ",a)

example\_209()

ตัวอย่าง **1.3.1**

การรับข้อมูลผ่านคีย์บอร์ด ทำได้ด้วยคำสั่ง input โดยเขียนอยู่ในรูป

Variable = input (“xxx”)

จะแสดงผล xxx บนหน้าจอ และเมื่อเรากดป้อนอะไรเข้าไป ค่าที่ป้อนเข้าไปจะไปเก็บอยู่ในตัวแปร Variable โดยมี type เป็น string

ซึ่งถ้าเราต้องการนำค่าในตัวแปรนี้ไปใช้ในการคำนวณทางคณิตศาสตร์ต่อก็ต้องแปลงค่าก่อนให้เป็น int หรือ float

**Example 1.3.1**

1. x= 10

2. x= 10 <class 'str'>

3. int(x) = 10 <class 'int'>

4. float(x) = 10.0 <class 'float'>

#Example 1.3.1

#Python 3.7.3

def example\_301():

x = input ("Input x ")

print ("1. x=",x)

print ("2. x=",x,type(x))

x = int(x)

print ("3. int(x) =",x,type(x))

x = float(x)

print ("4. float(x) =",x,type(x))

example\_301()

ตัวอย่าง **1.3.2**

การตรวจสอบเงื่อนไขด้วยคำสั่ง if

if x==10 คือการตรวจสอบว่า

x เท่ากับ 10 หรือไม่

ถ้าใช่ก็ทำบรรทัดต่อไปในบล๊อค คือ print (“yes, x =“,x)

if y==10: ตรวจสอบค่าของ y ว่าเท่ากับ 10 หรือไม่

ซึ่งในกรณีนี้ y ไม่เท่ากับ 10 เพราะเรากำหนดค่าตั้งต้นให้ y = 11

โปรแกรมก็จะข้ามคำสั่ง print (“Yes, y =“,10) ไป ไม่แสดงออกมา

แต่ถ้าอยากให้โปรแกรมพิมพ์บรรทัดนี้ออกมา ก็อาจจะลองเปลี่ยนค่าตรง y = 11 ให้เป็น y=10 แล้วลองเรียกโปรแกรมดูอีกที

**Example 1.3.2**

Yes, x = 10

ทบทวนเรื่อง indent หรือย่อหน้ากันอีกที ในตัวอย่างนี้ จะมีบล๊อคลูกคือ print (“Yes, x=“,x) เป็นบล๊อคลูกของ if x==10:

และ

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

ซึ่งบล๊อคลูกจะทำงานก็ต่อเมื่อเงื่อนไขของคำสังก่อนนั้นเป็นจริงเท่านั้น

#Example 1.3.2

#Python 3.7.3

def example\_302():

x = 10

y = 11

if x==10:

print ("Yes, x =",x)

if y==10:

print ("Yes, y =",10)

example\_302()

ตัวอย่าง **1.3.3**

การเปรียบเทียบว่าค่าสองค่าเท่ากันหรือไม่ นอกจากจะใช้เครื่องหมาย == แล้ว ยังสามารถใช้ is ได้ด้วย

ซึ่งจริงๆ แล้ว == กับ is นั้นเป็นคนละความหมายและใช้ต่างกัน แต่ในบางกรณี เช่นกรณีนี้ ก็สามารถนำ is มาใช้แทน == ได้

**Example 1.3.3**

Yes, x = 10

#Example 1.3.3

#Python 3.7.3

def example\_303():

x = 10

y = 11

if x is 10:

print ("Yes, x =",x)

if y is 10:

print ("Yes, y =",y)

example\_303()

ตัวอย่าง **1.3.4**

ใช้คำสั่ง input เพื่อรับค่าจากคีย์บอร์ด ลองมาทำโปรแกรมทายตัวเลขแบบง่ายๆ

เพิ่มเติมเรื่องคำสั่ง if else

If x==“10”:

<คำสั่งที่1>

else:

<คำสั่งที่2>

If คู่กับ else จะใช้ในกรณีตรวจสอบว่าถ้าเป็นตามเงื่อนไขใน if จะทำ <คำสั่งที่1> แต่ถ้าไม่ตรงเงื่อนไข จะทำ <คำสั่งที่2>

**Example 1.3.4**

Input x (guess 1-10) 11

Noooo, try again (try 10)

Input x (guess 1-10) 10

Yes, you win X=10

#Example 1.3.4

#Python 3.7.3

def example\_304():

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

if x=="10":

print ("Yes, you win X=10")

else:

print ("Noooo, try again (try 10)")

example\_304()

ตัวอย่าง **1.3.5**

เพิ่มเติมจาก if else

เป็น if elif else:

สามารถใช้ elif ตรวจสอบมากกว่าหนึ่งเงื่อนไขได้

เช่น

if a:

<คำสั่งที่1>

elif b:

<คำสั่งที่2>

elif c:

<คำสั่งที่3>

else:

<คำสั่งที่4>

โปรแกรมจะตรวจสอบทีละเงื่อนไข ถ้าตรงกับเงื่อนไข a ทำ <คำสั่งที่1> ถ้าไม่ก็ตรวจว่าตรงกับเงื่อนไข b หรือเปล่า ถ้าใช่ก็ทำ <คำสั่งที่2>

ถ้าไม่ใช่อีก ก็ตรวจสอบว่าตรงกับเงื่อนไข c หรือเปล่า ...ไปเรื่อยๆ

และถ้าไม่เข้าเงื่อนไขใดๆ เลย ก็ทำ <คำสั่งที่4>

**Example 1.3.5**

Input x (guess 1-10) 8

Noooo, try again (try 11)

Input x (guess 1-10) 11

Too much, try again (try 10)

Input x (guess 1-10) 10

Yes, you win x=10

#Example 1.3.5

#Python 3.7.3

def example\_304():

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

if x=="10":

print ("Yes, you win x=10")

elif x=="9":

print ("Too little, try again (try 11)")

elif x=="11":

print ("Too much, try again (try 10)")

else:

print ("Noooo, try again (try 11)")

example\_304()

ตัวอย่าง **1.3.6**

คำสั่งทำงานวนรอบ while <condition>:

เป็นคำสั่งให้ทำงานในบล็อค while ซ้ำไปเรื่อยๆ ตราบที่ <condition> ยังไม่เป็นจริง เช่นในตัวอย่างนี้คือ x เริ่มมาเป็น 0 โปรแกรมก็จะทำงานไปเรื่อยๆ

แต่ถ้าต้องการเบรคการทำงานเอง ก็กดปุ่ม control-c โปรแกรมก็จะหยุดการทำงาน

#Example 1.3.6

#Python 3.7.3

def example\_306():

**Example 1.3.6**

Input x 4

Too little, try again

Input x 11

Too much, try again

Input x 10

Yes, you win x=10

End Game!!!

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 much, try again")

elif x<10:

print ("Too little, try again")

print ("End Game!!!")

example\_306()

ตัวอย่าง **1.3.7**

คำสั่ง import ใช้สำหรับการนำ module ภายนอกมาใช้งาน ซึ่งอาจจะมีทั้ง module มาตรฐานที่ติดมากับตัวภาษา Python เอง หรือสามารถโหลดมาเพิ่มเติม หรือเขียนเองก็ได้

Import random คือการโหลด module random เข้ามา ทำให้เรามีคำสั่ง random หรือสร้างตัวเลขสุ่มไว้ใช้งาน

**Example 1.3.7**

ได้ตัวเลขสุ่มระหว่าง 1-10 แต่ละครั้งไม่เหมือนกัน

#Example 1.3.7

#Python 3.7.3

import random

def example\_307():

x = random.randint(1,10)

print (x)

example\_307()

ตัวอย่าง **1.3.8**

นำคำสั่งทั้งหมดที่ได้เรียนรู้มาถึงตอนนี้ มาสร้างเกมทายตัวเลขแบบง่ายๆ ได้แล้ว

#Example 1.3.8

#Python 3.7.3

import random

def example\_308():

x = 0

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 much, try again")

elif x<y:

print ("Too little, try again")

print ("End Game!!!")

example\_308()

ตัวอย่าง **1.4.1**

ตัวอย่างนี้สั้นๆ ง่ายๆ ว่าด้วยเรื่อง loop หรือการทำงานแบบวนรอบ โดยกำหนดรอบด้วยคำสั่ง for ตัวแปร in <ขอบแขตการนับ>

ในตัวอย่างก็ใช้ตัวแปร i เป็นตัวนับหรือทับศัพท์ว่า counter โดยนับในขอบเขต range(10) ซึ่งมีค่าตั้งแต่ 0-9

คำสั่ง

for i in range(10):

print (i))

โปรแกรมจะทำงานในบล๊อคลูกของคำสั่ง for คือ ให้ print (i) ไปจนกว่าจะหลุดจาก for ในกรณีนี้คือเมื่อนับครบ range(10) ซึ่งเท่ากับโปรแกรมจะ print(i) สิบรอบ โดยแต่ละรอบ ค่า i จะเปลี่ยนไปตามค่าที่อ่านได้จาก range(10) คือเริ่มตั้งแต่ 0, 1, 2, 3 ไปจนถึง 9

**Example 1.4.1**

0

1

2

3

4

5

6

7

8

9

ถ้าจะไม่ให้งง ทดลองเขียนแล้วดูจากตัวอย่างดีกว่า

#Example 1.4.1

#Python 3.7.3

def example\_401():

for i in range(10):

print (i)

example\_401()

ตัวอย่าง **1.4.2**

จากตัวอย่างเรื่องการใช้ for i in range(10) เราสามารถกำหนดค่าตั้งต้นใน range ได้

เช่นในตัวอย่าง for i in range(2,10): หมายถึงเริ่มต้นนับที่ 2 เอาค่าไปใส่ที่ i แล้วทำซ้ำไปเรื่อยๆ จนกว่าจะถึง 10 ...เมื่อถึง 10 ก็จะหยุดการทำงาน คือไม่ทำ ไม่เอาค่า 10 ไปแจก i และไม่ทำงานในบล๊อคลูก

ดังนั้นคำสั่ง print (i,i+1) จะ print ไปเรื่อยๆ ตั้งแต่ 2-9 เท่านั้น

**Example 1.4.2**

2 3

3 4

4 5

5 6

6 7

7 8

8 9

9 10

ในบล๊อคลูก คำสั่ง print(i,i+1) จะเห็นว่านอกจาก print(i) แล้ว เรายังนำ i ไปใช้ประโยชน์อื่นได้อีก เช่นเอาไปบวกกับ 1 แล้วแสดงผลออกมาก็ได้ดังตัวอย่าง

#Example 1.4.2

#Python 3.7.3

def example\_402():

for i in range(2,10):

print (i,i+1)

example\_402()

ตัวอย่าง **1.4.3**

ตัวอย่างนี้แสดงการกำหนดค่าใน range() จากคำสั่ง

For I in range(1,11,2):

ค่าใน range(x,y,z) มีสามตัวโดย

x จะเป็นค่าเริ่มต้น

y จะนับไม่เกินค่านี้

z แต่ละรอบให้ + เพิ่มไป z ซึ่งถ้าไม่ระบุ จะหมายถึงเพิ่มทีละ 1

ดังนั้นในตัวอย่างนี้ range (1,11,2) คือจะเริ่มจาก 1 นับไม่เกิน 11 และเพิ่มทีละ +2 ก็จะได้

**Example 1.4.3**

1

3

5

7

9

รอบแรก 1

รอบสอง 1+2 3

รอบสาม 3+2 5

รอบสี่ 5+2 7

รอบห้า 7+2 9

รอบหก 9+2 =11 สิ้นสุดการนับ ไม่ทำงานในบล๊อคลูก ไม่มีการนำ 11 ไปใส่ในตัวแปร i และไม่ทำคำสั่ง print (i)

#Example 1.4.3

#Python 3.7.3

def example\_403():

for i in range(1,11,2):

print (i)

example\_403()

ตัวอย่าง **1.4.4**

คำสั่ง for ทำงานตามค่าใน range() ซึ่งค่า ใน range() สามารถกำหนดให้เป็นค่าถอยหลังก็ได้ ดังตัวอย่าง

for i in range(10,1,-1)

range (10,1,-1) จะเริ่มที่ค่า 10 ถึงไม่เกิน 1 โดยลดค่าทีละ -1 ได้ผลตามตัวอย่างเลย

**Example 1.4.4**

10

9

8

7

6

5

4

3

2

#Example 1.4.4

#Python 3.7.3

def example\_404():

for i in range(10,1,-1):

print (i)

example\_404()

ตัวอย่าง **1.4.5**

ลองเล่นกับ range แปลกๆ บ้าง เช่น

range(-10,10,2)

ก็เข้าใจไม่ยากนะ เริ่มที่ -10 ถึงไม่เกิน 10 เพิ่มทีละ 2

**Example 1.4.5**

-10

-8

-6

-4

-2

0

2

4

6

8

#Example 1.4.5

#Python 3.7.3

def example\_405():

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

print (i)

example\_405()

ตัวอย่าง **1.4.6**

ลองเล่นแปลกๆ บ้าง เรารู้อยู่แล้วว่าคำสั่ง for จะเอาค่าจาก range() มาเติมให้กับตัวแปร i แล้วทีนี้ถ้าเรามาเปลี่ยนค่าของ i กลางทางจะเกิดอะไรขึ้น

จากตัวอย่าง

if i==5:

i=i+3

print (i)

ช่วงนี้ มีการตรวจสอบค่าของ i ด้วยคำสั่ง if ถ้าค่า i เป็น 5 ให้เอา i ไปบวกด้วย 3 แล้วเก็บค่าคืนเข้าไปใน i ซึ่งจะทำให้ i = 5+3 = 8

เมื่อสั่งให้โปรแกรมทำงาน จะเห็นว่าค่าที่ได้จะนับจาก 0 1 2 3 4 แล้วกระโดดไป 8 เลย ไม่มี 5

คำถามคือหลังจาก 8 แล้วเป็นอะไรต่อ 9 หรือ..ก็ไม่ จะเห็นว่า i กลับไปนับ 6 7 8 9 ต่อตามลำดับเดิม ดังนั้นก็จะเห็นได้ว่าการมาเปลียนค่าตัวแปร i กลางทาง กลาง loop ไม่ได้ส่งผลให้ค่าใน range() เปลี่ยนแปลงไปด้วย rang(10) ก็ยังประกอบด้วย 0 1 2 3 4 5 6 7 8 9 เหมือนเดิม เพียงแต่เอาค่าจาก range มาใส่ไว้ให้ใน i เท่านั้นเอง จะเอา i ไปทำอะไรต่อก็ได้ ไม่เกี่ยวกัน

**Example 1.4.6**

0

1

2

3

4

8

6

7

8

9

#Example 1.4.6

#Python 3.7.3

def example\_406():

for i in range(10):

if i==5:

i=i+3

print (i)

example\_406()

ตัวอย่าง **1.4.7**

การทำงานด้วยคำสั่งวนรอบในภาษา Python นอกจากคำสั่ง for ซึ่งเป็นการนับค่าในลิสต์แล้ว ยังมีอีกคำสั่งหนึ่ง คือคำสั่ง while

คำสั่ง while ไม่ได้ใช้การนับค่า แต่จะทำงานวนรอบไปเรื่อยๆ ตราบที่เงื่อนไขที่ตั้งไว้ยังเป็นจริง หรือยังมีค่าเป็น True อยู่

ตามตัวอย่างนี้ กำหนดค่าเริ่มต้น x=1

จากนั้นตรวจสอบค่าด้วยคำสั่ง while ถ้า x<10 ก็ให้ทำคำสั่งในบล๊อคลูกวนรอบไปเรื่อยๆ

บล๊อคลูกก็ถูกกำหนดโดย indentation หรือย่อหน้า ในที่นี้ก็มีสองคำสั่งคือ

print (“x=“,x)

**Example 1.4.7**

x= 1

x= 3

x= 5

x= 7

x= 9

และ

x+=2

คำสั่ง x+=2 เทียบเท่ากับ x=x+2 คือการนำค่าใน x มาบวกด้วย 2 แล้วนำไปเก็บไว้ในตัวแปร x ใช้ x เพื่อจะได้เห็นว่าตั้งชื่อตัวแปรอะไรก็ได้ ไม่ต้อง i ก็ได้

ดังนั้นผลที่ได้ก็คือ 1,3,5,7,9 รอบต่อไป 9+2=11 ก็มากกว่า 10 ตามเงื่อนไข โปรแกรมก็จะหลุดจาก loop ไม่ทำงานในบล๊อคลูกอีกต่อไป

#Example 1.4.7

#Python 3.7.3

def example\_407():

x=1

while x<10:

print ("x=",x)

x+=2

example\_407()

ตัวอย่าง **1.4.8**

ไม่มีอะไรแปลกใหม่ ใช้คำสั่ง while ตรวจสอบค่าของ y<10000 แต่ไม่ได้แปลว่าโปรแกรมต้องวน 10000 รอบ เพราะเรากำหนดให้ในแต่ละรอบ y\*=2 หรือ y=y\*2 ค่าของ y เพิ่มขึ้นสองเท่าในทุกๆ รอบ แป๊บเดียวก็เกิน 10000 แล้ว

จากตัวอย่างนี้ทำให้เห็นการใช้งานของ while ที่ต่างจาก for

ใน for loop นั้น จะทำงานเป็นจำนวนรอบเท่ากับผลที่ได้จากคำสั่ง range() แต่ใน while นั้นใช้การตรวจสอบเงื่อนไข ซึ่งอาจจะเป็นอะไรก็ได้

**Example 1.4.8**

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

#Example 1.4.8

#Python 3.7.3

def example\_408():

y=2

while y<10000:

print ("y=",y)

y\*=2

example\_408()

ตัวอย่าง **1.4.9**

ตัวอย่างนี้เป็นการใช้ while True

while True ก็คือเป็นจริง (True) ตลอดกาล โปรแกรมก็จะติด loop อนันต์ วนซ้ำไปเรื่อยๆ ไม่หลุดจาก loop ซึงเราสามารถใช้คำสั่ง break เพื่อออกจาก loop ได้ ...หรือถ้าพลั้งเผลอเขียนโปรแกรมแล้วติด loop อนันต์ โปรแกรมไม่ยอมหลุดจาก loop เราสามารถบังคับหลุดจาก loop ด้วยคีย์บอร์ดได้ด้วยการกดปุ่ม control c พร้อมๆ กัน หรือปุ่ม control d พร้อมๆ กัน ปุ่มนี้บางเครื่องเขียนว่า control บางเครื่องเขียนแค่ ctrl แต่มักจะอยู่ทางซ้ายมือ ใต้ปุ่ม shift หาไม่ยาก

**Example 1.4.9**

Input 1-10 (10 to Break) 9

x = 9

Input 1-10 (10 to Break) 8

x = 8

Input 1-10 (10 to Break) 10

End Loop

#Example 1.4.9

#Python 3.7.3

def example\_409():

while True:

x = input("Input 1-10 (10 to Break) ")

if x== "10":

print ("End Loop")

break

else:

print ("x = ",x)

#Can use Ctrl-C or Ctrl-D to break loop

example\_409()

*Tip*

ทำไมหลายคนชอบใช้ตัวแปร *i* กับคำสั่ง *for* เช่น *for i in range(n):*

โปรแกรมเมอร์บางคนอาจใช้ตัวแปรอื่นเช่น *for counter in range(n):* หรือ *for index in range(n)* ก็มีแต่หลายๆตัวอย่างก็จะเจอ *for i* โผล่ออกมาเรื่อยๆแถมมาในหลายๆภาษาด้วย

ถ้าลองค้นดูใน *google* ก็จะเจอคำตอบที่หลากหลายเช่น *i* แทนคำว่า *index* บ้างหรือ *i, j* ใช้ในภาษาทางคณิตศาสตร์บ้าง

แต่คำตอบที่เชื่อว่าน่าจะเป็นจุดเริ่มต้นจริงๆเพราะพอดีผู้เขียนเกิดทันยุคที่โปรแกรมเมอร์ทุกคนใช้ *for i* ตั้งแต่หลายสิบปีก่อน

คือในยุคโบร่ำโบราณนู้นสมัยที่คนจะเขียนโปรแกรมต้องเขียนลงในสมุดแล้วเอาคำสั่งไปเจาะรูปในกระดาษแข็งๆเรียกกันว่าการ์ดเจาะรูการ์ดหนึ่งแผ่นแทนหนึ่งคำสั่งโปรแกรมหนึ่งก็ใช้การ์ดปึ๊งหนึ่งแล้วเอาไปป้อนลงคอมพิวเตอร์ทีละใบๆให้คอมพิวเตอร์อ่านแล้วทำงานยุคนั้นมีภาษาคอมพิวเตอร์ที่เป็นที่นิยมอยู่ภาษาหนึ่งคือ *FORTRAN IV* ซึ่งก่อน *FORTRAN IV* ก็คงมีเวอร์ชันก่อนหน้านั้นเพราะคงไม่มีใครเกิดมาก็ตั้งชื่อเป็นเวอร์ชัน *IV* หรือ *4* เลยแต่ไม่เคยเห็นเวอร์ชันเก่ากว่านี้เกิดมาก็เจอเวอร์ชัน *IV* นี้แล้วเป็นที่นิยมมากซึ่งภาษานี้กำหนดให้ตัวแปรที่ตั้งชื่อนำหน้าด้วยอักษร *I* ถึง *N* เป็นตัวแปร *integer* หรือจำนวนเต็มโดยอัตโนมัตขณะที่ตัวแปรที่ขึ้นต้นด้วยอักษรอื่นเป็น *real (*หรือ *float* ในภาษา *Python)*

ซึ่งตัวแปรที่เราจะใช้มาเป็นตัวนับใน *loop* ได้ก็ต้องเป็นตัวแปรชนิดจำนวนเต็มหรือ *integer* นี้เองทำให้โปรแกรมเมอร์สมัยนั้นนิยมใช้ตัวแปร *i* แทนความหมายตัวแปรชนิด *integer* ทีนี้ภาษายุคที่ใกล้ๆกับ *FORTRAN* แล้วเริ่มมีการใช้ *for i* เท่าที่ได้ทันเห็นคือภาษา *BASIC* ซึ่งใช้รูปแบบคำสั่ง

*FOR nn = xx TO yy*

*…do something*

*NEXT nn*

โดย *nn* เป็นตัว *counter* เหมือนใน *Python* และด้วยความนิยมในการใช้ตัวแปร *i* จากภาษา *FORTRAN* ก็ติดมาถึงภาษา *BASIC* นี้ด้วยจากตัวอย่างโปรแกรมสมัยนั้นเป็น *for i* และถ้าเป็นและถ้ามีการใช้คำสั่ง *for* ซ้อนกันก็จะใช้ตัวแปรชื่อ *i* และ *j* ตามตัวอย่างนี้

*FOR I = 1 TO xx*

*FOR J = 1 TO yy*

*….do something*

*NEXT J*

*NEXT I*

*...*หลังจากนั้นไม่รู้ยังไงคำสั่ง *for i* นี้ก็สืบทอดต่อๆเป็นที่นิยมมาเรื่อยๆจนปัจจุบัน

ตัวอย่าง **1.5.1**

บทนี้ยาวหน่อย มาพูดกันเรื่องตัวแปรชนิด ลิสต์ (list) เรียกทับศัพท์ว่าลิสต์ก็แล้วกัน ไม่ต้องลำบากไปแปล

ตัวแปรลิสต์สร้างง่ายมาก คือกำหนดค่าเป็นชุดๆ ไว้ในวงเล็บก้ามปู หรือสัญลักษณ์แบบนี้ [ ] โดยคั่นข้อมูลแต่ละตัวด้วยเครื่องหมายมหัพภาค , หรือเรียกว่าลูกน้ำก็ได้

จากตัวอย่าง

a = [1,3,4,5]

และ

b = [“abc”,2,3,False]

แล้วลองแสดงค่า type (a) และ type (b) ซึ่งคอมพิวเตอร์ก็จะบอกออกมาเลยว่าเป็น class list หรือชนิดตัวแปรลิสต์ที่ว่ากันนี้

**Example 1.5.1**

1) a= [1, 2, 3, 4, 5]

2) b= ['abc', 2, 3, False]

3) type of a = <class 'list'>

4) type of b = <class ‘list'>

*TIP*

ภาษาอังกฤษเรียกเครื่องหมาย *[ ]* นี้ว่า *brackets* แต่ภาษาไทยเรียกวงเล็บทุกแบบว่านขลิขิตเหมือนกันหมดแล้วค่อยกำหนดชนิดย่อยของวงเล็บแบบนี้ว่าเป็นวงเล็บก้ามปู

#Example 1.5.1

#Python 3.7.3

def example\_501():

a = [1,2,3,4,5]

b = ["abc",2,3,False]

print ("1) a=",a)

print ("2) b=",b)

print ("3) type of a = ",type(a))

print ("4) type of b = ",type(b))

example\_501()

ตัวอย่าง **1.5.2**

ตัวอย่างนี้เข้าใจง่ายมาก คือเนื่องจากลิสต์นี่มีข้อมูลเป็นชุดๆ ดังนั้นจะอ้างอิงหรือจะเรียกข้อมูลตัวไหนออกมาก็อ้างด้วยตำแหน่งของข้อมูลในลิสต์นั้น โดยตั้งต้นนับจาก 0

การอ้างตำแหน่งก็ทำได้ด้วยการเขียนชื่อตัวแปรตามด้วยวงเล็บก้ามปูบอกตำแหน่ข้อมูลนั้น เช่น จากตัวอย่าง

a=[1,2,3,4,5]

a[0] ก็คือ 1 a[4] คือ 5

หรือ

b = [“abc”,2,3,False] b[0] ก็คือ “abc” b[1] คือ 2 b[3] คือ False

**Example 1.5.2**

1) a[1] = 2

2) b[0] = abc

และถ้าอ้างถึงข้อมูลที่ไม่มี เช่น b[10] โปรแกรมก็จะ error ลองทำดูก็ได้ เครื่องไม่ระเบิด แค่ error เฉยๆ

#Example 1.5.2

#Python 3.7.3

def example\_502():

a = [1,2,3,4,5]

b = ["abc",2,3,False]

print("1) a[1] =",a[1])

print("2) b[0] =",b[0])

print("3) a[-2] =",a[-2])

print("4 b[-1] =",b[-1])

example\_502()

ตัวอย่าง **1.5.3**

การแก้ไขค่าในตัวแปรลิสต์ทำได้ง่ายมาก คืออ้างถึงตัวแปรลิสต์ตำแหน่งที่ต้องการ แล้วก็กำหนดค่าเข้าไปตรงๆ เลย

เช่น

a=[1,2,3,4,5]

แล้วเรากำหนดค่า a[0] =7

ค่าของ a ก็จะกลายเป็น [7,2,3,4,5]

แถมเรายังเอาลิสต์ใส่ซ้อนเข้าไปในลิสต์ได้อีกด้วย ดูจากตัวอย่างได้เลย หรือลองแก้ไขค่าตำแหน่งอื่นๆ ในลิตส์ดู

**Example 1.5.3**

1) a = [1, 2, 3, 4, 5]

2) a = [7, 2, ['a', 'b', 'c,d,e', 8, 11], 4, 5]

#Example 1.5.3

#Python 3.7.3

def example\_503():

a = [1,2,3,4,5]

print ("1) a = ",a)

a[0]=7

a[2]=['a','b',"c,d,e",8,9+2]

print ("2) a = ",a)

example\_503()

ตัวอย่าง **1.5.4**

จากตัวอย่าง

a = [1, 2, [11, 12, 13], 4, (21, 22, 23)]

ค่าใน a มี 1,2, มีลิสต์ [11, 12,13] ตามด้วย 4 และ (21,22,23)

ค่าสุดท้าย (21,22,23) ดูคล้ายๆ ลิสต์แต่ดันอยู่ในวงเล็บธรรมดา ไม่ใช่วงเล็บก้ามปู มันต่างอย่าไร

จากตัวอย่าง

print (“1) a= “,a) ง่ายมาก ก็ print ค่า a ออกมาทั้งหมด

print (“2) a[2][1] = “,a[2][1]) พิสดารละ a[2][1] คืออะไร แต่ในเมื่อเรารู้ว่า a[2] คือ ลิสต์ [11,12,12] เราก็สามารถฉลาดรู้ได้เองว่า a[2][1] คือค่าในลิสต์ที่ซ้อนอยู่ในลิสต์นั่นเอง ในกรณีนี้ก็คือ ตัวที่ [1] ของลิสต์ [11,12,13] คำตอบคือ 12 ตามตัวอย่างเห็นกันอยู่แล้ว

print 3 ถึง 6 ก็ไม่ต้องอธิบายอะไรมาก คือตรวจสอบ type ของค่านั้นๆ

type(a) ก็คือ list

type(a[1]) ดูแล้วก็เป็น int

type(a[2]) ก็คือลิสต์ที่ซ้อนอยู่ ก็เป็น list แน่นอน

**Example 1.5.4**

1) a = [1, 2, [11, 12, 13], 4, (21, 22, 23)]

2) a[2][1] = 12

3) type of a is <class 'list'>

4) type of a[1] is <class 'int'>

5) type of a[2] is <class 'list'>

6) type of a[2][1] is <class 'int'>

7) type of a[4] is <class ‘tuple'>

type(a[2][1]) เรารู้ว่าค่า a[2][1]=12 ดังนั้น ถามแบบนี้ก็เหมือนถามหา type (12) ก็ย่อมเป็น int

พิสดารคือ type(a[4]) อันที่ค่ามันเป็น (21,22,23) เป็นตัวเลขในวงเล็บนี่ละคืออะไร คอมพิวเตอร์แสดงคำตอบออกมาว่าคือ tuple อ่านว่าทูเปิล เอาว่ารู้ว่ามี type ประหลาดๆ โผล่มาอีก type หนึ่ง เดี๋ยวค่อยมาทำความรู้จักกันละเอียดๆอีกที

#Example 1.5.4

#Python 3.7.3

def example\_504():

a = [1,2,[11,12,13],4,(21,22,23)]

print ("1) a = ",a)

print ("2) a[2][1] = ",a[2][1])

print ("3) type of a is",type(a))

print ("4) type of a[1] is",type(a[1]))

print ("5) type of a[2] is",type(a[2]))

print ("6) type of a[2][1] is",type(a[2][1]))

print ("7) type of a[4] is",type(a[4]))

example\_504()

ตัวอย่าง **1.5.5**

รู้จักกับคำสั่งที่ใช้ทำงานกับ list

append ใช้ใ่ข้อมูลเข้าไปในลิสต์ โดยข้อมูลที่ใส่เข้าไปจะไปต่อท้ายลิสต์เดิม นอกจากลิสต์นั้นเป็นลิสต์ว่าง ก็จะกลายเป็นข้อมูลแรกไป

extend เอาลิสต์ใหม่ไปต่อท้ายลิสต์เก่า

insert แทรกข้อมูลเข้าไปในลิสต์ ในตำแหน่งที่ต้องการ

index หาตำแหน่งของข้อมูลตัวนั้น โดยนับตำแหน่งแรกเป็น 0

ข้อสังเกต a.extend([“!”,7,8]) จะนำลิสต์ [“!”,7,8] ไป **“**ต่อกับ**”** ลิสต์ a

ส่วน a.append([0,0,0]) จะนำลิต์ [0,0,0] ไปใส่ไว้ **“**ใน**”** ลิสต์ a

#Example 1.5.5

#Python 3.7.3

def example\_505():

**Example 1.5.5**

1) a= []

2) append hello to a= ['hello']

3) append world to a= ['hello', 'world']

4) extend !,7,8 to a= ['hello', 'world', '!', 7, 8]

5) insert สวัสดี at position 2, a= ['hello', 'world', 'สวัสดี', '!', 7, 8]

6) find position of 7 = 4

7) find position of สวัสดี = 2

8) append [1,2,3] to a= ['hello', 'world', 'สวัสดี', '!', 7, 8, [0, 0, 0]]

a= []

print ("1) a=",a)

a.append("hello")

print ("2) append hello to a=",a)

a.append("world")

print ("3) append world to a=",a)

a.extend(["!",7,8])

print ("4) extend !,7,8 to a=",a)

a.insert (2,"สวัสดี")

print ("5) insert สวัสดี at position 2, a=",a)

print ("6) find position of 7 =",a.index(7))

print ("7) find position of สวัสดี =",a.index("สวัสดี"))

a.append ([0,0,0])

print ("8) append [1,2,3] to a=",a)

example\_505()

ตัวอย่าง **1.5.6**

เพิ่มคำสั่งในการทำงานกับลิสต์ count และ remove

count นับจำนวนข้อมูลในลิสต์ เช่น

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

a.count(3) จะได้ 3 เพราะมี 3 อยู่ในลิสต์ 3 ครั้ง

และ

a.count(6) จะได้ 2 เพราะมี 6 อยู่สองตัว

remove ใช้ลบข้อมูลออกจากลิสต์ ถ้ามีข้อมูลซ้ำกันหลายตัวก็จะลบออกทีละตัว โดยลบจากตำแหน่งน้อยที่สุดออกก่อน

ดังในตัวอย่าง

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

a.remove(3)

จะเหลือ a=[1,2,4,5,6,3,4,3,6] เป็นต้น

**Example 1.5.6**

1) a= [1, 2, 3, 4, 5, 6, 3, 4, 3, 6]

2) count 3? = 3

3) count 6? = 2

4) remove 3, a= [1, 2, 4, 5, 6, 3, 4, 3, 6]

5) remove 3, a= [1, 2, 4, 5, 6, 4, 3, 6]

6) remove 1, a= [2, 4, 5, 6, 4, 3, 6]

#Example 1.5.6

#Python 3.7.3

def example\_506():

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

print ("1) a=",a)

print ("2) count 3? =",a.count(3))

print ("3) count 6? =",a.count(6))

a.remove (3)

print ("4) remove 3, a=",a)

a.remove (3)

print ("5) remove 3, a=",a)

a.remove (1)

print ("6) remove 1, a=",a)

example\_506()

ตัวอย่าง **1.5.7**

คำสั่ง del x[n] ใช้ลบข้อมูลในตำแหน่งที่ n ออกจากลิสต์

pop() ใช้อ่านข้อมูลตัวสุดท้ายในลิสต์ และลบข้อมูลตัวนั้นออกจากลิสต์

pop (n) อ่านข้อมูลตำแหน่งที่ n จากลิสต์ และลบข้อมูลตัวนั้นออกไป

ซึ่ง pop() จะต่างกับ del ตรงที่ del ต้องกำหนดตัวที่จะลบ แล้วก็จะลบค่านั้นออกจากลิสต์ไปเลย ขณะที่ pop() เฉยๆ จะอ่านค่าตัวสุดท้าย หรือจากตำแหน่งที่ระบุออกมาให้ด้วยแล้วค่อยลบ

#Example 1.5.7

**Example 1.5.7**

1) x= ['a', 'b', 'c', 1, 3, 5, 7, [0, 1, 2]]

2) del x[0] x= ['b', 'c', 1, 3, 5, 7, [0, 1, 2]]

3) del x[3] x= ['b', 'c', 1, 5, 7, [0, 1, 2]]

4) x.pop()= [0, 1, 2] x= ['b', 'c', 1, 5, 7]

5) x.pop()= 7 x= ['b', 'c', 1, 5]

6) x.pop(1)= c x= ['b', 1, 5]

7) x.pop(0)= b x= [1, 5]

#Python 3.7.3

def example\_507():

x = ['a','b','c',1,3,5,7,[0,1,2]]

print ("1) x=",x)

del x[0]

print ("2) del x[0] x=",x)

del x[3]

print ("3) del x[3] x=",x)

y = x.pop()

print ("4) x.pop()=",y," x=",x)

print ("5) x.pop()=", x.pop()," x=",x)

print ("6) x.pop(1)=", x.pop(1)," x=",x)

print ("7) x.pop(0)=", x.pop(0)," x=",x)

example\_507()

ตัวอย่าง **1.5.8**

คำสั่ง sort() ใช้ในการเรียงข้อมูลจากน้อยไปมาก

คำสั่ง reverse() ใช้ในการสลับตำแหน่งข้อมูลจากหน้าไปหลังใหม่ ลองดูจากตัวอย่างก็เข้าใจเอง

บทนี้มีข้อสังเกตนิดหนึ่ง เมื่อกำหนดค่าตัวแปร a เป็นลิสต์ จากนั้นกำหนดให้ b=a เมื่อสั่ง a.sort() จะพบว่า b ก็จะถูก  sort ไปด้วย และที่สนุกสนานกว่านั้นคือเมื่อสั่ง b.reverse() หรือกลับตำแหน่งค่าของ b ก็จะพบว่า a ก็เปลียนแปลงไปด้วย นั่นคือ ลิสต์​ a และ b เป็นลิสต์เดียวกันที่มีข้อมูลชุดเดียวกันจริงๆ ไม่ใช่สองลิสต์ที่ค่าเท่ากัน ซึ่งเรื่องนี้จะอธิบายเพิ่มในบทถัดไป

**Example 1.5.8**

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

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

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

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

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

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

7) c= [7, 3, 2, 3, 5, 4, 2, 2, 1, 8]

8) reversed c= [8, 1, 2, 2, 4, 5, 3, 2, 3, 7]

9) revresed c= [7, 3, 2, 3, 5, 4, 2, 2, 1, 8]

#Example 1.5.8

#Python 3.7.3

def example\_508():

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

b = a

c =[7,3,2,3,5,4,2,2,1,8]

print ("1) a=",a)

print ("2) b=",b,"\n")

a.sort()

print("3) sorted a=",a)

print("4) b=",b,"\n")

b.reverse()

print("5) reversed b=",b)

print("6) a=",a,end="\n\n")

print ("7) c=",c)

c.reverse()

print ("8) reversed c=",c)

c.reverse()

print ("9) revresed c=",c)

example\_508()

*Tip*

ผู้ให้กำเนิดภาษา *Python* คือคุณ *Guido van Rossum* อ่านว่ากีโดฟานรอสสซัมนามสกุลคือ *van Rossum* หรือฟานรอสซัมไม่ใช่รอสซัมเฉยๆดังนั้นเราสามารถเรียกเขาว่าคุณกีโดหรือคุณฟานรอสซัมก็ได้แต่ไม่ควรเรียกว่าคุณรอสซัมเฉยๆหรือถ้าไปอ่านเป็นแวนรอสซัมก็จะเชยมาก

คุณกีโดเป็นโปรแกรมเมอร์ชาวดัทช์เวลาใครบอกว่าเป็นชาวดัทช์ก็เชื่อได้ว่าเขาเป็นคนประเทศเนเธอร์แลนด์หรือมีบรรพบุรุษเป็นชาวเนเธอร์แลนด์ซึ่งในกรณีนี้คือคุณกีโดก็เกิดในประเทศเนเธอร์แลนด์นี่ละสำเร็จการศึกษาระดับปริญญาโทจาก *University of Amsterdam* หรือมหาวิทยาลัยแห่งอัมสเตอร์ดัมในสาขาคณิตศาสตร์และวิทยาศาสตร์คอมพิวเตอร์ตั้งแต่ปีคศ*. 1982* หรือพ*.*ศ*. 2525* สมัยนั้นประเทศไทยก็มีสอนคอมพิวเตอร์ด้วยเหมือนกันแถมมีสร้างเองได้ด้วยทั้งคอมพิวเตอร์ทั้งซอฟต์แวร์ทั้งภาษาคอมพิวเตอร์เพราะเป็นยุครุ่งอรุณของวงการคอมพิวเตอร์พร้อมๆกันทั้งโลกยังไม่มีใครเก่งกว่าใครแต่เราสร้างคอมพิวเตอร์เองไปสักพักเราก็ตัดสินใจเลิกสร้างเองแล้วเน้นซื้อของที่คนอื่นสร้างแทน

*Tip*

คุณกีโดฟานรอสซัม *(Guido van Rossum)* ได้สร้างภาษา *Python* ขึ้นในเดือนพฤศจิกายนปีคศ*. 1989* ขณะที่กำลังหาอะไรทำเล่นๆแก้เซ็งช่วงวันหยุดคริสมาสเลยสร้างตัวแปลภาษาหรือ *interpreter* สำหรับภาษาสคริปต์ใหม่จากแนวภาษา *ABC* และเนื่องจากคุณกีโดเป็นแฟนตัวยงของคณะละครสัตว์ *Monty Python’s Flying Circus* ก็เลยตั้งชื่อภาษาใหม่นี้ว่า *Python* แบบไม่ได้กะว่างานนี้จะโด่งดังเป็นตำนานกันขนาดนี้แต่เนื่องจาก *Python* แปลตรงๆคืองูหลามดังนั้นเราจึงมักเห็นคนใช้สัญลักษณ์หรือรูปงูหลามเมื่อพูดถึงภาษา *Python*

*Tip*

ทำไมเราถึงมาเรียนภาษา *Python* กันถ้าค้นในเน็ตหรือไปถามอาจารย์แต่ละท่านก็คงได้คำตอบหลากหลายแต่คำตอบหนึ่งที่ต้องโผลออกมาแน่ๆก็คือ *“*เพราะภาษา *Pyhone* ง่าย*”* ง่ายคือเขียนง่ายอ่านง่ายดูเป็นระเบียบดีถึงแม้ว่าจะทำงานช้าๆไปบ้างหรือขาดๆเกินๆรูปแบบที่บางภาษาเค้ามองว่าดีงามไปบ้างแต่ด้วยความขาดๆเกินๆนี้เองสิ่งที่ปฏิเสธไม่ได้คือ *Python* นั้นง่ายและด้วยความง่ายของภาษา *Python* ทำให้กลายเป็นหนึ่งในภาษายอดนิยมโดยเฉพาะกับคนที่ทำเรื่องฉลาดๆอย่างเช่นการคำนวณทางคณิตศาสตร์หรือวิทยาศาสตร์รวมถึงคนทำ *AI* จะเห็นว่าคนฉลาดๆเขาก็ชอบของที่ง่ายๆเหมือนกันไม่ใช่ว่าฉลาดแล้วต้องทำอะไรที่ยากๆ

*...*ดังนั้นเมื่อรู้แล้วว่าภาษานี้ง่ายเราจึงไม่ควรมีปัญหาใดควรเรียนได้ง่ายๆสบายๆชิลๆ

*Tip*

ภาษา *Python* มีโครงสร้างหรือจะเรียกว่าเทคนิคที่ต่างกับภาษาอื่นอยู่เรื่องหนึ่งคือการใช้ *indentation* หรือการกำหนดบล็อคด้วยการใช้ย่อหน้าซึ่งต่างกับบางภาษาที่กำหนดบล็อคคำสั่งด้วยเครื่องหมาย *{ } (*ปีกกา*)* หรือบางภาษาก็มีการใช้คำสั่ง *begin end;* เป็นตัวกำหนดบล๊อค

บล๊อคที่ถูกกำหนดด้วยย่อหน้าจะถูกมองเสมือนเป็นคำสั่งลูกของคำสั่งก่อนนั้นเช่น

*def func1():*

*print(xxx)*

*print(yyy)*

*def func2():*

*print(iii)*

*print(jjj)*

*func1()*

เมื่อเรียกใช้ฟังก์ชัน *func1()* คอมพิวเตอร์จะทำคำสั่งในบล๊อคย่อหน้า

*print(xxx)* ต่อด้วย *print(yyy)* จบบล็อคที่ตรงนี้ไม่เกี่ยวกับ *def func2()* แล้ว

หรืออีกตัวอย่างหนึ่งเช่น

*x=11*

*if x==10:*

*print(xxx)*

*print(yyy)*

*print(zzz)*

โปรแกรมนี้คอมพิวเตอร์ตรวจสอบแล้วว่า *x* เป็น *11* ไม่เท่ากับ *10* ก็จะไม่ทำงานในบล๊อคย่อหน้าแต่จะข้ามลงมา *print (zzz)* เลย

แต่ถ้าเปลี่ยนตัวอย่างเป็น

*x=10*

*if x==10:*

*print(xxx)*

*print(yyy)*

*print(zzz)*

โปรแกรมนี้ *x=10* ดังนั้นโปรแกรมจะทำงานในบล๊อคย่อหน้าคือ *print(xxx)* ต่อด้วย *print(yyy)* จบบล๊อคจากนั้นก็มาทำงานคำสั่งนอกบล๊อคต่อคือ *print(zzz)*

เรื่อง *indentation* หรือย่อหน้านี้จริงๆแทบไม่ต้องอธิบายมากดูจากตัวอย่างก็เข้าใจไปเองเขียนเผื่อไว้กันงงเฉยๆ