Python 101

Contents

[Introduction 3](#_Toc495009842)

[First Program ‘Hello World’ 3](#_Toc495009843)

[Comments 3](#_Toc495009844)

[Variables 5](#_Toc495009845)

[Strings 5](#_Toc495009846)

[Data structures 5](#_Toc495009847)

[Condition expressions 5](#_Toc495009848)

[Loops 6](#_Toc495009849)

[Function 6](#_Toc495009850)

# Introduction

## First Program ‘Hello World’

## Comments

การแทรกคำอธิบาย(Comment)

ในกระบวนการพัฒนาโปรแกรมเมื่อมีการเขียนโค้ดที่ค่อนข้างยาว การพัฒนาโปรแกรมขนาดใหญ่ ที่มี

การทำงานร่วมกันหลายคน หรือการพัฒนาโปรแกรมที่ซับซ้อนนั้น การเขียนคำอธิบายไว้ระหว่างบรรทัดของ

การเขียนโปรแกรม เพื่อช่วยบอกว่าโค้ดบรรทัดนี้นั้น มีไว้ทำอะไร ผลที่ได้จากบรรทัดนี้คืออะไร ซึ่งจะทำให้เมื่อ

มีการกลับมาดูอีกครั้ง หรือมีการวิเคราะห์โปรแกรมอีกครั้งหนึ่งทำได้ง่าย และมีความถูกต้องขึ้นมาก

ในภาษา Python นั้น การแทรกคำอธิบายนั้นสามารถทำได้โดยใช้คำสั่ง ‘#’ ดังตัวอย่าง

print “Hello World1”

#print “Hello World2”

 Hello World1

print “Hello World1” #print “Hello World2”

 Hello World1

ตัวแปรประเภทข้อความ (string)

ใช้ (‘ ‘) (“ “) และ (“”” “””) เป็นการประกาศตัวแปร เพื่อใช้แทนข้อความที่เรากำหนดไว้ เช่น ใช้

แทนประโยค Hello World

x = “Hello World”

print x

 Hello World

x = ‘Hello World’

print x

 Hello World

x = “””Hello World”””

print x

 Hello World

ข้อแตกต่างระหว่าง (‘ ‘) (“ “) กับ (“”” “””) นั้นคือ การแสดงข้อความหลายบรรทัด ดังเช่นตัวอย่าง

x = “””H

e

l

l

o”””

print x

 H

e

l

l

o

แต่ หากใช้ (“ “) หรือ (‘ ‘)

x = ‘H

e

l

l

o’

หรือ

x = “H

e

l

l

o”

print x

 ระบบจะแจ้งว่า Error : EOL while scanning string literal หมายถึงว่า เราไม่ได้ปิดประโยคของ

ข้อความอย่างถูกต้อง เพราะ (‘ ‘) และ (“ “) ไม่สามารถรองรับการแสดงผลหลายบรรทัดจากคำสั่ง

print เพียงครั้งเดียว

# Variables

# Strings

# Data structures

# Condition expressions

Boolean operators

Boolean Operators order

If statement

Else ,elif part in if statement

# Loops

For Loop

For loop using string

While loop

Break keyword

Continue keyword

# Function

สรุปตัวอย่างคำสั่งต่างๆในภาษา Python

การประกาศตัวแปร ในรูปแบบต่างๆ

x = “””H

e

l

l

o”””

print x

 H

e

l

l

o

แต่ หากใช้ (“ “) หรือ (‘ ‘)

x = ‘H

e

l

l

o’

หรือ

x = “H

e

l

l

o”

print x

 ระบบจะแจ้งว่า Error : EOL while scanning string literal หมายถึงว่า เราไม่ได้ปิดประโยคของ

ข้อความอย่างถูกต้อง เพราะ (‘ ‘) และ (“ “) ไม่สามารถรองรับการแสดงผลหลายบรรทัดจากคำสั่ง

print เพียงครั้งเดียว

x = 'Hello World!'

print x # แสดงอักษรทั􀃊งหมดของ x

print x[0] # แสดงอักษรของ x ตั􀃊งแต่อักษรตัวแรก

print x[2:5] #แสดงอักษรของ x ตั􀃊งแต่ตัวอักษรที􀃉 3 ไปจนถึงตัวอักษรที􀃉 6

print x[2:] #แสดงตัวอักษรของ x ตั􀃊งแต่ลำดับ 3

print x \* 2 #แสดงตัวอักษรทุกตัวของ x 2ครั􀃊ง

print x + "TEST" #แสดงอักษรทั􀃊งหมดของ x และต่อด้วย “TEST”

print x[5],x[3],x[1]+"-"+x[6],x[4],x[2],x[0] #แสดงอักษรของ x เรียงลำดับ 5 – 3 – 1 – 6 – 4 – 2 – 0

 Hello World!

 H

 Llo

 llo World!

 Hello World!Hello World!

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 26 | 65

 Hello World!TEST

 l e-W o l H

ตัวแปรประเภทจำนวนเต็มและจำนวนเต็มขนาดยาว (Integer , Long)

เป็นการประกาศตัวแปรที่ใช้สำหรับเก็บข้อมูลตัวเลขที่เป็นจำนวนเต็ม เช่น

x = 5 #กำหนดให้ x มีค่าเท่ากับจำนวนเต็ม 5

y = 4L #กำหนดให้ y มีค่าเท่ากับจำนวนเต็มขนาดยาว 4

print x+y

print x-y

print y-x

print x\*y

print x/y

print y/x

print x%y

print y%x

 9

 1

 -1

 20

 1

 0

 1

 4

ข้อแตกต่างระหว่างตัวแปรแบบจำนวนเต็ม กับตัวแปรจำนวนเต็มขนาดยาวคือ เลขจำนวนสูงสุดที่มันรองรับได้

โดยเราสามารถดูจำนวนที่ตัวแปรเลขจำนวนเต็มรองรับได้สูงสุดจากคำสั่ง sys.maxint ดังตัวอย่าง

sys.maxint

2147483647

และถ้าหากจำนวนที่เราใช้มีขนาดใหญ่กว่านี้ เช่น sys.maxint +1 ระบบก็จะเปลี่ยนชนิดของตัวแปรไปเป็น

จำนวนเต็มขนาดยาวโดยอัตโนมัติ เช่น

a = sys.maxint

print a , type(a)

a = a + 1

print a , type(a)

2147483647 <type 'int'>

2147483648 <type 'long'>

ดังนั้นแล้วเราจึงอาจจะไม่ต้องกังวลเรื่องขนาดของตัวแปรเลย เนื่องจากระบบจะจัดการให้เราทุกอย่าง

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 27 | 65

ตัวแปรประเภทจำนวนทศนิยม (Float)

การประกาศตัวแปรที่ใช้สำหรับเก็บข้อมูลตัวเลขที่เป็นจำนวนทศนิยม เช่น

x = 5.0 # x เป็นตัวแปรประเภททศนิยม

y = 4 # y เป็นตัวแปรจำนวนเต็ม

print x+y

print x-y

print y-x

print x\*y

print x/y

print y/x

print x%y

print y%x

print y , type(y)

y = x/y

print y , type(y)

 9.0

 1.0

 -1.0

 20.0

 1.25

 0.8

 1.0

 4.0

 4 <type 'int'>

 1.25 <type 'float'>

ตัวแปรเลขจำนวนเชิงซ้อน (Complex)

การประกาศตัวแปรที่เป็นจำนวนเชิงซ้อน ในรูปแบบ a+bj (ซึ่ง j นั้นแทนจำนวนของ √−1 นั่นคือ

จำนวนที่เราเรียกว่า จำนวนจินตภาพ) a คือส่วนหนึ่งของจำนวนจริง และ b คือส่วนหนึ่งของจำนวนจินตภาพ

x = (5+1.25j) # x เป็นตัวแปรประเภทจำนวนเชิงซ้อน

y = 4 # y เป็นตัวแปรจำนวนเต็ม

print x , type(x)

print x.real,type(x.real)

print x.imag,type(x.imag)

print "x+y = ", x+y

print "x-y = ", x-y

print "y-x = ", y-x

print "x\*y = ", x\*y

print "x/y = ", x/y

print "y/x = ", y/x

print "x%y = ", x%y

print "y%x = ", y%x

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 28 | 65

 (5+1.25j) <type 'complex'>

 5.0 <type 'float'>

 1.25 <type 'float'>

 x+y = (9+1.25j)

 x-y = (1+1.25j)

 y-x = (-1-1.25j)

 x\*y = (20+5j)

 x/y = (1.25+0.3125j)

 y/x = (0.752941176471-0.188235294118j)

 x%y = (1+1.25j)

 y%x = (4+0j)

ผลลัทพ์ที่ได้จากการหารเอาเศษ และการหารเอาจำนวนเต็ม อาจจะให้ผลที่ผิดพลาดได้ จำเป็นต้องระวังในการ

ใช้คำสั่ง ‘%’ และ ‘//’

ตัวแปรประเภทชุดลำดับ(Lists)

ตัวแปรแบบชุดลำดับคือ ตัวแปรที่สามารถเก็บข้อมูล หลายๆรูปแบบ และหลายๆข้อมูล เรียงกันเป็น

ลำดับ และมีกระบวนการที่สามารถทำให้เราเข้าถึงข้อมูลชุดนั้นๆได้ การประกาศตัวแปรแบบชุดลำดับสามารถ

ทำได ้ โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และเรากำหนดให้เท่ากับกลุ่มของค่าต่างๆ จำนวนหนึ่ง เช่น

'abcd' 786 2.23 'john' 70.2 ในที่นี้มีทั้งหมด 5 จำนวน เราจะสามารถกำหนดได้ดังนี้ x = [ 'abcd', 786 ,

2.23, 'john', 70.2 ]

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “[ ]” และแต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์

“,” เช่น

x = [ 'abcd', 786 , 2.23, 'john', 70.2 ]

y = [123, 'john']

x จะมีค่าตามตารางข้างล่างนี้

X[0] X[1] X[2] X[3] X[4]

abcd 786 2.23 john 70.2

Y จะมีค่าตามตารางข้างล่างนี้

y[0] y[1]

123 john

print x , y # แสดงข้อมูลทั􀃊งหมดของ x และ y

print x[0] # แสดงเฉพาะข้อมูลชุดแรกของ x

print x[1:3] # แสดงข้อมูลของ x ตั􀃊งแต่ชุดที􀃉 2 ไปจนถึงชุดที􀃉 3

print x[2:] # แสดงข้อมูลของ x ตั􀃊งแต่ชุดที􀃉 3

print y \* 2 # แสดงข้อมูลทั􀃊งหมดของ y 2 ครั􀃊ง

print x + y # แสดงชุดข้อมูลทั􀃊งหมดของ x รวมกับ y

x = x + [‘1000’] # เพิ􀃉มข้อมูล ‘1000’ เข้าไปในชุดข้อมูล x

print x # แสดงข้อมูลทั􀃊งหมดของ x ซึ􀃉งจะมีข้อมูลใหม่ที􀃉เพิม􀃉 เข้ามาอยู่ด้วย

 ['abcd', 786, 2.23, 'john', 70.2] [123, 'john']

 abcd

 [786, 2.23]

 [2.23, 'john', 70.2]

 [123, 'john', 123, 'john']

 ['abcd', 786, 2.23, 'john', 70.2, 123, 'john']

 ['abcd', 786, 2.23, 'john', 70.2, ‘1000’]

ประเภทตัวแปรประเภท Tuples

Tuples เป็นตัวแปรลำดับขั้นอีกแบบหนึ่ง ซึ่งแตกต่างจากตัวแปรลำดับขั้นแบบ Lists ที่ได้กล่าวถึงใน

หัวข้อที่แล้ว ในส่วนที่ Tuples จะไม่อนุยาติให้มีการเขียนข้อมูลทับข้อมูลเดิม หรือเปลี่ยนแปลงข้อมูลจาก

ภายนอกได้ นี่คือความแตกต่างที่สำคัญของตัวแปรประเภท Tuples

การประกาศตัวแปร Tuples สามารถทำได้ โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และเรากำหนดให้เท่ากับ

กลุ่มของค่าต่างๆ จำนวนหนึ่ง เช่น 'abcd' 786 2.23 'john' 70.2 ในที่นี้มีทั้งหมด 5 จำนวน เราจะสามารถ

กำหนดได้ดังนี้ x = ( 'abcd', 786 , 2.23, 'john', 70.2 )

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “()” และแต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์ “,”

เช่น

x = ( 'abcd', 786 , 2.23, 'john', 70.2 )

y = (123, 'john')

print x,y # แสดงข้อมูลทั􀃊งหมดของ x และ y

print x[0] # แสดงข้อมูลชุดแรกของ x

print x[1:3] # แสดงข้อมูลของ x ตั􀃊งแต่ชุดที􀃉 2 ไปจนถึงชุดที􀃉 3

print x[2:] # แสดงข้อมูลของ x ตั􀃊งแต่ชุดที􀃉 3 ไปจนจบ

print y \* 2 # แสดงข้อมูลของ y 2 ครั􀃊ง

print x + y #แสดงข้อมูล x รวมกับ y

 ('abcd', 786, 2.23, 'john', 70.2) (123, 'john')

 abcd

 (786, 2.23)

 (2.23, 'john', 70.2)

 (123, 'john', 123, 'john')

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 30 | 65

 ('abcd', 786, 2.23, 'john', 70.2, 123, 'john')

แต่หากเราจะเปลี่ยนค่าใน ตัวแปร x เราจะพบว่าระบบจะแจ้ง Error ให้เราทราบ โดยบอกว่า TypeError:

'tuple' object does not support item assignment ซึ่งหมายถึง Tuples นั้นไม่รองรับการกำหนดค่าซ้ำ

หรือแก้ไขข้อมูลใหม่ เช่น

x = ( 'abcd', 786, 2.23, 'john', 70.2) #กำหนดค่า x

y = [ 'abcd', 786, 2.23, 'john', 70.2] #กำหนดค่า y

x[1] = 1000 #เปลี􀃉ยนค่า x ชุดที􀃉 2 และคำสัง􀃉 ในบรรทัดนี􀃊 จะเกิดปัญหา TypeError: 'tuple' object does

not support item assignment

y[1] = 1000 #คำสัง􀃉 นี􀃊จะเปลี􀃉ยนค่าของ y ชุดที􀃉 2 ให้เท่ากับ 1000 โดยไม่มีการแจ้งข้อผิดพลาด เนื􀃉องจาก y เป็นตัวแปรประเภท

Lists ซึ􀃉งยินยอมให้มีการเขียนข้อมูลทับข้อมูลเดิมได้

ประเภทตัวแปรประเภท Dictionary

ตัวแปรประเภทนี้ ชุดข้อมูลจะถูกเก็บในลักษณ์คู่อันดับแบบ คีย์ กับข้อมูล ที่ไม่เรียงลำดับ ซึ่งต่างจาก

ตัวแปรชุดแบบ Lists และ Tuples ที่มีลำดับข้อมูลเรียงกัน ดังนั้นแล้วเราจะไม่สามารถเข้าถึงข้อมูลด้วยวิธี

เดียวกันกับที่ใช้ใน Tuples หรือ Lists ได้ เราจะต้องเข้าถึงข้อมูลโดยผ่าน คีย์ เท่านั้น

การประกาศตัวแปรประเภทดิกชันนารี่นี้ เราสามารถทำได้โดย โดยการตั้งชื่อตัวแปร ในตัวอย่างนี้คือ x และ

เรากำหนดให้เท่ากับกลุ่มของค่าต่างๆ ที่ประกอบไปด้วยคีย์ และข้อมูล จำนวนหนึ่ง ในรูปแบบ คีย์ : ข้อมูล

เช่น 'key': 'data' , 'name' : 'abcd', 1 : 786 , 1.5 : 2.23 , 2 : 'john','test': 70.2 ในที่นี้มีทั้งหมด 6 จำนวน

เราจะสามารถกำหนดได้ดังนี้ x = {'key': 'data' , 'name' : 'abcd' , 1 : 786 , 1.5 : 2.23 , 2 : 'john','test':

70.2}

โดยทุกๆจำนวน จะต้องอยู่เรียงกันในกรอบสัญลักษณ์ “{ }” มีสัญลักษณ์ : คั่นแยกระหว่าง คีย์ กับข้อมูล และ

แต่ละค่านั้นจะแบ่งแยกกันด้วยสัญลักษณ์ “,” เช่น

a = {}

x = {'key': 'data' , 'name' : 'abcd' , 1 : 786 , 1.5 : 2.23 , 2 :

'john','test': 70.2}

y = ('y1','y2',1,2,3)

print x # Prints complete dictionary

print x.keys() # Prints all the keys

print x.values() # Prints all the values

 {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 'key': 'data', 'test': 70.2}

 [1.5, 1, 2, 'name', 'key', 'test']

 [2.23, 786, 'john', 'abcd', 'data', 70.2]

x['t'] = 0.5 #เพิ􀃉ม ‘t’ เป็นคีย์ใหม่ โดยให้มีข้อมูลเป็น 5

print x # Prints complete dictionary

print x.keys() # Prints all the keys

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 31 | 65

print x.values() # Prints all the values

 {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 't': 0.5, 'key': 'data', 'test': 70.2}

 [1.5, 1, 2, 'name', 't', 'key', 'test']

 [2.23, 786, 'john', 'abcd', 0.5, 'data', 70.2]

x['test'] = 25 #เปลี􀃉ยนข้อมูลของคีย์ ‘test’ ให้เท่ากับ 25

print x # Prints complete dictionary

print x.keys() # Prints all the keys

print x.values() # Prints all the values

 {1.5: 2.23, 1: 786, 2: 'john', 'name': 'abcd', 't': 0.5, 'key': 'data', 'test': 25}

 [1.5, 1, 2, 'name', 't', 'key', 'test']

 [2.23, 786, 'john', 'abcd', 0.5, 'data', 25]

a[y] = "Hello" #เพิ􀃉มคีย์ ชื􀃉อ y แบบ Tuples โดยให้มีค่าเท่ากับ “Hello”

print a # Prints complete dictionary

print a.keys() # Prints all the keys

print a.values() # Prints all the values

 {('y1', 'y2', 1, 2, 3): 'Hello'}

 [('y1', 'y2', 1, 2, 3)]

 ['Hello']

a["H"] = y #เพิ􀃉มคีย์ “H” ให้มีค่าเท่ากับ Tuples ของ y

print a # Prints complete dictionary

print a.keys() # Prints all the keys

print a.values() # Prints all the values

 {'H': ('y1', 'y2', 1, 2, 3), ('y1', 'y2', 1, 2, 3): 'Hello'}

 ['H', ('y1', 'y2', 1, 2, 3)]

 [('y1', 'y2', 1, 2, 3), 'Hello']

ตัวกระทำทางการคำนวณ

เป็นเครื่องหมายสำหรับใช้ทำงานทางการคำนวณ หรือเปรียบเทียบระหว่างข้อมูล เพื่อใช้ในการ

ตัดสินใจ ในกระบวนการทำงาน

เครื่องหมาย ความหมาย ตัวอย่าง

+ การบวก – นำสองจำนวนมาบวกกัน a = 10

b = 15

c = a + b

c = 25

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 32 | 65

- การลบ – นำจำนวนแรกลบด้วยจำนวนที่สอง a = 10

b = 15

c = a - b

c = -5

\* การคูณ – นำสองจำนวนคูณกัน a = 10

b = 15

c = a \* b

c = 150

/ การหาร – จำนวนแรกหารด้วยจำนวนที่สอง a = 20

b = 15

c = a / b

c = 1

a = 20.0

b = 15

c = a / b

c = 1.33..

% การหารเอาเศษ – คำตอบที่ได้จากการหาร คือเศษเหลือจากการหารนั้น a = 20

b = 15

c = a % b

c = 5

\*\* ยกกำลัง – เลขจำนวนแรก ยกกำลังด้วยเลขจำนวนที่สอง a = 20

b = 15

c = a \*\* b

c = 3.2768e+19

หรือเท่ากับ 3.2768x1019

// การหารเอาเลขจำนวนเต็ม – เมื่อให้การหารแบบนี้ คำตอบที่ไดจ้ ะเป็นเลขจำนวนเต็มเทา่ นั้น a = 20

b = 15

c = a // b

c = 1

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 33 | 65

a = 20.0

b = 15

c = a // b

c = 1.0

== เปรียบเทียบค่าทั้งสองจำนวนว่าเท่ากันหรือไม่ –

คำตอบที่ได้มีเพียง True หากเท่ากัน และ False หากไม่เท่า

a = 20.0

b = 15

(a == b) is not true.

!= เปรียบเทียบค่าทั้งสองจำนวนว่าเท่ากันหรือไม่ –

คำตอบที่ได้มีเพียง True หากไม่เท่ากัน และ False หากเท่ากัน

a = 20.0

b = 15

(a != b) is true.

<> เปรียบเทียบค่าทั้งสองจำนวนว่ามากกว่า หรือน้อยกว่า หรือไม่ –

คำตอบที่ได้มีเพียง True หากมากกว่า หรือน้อยกว่ากัน และ False หากเท่ากัน

จะเหมือน !=

a = 20.0

b = 15

(a <> b) is true.

> เครื่องหมายมากกว่า – ตรวจสอบจำนวนทางซ้ายว่ามากกว่าทางขวาหรือไม่ จะให้คำตอบเป็น

True เมื่อทางซ้ายมากกว่าทางขวา

a = 20.0

b = 15

(a > b) is true.

a = 20.0

b = 15

(b > a ) is false.

< เครื่องหมายน้อยกว่า – ตรวจสอบจำนวนทางซ้ายว่าน้อยกว่าทางขวาหรือไม่ จะให้คำตอบเป็น

True เมื่อทางซ้ายน้อยกว่าทางขวา

a = 20.0

b = 15

(a < b) is false.

a = 20.0

b = 15

(b < a ) is true.

>= เครื่องหมายเท่ากับ หรือมากกว่า – ตรวจสอบจำนวนทางซ้ายว่าเท่ากับ หรือมากกว่าทางขวา

หรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายเท่ากับ หรือมากกว่าทางขวา

a = 20.0

b = 15

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 34 | 65

(a >= b) is true.

a = 15.0

b = 15

(b >= a ) is true.

<= เครื่องหมายเท่ากับ หรือน้อยกว่า – ตรวจสอบจำนวนทางซ้ายว่าเท่ากับ หรือน้อยกว่าทางขวา

หรือไม่ จะให้คำตอบเป็น True เมื่อทางซ้ายเท่ากับ หรือนอ้ ยกว่าทางขวา

a = 20.0

b = 15

(a <= b) is false.

a = 15.0

b = 15

(b <= a ) is true.

= เครื่องหมายกำหนดค่า – ใช้สำหรับ กำหนดค่าทางซ้าย ให้เท่ากับค่าทางขวา c = a + b

กำหนดให้ c เท่ากับ a + b

+= เครื่องหมายเพิ่มค่า – ใช้สำหรับ เพิ่มค่าทางซ้าย ด้วยค่าทางขวา c += a

จะหมายถึง c = c + a

-= เครื่องหมายลดค่า – ใช้สำหรับ ลดค่าทางซ้าย ด้วยค่าทางขวา c -= a

จะหมายถึง c = c - a

\*= เครื่องหมายคูณค่า – ใช้สำหรับ คูณค่าทางซ้าย ด้วยค่าทางขวา c \*= a

จะหมายถึง c = c \* a

/= เครื่องหมายหารค่า – ใช้สำหรับ หารค่าทางซ้าย ด้วยค่าทางขวา c /= a

จะหมายถึง c = c / a

%= เครื่องหมายหารค่าแบบเอาเศษ – ใช้สำหรับ หารค่าแบบเอาเศษค่าทางซ้าย ด้วยค่าทางขวา c %= a

จะหมายถึง c = c % a

\*\*= เครื่องหมายยกกำลัง – ใช้สำหรับ ยกกำลังค่าทางซ้าย ด้วยค่าทางขวา c \*\*= a

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 35 | 65

จะหมายถึง c = c \*\* a

//= เครื่องหมายหารเอาจำนวนเต็ม– ใช้สำหรับ หารเอาจำนวนเต็มค่าทางซ้าย ด้วยค่าทางขวา c //= a

จะหมายถึง c = c // a

& เครื่องหมาย “และ” หมายถึงเอาค่าในแบบเลขฐานสองของทั้งสองฝั่งมา

“และ” กัน

a = 20

b = 15

c = a & b

a = 20 0b10100

b = 15 0b1111

c = 4 0b100

| เครื่องหมาย “หรือ” หมายถึงเอาค่าในแบบเลขฐานสองของทั้งสองฝั่งมา

“หรือ” กัน

a = 20

b = 15

c = a | b

a = 20 0b10100

b = 15 0b1111

c = 31 0b11111

^ Binary XOR Operator copies the bit if it is set in one operand but not both. a = 20

b = 15

c = a ^ b

a = 20 0b10100

b = 15 0b1111

c = 27 0b11011

~ Binary Ones Complement Operator is unary and has the efect of 'flipping' bits. a = 20

c = ~a

a = 20 0b10100

c = -21 -0b10101

<< เครื่องหมายเลื่อนบิทไปทางซ้าย – โดยจะเลื่อนบิทข้อมูลไปทางซ้ายตามจำนวนที่กำหนดทางขวา a = 20

c = a <<2

a = 20 0b10100

c = 80 0b1010000

>> เครื่องหมายเลื่อนบิทไปทางขวา – โดยจะเลื่อนบิทข้อมูลไปทางขวาตามจำนวนที่กำหนดทางขวา a = 20

c = a >>2

a = 20 0b10100

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 36 | 65

c = 5 0b101

and เปรียบเทียบ “และ” ทางตรรกศาสตร์

คำตอบที่ได้จะมีเพียง True หากทั้งสองข้างมีค่าเป็น True

และ False หากมีมากกว่า 1 ค่าที่เป็น False

a = 0

b = 1

(a and b) is false.

a = 1

b = 1

(a and b) is true.

or เปรียบเทียบ “หรือ” ทางตรรกศาสตร์

คำตอบที่ได้จะมีเพียง False หากทั้งสองข้างมีค่าเป็น False

และ True หากมีมากกว่า 1 ค่าที่เป็น True

a = 0

b = 1

(a or b) is true.

a = 0

b = 0

(a or b) is false.

not เปรียบเทียบ “ไม่” ทางตรรกศาสตร์

คำตอบที่ได้จะมีเพียง False หากค่านั้นเป็น True

และ True หากค่านั้นเป็น False

a = 0

b = 1

not(a and b) is true.

a = 1

b = 1

not(a and b) is false.

in จะให้คำตอบเป็น true หากค่าทางขวานั้นเป็นสมาชิกของชุดข้อมูลทางซ้าย a = 20.0

b = [0,2,20,3]

c = a in b

C = true

a = 20.0

b = [0,2,30,3]

c = a in b

c = false

not in จะให้คำตอบเป็น false หากค่าทางขวานั้นเป็นสมาชิกของชุดข้อมูลทางซ้าย a = 20.0

b = [0,2,20,3]

c = a not in b

C = false

a = 20.0

b = [0,2,30,3]

c = a not in b

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 37 | 65

c = true

is จะให้คำตอบเป็น true หากค่าทางขวานั้นเป็นค่าเดียวกันทางซ้าย a = 20.0

b = [15 , 20.0]

c = a is b[1]

C = true

a = 20.0

b = [15 , 20]

c = a is b[1]

c = false

is not จะให้คำตอบเป็น true หากค่าทางขวานั้นไม่เป็นค่าเดียวกันทางซ้าย a = 20.0

b = [15 , 20.0]

c = a is not b[1]

C = false

a = 20.0

b = [15 , 20]

c = a is not b[1]

c = true

ลำดับความสำคัญของตัวกระทำทางการคำนวณ

เครื่องหมาย ความหมาย ตัวอย่าง

‘\*\*’ การยกกำลัง a = 3

b = 5

c = b+a\*\*2

c = 14

‘~’ , ‘+’ , ‘-‘ ‘~’ – การกลับบิทข้อมูล

‘+’ – เครื่องหมายของจำนวนบวก

‘-‘ – เครื่องหมายของจำนวนลบ

a = 3

b = ~a

c = -a

d = +a

e = a + ~a

f = a + -a

g = a + +a

a = 3 , 0b11

b = -4 , -0b100

c = -3 , -0b11

d = 3 , 0b11

e = -1 , -0b1

f = 0 , 0b0

g = 6 , 0b110

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 38 | 65

‘\*’ , ‘/’ , ‘%’ , ‘//’ การคูณ และหาร ในแบบต่างๆ a = 3

b = 5

c = b+a\*2

c = 11

a = 3

b = 5

c = b+a/2

c = 6

‘+’ , ‘-‘ การบวกและลบ a = 3

b = 5

c = b<<4+a

c = 640 0b1010000000

a = 3

b = 5

c = b>>4-a

c = 2 0b10

‘>>’ , ‘<<’ การเลื่อนบิทข้อมูล a = 2

b = 14

c = b&13>>a

c = 2 0b10

a = 2

b = 14

c = b&13<<a

c = 4 0b100

‘&’ การ “และ” บิท ข้อมูล a = 2

b = 14

c = 13

d = 18

e = a ^ b | c & d

f = a | b ^ c & d

g = a | b & c ^ d

e = 12 0b1100

f = 14 0b1110

g = 30 0b11110

‘|’ , ‘^’ การ “หรือ” และการ “หรือ แบบ

พิเศษ”

a = 2

b = 14

c = 13

d = 18

e = a ^ b | c & d > a | b ^ c & d

f = a ^ b | c & d < a | b ^ c & d

e = False

f = True

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 39 | 65

‘<>’ , ‘==’ , ‘!=’ การเปรียบเทียบที่เท่ากัน หรือไม่

เท่ากัน

a = 2

b = 14

c = 13

d = 18

e = b > c == c

f = b < c == c

g = b <= b == b

e = True

f = False

g = True

‘<=’ , ‘<’ , ‘>’ , ‘>=’ การเปรียบเทียบที่มากกว่า และ น้อย

กว่า

a = 2

b = 14

c = 13

d = 18

e = 14 > a | b ^ c & d

f = 14 < a | b ^ c & d

g = 14 >= a | b ^ c & d

e = False

f = False

g = True

‘=’ , ‘%=’ , ‘/=’ , ‘//=’ ,

‘-=’ , ‘+=’ , ‘|=’ , ‘&=’ ,

‘>>=’ , ‘<<=’ , ‘\*=’ , ‘\*\*=’

การกำหนดค่า a = 2

b = 14

c = 10

c += b&13>>a

c = 12 0b1100

a = 2

b = 14

c = 10

c >>= b&13>>a

c = 2 0b10

‘is’ , ‘is not’ การตรวจสอบตัวแปร a = 2

b = 14

c = a is b&13>>a

d = b is b&13>>a

e = a is not b&13>>a

f = b is not b&13>>a

c = True

d = False

e = False

g = True

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 40 | 65

‘in’ , ‘not in’ การตรวจสอบสมาชิกข้อมูล a = 2

b = 14

c = ('123','abc',456,a)

d = ['456','def',123]

e = a in c

f = b in d + [b]

g = b not in d + [b]

e = True

f = True

g = False

‘not’ , ‘or’ , ‘and’ การตรวจสอบเงื่อนไข a = 2

b = 14

c = ('123','abc',456,a)

d = ['456','def',123]

e = not a in c

f = b in d + [b] and a in c

g = not b in d + [b] and a in c

h = b not in d + [b] or a in c

i = not b not in d + [b] or a in c

j = b not in d + [b] or not a in c

e = False

f = True

g = False

h = True

i = True

j = False

คำสั่งสำคัญในการทำงาน

คำสั่งการวนซ้ำ, วนลูป (Loop)

เป็นคำสั่งที่ใช้เพื่อให้โปรแกรมมีการทำงานซ้ำในช่วงคำสั่งการทำงานเดิมตามเงื่อนไขที่เราได้กำหนดไว้

ที่เรียกว่าการทำงานแบบวนลูป เช่น การสั่งให้โปรแกรมบวกเลขไปเรื่อยๆ ตราบใดที่ค่านั้นยังน้อยกว่า 100

เป็นต้น โดยเงื่อนไขในการวนลูปนั้น เงื่อนไขนั้นจะต้องเป็น จริง – ถูกตอ้ ง – ใช่ เสมอ และการวนลูปนั้นจะ

สิ้นสุดลงเมื่อ เงื่อนไขนั้นมีผลเป็นเท็จ – ผิด – ไม่ใช่ ซึ่งคำสั่งนั้น มี 2 คำสั่งคือ while และ for

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 41 | 65

คำสัง􀃉 /งาน

ตรวจสอบเงื􀃉อนไข

ใช่

ไม่ใช่

While – while <เงื่อนไข> :

คำสั่ง 1

คำสั่ง 2

คำสั่ง 3 ....

คำสั่ง 4

คำสั่ง while จะทำให้โปรแกรมทำงานวนลูปอยู่ในช่วงคำสั่งที่ 1 – คำสั่งที่ 3 ตราบใดที่ เงื่อนไขของ

คำสั่ง while ยังคงเป็นจริง และหากเงื่อนไของคำสั่ง while เป็นเท็จ โปรแกรมจึงจะมาทำคำสั่งที่ 4 ต่อไป

ตัวอย่างการใช้คำสั่ง while

count = 0

while (count < 9):

print 'The count is:', count

count = count + 1

print "Good bye!"

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 42 | 65

ผลการทำงาน

The count is: 0

The count is: 1

The count is: 2

The count is: 3

The count is: 4

The count is: 5

The count is: 6

The count is: 7

The count is: 8

Good bye!

ตัวอย่างนี้ จะเห็นว่าเงื่อนไขของการวนลูปคือ count จะต้อง “น้อยกว่า” 9 หาก count น้อยกว่า 9 แล้ว

โปรแกรมจะแสดงข้อความว่า “The count is : ” ตามด้วยค่าของ count และมีการเพิ่มค่าของ count ทีละ

1 ในทุกๆรอบที่มีการวนมาทำงาน

เมื่อ count นั้นมีการเพิ่มค่าจนเท่ากับ 9 แล้ว จะทำให้เงื่อนไขของลูป while เป็นเท็จ ที่ระบุวา่ count ต้อง

น้อยกว่า 9 จึงจะมีการวนลูปในช่วงของคำสั่งชุดนี้ เมื่อการวนลูปสิ้นสุดลง โปรแกรมก็จะออกจากลูปนั้น และ

มาทำงานในคำสั่ง print "Good bye!" ที่อยู่นอกวงลูปต่อไป

ลูปที่ไม่มีที่สิ้นสุด – Infinity Loop คือลูปที่เงื่อนไขนั้น ไม่มีทางเป็นเท็จได้ ดังนั้นโปรแกรมจะทำงานวนในลูป

นั้นอยู่ตลอด โดยไม่มีการออกมาทำงานนอกลูป ซึ่งลูปแบบนี้บ่อยครั้งที่มักถูกออกแบบเพื่อให้โปรแกรมทำงาน

อยู่ตลอดเวลาที่มีการทำงาน แต่บ่อยครั้งที่การเกิดลูปแบบไม่มีที่สิ้นสุดนี้ เกิดจากความผิดพลาดในการ

ออกแบบเงื่อนไขในการทำงาน ทำให้โปรแกรมค้างอยู่ในลูปนั้น

ตัวอย่าง ลูปที่ไม่มีที่สิ้นสุด – Infinity Loop

var = 1

while var == 1 : # This constructs an infinite loop

num = input("Enter a number :")

print "You entered: ", num

print "Good bye!"

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 43 | 65

ผลการทำงาน

Enter a number :1

You entered: 1

Enter a number :2

You entered: 2

Enter a number :5

You entered: 5

Enter a number :8

You entered: 8

Enter a number :4

You entered: 4

Enter a number :

จะเห็นว่าโปรแกรมนี้ วนรับค่าที่เราป้อนเข้าไปอยู่ตลอดเวลา เนื่องจากว่าเงื่อนไข var ==1 นั้น เป็นจริงเสมอ

เพราะคำสั่งในลุปนั้น ไม่ได้เข้าไปเปลี่ยนแปลงค่าใดๆของ var ทำให้เงื่อนไข var == 1 ไม่มีการเปลี่ยนแปลงไป

ด้วยเช่นกัน ดังนั้นมันจึงทำงานวนอยู่แบบนี้ตลอดไป

คำสั่งการวนลูปแบบ while มีการรองรับคำสั่ง else เพื่อให้ทำงานบนเงื่อนไขที่เป็นเท็จ โดยเฉพาะ เช่น

count = 0

while count < 5:

print count, " is less than 5"

count = count + 1

else :

count = count - 1

print count, " is not less than 5"

print "end for ",count

หาก count เริ่มต้นที่ 0 ผลที่ได้ก็จะเป็น

0 is less than 5

1 is less than 5

2 is less than 5

3 is less than 5

4 is less than 5

4 is not less than 5

end for 0

หาก count เริ่มต้นที่ 6 ผลที่ได้ก็จะเป็น

count = 6

while count < 5:

print count, " is less than 5"

count = count + 1

else :

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 44 | 65

count = count - 1

print count, " is not less than 5"

print "end for ",count

ผลที่ได้

5 is not less than 5

end for 5

For – for <คำชี้แจง> :

คำสั่งที่ 1

คำสั่งที่ 2

คำสั่งที่ 3....

คำสั่งที่ 4

คำสั่งวนลูปแบบ for นี้จะแตกต่างจากคำสั่ง while ตรงที่ คำสั่ง for จะใช้ “คำชี้แจง” ในการกำหนด

กรอบการวนลูป ดังนั้นเราจะไม่สามารถใช้ เงื่อนไขแบบ วนตราบใดที่ค่านั้นยังน้อยกว่า 9 ได้ แต่การวนลูปของ

การใช้ for นั้น จำนวนรอบของการวนจะถูกกำหนดอยู่ใน “คำชี้แจง” นั้นเลย เช่น วนนับตั้งแต่ 0 ไปจนถึง 9

ซึ่งจะมีระยะจำกัดอย่างชัดเจน โดยที่เราไม่สามารถแก้ไขคำชี้แจงนี้ได้ จากภายในลูปนั้น

ตัวอย่าง

for letter in 'Python': # First Example

print 'Current Letter :', letter

fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # Second Example

print 'Current fruit :', fruit

for count in range(10,20) : # Second Example

print 'Current count :', count

print "Good bye!"

ผลที่ได้

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : h

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 45 | 65

Current Letter : o

Current Letter : n

Current fruit : banana

Current fruit : apple

Current fruit : mango

Current count : 10

Current count : 11

Current count : 12

Current count : 13

Current count : 14

Current count : 15

Current count : 16

Current count : 17

Current count : 18

Current count : 19

Good bye!

และถึงแม้ว่าเราจะเพิ่มโค้ดเข้าไปเพื่อควบคุมตัวแปร ไม่ให้เปลี่ยนแปลงระหว่างการวนลูปเช่น ทำให้ตัวแปร

letter ให้มีค่าเท่ากับตัว t เสมอ fruit ให้มีค่าเท่ากับตัว apple เสมอ หรือ count ให้มีค่าเท่ากับ 15 เสมอ

เพื่อไม่ให้โปรแกรมวนจนครบทุกรอบได้ ก็ไม่ส่งผลกระทบต่อการวนลูปของโปรแกรม

for letter in 'Python': # First Example

print 'Current Letter :', letter

letter = 't'

fruits = ['banana', 'apple', 'mango']

for fruit in fruits: # Second Example

print 'Current fruit :', fruit

fruit = 'apple'

for count in range(10,20) : # Second Example

print 'Current count :', count

count = 15

print "Good bye!"

ผลที่ได้คือ

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : h

Current Letter : o

Current Letter : n

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 46 | 65

Current fruit : banana

Current fruit : apple

Current fruit : mango

Current count : 10

Current count : 11

Current count : 12

Current count : 13

Current count : 14

Current count : 15

Current count : 16

Current count : 17

Current count : 18

Current count : 19

Good bye!

จะเห็นว่าไม่มีความแตกต่าง เพราะว่าตัวทำงานของโปรแกรม จะไม่ยอมให้เราเข้าถึง เพื่อแก้ไขค่าของตัวแปร

ในส่วนของ “คำชี้แจง” ได้

คำสั่งการวนลูปแบบ for นั้น ก็มีการรองรับคำสั่ง else เช่นเดียวกับ while แต่สำหรับ for ลูปนั้น else จะ

ทำงานเมื่อตัวแปรนั้นอยู่นอกช่วงการทำงาน ตามคำชี้แจงใน for ลูปนั้น เช่น

for num in range(10,20): #to iterate between 10 to 20

for i in range(2,num): #to iterate on the factors of the number

if num%i == 0: #to determine the first factor

j=num/i #to calculate the second factor

print '%d equals %d \* %d' % (num,i,j)

break #to move to the next number, the #first FOR

else: # else part of the loop

print num, 'is a prime number'

ผลที่ได้

10 equals 2 \* 5

11 is a prime number

12 equals 2 \* 6

13 is a prime number

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 47 | 65

14 equals 2 \* 7

15 equals 3 \* 5

16 equals 2 \* 8

17 is a prime number

18 equals 2 \* 9

19 is a prime number

จะเห็นว่า ค่าที่เป็น prime number (จำนวนเฉพาะ) นั้น คือค่าที่อยู่นอกการวนของ for ลูป ที่มีคำชี้แจงว่า

วนในช่วง 2 ไปจนถึงค่า num – 1 หากมีตัวเลขใดในการวนที่มีการหารแล้วลงตัว โปรแกรมจะหยุดวน และ

หลุดออกจากลูป for นี้ด้วยคำสั่ง break แต่เมื่อไม่มีเลขใดหารลงตัว ในช่วง 2 ไปจนถึงค่า num – 1 ค่า

สุดท้ายของ i คือเท่ากับ num ก็จะทำให้เข้าเงื่อนไข else ของลูป for นี้ แล้วจึงแสดงผลว่านี้คือเลข prime

number ตามคำสั่งที่อยู่ใน else นั่นเอง

ดังนั้นหากเราไม่ใส่คำสั่ง break ในการวน for ลูปนั้น การวนก็จะเกิดกับเลขทุกชุด ทุกตัว และเลขทุกตัวก็จะ

เป็น prime number ไปด้วย

for num in range(10,20): #to iterate between 10 to 20

for i in range(2,num): #to iterate on the factors of the number

if num%i == 0: #to determine the first factor

j=num/i #to calculate the second factor

print '%d equals %d \* %d' % (num,i,j)

#break #to move to the next number, the #first FOR

else: # else part of the loop

print num, 'is a prime number'

ผลลัพท์ที่ได้

10 equals 2 \* 5

10 equals 5 \* 2

10 is a prime number

11 is a prime number

12 equals 2 \* 6

12 equals 3 \* 4

12 equals 4 \* 3

12 equals 6 \* 2

12 is a prime number

13 is a prime number

14 equals 2 \* 7

14 equals 7 \* 2

14 is a prime number

15 equals 3 \* 5

15 equals 5 \* 3

15 is a prime number

16 equals 2 \* 8

16 equals 4 \* 4

16 equals 8 \* 2

16 is a prime number

17 is a prime number

18 equals 2 \* 9

18 equals 3 \* 6

18 equals 6 \* 3

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 48 | 65

18 equals 9 \* 2

18 is a prime number

19 is a prime number

การซ้อนกันของลูป (nested loops)

ในบางเวลาที่เราต้องพัฒนาโปรแกรมที่มีความซับซ้อน มีการวนคำนวณค่าในกลุ่มจำนวนย่อย จาก

กลุ่มจำนวนใหญ่ หรือมีการตรวจสอบข้อมูลที่เป็นตารางหลายๆตาราง แม้กระทั่งการจัดเรียงข้อมูลให้เป็น

ระเบียบ เรามักจำเป็นที่จะต้องในการวนลูป หลายๆลูปซ้อนกันเพื่อให้สามารถเข้าถึงข้อมูลได้ทั้งหมด

ตัวอย่างการคำนวณตารางคูณค่า โดยใช้การซ้อนกันของ while ลูป

i = 2

while i < 10 :

print "Multiply of " , i

j = 1

while j <= 12 :

print "%d x %d = %d"%(i,j,i\*j)

j = j +1

i = i +1

ตัวอย่างการคำนวณตารางคูณค่า โดยใช้การซ้อนกันของ for ลูป

for i in range (2,10):

print "Multiply of ",i

for j in range (2,13):

print "%d x %d = %d" %(i,j,i\*j)

ผลลัพท์ที่ได้

Multiply of 2

2 x 1 = 2

2 x 2 = 4

2 x 3 = 6

2 x 4 = 8

2 x 5 = 10

2 x 6 = 12

2 x 7 = 14

2 x 8 = 16

2 x 9 = 18

2 x 10 = 20

2 x 11 = 22

2 x 12 = 24

Multiply of 3

3 x 1 = 3

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 49 | 65

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

3 x 10 = 30

3 x 11 = 33

3 x 12 = 36

Multiply of 4

4 x 1 = 4

4 x 2 = 8

4 x 3 = 12

4 x 4 = 16

4 x 5 = 20

4 x 6 = 24

4 x 7 = 28

4 x 8 = 32

4 x 9 = 36

4 x 10 = 40

4 x 11 = 44

4 x 12 = 48

Multiply of 5

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

5 x 11 = 55

5 x 12 = 60

Multiply of 6

6 x 1 = 6

6 x 2 = 12

6 x 3 = 18

6 x 4 = 24

6 x 5 = 30

6 x 6 = 36

6 x 7 = 42

6 x 8 = 48

6 x 9 = 54

6 x 10 = 60

6 x 11 = 66

6 x 12 = 72

Multiply of 7

7 x 1 = 7

7 x 2 = 14

7 x 3 = 21

7 x 4 = 28

7 x 5 = 35

7 x 6 = 42

7 x 7 = 49

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 50 | 65

7 x 8 = 56

7 x 9 = 63

7 x 10 = 70

7 x 11 = 77

7 x 12 = 84

Multiply of 8

8 x 1 = 8

8 x 2 = 16

8 x 3 = 24

8 x 4 = 32

8 x 5 = 40

8 x 6 = 48

8 x 7 = 56

8 x 8 = 64

8 x 9 = 72

8 x 10 = 80

8 x 11 = 88

8 x 12 = 96

Multiply of 9

9 x 1 = 9

9 x 2 = 18

9 x 3 = 27

9 x 4 = 36

9 x 5 = 45

9 x 6 = 54

9 x 7 = 63

9 x 8 = 72

9 x 9 = 81

9 x 10 = 90

9 x 11 = 99

9 x 12 = 108

ข้อระวังในการการวนลูปซ้อนของ while ลูปคือ ตัวแปรของเงื่อนไขในการวนลูป เนื่องจากมันสามารถถูกแก้ไข

ค่าได้ทุกๆที่ในช่วงลูปของมัน อาจจะทำให้โปรแกรมทำงานผิดพลาดได้ เช่น

i = 2

while i < 10 :

print "Multiply of " , i

j = 1

while j <= 12 :

print "%d x %d = %d"%(i,j,i\*j)

j = j +1

i = j

i = i +1

ผลลัพท์ที่ได้

Multiply of 2

2 x 1 = 2

2 x 2 = 4

3 x 3 = 9

4 x 4 = 16

5 x 5 = 25

6 x 6 = 36

7 x 7 = 49

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 51 | 65

8 x 8 = 64

9 x 9 = 81

10 x 10 = 100

11 x 11 = 121

12 x 12 = 144

แต่ปัญหานี้จะไม่เกิดขึ้นกับลูปแบบ for เพราะคำสั่ง for แต่ละคำสั่ง ตัวแปรใน คำชี้แจงของลูปนั้น เป็นคนละ

ตัว แยกจากกัน และไม่สามารถแก้ไขค่าได้

for i in range (2,10):

print "Multiply of ",i

for i in range (2,13):

print "%d x %d = %d" %(i,i,i\*j)

ผลลัพท์ที่ได้

Multiply of 2

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 3

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 4

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 5

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 52 | 65

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 6

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 7

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 8

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

Multiply of 9

2 x 2 = 24

3 x 3 = 36

4 x 4 = 48

5 x 5 = 60

6 x 6 = 72

7 x 7 = 84

8 x 8 = 96

9 x 9 = 108

10 x 10 = 120

11 x 11 = 132

12 x 12 = 144

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 53 | 65

คำสั่งควบคุมการวนซ้ำ

break – เป็นคำสั่งให้โปรแกรมออกจากลูปที่วนอยู่ และไปทำงานในคำสั่งถัดไปนอกลูปนั้น

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 54 | 65

for letter in 'Python': # First Example

if letter == 'h':

break

print "print after break"

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

var = var -1

if var == 5:

break

print "print after break"

print 'Current variable value :', var

print "Good bye!"

ผลลัพท์ที่ได้

Current Letter : P

Current Letter : y

Current Letter : t

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Good bye!

จะเห็นว่าโปรแกรมทำงานถึงแค่เงื่อนที่เราตั้ง break ไว้เท่านั้น แล้วออกจากการวนลูปไปทำงานในคำสั่งอื่นต่อ

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 55 | 65

continue – เป็นคำสั่งให้โปรแกรมข้ามรอบงานในลูปที่ทำอยู่นี้ไปทำในรอบถัดไปทันที

Loop <เงื􀃉อนไข>

continue

คำสัง􀃉 / งาน #1

คำสัง􀃉 / งาน #2

ใช่ - จริง

ไม่ - เท็จ

ทำคำสัง􀃉 อื􀃉นต่อไป

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 56 | 65

for letter in 'Python': # First Example

if letter == 'h':

continue

print "print after continue"

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

var = var -1

if var == 5:

continue

print "print after continue"

print 'Current variable value :', var

print "Good bye!"

ผลลัพท์ที่ได้

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : o

Current Letter : n

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Current variable value : 0

Good bye!

จะเห็นว่าโปรแกรมจะข้ามข้อความหลังคำสั่ง continue และไม่แสดงตัวอักษรที่เรามีเงื่อนไขตรงกับที่ใส่

continue

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 57 | 65

pass – เป็นคำสั่งที่หมายถึง ให้ ”ผ่าน” ไป นั่นคือโปรแกรมจะไม่ทำอะไรเลย และจะผ่านไปตามขั้นตอนปกติ

มักจะใช้เมื่อเราไม่ต้องการให้เงื่อนไขนี้มีการทำอะไร และผา่ นไปเฉยๆ แต่ภาษาไพท่อนนั้น ไม่ยินยอมให้มีการ

ใช้คำสั่งแบบตรวจสอบเงื่อนไข หรือการวนลูป โดยไม่มีงานใดๆในเงื่อนไขนั้น หรือในลูปนั้นได้ เราจึงต้องใช้

คำสั่ง pass

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 58 | 65

for letter in 'Python': # First Example

if letter == 'h':

pass

print "print after break"

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

var = var -1

if var == 5:

pass

print "print after break"

print 'Current variable value :', var

print "Good bye!"

ผลลัพท์ที่ได้

Current Letter : P

Current Letter : y

Current Letter : t

print after break

Current Letter : h

Current Letter : o

Current Letter : n

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

print after break

Current variable value : 5

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Current variable value : 0

Good bye!

คำสั่ง pass นี้ไม่มีผลใดๆต่อการทำงานของโปรแกรมเลย แต่มันสามารถใช้ในกรณีที่ เรามีเงื่อนไขที่เราต้องการ

ตรวจสอบ แต่ไม่ต้องให้มีผลอะไรเช่น

for letter in 'Python': # First Example

if letter == 'h':

pass

print 'Current Letter :', letter

var = 10 # Second Example

while var > 0:

var = var -1

if var == 5:

pass

print 'Current variable value :', var

print "Good bye!"

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 59 | 65

ผลลัพท์ที่ได้

Current Letter : P

Current Letter : y

Current Letter : t

Current Letter : h

Current Letter : o

Current Letter : n

Current variable value : 9

Current variable value : 8

Current variable value : 7

Current variable value : 6

Current variable value : 5

Current variable value : 4

Current variable value : 3

Current variable value : 2

Current variable value : 1

Current variable value : 0

Good bye!

ซึ่งในภาไพท่อน หลังการตรวจสอบเงื่อนไข หรือการวนลูปใดๆ จำเป็นที่จะต้องรองรับด้วยคำสั่งเพื่อทำงาน

เสมอ แต่ในบางกรณีเรายังไม่ต้องการให้มีการทำงาน แต่เราต้องการให้มีคำสั่งตรวจสอบ เราสามารถใช้ pass

มาเป็นคำสั่งแทนได้

คำสั่งเงื่อนไข

เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข เพื่อตัดสินใจว่าจะทำคำสั่งหรือไม่

คำสั่งประเภทเงื่อนไขนี้มี 2 คำสั่งคือ if และ if….else ข้อแตกต่างสำคัญของ if และ if-else นั้นคือ if

จะทำคำสั่งเมื่อเงื่อนไขเป็นจริงเพียงอย่างเดียว ส่วน if-else นั้น เมื่อเงื่อนไขเป็นจริง ก็จะทำคำสั่งใน if แต่หาก

เงื่อนไขเป็นเท็จ จะมาทำคำสั่งใน else แปลว่า if-else จะมีคำสั่งทำทุกครั้งที่มีการตรวจสอบเงื่อนไข

If – if <เงื่อนไข> : เป็นคำสั่งที่ใช้ในการตรวจสอบเงื่อนไข เพื่อตัดสินใจว่าจะทำคำสั่งหรือไม่ และจะทำเมื่อ

เงื่อนไขเป็นจริงเพียงอย่างเดียวเท่านั้น

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 60 | 65

If <เงื􀃉อนไข>

คำสั􀃉ง / งาน #1

ใช่ - จริง

ไม่ - เท็จ

ทำคำสั􀃉งอื􀃉นต่อไป

ตัวอย่างการใช้ if

for i in range (1,11):

print i

if i == 5 :

print "this is five"

ผลลัพท์ที่ได้

1

2

3

4

5

this is five

6

7

8

9

10

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 61 | 65

If-else – if <เงื่อนไข> : ... else : คำสั่งนี้จะมีการทำคำสั่งทั้งในส่วนของ if เมื่อเงื่อนไขเป็นจริง และ else

เมื่อเงื่อนไขเป็นเท็จ

If <เงื􀃉อนไข>

else

คำสัง􀃉 / งาน #1

ใช่ - จริง

ไม่ - เท็จ

ทำคำสัง􀃉 อื􀃉นต่อไป

else

คำสัง􀃉 / งาน #2

ตัวอย่างการใช้ if-else

for i in range (1,11):

if i == 5 :

print "this is five"

else :

print i

ผลลัพท์ที่ได้

1

2

3

4

this is five

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 62 | 65

6

7

8

9

10

การใช้คำสั่ง if-else ซ้อนกัน หลายเงื่อนไข เพื่อให้งา่ ยต่อการแยกเงื่อนไขการตัดสินใจของโปรแกรม ในกรณีที่

มีเงื่อนไขมากกว่า 1 เงื่อนไข เช่น

การแทนตัวเลขด้วยข้อความข้างต้น แต่มีการใช้เงื่อนไขเพิ่มขึ้น เช่น ในช่วง 3-7 นั้น ให้แสดงคำว่า “this is

number : “ แล้วตามด้วยตัวเลข ยกเว้นเลข 5 ที่ต้องแสดงข้อความว่า “this is five” เช่นเดิม และ ที่เลข 9

นั้นแสดงคำว่า “This is nine” แทนที่เลข 9

ตัวอย่างการใช้ if-else ซ้อนกัน

for i in range (1,11):

if i == 5 :

print "this is five"

elif i == 9 :

print "this is nine"

elif i < 8 and i > 1 :

print "this is number : ",i

else :

print i

ผลลัพท์ที่ได้

1

this is number : 2

this is number : 3

this is number : 4

this is five

this is number : 6

this is number : 7

8

this is nine

10

ซึ่งผลที่ได้จะต่างจากการใช้ if ธรรมดา

for i in range (1,11):

if i == 5 :

print "this is five"

if i == 9 :

print "this is nine"

if i < 8 and i > 1 :

print "this is number : ",i

else :

print i

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 63 | 65

ผลลัพท์ที่ได้

1

this is number : 2

this is number : 3

this is number : 4

this is five

this is number : 5

this is number : 6

this is number : 7

8

this is nine

9

10

ข้อควรระวังในการใช้ if-else แบบซ้อนกันคือ ลำดับการตรวจสอบเงื่อนไขของ if ที่ซ้อนกันอยู่ ซึ่งบางครั้งการ

สลับลำดับการตรวจสอบเงื่อนไข อาจจะทำให้ผลที่ได้เปลี่ยนไปด้วย เช่น

for i in range (1,11):

if i == 9 :

print "this is nine"

elif i < 8 and i > 1 :

print "this is number : ",i

elif i == 5 :

print "this is five"

else :

print i

ผลลัพท์ที่ได้

1

this is number : 2

this is number : 3

this is number : 4

this is number : 5

this is number : 6

this is number : 7

8

this is nine

10

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 64 | 65

เชิงอรรถ

1. “Hello World” เป็นประโยคที่เหล่านักพัฒนาโปรแกรมนิยมใช้กัน ในการทดลองเมื่อเริ่มต้นเรียนรู้ในภาษาใหม่ๆ หรือเครื่องมือใหม่ ที่

ใช้สำหรับพัฒนาโปรแกรม โดยเป้าหมายคือ สั่งให้โปรแกรมแสดงผลคำว่า “Hello World” บนหน้าจอ เพียงเท่านั้น แต่มันหมายถึงว่า

เราเข้าใจการทำงานของภาษาในเบื้องต้น และเข้าใจกระบวนการใช้งาน IDE ได้ถูกต้อง จุดเริ่มต้นของการใช้คำว่า “Hello World” ใน

การทดสอบนั้น มาจากชายที่ชื่อ Brian Kernighan ผู้แต่งหนังสือการเขียนโปรแกรมด้วยภาษา C ที่ชื่อว่า “The C Programming

Language” ในปี พ.ศ. 2512 (ค.ศ. 1978) ได้เขียนตัวอย่างโปรแกรมที่แสดงผลคำว่า “hello world” บนหน้าจอ ซึ่งหนังสือเล่นมีนี้มี

การใช้งานอย่างแพร่หลายมาก ส่งผลให้นักพัฒนาในยุคต่อมา มักจะเขียนโปรแกรมเริ่มต้นด้วยคำว่า “Hello World” ตามตัวอย่างใน

หนังสือตามไปด้วย

2. “Python” ถูกนำมาใช้ครั้งแรกในปี ค.ศ.1989 (พ.ศ.2532) โดย Guido van Rossum ผู้พัฒนาภาษา Python ออกแบบมาให้เป็น

ภาษาสคริปต์ระดับสูง คือมีความเข้าใกล้ภาษามนุษย์มากขึ้น เราสามารถพอจะคาดเดาผลจากการใช้งานของคำสั่งในแต่ละคำสั่งได้จาก

ภาษาที่ใช้ และรองรับรูปแบบการพัฒนาโปรแกรมสมัยใหม่ รวมไปถึงแนวความคิดโดยทั่วไป ในการพัฒนาโปรแกรมสมัยใหม่ ซึ่ง

สามารถนำมาใช้ได้กับการพัฒนาโปรแกรมบนภาษาอื่นๆได้เช่นเดียวกัน

3. “Zen of Python” - “คำพรรณวิถีแห่งไพท่อน”

Beautiful is better than ugly.

ความงามล้ำค่ากว่าไร้ระเบียบ

Explicit is better than implicit.

ความชัดแจ้งมีความหมายกว่าทุกนัยยะ

Simple is better than complex.

ความเรียบง่ายสมบูรณ์กว่าซับซ้อน

Complex is better than complicated.

แต่ความซับซ้อนก็ยังดีกว่าความสับสน

Flat is better than nested.

ความแบนเรียบมั่นคงกว่าการซ้อนทับ

Sparse is better than dense.

ความบางเบาสบายกว่าทึบแน่น

Readability counts.

จงเคารพต่อผู้อ่าน

Special cases aren't special enough to break the rules.

ไม่มีกรณีใด พิเศษพอที่จะอยู่เหนือกฎ

Although practicality beats purity.

แม้ว่าประสบการณ์จะทำลายความไร้เดียงสา

Errors should never pass silently.

ข้อผิดพลาดก็ไม่เคยที่ผ่านไปอย่างเงียบงัน

Unless explicitly silenced.

ยกเว้นว่ามันจะเงียบเฉียบ

In the face of ambiguity, refuse the temptation to guess.

การห้ามใจจากการคาดเดา เมื่อต้องเผชิญหน้ากับความไม่แน่ใจ

การเขียนโปรแกรมพืน􀃊 ฐาน 􀅙􀅘􀅙

P a g e 65 | 65

There should be one-- and preferably only one --obvious way to do it.

มีเพียงทางเดียว ทางเดียวที่ชัดเจน ทางเดียวที่ทำได้

Although that way may not be obvious at first unless you're Dutch.

แม้ว่าทางนั้น ดูเหมือนจะผิดในตอนแรก เว้นแต่คุณจะเขลาเกิน

Now is better than never.

และตอนนี้ก็ดีกว่าที่เคย

Although never is often better than \*right\* now.

แม้ว่าบ่อยครั้งที่ผ่านมาก็ดีกว่าตอนนี้

If the implementation is hard to explain, it's a bad idea.

ความคิดแย่ๆ คืออะไรก็ตามที่ไม่สามารถอธิบายได้

If the implementation is easy to explain, it may be a good idea.

ถ้าการทำงานนั้นเข้าใจได้ง่าย มันอาจจะเป็นความคิดที่ดี

Namespaces are one honking great idea -- let's do more of those!

เนมสเปซ คือวิธีการที่เยี่ยมในการประกาศความคิด -- จงทำให้มากขึ้นไปอีก

-19-Aug-2004-Tim Peters (วิศวกรคอมพิวเตอร์)

อ้างอิง

1. http://blog.hackerrank.com/the-history-of-hello-world/

2. https://www.tutorialspoint.com/python/python\_quick\_guide.htm

3. http://www.fordantitrust.com/files/python.pdf\_\_