PYTHON 101

เขียนโปรแกรมด้วยภาษาไพทอนเบื้องต้น

Python 101

CONTENTS

บทนำ

เริ่มต้นกับ ‘HELLO WORLD’

Python 101

เหมือนเป็นธรรมเนียมในการเขียนโปรแกรมไปแล้ว ไม่ว่าจะเขียนภาษาอะไร เรามักจะเริ่มต้นด้วยการเขียนคำสั่งให้โปรแกรมพิมพ์ข้อความว่า "Hello World!" ออกมาเสมอ หรือจะลองพิมพ์ประโยคอะไรก็ตามที่เราอยากพิมพ์ก็ได้นะ ลองเปลี่ยนดูเลย

File 1: helloworld.py

```
print("Hello, World!")
print('Hello, World!')
print("In programming the hard part isn't solving problems, but deciding what
problems to solve. - Paul Graham")
```

ผลที่ได้คือ

```
>>> %Run helloworld.py

  Hello, World!
  Hello, World!
  In programming the hard part isn't solving problems, but deciding what problems to solv
  e. - Paul Graham
```

โดยคำสั่งในการพิมพ์ข้อความคือ print แล้วตามด้วย (‘ ข้อความ ’) หรือ (“ ข้อความ ”)

การแทรกคำอธิบาย

การแทรกคำอธิบาย (Comment) ในกระบวนการพัฒนาโปรแกรมเมื่อมีการเขียนโค้ดที่ค่อนข้างยาว การพัฒนาโปรแกรมขนาดใหญ่ที่มีการทำงานร่วมกันหลายคนหรือการพัฒนาโปรแกรมที่ซับซ้อนนั้น การเขียนคำอธิบายไว้ระหว่างบรรทัดของการเขียนโปรแกรม เพื่อช่วยบอกว่าโค้ดบรรทัดนี้นั้น มีไว้ทำอะไร ผลที่ได้จากบรรทัดนี้คืออะไร ซึ่งจะทำให้เมื่อมีการกลับมาดูอีกครั้ง หรือมีการวิเคราะห์โปรแกรมอีกครั้งหนึ่งทำได้ง่าย และมีความถูกต้องขึ้นมากในภาษา Python นั้น การแทรกคำอธิบายนั้นสามารถทำได้โดยใช้สัญลักษณ์ hash ‘#’ นำหน้าบรรทัด

File 2: comments.py

```
# This is the comment for the first line.
print("Hello, World") # This is the comment for the second line
```

ตัวแปร

การกำหนดค่าตัวแปร

ตัวแปร (Variables) ถูกใช้เพื่อเก็บค่าที่เราสามารถอ้างถึงมาใช้ในภายหลังได้ ตัวแปรก็เหมือนการติดป้ายชื่อ แล้วใช้ ‘ = ’ เพื่อกำหนดค่าให้กับตัวแปร

การตั้งชื่อสามารถตั้งชื่อได้เฉพาะตัวอักษร ตัวเลข และ underscore(_) ไม่สามารถขึ้นต้นด้วยตัวเลขได้ ห้ามมีช่องว่างหรือเว้นวรรค ห้ามตั้งชื่อตัวแปรซ้ำกับคำสงวน (ดูคำสงวน)

a=2 คือให้ตัวแปร a มีค่าเป็นจำนวนเต็มเท่ากับ 2

b=c=5 คือการกำหนดค่า 5 ให้กับตัวแปร b และ c เรียกการกำหนดค่าแบบนี้ว่า **chained assignment**

my_name = "Emilia" คือการกำหนดให้ตัวแปร my_name มีค่าเป็นข้อความ Emilia ซึ่งในบรรทัดต่อมา my_name = "Thomas" เป็นการเปลี่ยนค่าที่เก็บเป็น Emilia ก่อนหน้าเป็นค่าใหม่คือ Thomas

File 3: variables.py

```
a = 5
c = b = 2
print("a = ", a)
print("b = ", b)
print("c = ", c)

my_name = "Emilia"
print("My name is = " + my_name)
my_name = "Thomas"
print("My name is = " + my_name)
```

ผลที่ได้

```
>>> %Run variables.py

  a =  5
  b =  2
  c =  2
  My name is = Emilia
  My name is = Thomas
```

ถ้าหากเราไม่ได้กำหนดค่าตัวแปรไว้ แล้วเรียกใช้ตัวแปรที่ไม่ได้สร้างไว้จะเกิด error เพราะหาตัวแปรที่อ้างอิงไม่เจอ

File 4: undefinevar.py

```
a = 1
print(b)
```

ผลที่ได้

```
>>> %Run undefinevar.py

  Traceback (most recent call last):
    File "F:\traintober2017\python101\undefinevar.py", line 2, in <module>
      print(b)
  NameError: name 'b' is not defined
```

ชนิดข้อมูลของตัวแปร

Python มีชนิดของตัวแปร 2 ชนิดหลักๆ ได้แก่

1. ตัวเลข (number) แบ่งเป็น
    a. จำนวนเต็ม (Integer) เช่น a= 10, b=3078
    b. จำนวนจริงหรือทศนิยม (Float) เช่น 3.14, 22/7, 2.31E5
    c. จำนวนตรรกะ (Boolean) คือ True และ False
    d. จำนวนเชิงซ้อน (Complex Numbers) เช่น 1+0j
2. ข้อความหรือสตริง (String)

เราสามารถตรวจสอบว่าในตัวแปรนั้นเมีค่าเป็นข้อมูลชนิดใดโดยใช้คำสั่ง type(ตัวแปร)

File 5: vaiable_type.py

```
a = 12
b = 12.3
c = "This is the string."

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

ผลที่ได้

```
>>> %Run variable_type.py

  <class 'int'>
  <class 'float'>
  <class 'str'>
```

ใน python มีฟังก์ชันในตัวที่สามารถแปลงชนิดข้อมูลเป็นชนิดอื่นได้ โดยคำสั่ง int(ตัวแปร) จะแปลงค่าจำนวนจริงเป็นชนิดจำนวนเต็มได้ float(ตัวแปร) สามารถแปลงชนิดข้อมูลจากจำนวนเต็มแสดงเป็นจำนวนจริงได้ และ str(ตัวแปร) สามารถแปลงค่าจากชนิดจำนวนเต็มและจำนวนจริงไปเป็นค่าแบบข้อความหรือสตริงได้

File 6: variable_convert.py

```
a = 9
print(type(a))

b = 9.23
print(type(b))

print(float(a))
print(int(b))
```

ผลที่ได้

```
>>> %Run variable_convert.py

  <class 'int'>
  <class 'float'>
  9.0
  9
```

ตัวดำเนินการทางคณิตศาสตร์

Arithmetic Operators หรือ ตัวดำเนินการทางคณิตศาสตร์ ประกอบไปด้วย บวก + (addition), ลบ – (subtraction), คูณ * (multiplication) , หาร / (division), ยกกำลัง ** (power) และหารเอาเศษ % (modulo)

สัญลักษณ์	การคำนวนณ	ตัวอย่าง
+	บวก	a = 5 , b = 4.5 c = a + b c = 9.5
-	ลบ	a = 5 , b = 4.5 c = a - b c = 0.5
*	คูณ	a = 9 , b = 2 c = a * b c = 18
/	หาร	a = 9 , b = 2 c = a / b c = 4.5
**	ยกกำลัง	a = 9 , b = 2 c = a ** b c = 81
%	หารเอาเศษ	a = 9 , b = 2 c = a % b c = 1

File 7: arithmetic_operators.py

```
a = 5
b = 9

add = a + 4.5
sub = a - 4.5
mul = a * b
power = b ** 2
mod = b % 2

print("Add = " + str(add))
print("Sub = " + str(sub))
print("Mul = " + str(mul))
print("Power = " + str(power))
print("Mod = " + str(mod))
```

ผลที่ได้

```
>>> %Run arithmetic_operators.py

   Add = 9.5
   Sub = 0.5
   Mul = 45
   Power = 81
   Mod = 1
```

ORDER OF OPERATIONS ลำดับการดำเนินการ

Order of Operations เป็นข้อตกลงกันร่วมกันในการดำเนินการทางคณิตศาสตร์ เพื่อให้การแก้ปัญหาทางคณิตศาสตร์เป็นไปอย่างถูกต้อง ไม่เช่นนั้นผลลัพธ์ที่ได้จะแตกต่างกัน

PEMDAS เป็นชื่ออย่างไม่เป็นทางการของ Order of Operations ย่อมาจาก Parenthesis, Exponents, Multiply, Divide, Add และ Subtract โดยลำดับการดำเนินการคือ **เมื่อเจอวงเล็บ (Parenthesis) ให้ทำในวงเล็บก่อน จากนั้นถ้าเจอยกกำลัง (Exponents) ก็ทำเป็นลำดับต่อมา ตามด้วยคูณ (Multiply) และหาร (Divide) ด้วยลำดับความสำคัญเท่ากัน ถ้าคูณมาก่อนก็ให้คูณ ถ้าหารมาก่อนก็ให้หาร สุดท้ายบวก(Add)และลบ(Subtract) ด้วยวิธีคิดเดียวกันกับคูณหาร อันไหนมาก่อนให้ทำอันนั้น**

P	E	M	D	U	S
Parenthesis ()	Exponents a^2	Multiply ×	Divide ÷	Add +	Subtract -

ตัวดำเนินการกำหนดค่า

ตัวดำเนินการกำหนดค่า (Assignment Operators) เป็นสัญลักษณ์ที่ใช้สำหรับกาหนดค่าหรือเปลี่ยนแปลงค่าให้แก่ตัวแปร ซึ่งตัวแปรทางด้านซ้ายมือจะถูกกาหนดค่าจากข้อมูลหรือตัวแปรจากด้านขวามือ โดยสมมุติว่า ตัวแปร a =10, b = 5, c = 9.0, d = 2.0, e = -3.5

+= เป็นการบวกค่าตัวแปรสองค่าจากฝั่งขวาไปเก็บไว้ที่ตัวแปรฝั่งซ้าย c += a จะเท่ากับ c = c + a

-= เป็นการลบค่าตัวแปรสองค่าจากฝั่งขวาไปเก็บไว้ที่ตัวแปรฝั่งซ้าย c -= a จะเท่ากับ c = c - a

File 8: assignments.py

```
a = 4.5
print("a = " + str(a))

a -= 3
print("a = " + str(a))

a += 10
print("a = " + str(a))
```

ผลที่ได้

```
>>> %Run assignments.py

  a = 4.5
  a = 1.5
  a = 11.5
```

ตัวดำเนินการเปรียบเทียบ

ตัวดำเนินการเปรียบเทียบ (COMPARISON OPERATORS) สามารถใช้เครื่องหมาย == (equality operator) ในการตรวจสอบว่าตัวแปรสองตัวเปรียบเทียบว่ามีค่าเหมือนกันหรือไม่ โดยผลที่ได้จากการเปรียบเทียบจะเป็น Boolean คือ True หรือ False

File 9: boolean_operators.py

```
a = 5
b = 18

is_equal = a == b

print(is_equal)
```

ผลที่ได้

```
>>> %Run boolean_operators.py

  False
```

นอกจากจะเปรียบเทียบด้วย == แล้ว เรายังสามารถเปรียบเทียบด้วย >= (มากกว่าหรือเท่ากับ) ,<= (น้อยกว่าหรือเท่ากับ), > (มากกว่า) , < (น้อยกว่า) และ != (ไม่เท่ากับ) ค่าที่ได้จากการเปรียบเทียบจะเป็นข้อมูลแบบ Boolean คือ True กับ False

File 10: comparison_operators.py

```
a = 1
b = 2
c = 3

print(a < b < c)

is_greater = c > b
is_less = b < a

print(is_greater)
print(is_less)
```

ผลที่ได้

```
>>> %Run comparison_operators.py

  True
  True
  False
```

ตัวจัดการข้อความ

การนำข้อความมาเรียงต่อกัน

การเอาข้อความมาเรียงต่อกัน (Concatenation) อย่างเช่นเราต้องเอาข้อความในตัวแปรมาต่อกัน จะใช้เครื่องหมาย + คั่นระหว่างตัวแปล

File 11: concatennation.py

```
hello = "Hello"
world = 'World'

hello_world = hello + world
print(hello_world)
hello_world = hello + " "+ world
print(hello_world)
```

ผลที่ได้

```
>>> %Run concatennation.py

  HelloWorld
  Hello World
```

STRING MULTIPLICATION

Python สามารถใช้ String คูณด้วย integer ได้ด้วย เช่น " * " * 10

File 12: string_multiplication.py

```
star = " * "
ten_of_stars = star * 10
print(ten_of_stars)
```

ผลที่ได้

```
>>> %Run string_multiplication.py
    *  *  *  *  *  *  *  *  *  *
```

STRING INDEXING

String ใน Python เก็บค่าแต่ละ characters เป็น index โดยเริ่มต้นจาก index = 0 เราสามารถเรียกตัวอักษรได้จากตำแหน่ง โดยใช้ str[index] เช่น

H	e	l	l	o
[0]	[1]	[2]	[3]	[4]

File 13: string_indexing.py

```
string = "Hello"
print("e : " + string[1])

h_letter = string[0]
print(h_letter)
```

ผลที่ได้

```
>>> %Run string_indexing.py

  e : e
  H
```

NEGATIVE INDEXING

สามารถใช้ค่าติดลบ ในการเข้าถึง index ของ String ได้ โดยจะเริ่มทำการนับจากด้านหลังมาด้านหน้า เช่น

H	e	l	l	o
[-5]	[-4]	[-3]	[-2]	[-1]

File 14: negative_indexing.py

```
string = "Hello, World!"
l_string = string[-1]
print(l_string)
```

ผลที่ได้

```
>>> %Run negative_indexing.py

  !
```

SLICING

Slicing หรือ substring คือการตัดคำออกจากข้อความ เช่น string[start:end]

File 15: Slicing.py

```
string = "Hello, World!"
hello = string[:5]
print(hello)
world = string[7:-1]
print(world)
```

ผลที่ได้

```
>>> %Run slicing.py

   Hello
   World
```

IN OPERATOR

เราสามารถตรวจสอบได้ว่าในตัวแปรมีข้อความหรือตัวอักษรที่เราต้องการหรือไม่ ซึ่งค่าที่ได้จากการตรวจสอบจะเป็น Boolean คือ True เมื่อมีข้อความที่ค้นหา หรือ false เมื่อไม่มีข้อความที่ค้นหา

File 16: in_operator.py

```
string = "Hello, World!"
print("Hello" in string)

contains = "foo" in string
print(contains)
```

ผลที่ได้

```
>>> %Run in_operator.py

   True
   False
```

STRING LENGTH

len() เป็นฟังค์ชันที่เอาไว้นับจำนวนความยาวของ String

""" (triple-quoted) เอาไว้สำหรับประกาศ String แบบหลายๆบรรทัด

File 17: string_length.py

```
poem = """Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
"""
print(poem)
poem_lenght = len(poem)
print("lenght : ", poem_lenght)
```

ผลที่ได้

```
>>> %Run string_length.py

  Beautiful is better than ugly.
  Explicit is better than implicit.
  Simple is better than complex.
  Complex is better than complicated.

  lenght :  132
```

CHARACTER ESCAPING

Backslash (\) เอาไว้ใช้กรณีต้องการพิมพ์ " หรือ ' ลงใน String ซึ่งโดยปกติ String จะมี " ที่เริ่มต้นและสุดของ String หากแทรก " ใน String โปรแกรมจะมอง " ใน String กลายเป็นจุดสิ้นสุดของ String กลายเป็น error ไป วิธีแก้ ก็คือต้องใช้ \ เช่น

File 18: char_escaping.py

```
print("\"In programming the hard part isn\'t solving problems, but deciding
what problems to solve.\" - Paul Graham")
print('"In programming the hard part isn\'t solving problems, but deciding
what problems to solve." - Paul Graham')
```

ผลที่ได้

```
>>> %Run char_escaping.py

  "In programming the hard part isn't solving problems, but deciding what problems to solve." - Paul Graham
  "In programming the hard part isn't solving problems, but deciding what problems to solve." - Paul Graham
```

STRING METHODS

lower() เป็นสำหรับทำให้ String กลายเป็นตัวพิมพ์เล็ก

upper() เอาไว้สำหรับเปลี่ยน String เป็นตัวพิมพ์ใหญ่

File 19: string_methods.py

```
string = "Hello, World!"
print(string)

print(string.lower())

print(string.upper())
```

ผลที่ได้

```
>>> %Run string_methods.py

  Hello, World!
  hello, world!
  HELLO, WORLD!
```

Python 101

STRING FORMATTING

string format ใช้ % ตามหลัง String ร่วมกับตัวแปร เพื่อกำหนด format ของ String ที่เราต้องการ หลักการคือ จะเปลี่ยนค่า %s ใน String กลายเป็นตัวแปรที่เรากำหนดไว้ เช่น

```
name = "Alice"
years = 25
print("My name is %s." % name)

print("I'm %d years old" % years)
```

ผลที่ได้

```
>>> %Run string_formatting.py

  My name is Alice.
  I'm 25 years old
```

DATA STRUCTURES

LIST INTRODUCTION

list คือ data structure ที่เอาไว้เก็บข้อมูลหลายๆข้อมูล ค่าที่เก็บใน list ต้องเป็นชนิดเดียวกัน เช่น

A list is a data structure you can use to store a collection of different pieces of information under a single variable name. A list can be written as an array of comma-separated values (items) between square brackets, e.g. lst = [item1, item2]. Lists might contain items of different types, but usually all the items in the list are of the same type. Like strings, lists can be indexed and sliced (see Lesson 3).

```
numbers = [1, 4, 9, 16, 25]
names = ['Alice', 'John', 'Dave', 'Lisa', 'Matt']
print(numbers)
print(names)

print(numbers[1:-1])
print(names[1:4])
```

ผลที่ได้

```
>>> %Run lists.py

  [1, 4, 9, 16, 25]
  ['Alice', 'John', 'Dave', 'Lisa', 'Matt']
  [4, 9, 16]
  ['John', 'Dave', 'Lisa']
```

LIST OPERATIONS

การเพิ่มข้อมูลลงไปใน list ใช้เมธอด append() และ list เป็น mutable เราสามารถเปลี่ยนข้อมูลใน list ได้ เช่น

You can add new items at the end of the list, by using the append() method and concatenation. Unlike strings, lists are a mutable type, i.e. it is possible to change their content using lst[index] = new_item

```
names = ['Alice', 'John']  # create new list
print(names)

names += ['Dave', 'Lisa']    # add two items to the list
print(names)

names.append("Matt")   # add one more item to the list using append() method
print(names)

names[-1] = 'Matthew'
print(names)
```

ผลที่ได้

```
>>> %Run list_operation.py

   ['Alice', 'John']
   ['Alice', 'John', 'Dave', 'Lisa']
   ['Alice', 'John', 'Dave', 'Lisa', 'Matt']
   ['Alice', 'John', 'Dave', 'Lisa', 'Matthew']
```

LIST ITEMS

การเพิ่มข้อมูลลงไปใน list ใช้เมธอด append() และ list เป็น mutable เราสามารถเปลี่ยนข้อมูลใน list ได้ เช่น

Assignment to slices is also possible, and this can even change the size of a list or clear it entirely.

Clear animals list.

```
names = ['Alice', 'John', 'Dave', 'Lisa', 'Matthew'] # create new list
print(names)

names[1:3] = ['Amy']    # replace 2 items 'John' and 'Dave' with 'Amy'
print(names)

names[1:3] = []     # remove 2 items 'Amy' and 'Lisa'
print(names)

names[0:] = [] #Clear List Items
print(names)
```

ผลที่ได้

```
>>> %Run list_items.py

  ['Alice', 'John', 'Dave', 'Lisa', 'Matthew']
  ['Alice', 'Amy', 'Lisa', 'Matthew']
  ['Alice', 'Matthew']
  []
```

TUPLES

Tuples คือ list ที่ไม่สามารถเปลี่ยนแปลงข้อมูลได้ ไม่สามารถเพิ่ม ลบ หรือแก้ไขข้อมูลเหมือน list ได้ มี syntax ง่ายๆคือ ขึ้นต้นและปิดด้วย (และ) ขั้นข้อมูลด้วย comma เช่น

Tuples are almost identical to lists. The only significant difference between tuples and lists is that tuples cannot be changed: you cannot add, change, or delete elements from the tuple. Tuples are constructed by a comma operator enclosed in parentheses, for example (a, b, c). A single item tuple must have a trailing comma, such as (d,).

```
moons_of_saturn = ('Mimas', 'Enceladus', 'Tethys', 'Dione', 'Rhea', 'Titan',
'Iapetus')

print(len(moons_of_saturn))
```

ผลที่ได้

```
>>> %Run tuples.py

  7
```

DICTIONARIES

Dictionary คล้ายคลึงกับ list แต่ต่างกันที่ Dictionary สามารถเข้าถึงข้อมูลได้ด้วยการใช้ key แทนที่จะเป็น index แบบ list (คล้ายคลึงกับ Ruby Hash, Java HashMap หรือ JSON ท่าเทียบกับภาษาอื่นๆ)

A dictionary is similar to a list, except that you access its values by looking up a key instead of an index. A key can be any string or a number. Dictionaries are enclosed in curly braces e.g. dct= {'key1' : "value1", 'key2' :"value2"}.

Print Jane's phone number from phone_book.

```
# create new dictionary.
moons_of_saturn = {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062,
                   'Dione': 1123, 'Rhea': 1527, 'Titan': 5150}
print(moons_of_saturn)

# Add new item to the dictionary
moons_of_saturn['Iapetus']= 1470
print(moons_of_saturn)

# Remove key-value pair from moons_of_saturn
del moons_of_saturn['Titan']

print(moons_of_saturn['Rhea'])
```

ผลที่ได้

```
>>> %Run dictionary.py
  {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062, 'Dione': 1123, 'Rhea': 1527, 'Titan': 5150}
  {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062, 'Dione': 1123, 'Rhea': 1527, 'Titan': 5150, 'Iapetus': 1470}
  1527
```

DICTIONARY KEYS() AND VALUE()

There are a lot of useful methods in dictionaries such as keys() and values(). You can explore the rest using Ctrl + Space after a dict_namefollowed by a dot.

```
# create new dictionary.
moons_of_saturn = {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062,
                   'Dione': 1123, 'Rhea': 1527, 'Titan': 5150}
print(moons_of_saturn)

print(moons_of_saturn.keys())
print(moons_of_saturn.values())
```

ผลที่ได้

```
>>> %Run dict_key_value.py
  {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062, 'Dione': 1123, 'Rhea': 1527, 'Titan': 5150}
  dict_keys(['Mimas', 'Enceladus', 'Tethys', 'Dione', 'Rhea', 'Titan'])
  dict_values([396, 504, 1062, 1123, 1527, 5150])
```

IN KEYWORD

in นอกจากเช็คตัวอักษรใน String ได้แล้ว ก็ยังสามารถเช็คว่ามี item ที่ต้องการใน list หรือ dictionary ได้อีกด้วย

The **in** keyword is used to check if a list or a dictionary contains a specific item. You can apply **in** to lists or dictionaries the same way as you did with strings.

Check if grocery_dict keys contain **"fish"**

```
moons_of_saturn_list = ['Mimas', 'Enceladus', 'Tethys']
moons_of_saturn_dict = {'Mimas': 396, 'Enceladus': 504, 'Tethys': 1062}

print('Enceladus' in moons_of_saturn_list)
print('Mimas' in moons_of_saturn_dict)
```

ผลที่ได้

```
>>> %Run in_keyword.py

  True
  True
```

CONDITION EXPRESSIONS

LOGICAL OPERATORS

ใช้สำหรับเปรียบเทียบค่าซึ่งผลลัพธ์ที่ได้จะเป็น บูลลีน โดย

ถ้าใช้ And (และ) จะได้ผลลัพธ์เป็น True ก็ต่อเมื่อทั้งสองฝั่งเป็น True

ถ้าใช้ Or (หรือ) จะได้ผลลัพธ์เป็น True โดยที่ฝั่งใดฝั่งหนึ่งเป็น True หรือ เป็น True ทั้งสองฝั่ง

ถ้าเติม not (ไม่) ข้างหน้าค่าที่เป็น True จะกลายเป็น False ถ้าเติมหน้าค่าที่เป็น False กลายเป็น True

ลำดับในการดำเนินการไม่ได้เริ่มจากซ้ายไปขวา แต่นับจาก not and และ or ตามลำดับ

```
name = "John Perry"
age = 75

print(name == "John Perry" and age != 57)
print(name == "John Scalzi" and age == 75)

print(name == "John Perry" or age != 57)
print(name == "John Scalzi" or age == 75)

print(name == "John Perry" or not age > 57)
print(name == "John Scalzi" or not (name == "John Perry" and age == 75))
```

ผลที่ได้

```
>>> %Run logical_operators.py

  True
  False
  True
  True
  True
  False
```

IF STATEMENT

The if keyword is used to form a conditional statement that executes some specified code after checking if its expression is True. Python uses indentation to define code blocks.

```
name = "John Perry"
age = 75

if name == "John Perry" or age == 75:
    print("His name is John Perry.")
    print("John Perry is 75 years old.")
```

ผลที่ได้

```
>>> %Run if_statement.py

  His name is John Perry.
  John Perry is 75 years old.
```

ELSE ,ELIF PART IN IF STATEMENT

The else statement complements the if statement. The elif keyword is short for "else if".

```
point = 75

if point >= 80:
    print('Grade A')
elif point >= 70:
    print('Grade B')
elif point >= 60:
    print('Grade C')
elif point >=50:
    print('Grade D')
else:
    print('Grade F')
```

```
>>> %Run else_elif.py

  Grade B
```

LOOPS

FOR LOOP

for loops are used to iterate over a given sequence. On each iteration, the variable defined in the for loop will be assigned to the next value in the list.

```
for i in range(5): # range(5) => list [0, 1, 2, 3, 4]
    print(i)

# traversal of List sequence
numbers = [7, 45, 85, 12, 43]
for num in numbers:
    print('Number : ' + str(num))
```

ผลที่ได้

```
>>> %Run for_loops.py

  0
  1
  2
  3
  4
  Number : 7
  Number : 45
  Number : 85
  Number : 12
  Number : 43
```

FOR LOOP USING STRING

Strings are very similar to lists in Python. You can use string to iterate over it.

```
# traversal of a string sequence
planets = 'Planets'
for letter in planets:
    print('Letter : '+ letter)

# initialize length variable
length = 0
for i in planets:
    # add 1 to the length on each iteration
    length += 1

print(len(planets) == length)
```

ผลที่ได้

```
>>> %Run for_string.py

  Letter : P
  Letter : l
  Letter : a
  Letter : n
  Letter : e
  Letter : t
  Letter : s
  True
```

WHILE LOOP

A while loop is similar to an if statement: it executes some code if some condition is true. The key difference is that it will continue to execute indented code for as long as the condition is True.

```
count = 0
while (count < 5):
    print ('count :', count)
    count = count + 1
```

ผลที่ได้

```
>>> %Run while_loop.py

  count : 0
  count : 1
  count : 2
  count : 3
  count : 4
```

BREAK KEYWORD

An infinite loop is a loop that never exits. If the loop condition happens to always be True, such a loop becomes infinite. The break keyword is used to exit the current loop.

```
for letter in 'Planets':     # First Example
   if letter == 't':
      break
   print ('Current Letter :', letter)

var = 10                    # Second Example
while var > 0:
   print ('Current val :', var)
   var = var-1
   if var== 5:
      break
```

ผลที่ได้

```
>>> %Run break_keyword.py

   Current Letter : P
   Current Letter : l
   Current Letter : a
   Current Letter : n
   Current Letter : e
   Current val : 10
   Current val : 9
   Current val : 8
   Current val : 7
   Current val : 6
```

CONTINUE KEYWORD

The continue keyword is used to skip the rest of the code inside the loop for the currently executed loop and return to the for or while statement.

```
for x in range(10):
    if x % 2 == 0:
        continue
    print(x)
```

ผลที่ได้

```
>>> %Run continue_keyword.py

   1
   3
   5
   7
   9
```

FUNCTION

FUNCTION

Functions are a convenient way to divide your code into useful blocks, make it more readable and help reuse it. Functions are defined using the keyword def, followed by the function's name.

```
def hello_world():
    print("Hello, World!")

for i in range(5):
    hello_world()
```

ผลที่ได้

```
>>> %Run functions.py

  Hello, World!
  Hello, World!
  Hello, World!
  Hello, World!
  Hello, World!
```

PARAMETER AND CALL ARGUMENT

Function parameters are defined inside the parentheses (), following the function name. A parameter acts as a variable name for the passed argument.

```
def square(x):
    print(x ** 2)

square(5)
square(12)
```

ผลที่ได้

```
>>> %Run parameter_arg.py

  25
  144
```

RETURN VALUE

Functions may return a value to the caller, using the keyword return. You can use the returned value to assign it to a variable or just print it out.

```
def area(width, height):
    a = width * height
    return a

print('Area = %d' % area(5,2))
```

ผลที่ได้

```
>>> %Run return_keyword.py

  Area = 10
```

DEFAULT PAREMETER

Sometimes it's useful to specify a default value for one or more function parameters. This creates a function that can be called with fewer arguments than it is defined to allow.

```
def area(width, height = 10):
    a = width * height
    return a

print('Area = %d' % area(5,2))
print('Area = %d' % area(5))
```

ผลที่ได้

```
>>> %Run default_parameter.py

  Area = 10
  Area = 50
```

ABOUT PYTHON

BRIEF HISTORY OF PYTHON

ย้อนกลับไปเดือนธันวาคมปี ค.ศ. 1989 โปรแกรมเมอร์ชาวดัชต์นามว่า กีโด ฟาน รอสซัม (Guido van Rossum) มองหาโปรเจคอดิเรกทำช่วงวันหยุดยาวคริสต์มาส เขาตัดสินใจเขียนตัวแปลคำสั่ง (Interpreter) สำหรับภาษาใหม่ที่เคยคิดไว้ และตั้งชื่อภาษาใหม่ว่า **"Python"** (อ่านว่า ไพทอน) ซึ่งมาจากชื่อรายการตลก Monty Python's Flying Circus

ภาษา Python รับอิทธิพลจากหลายภาษา ABC, Modula-3, Icon, ANSI C, Perl, Lisp, Smalltalk และ Tcl โดยเฉพาะอย่างยิ่งจากภาษา ABC ซึ่งรอสซัมเองก็เคยช่วยพัฒนาภาษา ABC ช่วงต้นของปี ค.ศ. 1980 ภาษา ABC เป็นภาษาที่สง่างามและทรงพลังที่ถูกสร้างขึ้นสำหรับสอนโปรแกรมมิ่ง แต่กระนั้นก็ไม่เคยได้รับความนิยม รอสซัมคิดว่าเหตุผลที่ ABC ไม่ได้รับความนิยมก็เพราะเป็นการยากที่เพิ่มคำสั่งพื้นฐานใหม่ๆ ให้ ABC มันมีขนาดใหญ่ เป็นระบบปิด ทำได้เพียงสั่งงานงาน I/O พื้นฐาน อย่างเช่น อ่านข้อความจากคอนโซล เขียนข้อความไปยังคอนโซล รอสซัมตัดสินใจจะไม่ทำพลาดเช่นนี้อีกกับ Python

และนี่อาจเป็นเหตุให้ Python นิยมใช้ในการศึกษา เพราะจุดเริ่มต้นออกแบบภาษามาเพื่อใช้ในการสอนและมุ่งหมายไปยังผู้ที่ไม่ใช่โปรแกรมเมอร์อาชีพ ไม่เพียงเท่านั้น Python ยังเป็นโครงการเปิด ส่งผลให้ชุมชนผู้พัฒนามีขนาดใหญ่และยืดหยุ่น python ดึงดูดทั้งนักพัฒนาหน้าใหม่และนักพัฒนาอาชีพด้วย

PYTHON ในทุกๆที่

Python ได้รับความนิยมในวงกว้าง ถูกนำไปพัฒนา software ในด้านต่างๆมากมาย

Python 101

บริษัทและองค์กรจำนวนมากใช้ python ในการพัฒนาโปรแกรม เช่น Google, NASA, Bank of America, Disney, CERN, YouTube, Mozilla หรือแม้กระทั่งสื่ออย่าง The Guardian

“THE ZEN OF PYTHON” - คำพรรณวิถีแห่งไพทอน

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.

ยกเว้นว่ามันจะเงียบเชียบ

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

การห้ามใจจากการคาดเดา เมื่อต้องเผชิญหน้ากับความไม่แน่ใจ

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)

Spaggri แปล

เชิงอรรถ

1. “Hello World” เป็นประโยคที่เหล่านักพัฒนาโปรแกรมนิยมใช้กัน ในการทดลองเมื่อเริ่มต้นเรียนรู้ในภาษาใหม่ๆ หรือเครื่องมือใหม่ ที่ใช้สำหรับพัฒนาโปรแกรม โดยเป้าหมายคือ สั่งให้โปรแกรมแสดงผลคำว่า “Hello World” บนหน้าจอ เพียงเท่านั้น แต่มันหมายถึงว่าเราเข้าใจการทำงานของภาษาในเบื้องต้น และเข้าใจกระบวนการใช้งาน IDE ได้ถูกต้อง จุดเริ่มต้นของการใช้คำว่า “Hello World” ในการทดสอบนั้น มาจากชายที่ชื่อ Brian Kernighan ผู้แต่งหนังสือการเขียนโปรแกรมด้วยภาษา C ที่ชื่อว่า “The C Programming Language” ในปี พ.ศ. 2512 (ค.ศ. 1978) ได้เขียนตัวอย่างโปรแกรมที่แสดงผลคำว่า “hello world” บนหน้าจอ ซึ่งหนังสือเล่มนี้มี

การใช้งานอย่างแพร่หลายมาก ส่งผลให้นักพัฒนาในยุคต่อมา มักจะเขียนโปรแกรมเริ่มต้นด้วยคำว่า “Hello World” ตามตัวอย่างในหนังสือตามไปด้วย

2. “Python” ถูกนำมาใช้ครั้งแรกในปี ค.ศ.1989 (พ.ศ.2532) โดย Guido van Rossum ผู้พัฒนาภาษา Python ออกแบบมาให้เป็นภาษาสคริปต์ระดับสูง คือมีความเข้าใกล้ภาษามนุษย์มากขึ้น เราสามารถพอจะคาดเดาผลจากการใช้งานของคำสั่งในแต่ละคำสั่งได้จากภาษาที่ใช้ และรองรับรูปแบบการพัฒนาโปรแกรมสมัยใหม่ รวมไปถึงแนวความคิดโดยทั่วไป ในการพัฒนาโปรแกรมสมัยใหม่ ซึ่งสามารถนำมาใช้ได้กับการพัฒนาโปรแกรมบนภาษาอื่นๆได้เช่นเดียวกัน

อ้างอิง

1. http://blog.hackerrank.com/the-history-of-hello-world/
2. http://www.fordantitrust.com/files/python.pdf
3. https://www.tutorialspoint.com/python3/index.htm
4. PyCharm Edu https://www.jetbrains.com/help/pycharm-edu/pycharm-edu.html
5. https://sites.google.com/site/dotpython/
6. http://keancode.github.io/python/
7. http://marcuscode.com/lang/python
8. http://www.cs.su.ac.th/~tasanawa/cs517321/python/PythonOperator.pdf
9. Tollervey, Nicholas H.. (2015). Python in Education Teach, Learn, Program. California: O’Reilly Media, Inc.