Title: ตำราวิชา Problem Solving and Computer Programming (PSCP) - PSCP Book

Author: รศ.ดร. โชติพัชร์ ภรณวลัย

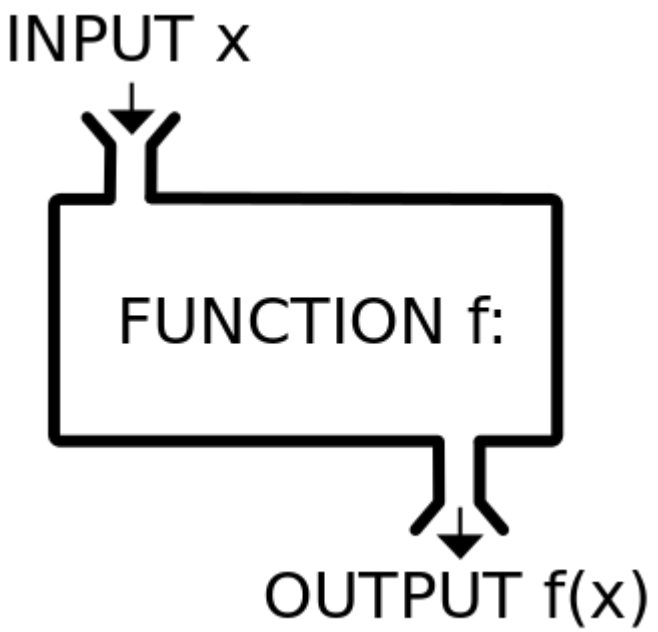
Language: th-TH

Date: 22 สิงหาคม 2565

Chapter 2: Functions

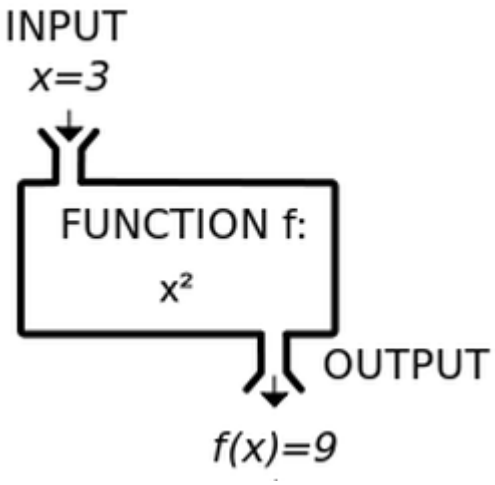
Built-in Functions

ในการเขียนโปรแกรมคอมพิวเตอร์ `Function` คือ ชุดของคำสั่งหลายๆคำสั่งที่เรียงลำดับต่อกัน และมีชื่อที่สามารถเรียกใช้ได้ โดยเมื่อต้องการให้ทำงานตามชุดคำสั่งนั้น ก็เพียงแต่เรียกชื่อ `Function` นั้น โดยอาจจะส่งค่า `argument` ไปด้วยหรือไม่ก็ได้ ดังรูปด้านล่างนี้ `f` คือ ชื่อของ `function` และค่าของ `x` ที่ส่งเข้าไปให้กับ `function f` เรียกว่า `argument`



อ้างอิงรูปจาก https://en.wikipedia.org/wiki/Function_(mathematics)

เมื่อส่งค่า 3 หรือที่เรียกว่า `argument` ให้กับ `function f(x) = x**2` ตัวแปร `x` ก็จะมีค่าเท่ากับ 3 เป็น Input ของ `function f` และจะได้ Output หรือ `f(x)` เป็น 9 ดังแสดงในรูปด้านล่าง



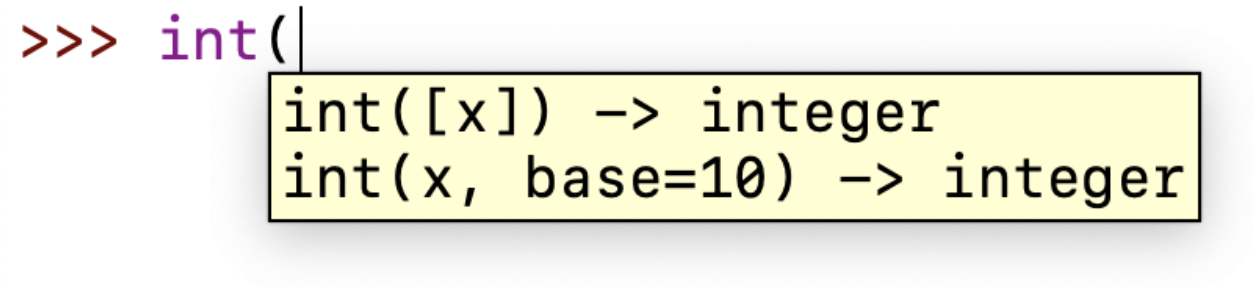
อ้างอิงรูปจาก https://en.wikipedia.org/wiki/Function_(mathematics)

ในบทที่ผ่านมา เราได้กล่าวถึง Function ไปบ้างแล้ว ยกตัวอย่างเช่น ที่ผ่านมา เราได้รู้จัก Function ชื่อว่า `print()` `type()` `input()` `int()` `float()` และ `str()` สังเกตว่าหลังชื่อ function จะมีเครื่องหมายวงเล็บ `()`ตามมาด้วยเสมอ

Function `print()` อาจจะมีการส่งค่าเข้าไปที่เรียกว่า `argument` ได้หลายค่าเช่น `print(123, 10.11, 'Hello')` จะมีการส่งค่า `argument` เข้าไป 3 ค่า ค่าแรกเป็น Integer (123) ค่าที่สองเป็น Float (10.11) และค่าที่สามเป็น String ('Hello')

เราอาจจะเลือกส่ง `argument` เดียวไปยัง `print()` ก็ได้ เช่น `print('Hello')` หรือไม่ส่งเลยก็ได้เช่น `print()` ซึ่งจะมีความหมายว่าไม่ต้องพิมพ์อะไรออกที่หน้าจอ และให้ขึ้นบรรทัดใหม่ไปเลย เพราะค่า `end='\n'` เป็นค่า `default` ของ `print()`

แม้ว่า `print()` จะเป็น function ที่สามารถส่ง `argument` ไปจำนวนเท่าไหร่ก็ได้ แต่บาง function จะต้องส่ง `argument` ไปตามจำนวนที่กำหนดเท่านั้น เช่น function `int()` ถ้าเราลองพิมพ์คำว่า `int(` คือพิมพ์คำว่า int และสัญลักษณ์วงเล็บเปิด ทิ้งไว้ โปรแกรม 'IDLE' จะแสดงตัวช่วย (`Call Stack Visibility`) ซึ่งแสดงในกรอบสีเหลืองดังในรูปด้านล่าง



จากรูป จะเห็นได้ว่า function `int` สามารถรับ `argument` ได้ 0 ตัว หรือ 1 ตัว หรือ 2 ตัว เท่านั้น ที่สามารถตอบได้ว่ามี 0 หรือ 1 ตัว คือจะเห็นว่าในบรรทัดแรก `x` จะถูกล้อมรอบด้วยเครื่องหมายก้ามปู `[]` ซึ่งหมายความว่า อาจจะมี `x` หรือไม่มีก็ได้ ถ้ามีก็มีได้ 1 ตัว คือ `x` นั่นเอง

ส่วนบรรทัดที่สอง จะเห็นได้ว่า function `int` รับ `argument` ได้ 2 ตัว (ค่า) ค่าแรกคือ `x` และค่าที่สองคือ `base` หรือเลขฐาน ซึ่งมีค่าเริ่มต้นเป็น 10

จากสองบรรทัดที่แสดงใน Call Stack Visiblity จะเห็นว่ามีการเขียน มีการเขียน -> integer แสดงว่า function int() จะคืนค่า หรือ return ค่ากลับมาเป็นชนิด integer เสมอ

ลองดูตัวอย่างการใช้งาน `int` กัน ดังรูปด้านล่าง

```
>>> int()
0
>>> int('111')
111
>>> int(111)
111
>>> int('111', 2)
7
>>> int('111', '222', '333')
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    int('111', '222', '333')
TypeError: int() takes at most 2 arguments (3 given)
>>>
```

ถ้าไม่ส่ง `argument` เข้าไปที่ `int` เลย function `int` จะส่งค่า หรือ return ค่า 0 กลับมา

เมื่อส่ง `'111'` ซึ่งเป็น string เข้าไป จะได้ `111` กลับมา และหากส่ง integer `111` เป็น `argument` เข้าไป ก็จะได้ Integer `111` กลับมาเช่นกัน

เมื่อส่ง `argument` เข้าไป 2 ค่า ได้แก่ `111` และ `2` จะได้ค่า `7` เพราะ `7` คือ เลขฐาน 2 ของ 111

และหากส่งเข้าไป 3 ค่า เช่น `'111' '222' '333'` จะได้ `TypeError` กลับมา โดยมีข้อความว่า `int()` สามารถรับสูงสุด 2 argument แต่ว่าส่งมา 3 ค่า

อย่างไรก็ตามไม่จำเป็นว่าถ้าเราส่ง 'argument' เข้าไป 2 ค่าแล้ว จะไม่มี Error เลย ยกตัวอย่างเช่น ดังรูปด้านล่าง จะเห็นได้ว่า ค่า `base` ที่เป็นไปได้จะต้องมีค่ามากกว่าหรือเท่ากับ 2 และ น้อยกว่าหรือเท่ากับ 36 หรือ 0 เท่านั้น ดังนั้นหากส่งค่า 50 เข้าไปที่ `base` จะได้ `ValueError` กลับมา

```
>>> int('111', 50)
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    int('111', 50)
ValueError: int() base must be >= 2 and <= 36, or 0
>>>
```

Function `print()` `type()` `int()` `float()` และ `str()` เป็น Function ที่มีมากับภาษา Python อยู่แล้ว โดยเราไม่จำเป็นต้องไป download อะไรเพิ่มเติม และนำมาติดตั้งเพื่อเรียกใช้งาน เราจึงเรียก Function เหล่านี้ว่า `Built-in Function` ในภาษา Python มี `Built-in Function` อีกมากมายให้เรียกใช้งาน ซึ่งเราจะค่อยๆได้ศึกษาต่อไป

Importing Modules

มี `Built-in Function` อื่นๆ อีกที่เราสามารถเรียกใช้งานได้ แต่จำเป็นที่จะต้องมีการ `import` เข้ามาก่อน ยกตัวอย่างเช่น หากเราต้องการใช้ Function ทางคณิตศาสตร์ เช่น `sin(x)` `cos(x)` เราสามารถใช้งานได้ แต่ต้อง import Module ที่ชื่อว่า `math` เข้ามาก่อน ซึ่งใน `Module math` นี้จะมีการรวบรวม Function ทางคณิตศาสตร์จำนวนมากให้สามารถเลือกใช้งาน

หากเราต้องการทราบว่า Module Math มี Function อะไรให้ใช้งานได้บ้าง สามารถทำได้โดยการพิมพ์ `import math` เพื่อทำการ import module math เข้ามา และพิมพ์ `help(math)` จะแสดงผลดังรูปด้านล่าง (ไม่ได้แสดงผลทั้งหมด)

```
>>> import math
>>> help(math)
Help on module math:

NAME
    math

MODULE REFERENCE
    https://docs.python.org/3.8/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module provides access to the mathematical functions
    defined by the C standard.

FUNCTIONS
    acos(x, /)
        Return the arc cosine (measured in radians) of x.

    acosh(x, /)
        Return the inverse hyperbolic cosine of x.

    asin(x, /)
        Return the arc sine (measured in radians) of x.

    asinh(x, /)
        Return the inverse hyperbolic sine of x.

    atan(x, /)
        Return the arc tangent (measured in radians) of x.

    atan2(y, x, /)
```

Ln: 866 Col: 4

และถ้าลองเลื่อนลงไปเรื่อยๆจะพบว่ามี function ทางคณิตศาสตร์ให้เลือกใช้มากมาย โดยเรียงชื่อ Function ตามตัวอักษร ตั้งแต่ a จนถึง z ตามลำดับ โดยด้านล่างสุดจะแสดงค่า DATA หรือค่า constant ทางคณิตศาสตร์ เช่น ค่า `pi` ให้สามารถใช้ได้ด้วย

```
        Difference between x and the closest integer multiple of y.

        Return x - n*y where n*y is the closest integer multiple of y.
        In the case where x is exactly halfway between two multiples of
        y, the nearest even value of n is used. The result is always exact.

    sin(x, /)
        Return the sine of x (measured in radians).

    sinh(x, /)
        Return the hyperbolic sine of x.

    sqrt(x, /)
        Return the square root of x.

    tan(x, /)
        Return the tangent of x (measured in radians).

    tanh(x, /)
        Return the hyperbolic tangent of x.

    trunc(x, /)
        Truncates the Real x to the nearest Integral toward 0.

        Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    inf = inf
    nan = nan
    pi = 3.141592653589793
    tau = 6.283185307179586

FILE
    /Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/lib-dynload/math.cpyth
on-38-darwin.so
```

ใน module math จะมี function หนึ่ง ชื่อว่า `hypot()` ถ้าอยากทราบวิธีการใช้งาน function นี้สามารถทำได้ ดังรูปด้านล่าง รูปนี้เป็นผลลัพธ์ของการ `run` บน Python version 3.8.5 หาก `run` บน version อื่น อาจจะได้ผลลัพธ์ไม่เหมือนกัน

```
>>> help(math.hypot)
Help on built-in function hypot in module math:

hypot(...)
    hypot(*coordinates) -> value

    Multidimensional Euclidean distance from the origin to a point.

    Roughly equivalent to:
        sqrt(sum(x**2 for x in coordinates))

    For a two dimensional point (x, y), gives the hypotenuse
    using the Pythagorean theorem:  sqrt(x*x + y*y).

    For example, the hypotenuse of a 3/4/5 right triangle is:

        >>> hypot(3.0, 4.0)
        5.0

>>>
```

จากคำอธิบายในรูป Function `hypot()` นี้เป็น `Built-in function` ที่อยู่ใน module `math` และใช้ในการหาค่า **'Multidimensional Euclidean distance'** จากจุด `origin` ไปยังจุด (`point`) หรือ `coordinate` ที่ต้องการ โดยเรา

สามารถส่ง `coordinate` เป็น `argument` ไปยัง function `hypot()`

หมายเหตุ

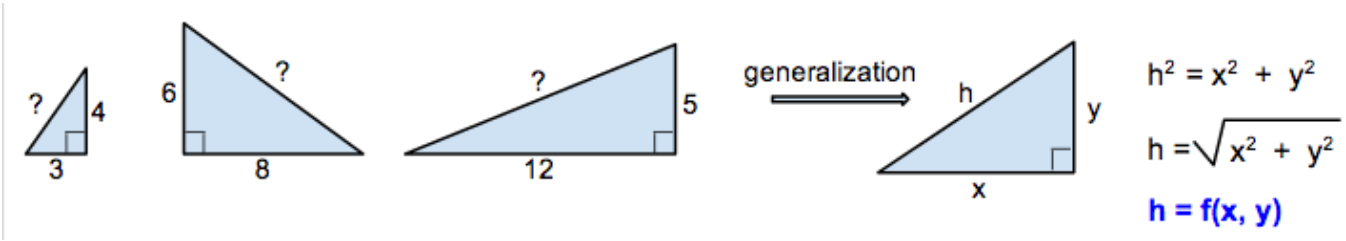
1. คำว่า `hypot()` ย่อมาจากคำว่า `hypotenuse` ซึ่งแปลว่าเส้นแทยงมุม

2. สามารถศึกษาข้อมูลเพิ่มเติมเกี่ยวกับ Euclidean Distance ได้ที่ https://en.wikipedia.org/wiki/Euclidean_distance

ตัวอย่างการใช้งาน `hypot()` เป็นไปดังรูปด้านล่าง

```
>>> # 1-dimensional Euclidean distance from origin (0) to point (1)
>>> math.hypot(1)
1.0
>>> # 1-dimensional Euclidean distance from origin (0) to point (2)
>>> math.hypot(2)
2.0
>>> # 2-dimensional Euclidean distance from origin (0, 0) to point (3, 4)
>>> math.hypot(3, 4)
5.0
>>> # 2-dimensional Euclidean distance from origin (0, 0) to point (5, 12)
>>> math.hypot(5, 12)
13.0
>>> # 2-dimensional Euclidean distance from origin (0, 0) to point (1, 1)
>>> math.hypot(1, 1)
1.4142135623730951
>>> # 3-dimensional Euclidean distance from origin (0, 0, 0) to point (1, 1, 1)
>>> math.hypot(1, 1, 1)
1.7320508075688772
>>> # 4-dimensional Euclidean distance from origin (0, 0, 0, 0) to point (1, 1, 1, 1)
>>> math.hypot(1, 1, 1, 1)
2.0
```

การคำนวณหาค่า 2-dimensional Euclidean distance ก็คือการหาค่า ด้านตรงข้ามมุมฉากของ สามเหลี่ยมมุมฉาก ตามทฤษฎีของพีธากอรัส ดังนั้นถ้าด้านประกอบมุมฉากของสามเหลี่ยม 2 ด้าน มีค่าเท่ากับ 3 และ 4 ด้านตรงข้ามมุมฉาก (hypotenuse) จะมีค่าเท่ากับ 5 ซึ่งคำนวณได้จาก ($\sqrt{3^2 + 4^2}$)



อ้างอิงรูปจาก https://cs.wellesley.edu/~cs110/reading/L16/

เนื่องจาก `hypot()` เป็น function ที่อยู่ใน module `math` และการใช้งาน module `math` จำเป็นต้อง `import` ก่อน ดังนั้นการเรียกใช้งาน `hypot()` จำเป็นที่จะต้องมีการเรียกชื่อ module `math` และตามด้วยเครื่องหมายจุด แล้วจึงตามด้วยชื่อ Function ใน module `math` นั้น เช่น `math.hypot(3, 4)` เป็นต้น

ที่เราจำเป็นต้องระบุชื่อ module ก่อนหน้าชื่อ function เป็นเพราะว่า อาจจะมีชื่อ function ซ้ำกัน แต่อยู่คนละ module

ยกตัวอย่างเช่น สมมุติว่า มี module `x` (ไฟล์ `x.py`) และ module `y` (ไฟล์ `y.py`) ทั้ง module `x` และ `y` มี function ชื่อ `hypot()` ซ้ำกัน ถ้าหากว่าการที่เรา `import x` และ `import y` เข้ามาในโปรแกรมของเรา ก็อาจจะเกิดความสับสนได้ว่า หากเราเรียกใช้งาน `hypot()` โดยไม่ระบุชื่อ module ว่าเป็น `hypot()` ของ module `x` หรือ `y` ก็จะทำให้ Python ไม่ทราบว่าผู้เขียนโปรแกรมนี้ เจตนาต้องการใช้ `hypot()` ของ module `x` หรือ `y` กันแน่

ดังนั้น Python จึงต้องให้ผู้เขียนโปรแกรมระบุชื่อ module ที่ต้องการไว้ก่อนหน้าชื่อ function แล้วคั่นกันด้วยเครื่องหมายจุด เช่น หากต้องการใช้ `hypot()` ของ module `x` ให้เรียก `x.hypot()` หากต้องการใช้ `hypot()` ของ module `y` ก็

ให้เรียกใช้ด้วย `y.hypot()`

อย่างไรก็ตามหากผู้เขียนโปรแกรมไม่ต้องการระบุชื่อ module ก็สามารถทำได้เช่นกัน โดยเวลา `import` แทนที่จะพิมพ์ว่า `import x` ก็ให้พิมพ์ว่า `from x import hypot` แทน และการเรียกใช้ `hypot()` ก็สามารถเรียกใช้ได้โดยตรงเลย ไม่ต้องระบุชื่อ module `x` นำหน้าอีกแล้ว

แต่การทำเช่นนี้ ผู้เขียนโปรแกรมต้องมีความระมัดระวังเพิ่มเติม เพราะหากมีการพิมพ์ `from y import hypot()` เข้าไปเพิ่มในโปรแกรมนี้ด้วย การเรียกใช้ `hypot()` โดยไม่ระบุชื่อ module จะทำให้ Python เรียกใช้ `hypot()` ของ module `y` เพราะว่าเราได้ทำการ import ตัว `hypot()` ของ module `y` เข้ามาทีหลัง ดังนั้น `hypot()` ของ module `y` จะไปแทนที่ `hypot()` ของ module `x` และทำให้เราไม่สามารถเรียกใช้งาน `hypot()` ของ module `x` ในโปรแกรมนั้นได้อีก

และหากต้องการจะ import ทุก function ใน module ใดๆ เช่น module `x` ให้พิมพ์ คำว่า `from x import *` โดยเครื่องหมาย `*` แทนทุก function ที่อยู่ใน module `x`

จากตัวอย่างข้างบนจะเห็นได้อีกอย่างว่า `hypot()` สามารถรับ argument ได้หลายจำนวน (ตามจำนวน dimension) ให้ผู้เรียนลองทดสอบดูว่า กรณีต้องการคำนวณหา `Euclidean distance` สำหรับ 0 dimension จะได้ค่าเท่าไร

ใน help ของ `hypot()` มีการเขียนว่า `hypot(*coordinates) -> value` แสดงว่า `hypot()` สามารถรับ `argument` ได้หลายค่า เนื่องจากมีเครื่องหมาย `*` อยู่ การใช้งานลักษณะนี้เรียกว่า `scatter` ซึ่งจะได้กล่าวถึงภายหลังต่อไป การใช้เครื่องหมาย `->` เป็นการบอกว่า `hypot()` จะ return ค่า (`value`) กลับมา 1 ค่า

นอกจาก `Built-in function` แล้วในภาษา Python ยังมีผู้พัฒนาอิสระที่ได้พัฒนา Module อื่นๆ เพิ่มเติม ที่ไม่ได้ถูกรวมไว้ใน Python แต่หากเราต้องการใช้ function ใน module นั้น ก็สามารถทำได้ โดยการ download module ที่ผู้พัฒนานั้นได้แจกจ่าย เพื่อมาติดตั้งในเครื่องคอมพิวเตอร์ของเรา เพื่อให้ Python รู้จัก Module นั้น และสามารถเรียกใช้งานได้ การใช้งาน Module ลักษณะนี้ จะเรียกว่าเป็น `Third-pary Libraries` หมายถึงการใช้ Library หรือ Module จากผู้พัฒนาภายนอก มาใช้ในโปรแกรมของเรา

Exercise 1 (MyMath)

ให้ผู้เรียนลองทดลองทำโจทย์ข้อ `MyMath` ซึ่งมีการกำหนด `Input Specification` และ `Output Specification` ไว้ในรูปด้านล่าง

Specification

Input Specification	Output Specification
ไม่มีค่าส่งเข้าในโจทย์ข้อนี้	5 บรรทัด ตามฟังก์ชั่นดังนี้ ปล.ฟังก์ชั่นตรีโกณมิติที่ใช้ในโจทย์ข้อนี้มีหน่วยเป็นองศาทั้งหมด $\frac{\sin(90)+\sin^2(60)}{\cos(245^2)+\cos(45+135)}$ (1) $\frac{16!4!}{8!}$ (2) $\frac{15+25}{\sqrt{(25-12)^2+(12-15)^2}}$ (3) $\log_{10}(1234^4)$ (4) $\frac{\log_5 4234+\log_2 8191+71\log_{10}156154}{\log_7 777-\log_8 888-\log_9 999}$ (5)

ข้อควรระวัง การใช้งาน function ใดๆ ควรศึกษาก่อนว่า function นั้นมีวิธีการใช้งานอย่างไร ยกตัวอย่างเช่น function `sin()` มีการรับค่า `argument` อะไร รับ `argument` ได้กี่จำนวน และหน่วยของ `argument` เป็นอะไร เช่น หน่วยเป็น `radian` (เรเดียน) หรือ `degree` (องศา) เป็นต้น

Adding a New Function

เราสามารถสร้าง function ขึ้นมาใช้งานเองได้ด้วยเช่นเดียวกัน วิธีการสร้างจะมีรูปแบบดังนี้

```
"""Module Docstring"""
def function_name(parameter1, parameter2, ...):
    """Function Docstring"""
    statements
    function_call
    ...

function_name(argument1, argument2, ...)
```

ในบรรทัดแรกของไฟล์ เราจะเขียน `Module Docstring` เพื่ออธิบายโดยย่อว่า `Module` นี้คืออะไร และใน 1 ไฟล์ หรือ 1 module นี้ เราสามารถสร้าง function ได้หลายๆ function โดยแต่ละ function จะมีรูปแบบเที่เหมือนกัน คือ

1. เริ่มต้นด้วยคำว่าคำสงวน `def` ซึ่งย่อมาจากคำว่า `define` หรือ นิยาม เนื่องจากว่า `def` เป็นคำสงวน ดังนั้นจึงไม่สามารถเอาคำว่า `def` ไปเป็นชื่อตัวแปรในโปรแกรมอีก
2. `function_name` จะเป็นชื่อของ function ตัวอย่างชื่อ function ที่เรารู้จักแล้ว เช่น `print` และ `type` เป็นต้น โดยปกติชื่อ function มักจะเป็นคำกริยาในภาษาอังกฤษ เพื่อเป็นการบอกว่า function นี้จะทำ (กริยา) อะไร การตั้งชื่อ function ควรตั้งตามหลักการตั้งชื่อ function ที่ดี โดยทั่วไปจะเหมือนกับหลักการตั้งชื่อตัวแปร เช่น ชื่อ

function จะเป็นตัวพิมพ์เล็กในภาษาอังกฤษทั้งหมด และมีอักขระพิเศษได้อย่างเดียวคือ `underscore` หรือ `_` เท่านั้น นอกจากนี้ตัวอักษระตัวแรกของชื่อ function ไม่สามารถเป็นตัวเลขได้ เป็นต้น

3. ถัดจากชื่อ function (ซึ่งอยู่ในบรรทัดเดียวกัน) จะเป็นวงเล็บ และมีตัวแปร ที่เรียกว่า `parameter` อยู่ภายในวงเล็บ โดยอาจจะมีหรือไม่มี `parameter` ก็ได้ ถ้ามีอาจจะมี 1 parameter หรือมากกว่าก็ได้ เมื่อเราเรียกใช้ function จะต้องมีการส่งค่า `argument` มาให้ตรงกับที่ function กำหนดไว้ ค่า `argument` ที่ function รับเข้ามาก็จะถูก assign หรือใส่ให้ตรงกับ `parameter` ที่ได้นิยามไว้ และหลังวงเล็บปิดของบรรทัดนี้ให้เขียนเครื่องหมาย colon หรือ `:` เราจะเรียกบรรทัดที่ใช้ในการนิยามชื่อ function และ parameter (ในข้อ 2-3) ทั้งหมดว่า `Header`

4. ทุกๆ function จะต้องมีการสร้าง `function docstring` เพื่ออธิบายโดยย่อว่า `function` นี้คือ อะไร ใช้งานอย่างไร เป็นต้น จะสังเกตเห็นว่า **สิ่งที่อยู่ภายใน function หรือบรรทัดต่อๆไปทั้งหมดของ function นี้ จะต้องถูก indent หรือขยับเข้าไปทางขวา 4 ช่องว่าง เสมอ** สิ่งที่อยู่ในภายใน function จะมีทั้ง statement และการเรียกใช้งาน function อื่นๆ เราจะเรียกสิ่งที่อยู่ภายใน function นี้ทั้งหมดว่า `Body` เมื่อเขียน `Body` เสร็จแล้ว ให้เว้นบรรทัดว่างอย่างน้อย 1 บรรทัด เพื่อเป็นบอกว่าได้นิยาม function นี้เสร็จแล้ว

เมื่อต้องการใช้งาน function นั้นให้ทำงาน ก็ให้เรียกใช้งาน function_name นั้น และมีการส่งค่า `argument` ไป ตัวอย่างในรูปด้านบน ค่า `argument1` ก็จะถูกส่งไปยัง `parameter1` และ ค่า `argument2` ก็จะถูกส่งไปยัง `parameter2` ตามลำดับ

ยกตัวอย่าง เราจะทำโจทย์ข้อ `Entryway` โดยการเขียนเป็น Function ได้ดังนี้

```
"""Entryway problem"""

def entryway():
    """Print a string 'Output'"""
    print('Output')

entryway()
```

ในตัวอย่างนี้

1. `"""Entryway problem"""` คือ `module docstring`
2. `entryname` คือ `function_name` โดย function นี้ไม่มีการกำหนด `parameter` เลย
3. `def entryway():` บรรทัดนี้เรียกว่า `Header`
4. `"""Print a string 'Output'"""` คือ `function docstring` และเป็นส่วนนึงของ Body ของ function `entryway`
5. `print('Output')` เป็นส่วนของ `Body` ของ function นี้
6. การเรียกใช้งาน function `entryway` ก็ต้องมีการใส่วงเล็บด้วย แม้ว่าจะไม่ต้องส่ง `argument` ใดๆ ไปให้ `entryway` เลยก็ตาม

Void vs. Fruitful Functions

Function สามารถแบ่งได้เป็น 2 ประเภท คือ

1. Function ที่ไม่มีการคืนค่า เรียกว่า `Void function`

2. Function ที่มีการคืนค่า เรียกว่า `Fruitful function`

Function `entryway` ในรูปข้างบน จัดเป็นประเภท `Void function` เนื่องจาก `entryway` ไม่มีการคืนค่า (`return`) อะไรกลับมา สิ่งที่ `entryway` ทำ คือการแสดงผลคำว่า `'Output'` บนหน้าจอเท่านั้น

```
>>> entryway()
Output
>>> x = entryway()
Output
>>> x
>>> type(x)
<class 'NoneType'>
>>>
```

ดังตัวอย่างในรูปด้านบน หากเราสร้างตัวแปร `x` มารับค่าผลจากการเรียก function `entryway` จะพบว่าเมื่อเรียกดูค่า `x` ในบรรทัดถัดไป จะพบว่าไม่มีการแสดงค่าใดๆออกมา แต่หากดูชนิดของข้อมูลที่จัดเก็บในตัวแปร `x` ด้วย `type` จะพบว่า `x` เก็บข้อมูลชนิดที่เรียกว่า `NoneType` ซึ่งเป็นชนิดของข้อมูลที่แสดงว่าไม่มีข้อมูลอะไรภายใน เนื่องจาก `entryway` เป็น `void function` จึงไม่มีอะไร `return` กลับมาให้ `x`

ลองเปลี่ยนวิธีการเขียน Function `entryway` จาก `Void function` เป็น `Fruitful function` ด้วยการเขียนดังในรูปด้านล่าง

```
"""Entryway problem with fruitful function"""

def entryway():
    """Return a string 'Output'"""
    return 'Output'

print(entryway())
```

จะเห็นว่าใน Function `entryway` ไม่ได้มีการ `print` คำว่า `Output` ออกที่หน้าจอ แต่จะมีการคืนค่า (return) string คำว่า 'Output' กลับมา ซึ่ง string 'Output' นี้ถูกคืนกลับมานี้จะถูกส่งมาเก็บไว้ในตัวแปร หรือเป็น argument ให้กับ function `print` เพื่อพิมพ์ข้อความออกทางหน้าจอได้

ลักษณะการเรียกใช้ function `entryway` โดยอยู่ภายในวงเล็บของ function `print` แบบนี้ เราเรียกว่า `Composition`

รูปด้านล่างแสดงการใช้งาน `entryway` แบบที่เขียนเป็น `Fruitful function`

```
>>> print(entryway())
Output
>>> x = entryway()
>>> x
'Output'
>>> print(x)
Output
>>> print(entryway)
<function entryway at 0x7f80a81c31f0>
>>> type(entryway)
<class 'function'>
```

จะเห็นได้ว่า เมื่อเขียน `x = entryway()` ค่า `x` จะเก็บค่า string `'Output'` เมื่อ `print(x)` ก็จะแสดงผลคำว่า `Output` เหมือนกับเรียกด้วย `print(entryway())`

หากเราลองพยายาม `print(entryway)` โดยไม่ใส่วงเล็บด้านหลัง `entryway` จะเป็นการพิมพ์ข้อความแสดงว่า `entryway` เป็น Function และตำแหน่งที่จัดเก็บ `entryway` ในหน่วยความจำ

`0x7f80a81c31f0` คือตำแหน่งของหน่วยจำในเครื่องคอมพิวเตอร์ เขียนเป็นเลขฐาน 16 หากผู้เรียนลองทำตาม อาจจะได้ตำแหน่งที่เก็บของ `entryway` ไม่เหมือนกับในรูปด้านบน เนื่องจากระบบปฏิบัติการจะเป็นคนจัดสรรที่ว่างในหน่วยความจำให้โดยอัตโนมัติ

และหากลองใช้ `type(entryway)` ก็จะได้ผลลัพธ์คล้ายกันคือ การแสดงว่า `entryway` เป็น `Function`

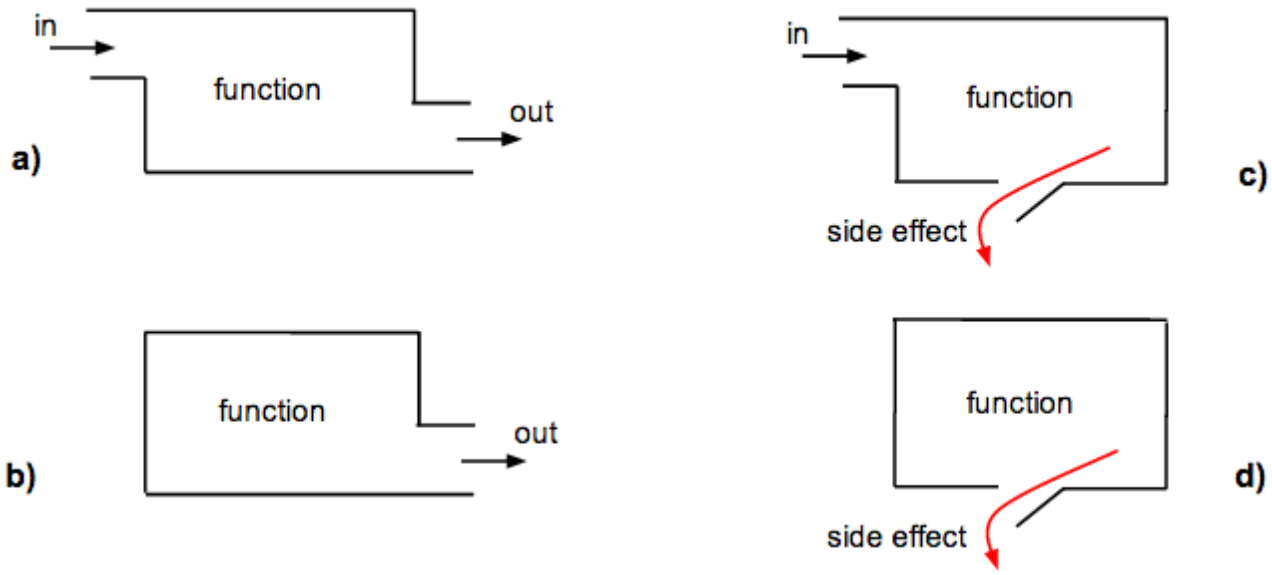
ข้อควรระวังอีกประการหนึ่งของการใช้ `return` ดังตัวอย่างในรูปด้านล่างคือ เมื่อ `entryway` ได้ return ค่า `'Output'` ไปแล้ว ก็จะเป็นการสิ้นสุดการทำงานของ `entryway` ทันที ดังนั้น `statement` ทั้งหมดที่อยู่ด้านล่างของ `entryway` ก็จะไม่ถูกเรียกใช้งานอีกต่อไป ในตัวอย่างนี้ ข้อความว่า `This will never be printed` จะไม่ถูกแสดงผลทางหน้าจอ

```
"""Entryway problem with fruitful function"""

def entryway():
    """Return a string 'Output'"""
    return 'Output'
    print('This will never be printed')

print(entryway())
```

รูปด้านล่างนี้แสดงให้เห็นว่า function สามารถแบ่งได้เป็น 4 ประเภท โดยในรูป a) และ b) จะมีการ return ค่ากลับไป (สังเกตจากที่รูปมี -> out) แสดงว่ารูป a) และ b) จัดเป็น Fruitful functions

ส่วนรูป c) และ d) ไม่มีการ return ค่าใดๆ แต่มี side effect เกิดขึ้น เช่นอาจจะมีการเปลี่ยนแปลงค่าตัวแปรบางตัว หรืออาจจะมีการแสดงผล (print) ออกทางหน้าจอ โดยไม่ได้มีการ return ค่าใดๆ ดังนั้นรูป c) และ d) จะถูกจัดเป็น Void functions

กรณีรูป a) และ b) จะแตกต่างกันที่มีการรับ input (in) เข้ามาผ่านทาง parameter ของ function หรือไม่ เช่นเดียวกับรูป c) และ d)



อ้างอิงรูปจาก https://cs.wellesley.edu/~cs110/reading/L16/

รูปด้านล่าง แสดงตัวอย่างการเขียน Function `stilljumping` ในโจทย์ข้อ `StillJumping` ในบทที่แล้ว ด้วยการเขียนแบบเป็น `Void function` ในตัวอย่าง การนิยาม `stilljumping` จะไม่มีการกำหนด `parameter` เลย (จำนวน `parameter` เท่ากับ 0) ในตัวอย่างนี้เทียบเท่ากับแบบ d) เพราะไม่มีการ return ค่าใดๆกลับไป แต่มี side effect คือมีการ print(word) ใน function

```
"""StillJumping problem with void function"""

def stilljumping():
    """Get a string and print that string on the screen"""
    word = input()
    print(word)

stilljumping()
```

รูปด้านล่าง แสดง Function `stiljumping` โดยเขียนเป็น `Void function` เช่นเดียวกับรูปที่แล้ว แต่ว่าในรูปนี้ Function `stilljumping` ถูกนิยามโดยมีการสร้าง `parameter` 1 ตัว ชื่อว่า `word` ในตัวอย่างนี้เทียบเท่ากับแบบ c) เพราะไม่มีการ return ค่าใดๆ และมี side effect โดยการ print(word)

```
"""StillJumping problem with void function"""

def stilljumping(word):
    """Print a string word on the screen"""
    print(word)

stilljumping(input())
```

บรรทัดล่างสุดในตัวอย่างนี้จะเห็นว่า ผลลัพธ์ของ Function `input()` ที่ return กลับมา จะถูกส่งไปเป็น `argument` ให้กับ Function `stilljumping` โดยตัวแปร `word` ซึ่งเป็น `parameter` ของ `stilljumping` จะรับค่าไปเก็บ เช่น หาก

`input()` รับค่าเข้ามาเป็น `'hello'` ค่า string `'hello'` ก็จะเป็น `argument` ถูกส่งไปยัง Function `stilljumping` ที่นิยามไว้ด้านบน และค่า `word` ก็จะเก็บค่า `'hello'` ด้วยเช่นกัน

เนื่องจาก Function `input()` มีการคืนค่ากลับไป เพื่อเป็น `argument` ใน `stilljumping` ดังนั้น `input()` ก็จัดอยู่ในประเภท `Fruitful function`

รูปด้านล่าง แสดง Function `stiljumping` โดยเขียนเป็น `Fruitful function` โดย `stilljumping` จะ return ค่าที่ได้จาก `input()` กลับไปให้กับ `print()` เพื่อแสดงผลออกทางหน้าจอ ในตัวอย่างนี้เทียบเท่ากับแบบ b) เพราะมีการ return ค่ากลับไป แต่ไม่มีการรับค่าเข้ามาใน function ผ่านทาง parameter เลย (จำนวน parameter = 0)

```
"""StillJumping problem with fruitful function"""

def stilljumping():
    """Return a string from input()"""
    return input()

print(stilljumping())
```

ทั้ง 3 แบบข้างบนของ `stilljumping` จะสามารถจัดได้ว่า แบบ d) และ c) และ b) ตามลำดับ โดยทั้ง 3 แบบนั้นได้ผลลัพธ์ที่เหมือนกัน คือมีการรับข้อความที่รับเข้ามาจาก `input()` และแสดงข้อความนั้นออกมาที่หน้าจอ ดังนั้นจะเขียนแบบไหนส่งขึ้น eJudge ก็ได้

Exercise 2 (Create Functions)

โจทย์แบบฝึกหัดในบทที่แล้ว จะมีการสร้างตัวแปรในบางข้อ เนื่องจากเรายังไม่สร้าง Function ขึ้นมาเอง ดังนั้นการสร้างตัวแปรเหล่านั้นจะถือว่าอยู่ภายนอก function ทั้งหมด เราจะเรียกตัวแปรที่อยู่ภายนอก function ว่า `Global variables` ซึ่งในการเขียนโปรแกรมที่ดี ควรหลีกเลี่ยงการสร้างตัวแปรไว้ภายนอก function ถ้าจะมีได้ ก็อาจจะเป็นตัวแปรเพื่อไว้เก็บค่าคงที่ หรือที่เราเรียกว่า `constant` เท่านั้น ดังนั้นเมื่อตรวจสอบคุณภาพ Quality ของโปรแกรม ด้วย PEP8 จะทำให้โปรแกรมในแบบฝึกหัดบางข้อในบทที่ผ่านมา ไม่เต็ม 100%

ให้ผู้เรียนกลับไปแก้ไขโปรแกรมตั้งแต่ข้อ `Entryway` จนถึงข้อ `Regulation` ในบทที่แล้ว โดยให้มีการสร้าง Function ขึ้นมา โดยอาจจะสร้างขึ้นเป็นแบบ `Void function` หรือ `Fruitful function` ก็ได้ (ควรลองทั้ง 2 แบบ) และส่งขึ้น eJudge ใหม่ทั้งหมด โดยให้มี Quality ครบ 100% ทุกข้อ

เนื่องจากข้อ `stilljumping` ได้แสดงให้ดูการเขียนแบบ d) และ c) และ b) ในรูปด้านบนแล้ว ข้อ `stilljumping` ให้ผู้เรียนทดลองเขียน function `stilljumping` โดยเขียนเป็นแบบ a)

หากมี Quality ไม่ครบ 100% เนื่องจาก การเขียนไม่เป็นไปตามที่ PEP8 กำหนด ให้ click ที่ `How to improve your code` เพื่อตรวจสอบข้อความแนะนำ และทำการแก้ไขจนกว่าจะได้ Quality 100% ทุกข้อ

Exercise 3 (Train)

ให้ผู้เรียนทำข้อ `Jumping` ใน eJudge โดยให้สร้าง Function ชื่อ `train()` เพื่อแสดงผลตามที่โจทย์ต้องการ

Hint

1. ข้อมูลชนิด string สามารถนำมาต่อกัน (เรียกว่า `Contatenation`) โดยใช้ operator `+` ได้ ยกตัวอย่างเช่น `'Hello' + 'World'` จะได้ผลลัพธ์เป็น `'HelloWorld'` หากต้องการให้ผลลัพธ์เป็น `'Hello World'`

กล่าวคือมีการเว้นวรรคระหว่าง `'Hello'` กับ `'World'` สามารถทำได้หลายวิธีเช่น `'Hello' + ' ' + 'World'` เป็นต้น

2. ข้อมูลชนิด string สามารถนำมาคูณกับตัวเลข Integer ได้ ยกตัวอย่างเช่น `'Hello'*3` จะได้ผลลัพธ์เป็น `'HelloHelloHello'` และเราสามารถสลับที่การคูณได้ เช่น `3*'Hello'` ก็ได้ผลลัพธ์เป็น `'HelloHelloHello'` เช่นกัน

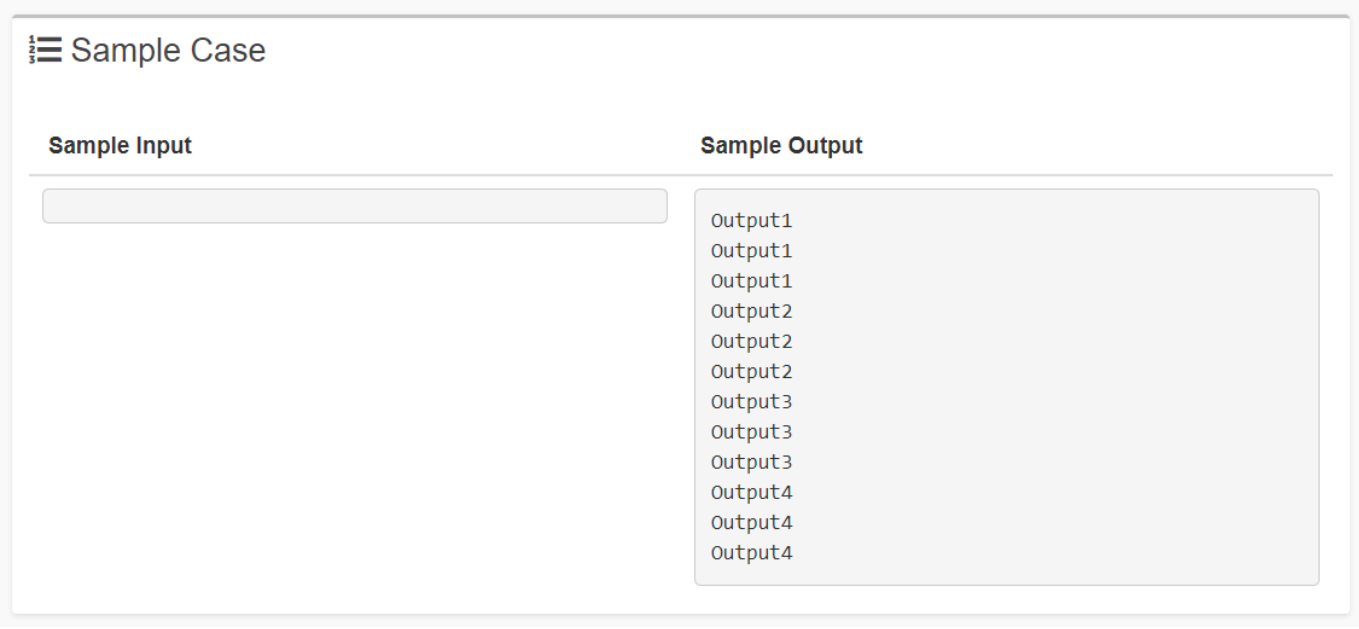
Exercise 4 (Jumping)

ให้ผู้เรียนทำข้อ `Jumping` ใน eJudge โดยให้สร้าง Function ทั้งหมด 2 function ดังนี้

1. Function ชื่อ `jumping()`
2. Function ชื่อ `print_output(num)`

โดย `jumping` จะทำการเรียก (call function) `print_output(num)` จำนวน 4 ครั้ง แต่ละครั้งให้ส่งเลข 1, 2, 3, 4 ไป (`num` จะมีค่าเป็น 1, 2, 3, 4 ในแต่ละครั้งที่เรียกไป)

และใน `print_output(num)` แต่ละครั้ง ก็จะพิมพ์คำว่า `'Output'` และตัวเลขตามหลัง ออกทางหน้าจอ จำนวน 3 บรรทัด ดังตัวอย่างที่อยู่ใน Sample Case



Sample Case

Sample Input	Sample Output
	Output1 Output1 Output1 Output2 Output2 Output2 Output3 Output3 Output3 Output4 Output4 Output4

Exercise 5 (F2C)

ให้ผู้เรียนทำข้อ `F2C` ใน eJudge

Exercise 6 (Rectangle)

ให้ผู้เรียนทำข้อ `Rectangle` ใน eJudge

Exercise 7 (BMI)

ให้ผู้เรียนทำข้อ `BMI` ใน eJudge

Exercise 8 (CompositeFunction)

ให้ผู้เรียนทำข้อ `CompositeFunction` ใน eJudge

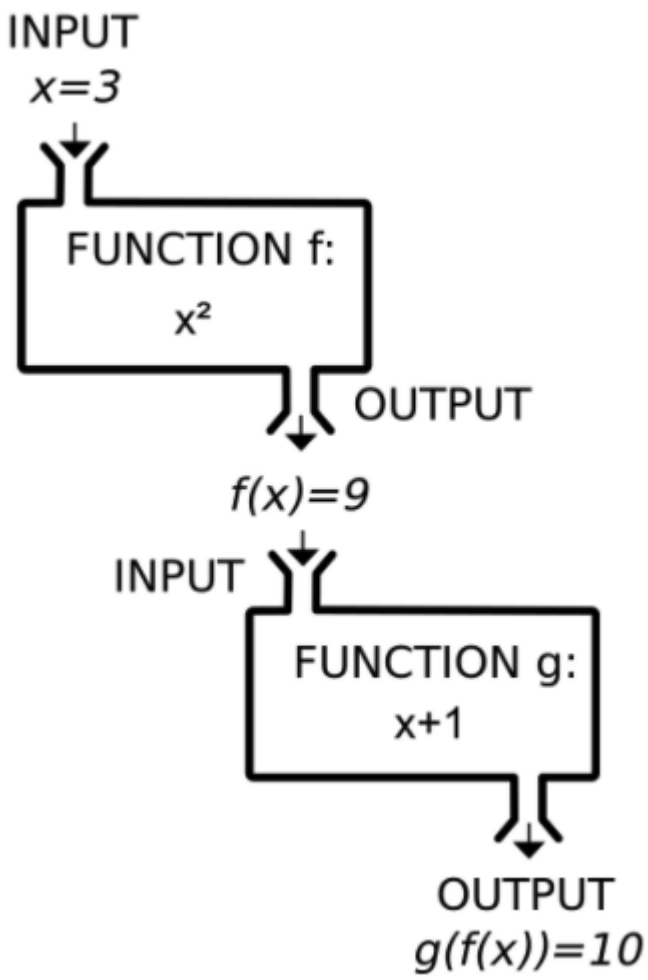
Hint

`Composite Function` หรือที่เรียกว่าฟังก์ชั่นประกอบ เป็นการสร้าง `Function` จาก `Function` อื่นๆ (จำนวนมากกว่าหรือ 2 `Function`) มาประกอบ กัน (`Chained`) ยกตัวอย่างเช่น มี 2 Function ได้แก่

1. Function y = f(x) หรือเขียนได้อย่างหนึ่งได้เป็น f: x->y (หมายความว่า function ชื่อ f โดยมี input เป็น x และ output เป็น y)
2. Function z = g(x) หรือเขียนได้อย่างหนึ่งได้เป็น g: x->z (หมายความว่า function ชื่อ g โดยมี input เป็น x และ output เป็น z)

เราสามารถสร้าง `Composite Function` หรือฟังก์ชั่นประกอบ จาก 2 Function ข้างบนได้

ดังตัวอย่างในรูปด้านล่าง `Composite Function` ที่สร้างขึ้นมาจะเรียกว่า `g o f` โดยที่ `(g o f)(x) = g(f(x))` กล่าวคือ Ouput ของ `Function f(x)` จะเป็น Input ให้กับ `Function g(y)` ในลำดับต่อไป ในรูปด้านล่างนี้ `f(x) = x**2` เมื่อ x = 3 จะได้ค่า Output หรือค่า `f(x) = 9` หรือ `y = 9` โดยค่า `y = 9` นี้จะเป็น Input หรือค่า x ให้กับ Function `g(x) = x + 1` และจะให้ผลลัพธ์ `g(f(x))` มีค่าเท่ากับ 10



อ้างอิงรูปจาก https://en.wikipedia.org/wiki/Function_(mathematics)

นอกเหนือจาก Function `g o f` ในตัวอย่างด้านบนแล้ว เราสามารถสร้าง `Function f o g` ได้เช่นกัน ในกรณีนี้หาก `x = 3` จะได้ค่า `f o g` หรือ `f(g(x))` มีค่าเท่ากับ 16

Exercise 9 (TheFunctionWithin)

ให้ผู้เรียนทำข้อ `TheFunctionWithin` ใน eJudge

Exercise 10 (EuclideanDistance2D)

ให้ผู้เรียนทำข้อ `EuclideanDistance2D` ใน eJudge โดยให้ทดลองสร้าง Function โดยไม่ใช้ Function `hypot()`