

w02-Lab1

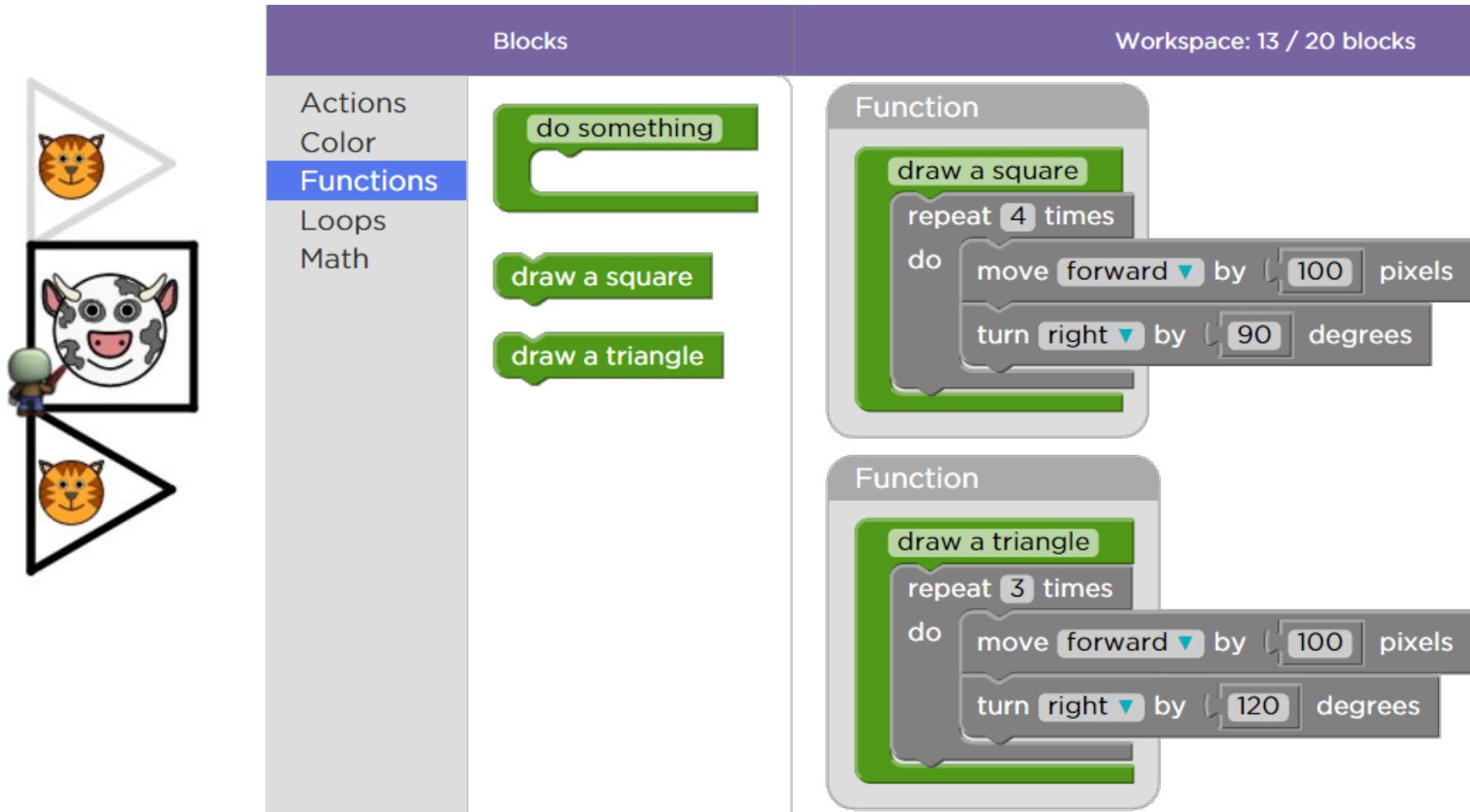
Functions

Part I

for 204111

by Kittipitch Kuptavanich

The Artist (Revisited)



The image shows a Scratch workspace with a drawing of a cow and two triangles. The code editor on the right contains two functions: 'draw a square' and 'draw a triangle'.

Workspace: 13 / 20 blocks

Blocks:

- Actions
- Color
- Functions**
- Loops
- Math

Function: draw a square

```
repeat 4 times
do
  move forward by 100 pixels
  turn right by 90 degrees
```

Function: draw a triangle

```
repeat 3 times
do
  move forward by 100 pixels
  turn right by 120 degrees
```

What is a Function?

- ในการเขียนโปรแกรม ฟังก์ชัน คือ ชุดคำสั่ง ที่มีการ กำหนดชื่อ เพื่อทำหน้าที่ย่างใดอย่างหนึ่ง (หรือมากกว่า) เช่น

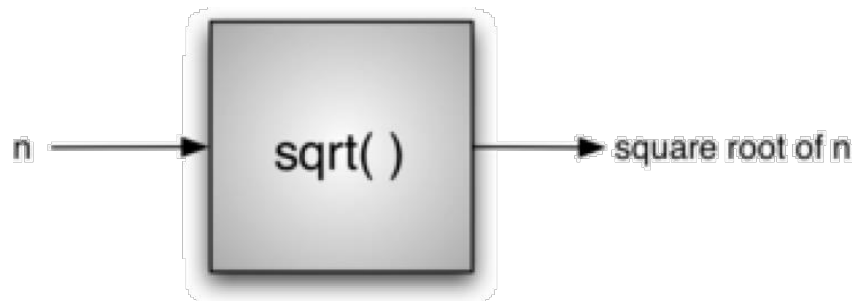
```
>>> type(32)
<class 'int'>
```

- ในกรณีนี้ ชื่อของฟังก์ชันคือ **type**
- Expression ที่อยู่ในวงเล็บ (ตัวเลข 32) เรียกว่า **Argument**
- ผลที่ได้ (result) ของการเรียกใช้ฟังก์ชัน **type** คือ ชนิด ของ **Argument** ในที่นี้คือตัวเลข **32**
- สรุป
 - ฟังก์ชันรับค่า **Argument**
 - ฟังก์ชันคืนค่า **Result**

Guideline ในการตั้งชื่อฟังก์ชันใน Python คือใช้ตัวอักษรพิมพ์เล็กทั้งหมด (สามารถคั่นระหว่างคำด้วย Underscore)

What is a Function? [2]

- จากตัวอย่างในฟังก์ชัน `type()` หรือ `draw_a_rectangle()` จะเห็นได้ว่า
 - ในบางกรณี เราไม่จำเป็นต้องทราบถึงกระบวนการที่เกิดขึ้นภายในฟังก์ชัน (**Black Box View**)
 - ทราบแค่ชื่อฟังก์ชันและ
 - วิธีใช้ (ต้องการ **Argument** อะไร และ คืนค่าอะไร)



What is a Function? [3]

- และในบางกรณีในฐานะโปรแกรมเมอร์เราจำเป็นต้องสร้างฟังก์ชันขึ้นเอง เพื่อเรียกใช้ในภายหลัง
 - ต้องกำหนดการรับค่า และการคืนค่า
 - ต้องเข้าใจกระบวนการที่เกิดขึ้นภายใน

Python Built-in Functions

<code>abs()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>all()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>any()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>ascii()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bin()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>bool()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	
<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>	

Type Conversion Functions

- เราสามารถใชฟังก์ชันเพื่อเปลี่ยนชนิดข้อมูลได้ โดยฟังก์ชันจะคืนค่าเป็นชนิดข้อมูลใหม่

```
>>> int('32')
32
>>> int('hello')
ValueError: invalid literal for int() with base 10: 'hello'
```

- `int()` สามารถรับค่า **Argument** เป็นตัวเลขจำนวนจริง (Floating-point) ได้ แต่การเปลี่ยนค่าจะเป็นการปัดเศษทิ้ง ทุกกรณี e.g. `int(10.8) = 10` `int(-10.8) = -10`
- `float()` และ `str()` เปลี่ยนค่าให้เป็น float และ string ตามลำดับ

```
108 // 10 == 10
-108 // 10 == -11
```

Math Functions

- นอกจาก **Function Built-in** เบื้องต้นแล้ว **Python** ยังมี **Function** อื่น ๆ ให้เรียกใช้โดยมีการจัดกลุ่มไว้เป็น **Module**
- จะต้องมีการ **import Module** นั้น ๆ เสียก่อน เช่น โดยสามารถ **import** ทั้ง **Module** หรือเฉพาะฟังก์ชัน หรือ **Class** ได้

```
>>> import math
>>> print(math)
<module 'math' (built-in)>
>>> math.sqrt(3)
1.7320508075688772
```

```
05 from math import sqrt
06
07 print(sqrt(2))
```


Math Functions [2]

```
math.ceil(x)
```

```
math.fabs(x)
```

```
math.factorial(x)
```

```
math.floor(x)
```

```
math.trunc(x)
```

```
math.exp(x)
```

```
math.log(x[, base])
```

```
math.log2(x)
```

```
math.log10(x)
```

```
math.pow(x, y)
```

```
math.sqrt(x)
```

Math Functions [3]

```
math.acos(x)
```

```
math.asin(x)
```

```
math.atan(x)
```

```
math.cos(x)
```

```
math.sin(x)
```

```
math.tan(x)
```

```
math.degrees(x)
```

```
math.radians(x)
```

```
math.pi
```

```
math.e
```

What Does a Module Export?

```
>>> import math
>>> dir(math)
['__doc__', '__loader__', '__name__', '__package__', '__spec__',
'acos', 'acosh', 'asin', 'asinh', 'atan', 'atan2', 'atanh',
'ceil', 'copysign', 'cos', 'cosh', 'degrees', 'e', 'erf', 'erfc',
'exp', 'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'hypot', 'isfinite', 'isinf', 'isnan', 'ldexp',
'lgamma', 'log', 'log10', 'log1p', 'log2', 'modf', 'pi', 'pow',
'radians', 'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
```

```
>>> help(math.factorial)
```

Help on built-in function factorial in module math:

```
factorial(...)
    factorial(x) -> Integral
```

Find x!. Raise a ValueError if x is negative or non-integral.

```
>>> help(math) จะแสดง
รายละเอียดของทุกฟังก์ชันใน Module
```

The `print()` Function

- โดยปกติแล้วฟังก์ชัน `print()` จะเพิ่มอักขระพิเศษคือ Newline Character (`\n`) - ขึ้นบรรทัดใหม่ หลังทุกข้อความที่แสดง

```
# script hello.py
print("hello")
print("Jon Snow")
```



```
$ python3 hello.py
Hello
Jon Snow
```

- หากต้องการให้แสดงข้อความจากฟังก์ชัน `print()` หลาย ๆ ข้อความในบรรทัดเดียวกัน สามารถทำได้โดยการระบุ Argument `end=""` เมื่อเรียกใช้ฟังก์ชัน เช่น

```
# script hello.py
print("hello", end="")
print("Jon Snow")
```



```
$ python3 hello.py
HelloJon Snow
```



`end` ถือเป็น Argument แบบพิเศษ คือจะระบุหรือไม่ระบุก็ได้เมื่อมีการเรียกใช้ฟังก์ชัน เราเรียก Argument ลักษณะนี้ว่า Optional Argument

The `print()` Function [2]

- หากต้องการคั่นระหว่าง Output ของฟังก์ชัน `print()` ด้วย Space หรืออักขระอื่น ๆ เราสามารถระบุได้ด้วย Argument `end` เช่นกันดังแสดงด้านล่าง

```
# script hello.py
print("hello", end="**")
print("Jon Snow")
```



```
$ python3 hello.py
Hello**Jon Snow
```

- ในลักษณะเดียวกันกับ Argument `end` ฟังก์ชัน `print()` ใช้ Argument `sep` เพื่อระบุอักขระที่ใช้แยกระหว่าง Argument

```
# script number.py
print(1, 2, 3)
print(1, 2, 3, sep="")
print(1, 2, 3, sep="**")
```



```
$ python3 numbers.py
1 2 3
123
1**2**3
```

Escape Characters

Escape Sequence	Meaning	Notes
<code>\\</code>	Backslash (<code>\</code>)	
<code>\'</code>	Single quote (<code>'</code>)	
<code>\"</code>	Double quote (<code>"</code>)	
<code>\a</code>	ASCII Bell (BEL)	
<code>\b</code>	ASCII Backspace (BS)	
<code>\f</code>	ASCII Formfeed (FF)	
<code>\n</code>	ASCII Linefeed (LF)	
<code>\r</code>	ASCII Carriage Return (CR)	
<code>\t</code>	ASCII Horizontal Tab (TAB)	
<code>\v</code>	ASCII Vertical Tab (VT)	
<code>\ooo</code>	Character with octal value <i>ooo</i>	
<code>\xhh</code>	Character with hex value <i>hh</i>	

The `print()` Function [3]

- เราสามารถใช้ Method (เมธอด) `str.format()` (Method เป็นชื่อใช้เรียกฟังก์ชันประเภทหนึ่ง) ร่วมกับฟังก์ชัน `print()` เพื่อจัดรูปแบบการแสดงผลได้

```
>>> print('{0} and {1}'.format('spam', 'eggs'))
spam and eggs
>>> print('{1} and {0}'.format('spam', 'eggs'))
eggs and spam
```



- ตัวเลขในวงเล็บปีกกาแทนตำแหน่งของ **Argument** ของ `format()` โดยเริ่มนับจาก 0 เสมอ

The `print()` Function [4]

- การจัดรูปแบบการแสดงผลอื่น ๆ สามารถทำได้โดยใช้เครื่องหมาย : (colon)

```
>>> print('PI is approximately {0:.3f}'.format(math.pi))
PI is approximately 3.142.
```

จัดการการแสดงผล

- `.3f` เป็นการระบุทศนิยม 3 ตำแหน่ง

```
>>> print('PI is approximately {0:09.3f}'.format(math.pi))
PI is approximately 00003.142.
```

- `09` เป็นการระบุจำนวนอักขระทั้งหมดที่ต้องการแสดง รวมจุดทศนิยม หากไม่ครบให้เติม 0 นำหน้า
- หากเปลี่ยน `09` เป็น `#9` จะเติมช่องว่าง (อักขระ Space) จนครบ 9 หลักแทน

More `format()` Examples

- Aligning the text and specifying a width:

```
>>> '{:<30}'.format('left aligned')
'left aligned'
>>> '{:>30}'.format('right aligned')
'right aligned'
>>> '{:^30}'.format('centered')
'centered'
# use '*' as a fill char
>>> '{:*^30}'.format('centered')
'*****centered*****'
```

More `format()` Examples [2]

- Replacing `%+f`, `%-f`, and `%f` and specifying a sign:

```
# specify decimal point
>>> '{:5.2f}; {:5.3f}'.format(3.14, -3.14)
3.14; -3.140
# show it always
>>> '{:+f}; {:+f}'.format(3.14, -3.14)
+3.140000; -3.140000
# show a space for positive numbers
>>> '{: f}; {: f}'.format(3.14, -3.14)
 3.140000; -3.140000
# show only the minus -- same as '{:f}; {:f}'
>>> '{:-f}; {:-f}'.format(3.14, -3.14)
3.140000; -3.140000
```

More `format()` Examples [3]

- Using the comma as a thousands separator:

```
>>> '{:,}'.format(1234567890)
'1,234,567,890'
```

- Expressing a percentage:

```
>>> points = 19
>>> total = 22
>>> 'Correct answers: {:.2%}'.format(points/total)
'Correct answers: 86.36%'
```

- Using type-specific formatting:

```
>>> import datetime
>>> d = datetime.datetime(2015, 7, 4, 12, 15, 58)
>>> '{:%Y-%m-%d %H:%M:%S}'.format(d)
'2015-07-04 12:15:58'
```

Formatting with the % Operator

- **Similar to C**

```
>>> s = "The %s have won %d Super Bowls" % ("Steelers", 6)
```

```
>>> print(s)
```

```
The Steelers have won 6 Super Bowls
```

```
>>> s1 = "The square root of %d is about |%.2f|" % (5, 5**0.5)
```

```
>>> print(s1)
```

```
The square root of 5 is about |2.24|
```

```
>>> s2 = "The square root of %d is about |%6.2f|" % (5, 5**0.5)
```

```
>>> print(s2)
```

```
The square root of 5 is about | 2.24|
```

```
>>> s3 = "The square root of %d is about |%06.2f|" % (5, 5**0.5)
```

```
>>> print(s3)
```

```
The square root of 5 is about |002.24|
```

Formatting with the % Operator [2]

• Conversion Flags

flag	Meaning	Note
'#'	The value conversion will use the "alternate form" (defined in next page).	
'0'	The conversion will be zero padded for numeric values.	
'-'	The converted value is left adjusted (overrides the '0' conversion if both are given).	
' '	(a space) A blank should be left before a positive number (or empty string) produced by a signed conversion.	
'+'	A sign character ('+' or '-') will precede the conversion (overrides a "space" flag).	
'%'	No argument is converted, results in a '%' character in the result	

Formatting with the % Operator [3]

• Conversion Types

type	Meaning	Note
'd', 'i'	Signed integer decimal.	
'o'	Signed octal value.	
'x', 'X'	Signed hexadecimal	
'e', 'E'	Floating point exponential	
'f', 'F'	Floating point decimal.	
'g', 'G'	Floating point (if $5 < \text{exp} < -4$) or decimal	
'c'	Single character (accepts integer or single character string).	
's'	String (converts any Python object using str()).	
'%'	No argument is converted, results in a '%' character in the result	

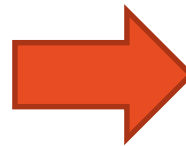
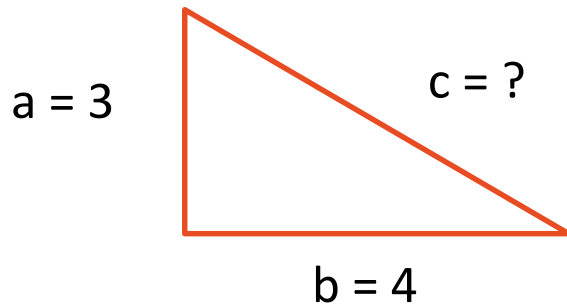
Void and Fruitful Function

- จากข้อสังเกตจะพบว่าฟังก์ชันมี 2 ประเภท คือ
 - ฟังก์ชันที่มีการคืนค่า (Non-void Function or Fruitful Function)
 - เช่น `abs()` *ตัวอย่างตัวเลข และ '=' ในการเรียนรู้*
 - ฟังก์ชันที่ไม่มีการคืนค่า (Void Function)
 - เช่น `print()` ที่ดำเนินการแต่ไม่คืนค่าอะไร
- ใน **Script Mode** ถ้าเราเรียกใช้ฟังก์ชันที่มีการคืนค่า เราจำเป็นต้องนำตัวแปรอื่น ๆ มารับค่าที่ถูกคืนมา เพื่อดำเนินการต่อ (หากไม่นำตัวแปรมารับค่า ก็จะนำค่าที่ถูกคืนมาไปใช้ต่อไม่ได้) เช่น

```
y = -45
x = abs(-45)
print("abs of {0} is {1}".format(y,x))
```

User Defined Function

- การหาด้านตรงข้ามมุมฉากของสามเหลี่ยม (Hypotenuse)



$$\begin{aligned} c^2 &= a^2 + b^2 \\ c &= \sqrt{a^2 + b^2} \\ &= (a^2 + b^2)^{\frac{1}{2}} \end{aligned}$$

- การวิเคราะห์ปัญหา

- Input:

- จำนวนข้อมูล 2 ชนิดข้อมูล int

- Output:

- จำนวนข้อมูล 1 ชนิดข้อมูล float

User Defined Function [2]

- จากตัวอย่างการคำนวณความยาวตรงข้ามมุมฉาก (3, 4, 5) เราสามารถสร้างฟังก์ชันขึ้นเพื่อใช้กับสามเหลี่ยมมุมฉากใด ๆ ได้

```
def hypotenuse(a, b):
    return ((a**2) + (b**2))**0.5
```

รับค่า

คืนค่า

- Syntax

ทำให้ง่ายกว่าของ function ส่งอะไรก็ได้ สร้างใน

```
def functionName(formal parameters):
    □□□□functionBody
```

เรียกใช้ = Argument

- Note:

- Keyword ที่ใช้ในการสร้างฟังก์ชันคือ **def**
- บรรทัดแรกเป็นการกำหนดชื่อและ พารามิเตอร์ จะต้องตามด้วย **:**
- ทุกบรรทัดของ Function Body จะต้องมีการย่อหน้า (Indent)
 - ไม่ควรใช้อักขระ Tab (**\t**)
 - Python Editor โดยมากจะใช้อักขระ Space 4 ตัวเมื่อกด Tab
- การคืนค่า ทำได้โดยใช้คำสั่ง **return**
 - ฟังก์ชันจะจบการทำงานทันทีเมื่อดำเนินการมาถึงคำสั่ง **return**

Why Function?

- ตั้งชื่อให้ชุดคำสั่ง เพื่อง่ายต่อการอ่านทำความเข้าใจและ debug
 - เช่น เมื่ออ่านชื่อฟังก์ชัน `draw_a_rectangle()` ก็จะเข้าใจได้ว่ามีไว้เพื่อวาดสี่เหลี่ยมมุมฉาก
- ชุดคำสั่งชุดเดียวสามารถใช้ได้กับหลาย ๆ กรณี (Generalization) เมื่อเขียนเป็นฟังก์ชัน
 - ทำให้โปรแกรมมีขนาดเล็กลง โดยนำชุดคำสั่งที่ซ้ำกันมาแยกไว้เป็นฟังก์ชันแล้วเรียกใช้
 - ง่ายต่อการแก้ไข (แก้ทีเดียว)

Why Function? [2]

- การแบ่งโปรแกรมขนาดใหญ่ ให้เป็นฟังก์ชันย่อย ๆ จะทำให้สามารถ debug แต่ละ ฟังก์ชันแยกกันได้ในกรณีพบข้อผิดพลาด
- ฟังก์ชันที่เขียนไว้และผ่านการทดสอบไว้อย่างดีแล้วสามารถนำไปเรียกใช้ในโปรแกรมอื่น ๆ ได้

User Defined Function [3]

- เมื่อมีการกำหนดฟังก์ชันไว้แล้วใน **script** เมื่อ **run python script** ดังกล่าว ก็จะสามารถเรียกใช้ **function** นั้น ๆ ได้

```
$ python3 -i hypotenuse.py  
>>> hypotenuse(5,12)  
13.0  
>>>
```

- ในกรณีนี้เราเรียกใช้ คำสั่ง **Python** ด้วย **flag -i** เพื่อให้ **Python** เปิด **Interactive Prompt** ค้างไว้
- หากใน **Script** มีแต่ **Function Definition** แต่ไม่มีการเรียกใช้ (**Function Call**) เมื่อ **run Script** ดังกล่าวก็จะไม่เกิดผลอะไร

Function Call

ฟังก์ชัน

- เราสามารถเรียกใช้ฟังก์ชันจากใน script ได้โดยตรง

```
01 #!/usr/bin/env python3 hypotenuse.py
02
03 def hypotenuse(a, b):
04     h = ((a ** 2) + (b ** 2)) ** 0.5
05     return h
06
07 s1 = float(input("input a: "))
08 s2 = float(input("input b: "))
09 h = hypotenuse(s1, s2)
10
11 print("hypotenuse = {0:.2f}".format(h))
```

- หากไม่มีการเรียกใช้ฟังก์ชันที่สร้างขึ้นมา เมื่อ run Script ก็จะไม่มีการดำเนินการใดๆ
- ในภาษา Python การเรียกใช้ฟังก์ชันหรือ Function Call จะต้องเกิดขึ้นหลังจาก Function Definition เสมอ

Function Call [2]

Keywords

Local

หากมี **Argument** หรือ **Parameter** มากกว่า 1 ตัว
จะต้องคั่นด้วยเครื่องหมาย **comma**
(สังเกตการใช้ **Space** ก่อนและหลัง **Comma**)

Definition
Function

Global

Call Function

```

01 #!/usr/bin/env python3
02
03 def hypotenuse(a, b):
04     h = ((a ** 2) + (b ** 2)) ** 0.5
05     return h
06
07 s1 = float(input("input a: "))
08 s2 = float(input("input b: "))
09
10 h = hypotenuse(s1, s2)
11
12 print("hypotenuse = {0:.2f}".format(h))
  
```

hypotenuse.py

formal parameter

Return Value

Argument

Function Call [3]

Python ไม่ได้จำกัดให้ใช้ชื่อ `main()`

- แต่เป็นชื่อ `function` หลักในภาษาอื่น ๆ
- การใช้ชื่อ `main()` ทำให้เข้าใจหน้าที่ของฟังก์ชันทันทีที่เห็น

- โดยปกติแล้ว ในโปรแกรมหนึ่ง ๆ จะต้องทำงานกับหลาย ๆ ฟังก์ชัน การนำคำสั่งดำเนินการหลักไว้ที่ส่วนท้ายสุดของไฟล์ อาจจะไม่สะดวกต่อการอ่านและแก้ไข

- เราสามารถนำคำสั่งดำเนินการหลักรวมไว้ในฟังก์ชัน `main()`

ด้านบน

- แล้วเรียกใช้

`main()`

ที่ส่วนล่าง

สุดของ

โปรแกรม

```

02 def main():
03     s1 = float(input("input a: ")) hypotenuse.py
04     s2 = float(input("input b: "))
05     h = hypotenuse(s1, s2)
06     print("hypotenuse = {0:.2f}".format(h))
07
08 def hypotenuse(a, b):
09     h = ((a ** 2) + (b ** 2)) ** 0.5
10     return h
11
12 if __name__ == '__main__':
13     main() # เรียกใช้ main() ใน global scope
  
```

Function Call [4]

- เพื่อการรองรับการเรียกใช้ฟังก์ชันที่เราเขียนจากไฟล์ Script อื่น เราควรใส่เงื่อนไขในการเรียกใช้ฟังก์ชัน `main()` ดังนี้เสมอ (จะอธิบายเพิ่มเติมในภายหลัง)

```
12 if __name__ == '__main__':  
13     main()
```


References

- <http://www.greenteapress.com/thinkpython/thinkCSpy/html/chap03.html>
- <http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-writing-functions-examples.html>
- <https://docs.python.org/3/tutorial/controlflow.html#defining-functions>
- <http://www.minich.com/education/wyo/stylesheets/pseudocode.htm>

References [2]

- <http://www.cs.cmu.edu/~./15110/lectures/lec4-ProgrammingPart1.pdf>
- <http://www.cs.cmu.edu/~./15110/lectures/lec5-ProgrammingPart2.pdf>
- <https://docs.python.org/3/tutorial/inputoutput.html>
- <https://docs.python.org/3/library/stdtypes.html#old-string-formatting>
- <https://docs.python.org/3.4/library/functions.html>