

w03-Lec2

# Functions Part I

Assembled for 204111  
by Kittipitch Kuptavanich

## What is a Function?

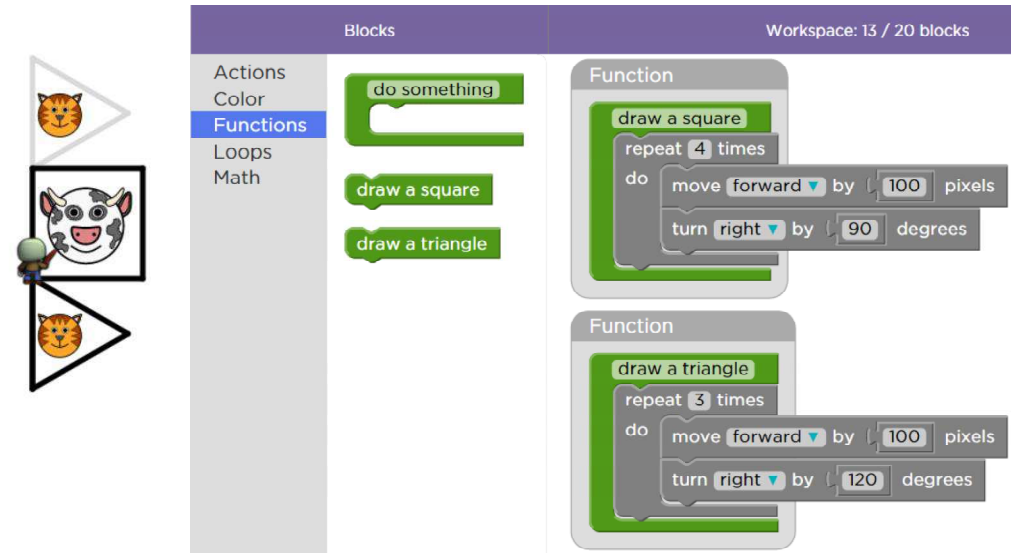
- ในการเขียนโปรแกรม ฟังก์ชัน คือชุดคำสั่งที่มีการกำหนดชื่อ เพื่อทำหน้าที่ย่างใดอย่างหนึ่ง (หรือมากกว่า) เช่น

```
>>> 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)

## The Artist (Revisited)

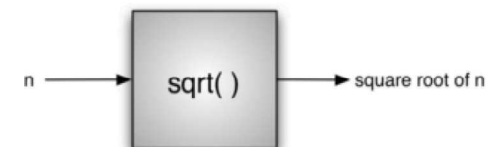


Think Python: How to Think Like a Computer Scientist

2

## 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

## Type Conversion Functions

- เราสามารถใช้ฟังก์ชันเพื่อเปลี่ยนชนิดข้อมูลได้ ฟังก์ชันจะคืนค่าเป็นชนิดข้อมูลใหม่

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

- `int()` สามารถรับค่า Argument เป็นตัวเลขจำนวนจริง (Floating-point) ได้ แต่การเปลี่ยนค่าจะเป็นการปัดเศษทิ้งทุกกรณี e.g. `int(10.8) = _____` `int(-10.8) = _____`
- `float()` และ `str()` เปลี่ยนค่าให้เป็น float และ string ตามลำดับ

# 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>	

## 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)
```

## 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
```

# 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
```

## The `print()` Function

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

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

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

- หากต้องการให้แสดงข้อความจากฟังก์ชัน `print()` หลายๆ ข้อความในบรรทัดเดียวกัน สามารถทำได้โดยการระบุ พารามิเตอร์ `end=""` เมื่อเรียกใช้ฟังก์ชัน เช่น

```
print("hello", end="")
print("Jon Snow")
```

```
$ python hello.py
HelloJon Snow
```

`end` ถือเป็น พารามิเตอร์ แบบพิเศษ คือจะระบุหรือไม่ระบุก็ได้เมื่อมีการเรียกใช้ฟังก์ชัน เราเรียก พารามิเตอร์ ลักษณะนี้ว่า **Optional Parameters**

## The `print()` function [2]

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

```
print("hello", end="**")
print("Jon Snow")
```

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

- ในลักษณะเดียวกันกับ พารามิเตอร์ `end` ฟังก์ชัน `print()` ใช้พารามิเตอร์ `sep` เพื่อระบุอักขระที่ใช้แยกระหว่างพารามิเตอร์

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

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

Think Python: How to Think Like a Computer Scientist

13

## 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

Think Python: How to Think Like a Computer Scientist

15

## Special Characters

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

Think Python: How to Think Like a Computer Scientist

14

## The `print()` Function [4]

- การจัดรูปแบบการแสดงผลอื่น ๆ สามารถทำได้โดยใช้เครื่องหมาย : (colon)

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

- `.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

Think Python: How to Think Like a Computer Scientist

16

## 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 [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'
```

## 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'
```

## 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 Type

Conversion	Meaning	Note
'd'	Signed integer decimal.	
'x'	Signed hexadecimal (lowercase).	
'f'	Floating point decimal format.	
'c'	Single character (accepts integer or single character string).	
's'	String (converts any Python object using <code>str()</code> ).	
'%'	No argument is converted, results in a '%' character in the result	

21

## Void and Fruitful Function

- จากข้อสังเกตจะพบว่าฟังก์ชันมี 2 ประเภท คือ
  - ฟังก์ชันที่มีการคืนค่า (Non-void Function or Fruitful Function)
    - เช่น `abs()`
  - 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))
```

23

## Formatting with the % Operator [3]

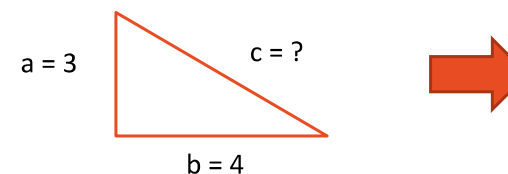
### • Conversion Type

flag	Meaning	Note
'#'	The value conversion will use the "alternate form" (where defined below).	
'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	

22

## 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 \_\_\_\_\_ ชนิดข้อมูล \_\_\_\_\_

#### • Output:

• จำนวนข้อมูล \_\_\_\_\_ ชนิดข้อมูล \_\_\_\_\_

24

## User Defined Function [2]

- จากตัวอย่างการคำนวณความยาวตรงข้ามมุมฉาก (3, 4, 5) เราสามารถสร้างฟังก์ชันขึ้นเพื่อใช้กับสามเหลี่ยมมุมฉากใดๆ ได้

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

รับค่า      คืนค่า

### Syntax

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

### 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**

25

## User Defined Function [3]

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

```
$ python -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

27

## Why Function?

- ตั้งชื่อให้ชุดคำสั่ง เพื่อง่ายต่อการอ่านทำความเข้าใจและ debug
  - เช่น เมื่ออ่านชื่อฟังก์ชัน `draw_a_rectangle()` ก็จะเข้าใจได้ว่ามีไว้เพื่อวาดสี่เหลี่ยมมุมฉาก
- ชุดคำสั่งชุดเดียวสามารถใช้ได้กับหลายๆ กรณี (Generalization) เมื่อเขียนเป็นฟังก์ชัน
  - ทำให้โปรแกรมมีขนาดเล็กลง โดยนำชุดคำสั่งที่ซ้ำกันมาแยกไว้เป็นฟังก์ชันแล้วเรียกใช้
  - ง่ายต่อการแก้ไข (แก้ทีเดียว)
- การแบ่งโปรแกรมขนาดใหญ่ ให้เป็นฟังก์ชันย่อยๆ จะทำให้สามารถ debug แต่ละ ฟังก์ชันแยกกันได้ในกรณีพบข้อผิดพลาด
- ฟังก์ชันที่เขียนไว้และผ่านการทดสอบไว้อย่างดีแล้วสามารถนำไปใช้กับโปรแกรมอื่นๆ ได้

Think Python: How to Think Like a Computer Scientist

26

## Function Call

- เราสามารถเรียกใช้ฟังก์ชันจากใน script ได้โดยตรง

```
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 = int(input("input a: "))
08 s2 = int(input("input b: "))
09 h = hypotenuse(s1, s2)
10
11 print("hypotenuse = {:.2f}".format(h))
```

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

28



## Function Call [2]

### Keywords

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

```
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 = int(input("input a: "))
08 s2 = int(input("input b: "))
09
10 h = hypotenuse(s1, s2)
11
12 print("hypotenuse = {0:.2f}".format(h))
```

Think Python: How to Think Like a Computer Scientist

29

## Top-Down Design

### เขียนฟังก์ชันจากใหญ่ไปเล็ก

Write functions top-down

- ให้สมมติว่า Helper Function (ฟังก์ชันย่อย) ต่าง ๆ เขียนเสร็จแล้ว  
Assume helper functions already exist!

### ทำการ Test Function จากเล็กไปใหญ่

Test functions bottom-up

- ไม่นำฟังก์ชันใด ๆ มาเรียกใช้ จนกว่าจะผ่านการ test อย่างถี่ถ้วน  
Do not use a function before it has been thoroughly tested
- ในทางปฏิบัติสามารถเขียน Stub Function มาใช้ชั่วคราวขณะทำ  
Top-down Design

Practicality: May help to write stubs (simulated functions as temporary placeholders in top-down design)

32

## Function Call [3]

Python ไม่ได้จำกัดให้ใช้ชื่อ `main()`  
• แต่เป็นชื่อ function หลักในภาษาอื่น ๆ  
• การใช้ชื่อ `main()` ทำให้เข้าใจหน้าที่  
ของฟังก์ชันทันทีที่เห็น

- โดยปกติแล้ว ในโปรแกรมหนึ่ง ๆ จะต้องทำงานกับหลาย ๆ ฟังก์ชัน  
การนำคำสั่งดำเนินการหลักไว้ที่ส่วนท้ายสุดของไฟล์ อาจจะไม่  
สะดวกต่อการอ่านและแก้ไข

- เราสามารถนำคำสั่งดำเนินการหลักรวมไว้ในฟังก์ชัน `main()`  
ด้านบน

- แล้วเรียกใช้  
`main()`  
ที่ส่วนล่าง

ชุดของ  
โปรแกรม

```
02 def main():
03     s1 = int(input("input a: "))
04     s2 = int(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 main() # เรียกใช้ฟังก์ชันใน global scope
```

Think Python: How to Think Like a Computer Scientist

31

## References

- <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>

33