w02-Lec2

Functions

Assembled for 204217

2015 S1

by Kittipitch Kuptavanich

What is a Function?

- ในการเขียนโปรแกรม ฟังก์ชัน คือชุดคำสั่งที่มีการกำหนดชื่อ เพื่อทำหน้าที่อย่างใดอย่างหนึ่ง (หรือมากกว่า) เช่น

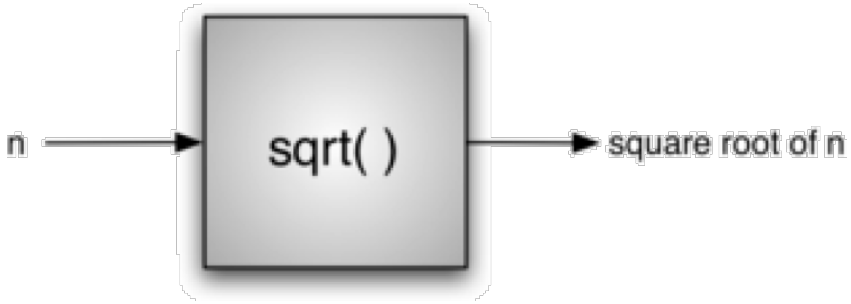
```
>>> type(32)
<class 'int'>
```

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



What is a Function? [3]

- และในบางกรณีในฐานะโปรแกรมเมอร์เราจำเป็นต้องสร้างฟังก์ชันขึ้นเอง เพื่อเรียกใช้ในภายหลัง
  - ต้องกำหนดการรับค่า และการคืนค่า
  - ต้องเข้าใจกระบวนการที่เกิดขึ้นภายใน

https://docs.python.org/3.4/library/functions.html Think Python / How to Think like a Computer Scientist

Python Built-in functions

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

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() หลาย ๆ 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
```

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 *ooo*	
\xhh	Character with hex value *hh*	

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 สามารถทำได้โดยการใช้เครื่องหมาย : (colon)

```
>>> print('PI is approximately {0:.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 0000003.142.
```

- 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 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 str()).	
'%'	No argument is converted, results in a '%' character in the result	

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	

Void and Fruitful Function

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

Why Function?

- ตั้งชื่อให้ชุดคำสั่ง เพื่อง่ายต่อการอ่านทำความเข้าใจและ debug
  - เช่น เมื่ออ่านชื่อฟังก์ชัน `draw_a_rectangle()` ก็จะเข้าใจได้ว่ามีไว้เพื่อวาดสี่เหลี่ยมมุมฉาก
- ชุดคำสั่งชุดเดียวสามารถใช้ได้กับหลาย ๆ กรณี (Generalization) เมื่อเขียนเป็นฟังก์ชัน
  - ทำให้โปรแกรมมีขนาดเล็กลง โดยนำชุดคำสั่งที่ซ้ำกันมาแยกไว้เป็นฟังก์ชันแล้วเรียกใช้
  - ง่ายต่อการแก้ไข (แก้ที่เดียว)
- การแบ่งโปรแกรมขนาดใหญ่ ให้เป็นฟังก์ชันย่อย ๆ จะทำให้สามารถ debug แต่ละ ฟังก์ชันแยกกันได้ในกรณีพบข้อผิดพลาด
- ฟังก์ชันที่เขียนไว้และผ่านการทดสอบไว้อย่างดีแล้วสามารถนำไปใช้กับโปรแกรมอื่น ๆ ได้

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))
```

Keyword Arguments and Default Value

- **In python there are two ways for formal parameters to get bound to actual parameters**
  - **Positional (ใช้ตำแหน่ง):**
    - **First formal parameter is bound to the first actual parameter, the 2nd formal to the 2nd actual, and so on**

```
max(5, 2)
```

    - **x is bound to 5**
    - **y is bound to 2**

```
def max(x, y):
  if x > y:
    return x
  else:
    return y
```

Keyword Arguments and Default Value [2]

- **Keyword Arguments (ใช้ Keyword):**
  - **Binding using the name of the formal parameters**

```
max(y=5, x=2)
```

  - **x is bound to 2**
  - **y is bound to 5**

```
def max(x, y):
  if x > y:
    return x
  else:
    return y
```

Keyword Arguments and Default Value [3]

- **The most useful form is to specifying a default value for one or more arguments**
- **Example:**

```
01 def ask_ok(prompt, retries=4, complaint='Yes or no, please!'):
02     while True:
03         ok = input(prompt)
04         if ok in ('y', 'ye', 'yes'):
05             return True
06         if ok in ('n', 'no', 'nop', 'nope'):
07             return False
08         retries = retries - 1
09         if retries < 0:
10             raise OSError('uncooperative user')
11         print(complaint)
```

สังเกตการใช้ **in** เพื่อตรวจสอบ ค่าที่สนใจ ว่าอยู่ในกลุ่มใด ๆ หรือไม่

Keyword Arguments and Default Value [4]

- **So we can call this function with**

```
03 # giving only the mandatory argument:
04 ask_ok('Do you really want to quit?')
05
06 # giving one of the optional arguments:
07 ask_ok('OK to overwrite the file?', 2)
08
09 # or even giving all arguments:
10 ask_ok('OK to overwrite the file?', 2,
11         'Come on, only yes or no!')
```

http://www.kosbie.net/cmu/spring-13/15-112

Keyword Arguments and Default Value [5]

```
01 def parrot(voltage, state='a stiff', action='voom',
02               type='Norwegian Blue'):
03     functionBody
```

- **The function accepts one required argument (voltage) and three optional arguments (state, action, and type).**
- **Invalid Calls**

```
# required argument missing
>>> parrot()
# non-keyword argument after a keyword argument
>>> parrot(voltage=5.0, 'dead')
# duplicate value for the same argument
>>> parrot(110, voltage=220)
# unknown keyword argument
>>> parrot(actor='John Cleese')
```

http://www.kosbie.net/cmu/spring-13/15-112

Keyword Arguments and Default Value [6]

- **The default values are evaluated at the point of function definition in the defining scope, so that**

```
01 i = 5
02
03 def f(arg=i):
04     print(arg)
05
06 i = 6
07 f()
```

Will print

```
>>>
5
```

Variable Scope

```
04 def f(x):
05     y = 1
06     x = x + y
07     print("x = ", x)
08     return x
09
10
11 x = 3                    Global Scope
12 y = 2
13 z = f(x)
14
15 print("z = ", z)
16 print("x = ", x)
17 print("y = ", y)
18
19
```

```
>>>
x = 4
z = 4
x = 3
y = 2
```

ในการเรียกใช้ฟังก์ชัน `f()` ตัวแปร y และ x ในฟังก์ชัน `f()` เป็น **Local Variable** และมี **Scope** (หรือ **Name Space**) ของ **Variable** แค่ภายในฟังก์ชัน `f()`

การ **Assign** ค่า หรือ **Operation** ใด ๆ ในฟังก์ชัน `f()` ไม่มีผล ต่อ x และ y ที่อยู่ด้านนอก (`__main__`) เนื่องจากเป็น **Variable** คนละตัว

Variable Scope [2]

```
x = 8
def f():
    x = 5

f()
print(x)
```

```
>>>
8
```

```
def f():
    print(x)

def g():
    print(x)
    x = 1

x = 3
f()
x = 3
g()
```

ทำให้ x มี scope เป็น local ดังนั้น `print(x)` จึงเกิด Error เพราะว่า `print()` ก่อน Assign

```
>>>
3
UnboundLocalError: local variable
'x' referenced before assignment
```

For now: หลีกเลี่ยงการใช้ global variable

http://www.kosbie.net/cmu/spring-13/15-112/handouts/notes-writing-functions-examples.html

Example 1: Ones Digit

- **Problem Statement**
  - ต้องการเขียนฟังก์ชันที่รับค่าเลขจำนวนเต็มใด ๆ แล้วคืนค่า*หลักหน่วย* (ones digit) ของจำนวนนั้น ๆ
  - การวิเคราะห์ปัญหา
    - Input: (parameter)
      - จำนวนข้อมูล_____ชนิดข้อมูล_____
    - Output (return value)
      - จำนวนข้อมูล_____ชนิดข้อมูล_____
  - ก่อนที่จะเริ่มเขียนฟังก์ชัน เราจะเริ่มจากการพิจารณา **Test Case** ต่าง ๆ ก่อน ให้แน่ใจว่าเราเข้าใจหน้าที่ของฟังก์ชันที่จะเขียน

Example 1: Ones Digit [2]

- **Test Cases**
  - อะไรคือผลลัพธ์ของ `ones_digit(123)`?
    - `3 # หลักหน่วยของ 123`
  - `ones_digit(7890)`
    - `0`
  - `ones_digit(6)`
    - `6`
  - `ones_digit(-54)`
    - `4`

Example 1: Ones Digit [3]

- เราจะสร้างฟังก์ชันเพื่อทดสอบฟังก์ชัน `ones_digit()` (ที่ยังไม่ได้เขียน ณ จุดนี้)
  - เราจะสร้างฟังก์ชันชื่อ `test_ones_digit()`
  - ในขั้นตอนนี้เราจะนำ **Test Case** ที่สร้างไว้มาทดสอบโดยการใช้ฟังก์ชัน **assert()**
    - ค่าที่ส่งเป็น **argument** ใน `assert()` **function** ควรเป็นค่าที่เป็น **True**
    - มิฉะนั้นฟังก์ชันจะฟ้องโดยการแสดง ***AssertionError Exception*** (แสดงว่าเกิดความผิดพลาด)

```
03 def test_ones_digit():
04     print("Testing ones_digit... ",end='')
05     assert(ones_digit(123) == 3)
06     assert(ones_digit(7890) == 0)
07     assert(ones_digit(6) == 6)
08     assert(ones_digit(-54) == 4)
09     print("Passed all tests!")
```

Example 1: Ones Digit [4]

- **Stub Solution**
  - ในขั้นตอนนี้เราจะเขียน **Function Body** ปลอม ๆ (or "stub") ขึ้นมาซึ่งไม่ได้ทำหน้าแก้ปัญหาตามโจทย์ **แต่** ทำหน้าที่คืนค่าคำตอบปลอม ๆ
  - **Why?** เพื่อที่จะได้ลอง run ฟังก์ชันทดสอบ

```
def ones_digit(x):
  return 3                    # stub, for testing
test_ones_digit()             # actually run the test!
```

```
assert(ones_digit(7890) == 0)
AssertionError
```

Surprised?

Example 1: Ones Digit [5]

Test cases:
123
7890
6
-54

- แก้ปัญหา, ทดสอบ, ทำซ้ำ
  - ตอนนี้เหลือเพียงขั้นตอนเดียวคือการแก้ปัญหา
  - And how do we do that?
    - Quite simple: the x % y gives the remainder
    - So 1's digit is just x % 10

```
def ones_digit(x):
  return x % 10          # first attempt!
```

```
assert(ones_digit(-54) == 4)
AssertionError                              -_-
```

Example 1: onesDigit [6]

- แก้ปัญหา, ทดสอบ, ทำซ้ำ [2]
  - (อย่างน้อยก็) ผ่าน 3 Test Case แรก
  - ดูเหมือนฟังก์ชันของเราจะใช้ได้กับจำนวนบวกแต่ทำงานพลาดถ้าเป็นจำนวนลบ
  - Why? ลองพิจารณาการทำงานของ modulo

```
def ones_digit(x):
    return abs(x) % 10      # second attempt!
```

```
Testing ones_digit... Passed all tests!    ^_^
```

Top-Down Design (Recap)

- 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)

Pseudocode

- **Pseudocode หรือ รหัสเทียมคือข้อความที่เขียน ขึ้นเพื่อทำการวางโครงร่าง (Outline) ของโปรแกรม**
  - **ใช้ภาษาอังกฤษ (หรือ ภาษาธรรมชาติ – Non Formal) ไม่มีรูปแบบตายตัว – แต่มี Notation (เครื่องหมาย หรือสัญลักษณ์) กลางเพื่อให้ผู้อ่านส่วนมากเข้าใจ**
- **Why Pseudocode?**
  - **ใช้เพื่อออกแบบ วางแผนและอธิบาย ขั้นตอนการทำงานของโปรแกรมก่อนที่จะมีการ Code จริง**
  - **หลังจาก Code แล้ว Pseudocode จะกลายเป็น Comments**

Example 1: Sum of Integers

Example 1

- **Problem Statement:**
  - **เขียนโปรแกรม เพื่อรับจำนวนเต็มสองจำนวน จาก user แล้ว แสดงผลบวกของตัวเลขทั้งสอง**
- **Pseudocode:**
  - Prompt the user to enter the first integer
    รับจำนวนเต็มตัวแรกจาก User
  - Prompt the user to enter a second integer
    รับจำนวนเต็มตัวที่สองจาก User
  - Compute the sum of the two user inputs and save to a variable
    คำนวณผลบวกของทั้งสองจำนวน แล้วเก็บไว้ใน Variable
  - Display an output prompt that explains the answer as the sum
    แสดง Output โดยอธิบายว่าจำนวนที่จะแสดงคือผลบวก
  - Display the result
    แสดงค่าของ Variable ที่เก็บผลบวก

Example 2: Sphere Volume

- **Problem Statement** (`Lab_03_1_XXXXXXXXX.py`)
  - **เขียนโปรแกรมเพื่อรับค่าพื้นที่ผิวของทรงกลมจาก user แล้วคำนวณปริมาตรของทรงกลมนั้น**
- **การวิเคราะห์ปัญหา**
  - **Input:**
    - **จำนวนข้อมูล_____ชนิดข้อมูล_____**
  - **Output**
    - **จำนวนข้อมูล_____ชนิดข้อมูล_____**

Example 2: Sphere Volume [3]

sphere.py

```
01 #!/usr/bin/env python
02
03
04
05 def main():
06     # รับข้อมูลพื้นที่ผิวจาก user
07
08     # นำพื้นที่ผิวที่ได้มาคำนวณหารัศมี
09
10     # นำรัศมีที่คำนวณได้มาคำนวณหาปริมาตร
11
12     # แสดงปริมาตรทรงกลม
13
14
15 main()
```

Example 2: Sphere Volume [4]

sphere.py

```
01 #!/usr/bin/env python
02
03 import math
04
05 def main():
06     # รับข้อมูลพื้นที่ผิวจาก user
07     surface_area = float(input("input surface area: "))
08     # นำพื้นที่ผิวที่ได้มาคำนวณหารัศมี
09     radius = find_r_from_surface_area(surface_area)
10     # นำรัศมีที่คำนวณได้มาคำนวณหาปริมาตร
11     volume = sphere_volume(radius)
12     # แสดงปริมาตรทรงกลม
13     print("volume = {0:.2f}".format(volume))
14
15 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