Python: Functions

01418112: Fundamentals Programming Concept

เนื้อหาในสไลด์นี้ อ้างอิงจากสไลด์ของ ผศ.ดร.ชัยพร ใจแก้ว ภาควิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเกษตรศาสตร์

Agenda

- ฟังก์ชันที่เคยใช้มาแล้ว
- ประโยชน์ของฟังก์ชัน
- การนิยามฟังก์ชัน
- ส่วนประกอบของฟังก์ชัน
- Program flow เมื่อมีฟังก์ชัน
- ตัวอย่าง
- Exercise
- Scope/ขอบเขตของตัวแปร
  - Global vs. Local variables

ฟังก์ชันที่เคยใช้มาแล้ว

ฟังก์ชัน	Parameters	การทำงาน	Return value
print( )	นิพจน์ใด ๆ	แสดงค่าของนิพจน์ออกจอภาพ	-
input( )	ข้อความ	รอรับข้อมูลจากผู้ใช้	สตริงที่เก็บข้อความที่ผู้ใช้
int( )	สตริงหรือตัวเลข	เปลี่ยนเป็นจำนวนเต็ม	ค่าจำนวนเต็ม
float( )	สตริงหรือตัวเลข	เปลี่ยนเป็นจำนวนจริง	ค่าจำนวนจริง

ประโยชน์ของฟังก์ชัน

- ลดความซับซ้อนในกระบวนการแก้ปัญหา โดยแตกปัญหาใหญ่ ให้เป็นปัญหาย่อย ที่แก้ได้ในไม่กี่ขั้นตอน
- ลดการเขียน โปรแกรม ที่ซับซ้อน
- ลดรายละเอียดในส่วนของโปรแกรมหลัก ทำให้โปรแกรมมีระเบียบเข้าใจได้ง่าย
- แต่ แต่ละภาษาไม่สามารถเตรียมฟังก์ชันได้ครบถ้วนตามที่โปรแกรมเมอร์ ต้องการ
- User-defined function: ฟังก์ชันที่ผู้เขียนโปรแกรมสร้างเองได้

ตัวอย่าง: คำนวณพื้นที่สามเหลี่ยม

- เขียนโปรแกรมเพื่อคำนวณ และแสดงพื้นที่สามเหลี่ยม
  - โดยรับค่า ความยาว และ ความสูง จากผู้ใช้

```
b=int(input("Enter base: "))
h=int(input("Enter height: "))
a=b*h/2
print("Triangle area is",a)
```

คำนวณพื้นที่สามเหลี่ยม: Functions

- จะดีกว่าหรือไม่ ถ้าสามารถเขียนโปรแกรมให้อ่านเข้าใจง่าย ๆ ได้

```
b=read_base()
h=read_height()
a=compute_area(b,h)
show_area(a)
```

- นั่นคือ
  - ใช้ฟังก์ชันที่สร้างขึ้นเอง read_base( ) เพื่อรับความยาวฐาน
  - ใช้ฟังก์ชันที่สร้างขึ้นเอง read_height( ) เพื่อรับความสูง
  - ใช้ฟังก์ชันที่สร้างขึ้นเอง compute_area( ) เพื่อคำนวณ
  - ใช้ฟังก์ชันที่สร้างขึ้นเอง show_area( ) เพื่อแสดงผลลัพธ์

การนิยามฟังก์ชันในไพทอน

- ต้องเว้นช่องว่างด้านหน้า (ย่อหน้า) เพื่อระบุขอบเขตของฟังก์ชัน
- พารามิเตอร์ ไม่จำเป็นต้องมี ขึ้นอยู่กับการทำงานของฟังก์ชัน
- ฟังก์ชันไม่จำเป็นต้องคืนค่า (ไม่มีคำสั่ง return)

```
def ชื่อฟังก์ชัน(พารามิเตอร์, ...):
    ...
    ลำดับคำสั่ง กำหนดการทำงานของฟังก์ชัน
    ...
    return ค่าที่ต้องการคืนกลับ (ถ้ามี)
```

นิยาม (สร้าง) ฟังก์ชันที่ต้องการ

```
def read_base():
    b = input("Enter base: ")
    return int(b)

def read_height():
    h = input("Enter height: ")
    return int(h)

def compute_area(base,height):
    return 0.5*base*height

def show_area(area):
    print("Triangle area is", area)
```

วิเคราะห์ส่วนประกอบของฟังก์ชัน

```
def compute_area(base,height):
    return 0.5*base*height
```

- ชื่อฟังก์ชัน: compute_area( )
  - ชื่อ - มีข้อกำหนดการตั้งชื่อเช่นเดียวกับชื่อตัวแปร ไม่ควรซ้ำกับคำสั่ง (reserved words) หรือชื่อฟังก์ชันที่มีอยู่แล้วของไพทอน
- ทำหน้าที่:
  - คำนวณพื้นที่สามเหลี่ยม จาก base และ height

วิเคราะห์ส่วนประกอบของฟังก์ชัน

```
def compute_area(base,height):
    return 0.5*base*height
```

- พารามิเตอร์:
  - ค่า หรือนิพจน์ ที่ฟังก์ชัน รับเข้ามาเพื่อใช้งานในฟังก์ชัน
  - ตอนเรียกใช้ฟังก์ชัน ต้องระบุค่า หรือนิพจน์ ให้มีจำนวนเท่ากันกับจำนวนพารามิเตอร์ของฟังก์ชัน
  - ภายในฟังก์ชัน พารามิเตอร์ ถูกนำไปใช้งานได้เสมือนเป็นตัวแปรที่ชี้ไปยังค่าหรือนิพจน์ ที่ถูกกำหนดให้ตอนเรียกใช้ฟังก์ชัน เช่น
    - compute_area(32,80)
    - จะมีตัวแปร base ชี้ไปที่ค่า 32 และ height ชี้ไปที่ค่า 80
  - พารามิเตอร์เหล่านี้ จะหายไปเมื่อฟังก์ชันจบการทำงาน

วิเคราะห์ส่วนประกอบของฟังก์ชัน

```
def compute_area(base,height):
    return 0.5*base*height
```

- Returned value (ค่าที่ส่งกลับ)
  - นิพจน์ที่อยู่หลังคำสั่ง return จะถูกส่งค่ากลับไปให้กับส่วนของโปรแกรมที่มาเรียกฟังก์ชันนี้
- ในที่นี้ ค่าที่ส่งกลับ คือ
  - 0.5 * base * height
  - พื้นที่สามเหลี่ยม ที่สัมพันธ์กับ base และ height ที่รับมา

วิเคราะห์เพิ่มเติม

- วิเคราะห์ฟังก์ชันอื่น ๆ ในโปรแกรมตัวอย่าง ในลักษณะเดียวกันกับตัวอย่างการวิเคราะห์ฟังก์ชัน compute_area( )
  - ฟังก์ชันชื่ออะไร?
  - ทำหน้าที่อะไร?
  - รับข้อมูลใดเข้า (หรืออะไรคือพารามิเตอร์)?
  - ส่งข้อมูลใดกลับ?

```
def read_base():
    b = input("Enter base: ")
    return int(b)

def read_height():
    h = input("Enter height: ")
    return int(h)

def show_area(area):
    print("Triangle area is", area)
```

ข้อสังเกต

- โปรแกรมที่สมบูรณ์ คือ
- สังเกตว่า คำสั่งที่อยู่ภายในนิยามของฟังก์ชัน จะไม่ถูกดำเนินการ (รัน) ในขณะนิยาม แต่จะถกรันเมื่อฟังก์ชันถูกเรียกใช้เท่านั้น

```
Python 3.4.0 (default, Apr 11 2014,
13:05:11)
[GCC 4.8.2] on linux
 : def read_base():
..     b = input("Enter base: ")
..     return int(b)
..
=> None
 : b=read_base()
Enter base:  4
=> None
 :
```

```
def read_base():
    b = input("Enter base: ")
    return int(b)

def read_height():
    h = input("Enter height: ")
    return int(h)

def compute_area(base,height):
    return 0.5*base*height

def show_area(area):
    print("Triangle area is", area)

b=read_base()
h=read_height()
a=compute_area(b,h)
show_area(a)
```

ข้อสังเกต (อีกตัวอย่าง)

```
 : print("Hello")
 Hello
=> None
 : def sayHi():
..    print("Hi")
..
=> None
 : sayHi()
 Hi
=> None
 :
```

ฟังก์ชัน print( ) ทำงานทันที

การนิยามฟังก์ชัน sayHi( ) ฟังก์ชัน print( ) ในนี้ยังไม่ทำงาน

ต่อเมื่อเรียกใช้ฟังก์ชันsayHi( ), print( ) จึงถูกเรียกให้ทำงาน

Program Flow

read_base( ) เริ่มทำงาน จนจบฟังก์ชัน แล้วย้อนกลับไปทำงานต่อจาก 1

```
2 def read_base():
      b = input("Enter base: ")
      return int(b)

4 def read_height():
      h = input("Enter height: ")
      return int(h)

6 def compute_area(base,height):
      return 0.5*base*height

8 def show_area(area):
      print("Triangle area is", area)
```

เรียกใช้ read_base( )

```
1 b=read_base()
3 h=read_height()
5 a=compute_area(b,h)
7 show_area(a)
```

ตัวอย่าง: cm to inch

- ให้นิยามฟังก์ชัน ชื่อ cm_to_inch( )
  - รับพารามิเตอร์หนึ่งตัว คือ ความยาวเป็น cm
  - คืนค่าความยาวเป็น นิ้ว ที่สัมพันธ์กัน
  - 1 นิ้ว = 2.54 cm

```
def cm_to_inch(x):
    return x/2.54

def cm_to_inch(x):
    inch = x/2.54
    return inch
```

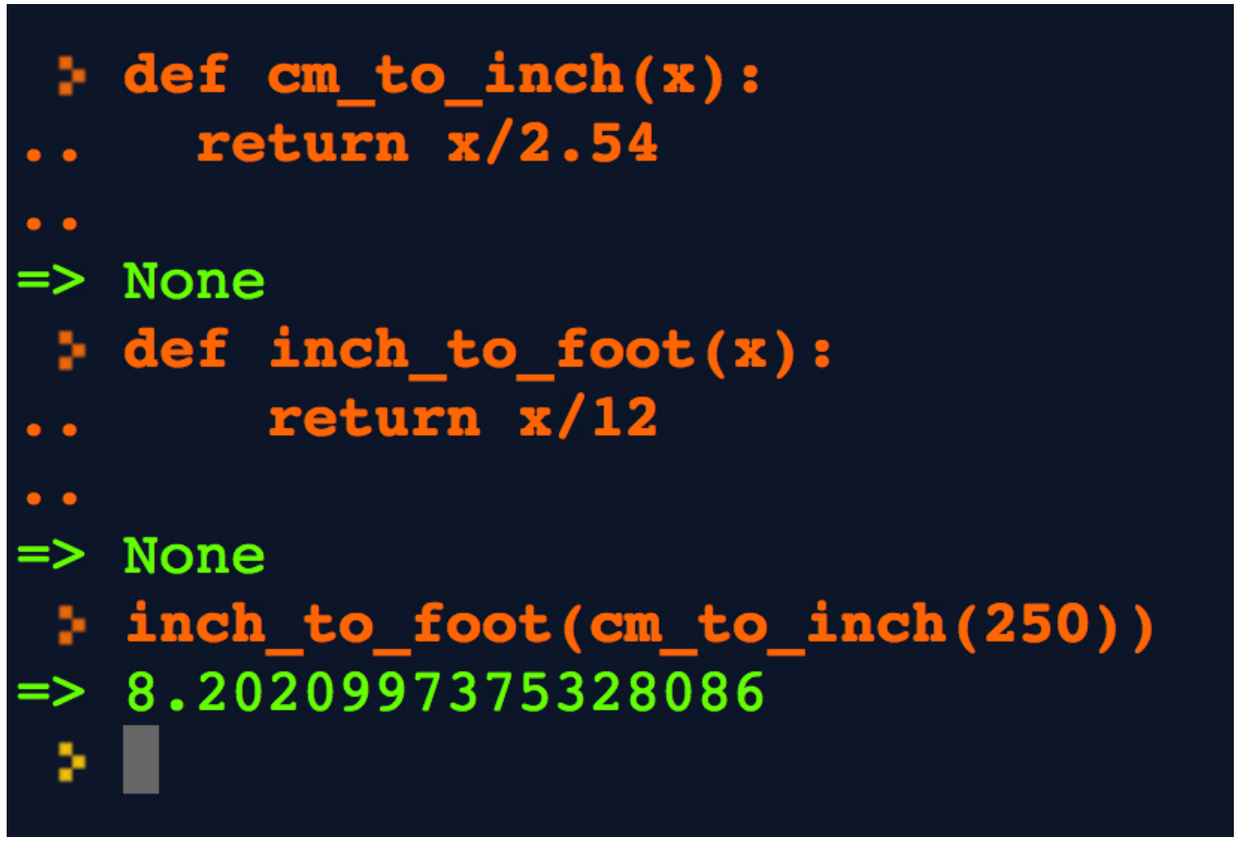
ตัวอย่าง: inch to foot

- ให้นิยามฟังก์ชัน ชื่อ inch_to_foot( )
  - รับพารามิเตอร์หนึ่งตัว คือ ความยาวเป็น inch
  - คืนค่าความยาวเป็น ฟุต ที่สัมพันธ์กัน
  - 1 ฟุต = 12 นิ้ว

```
def inch_to_foot(x):
    ...
```

ตัวอย่าง: cm to foot

- การเรียกฟังก์ชันที่มีการคืนค่า จะถูกมองเป็นนิพจน์
- นิพจน์นี้ สามารถนำไปใช้งาน หรือประกอบเป็นส่วนหนึ่ง ของนิพจน์อื่นได้ หรือนำไปเป็นพารามิเตอร์ของฟังก์ชัน ก็ได้
- ถ้าต้องการแปลง cm เป็น foot อาจเรียกใช้ฟังก์ชันซ้อนกันได้



ตัวอย่าง: cm to foot

- ให้นิยามฟังก์ชัน ชื่อ cm_to_foot( )
  - โดยต้องไปเรียกใช้ฟังก์ชัน cm_to_inch( ) และ inch_to_foot( ) ที่ได้นิยามไว้ก่อนหน้านี้แล้ว

Exercise

- ให้เขียนฟังก์ชัน ring_area(r1, r2) ทำหน้าที่หาพื้นที่ของวงแหวน รับพารามิเตอร์ 2 ตัว คือ ค่าจำนวนจริงรัศมีของวงกลมสองวงที่ประกอบกันเป็นวงแหวน แล้วส่งค่ากลับเป็นพื้นที่วงแหวน

```
import math
def ring_area(r1,r2):
    ...

rA=float(input("Enter 1st radius: "))
rB=float(input("Enter 2nd radius: "))
print("Area of the ring is",ring_area(rA,rB))
```

ขอบเขตของตัวแปร

- ตัวแปรที่ประกาศไว้ ภายในฟังก์ชัน จะมีตัวตน จากจุดที่มีการกำหนดให้ครั้งแรก จนกระทั่งฟังก์ชันจบการทำงาน เท่านั้น
  - เรียกว่า ตัวแปรแบบ**โลคอล** (Local variable)
- ตัวแปรที่ประกาศไว้ ภายนอกฟังก์ชัน จะมีตัวตน นับจากจุดที่มีการกำหนดค่าให้ครั้งแรก จนกระทั่งจบโปรแกรม
  - เรียกว่า ตัวแปรแบบ**โกลบอล** (Global variable)

Global vs. Local Variables

```
1   x = 8
2   y = 3
3
4   def myfunc():
5       y = 5
6       a = x
7
8   myfunc()
9   print(x,y)
10  print(a)
```

y นี้เป็นคนละตัวกับ y ในบรรทัดที่ 2

x นี้เป็นตัวเดียวกันกับที่นิยามใน 1

พิมพ์ 8 3 นั่นคือ y นอกฟังก์ชันไม่มีการเปลี่ยนแปลง

error เพราะ ไม่เคยกำหนดค่า a มาก่อน ในโปรแกรมหลัก

ข้อควรระวัง

```
1  x = 8
2
3  def myfunc():
4      print(x)
5
6  myfunc()
```

x ใน myfunc() ถือเป็น โกลบอล

```
8
=> None
```

```
1  x = 8
2
3  def myfunc():
4      print(x)
5      x = 3
6
7  myfunc()
```

บรรทัดนี้ ทำให้ x ใน myfunc() เป็น โลคอล

ไม่สามารถอ้างถึง x ได้ ณ บรรทัดนี้ เพราะ x ยังไม่มีตัวตน

```
Traceback (most recent call
last):
  File "python", line 7, in
<module>
  File "python", line 4, in
myfunc
UnboundLocalError: local
variable 'x' referenced before
assignment
```

คำสั่ง global

- ใช้ภายในฟังก์ชัน เพื่อระบุว่าตัวแปรดังกล่าวเป็นตัวแปรตัวเดียวกับที่ใช้นอกฟังก์ชัน

```
1   x = 8
2   y = 3
3
4   def myfunc():
5       global y
6       y = 5
7       a = x
8
9   myfunc()
10  print(x,y)
```

y นี้ เป็นตัวเดียวกับ y ในโปรแกรมหลัก

```
 8 5
=> None
```

ข้อควรระวังการใช้ global

- ควรหลีกเลี่ยงการใช้ตัวแปรโกลบอล ภายในฟังก์ชัน เนื่องจาก
  - พฤติกรรมของฟังก์ชันไม่ได้ถูกกำหนด แค่ค่าพารามิเตอร์ แต่ยังขึ้นอยู่กับค่า ของตัวแปรโกลบอลที่อ้างถึงด้วย
  - ไล่การทำงานของโปรแกรมยาก หากมี หลายฟังก์ชันที่สามารถเปลี่ยนค่าตัว แปรโกลบอลไปมาได้