Python Advanced Function

by Dr. Sethavidh Gertphol

Outline

* Immutable / Mutable Arguments
* Argument Matching
* Function as object
* Map
* Lambda

copyright Sethavidh Gertphol

Immutable / Mutable arguments

* **immutable objects:** แก้ไขข้อมูลตัวเองไม่ได้ ต้องสร้างอ๊อปเจ็คใหม่ เสมอ
  * เช่น int, float, str
* **mutable objects:** แก้ไขข้อมูลตัวเองได้
  * list, dictionary
* ส่งอ๊อปเจ็คทั้งสองแบบไปให้ฟังก์ชันได้ แต่ถ้ามีการแก้ไขในฟังก์ชัน จะแตกต่างกัน

copyright Sethavidh Gertphol

Immutable arguments

* assign ตัวแปรในฟังก์ชันเฮดเดอร์ไปที่ **immutable** อ๊อปเจ็คที่ส่งมา
* เนื่องจากแก้ไขอ๊อปเจ็คไม่ได้
  * การเปลี่ยนแปลงข้อมูลจะเป็นการสร้างอ๊อปเจ็คใหม่เสมอ
  * ผลคือ**อ๊อปเจ็คเดิมที่ส่งมาจะไม่เปลี่ยนแปลง**
* เป็นรูปแบบการส่งแบบ **pass-by-value**

copyright Sethavidh Gertphol

Mutable arguments

* assign ตัวแปรในฟังก์ชันเฮดเดอร์ไปที่ **mutable** อ๊อปเจ็คที่ส่งมาเหมือนกัน
* แต่เนื่องจากแก้ไขอ๊อปเจ็คได้
  * การเปลี่ยนแปลงข้อมูลผ่านตัวแปร**จะแก้ไขอ๊อปเจ็คเดิมด้วย**
* เป็นรูปแบบการส่งแบบ **pass-by-reference**

copyright Sethavidh Gertphol

ตัวอย่าง

```
def func(x, ls):
    x = 50
    ls[0] = 100
    print("In function: x is %s, ls is %s" % (x, ls))

a = 20
l = [10,20,30]
print("Before function: a is %s, l is %s" % (a, l))
func(a,l)
print("After function: a is %s, l is %s" % (a, l))
```

```
Before function: a is 20, l is [10, 20, 30]
In function: x is 50, ls is [100, 20, 30]
After function: a is 20, l is [100, 20, 30]
```

copyright Sethavidh Gertphol

Argument Matching

```
def func(denom, divider, adder=0):
    return denom/divider+adder
```

* Positional

```
func(10,2,3)
```

```
8.0
```

* Keyword

```
func(divider=5,adder=2,denom=20)
```

```
6.0
```

* Mix (positional must come first)

```
func(30,adder=10,divider=3)
```

```
20.0
```

copyright Sethavidh Gertphol

Default Argument

* ระบุค่า default ของ argument นั้นได้โดย assign ค่าไว้ที่ function header เลย

```
def func(denom, divider=2, adder=0):
    return denom/divider+adder
```

* ตอนเรียกฟังก์ชันจะไม่ใส่ค่าของ adder ก็ได้ ถ้าไม่ใส่จะใช้ค่า default

```
func(10,2)
```

```
5.0
```

* ถ้าใส่ค่าของ adder ก็จะใช้ค่าที่ใส่มาตามปกติ

```
func(10,2,3)
```

```
8.0
```

* ใช้ keyword argument ข้าม argument ตัวที่มี default ได้

```
func(10,adder=3)
```

```
8.0
```

copyright Sethavidh Gertphol

Varargs

* เทคนิคสำหรับส่งรับ arguments แบบไม่จำกัดจำนวน โดยใช้ * และ ** ช่วย
* * ใช้เก็บ positional arguments ลง tuples
* ** ใช้เก็บ keyword arguments ลง dictionary

copyright Sethavidh Gertphol

ตัวอย่าง (*)

```
def my_product(first, second, *rest):
    print(first, second, rest)
    result = first*second
    for x in rest:
        result *= x
    return result
```

```
my_product(1,2,3,4,5)
```

```
1 2 (3, 4, 5)
120
```

```
my_product(1,2,3)
```

```
1 2 (3,)
6
```

copyright Sethavidh Gertphol

ตัวอย่าง (**)

```
def my_product(first, second, **rest):
    print("first =",first)
    print("second =", second)
    for key in rest:
        print(key, "=",rest[key])
```

```
my_product(second=2,first=1,third=3,forth=4,fifth=5)
```

```
first = 1
second = 2
third = 3
forth = 4
fifth = 5
```

copyright Sethavidh Gertphol

ลำดับการ Match

* คร่าว ๆ

1. positional
2. keyword
3. *
4. **
5. check for missing and multiple, use default

copyright Sethavidh Gertphol

Function object

* ชื่อฟังก์ชันที่ไม่มี () และไม่มีการส่ง argument จะเป็นฟังก์ชันอ๊อปเจ็ค
  * สามารถนำไป assign ใส่ตัวแปร หรือส่งเป็น argument ก็ได้
* สามารถเรียกฟังก์ชันผ่านตัวแปรโดยใส่ () กับ argument ได้เหมือนเรียกฟังก์ชันด้วยชื่อ

copyright Sethavidh Gertphol

ตัวอย่าง

variable

```
def my_calculation(calc_fn, *rest):
    result = rest[0]
    for x in rest[1:]:
        result = calc_fn(result,x)
    return result

def my_add(a,b): return a+b
def my_mul(a,b): return a*b
```

function call through variable

```
my_calculation(my_add, 1,2,3,4,5)
```

15

```
my_calculation(my_mul, 1,2,3,4,5)
```

120

pass function object without ()

copyright Sethavidh Gertphol

Map

* ฟังก์ชัน **map** ใช้เรียกฟังก์ชันอื่น โดยส่ง **arguments** จาก **list** (หรือ **iterable** อื่น) ไปให้ทีละตัว
* ใช้ในการเรียกฟังก์ชันซ้ำ ๆ หลาย ๆ ครั้ง โดย **arguments** เปลี่ยนไปในแต่ละครั้ง
* เป็นลักษณะการทำงานแบบ **functional programming**

copyright Sethavidh Gertphol

ตัวอย่าง

* สมมติว่าถ้ามีคะแนนสอบกลางภาคและปลายภาค อยู่ใน **list** สองอัน เราจะบวกคะแนนรวมได้อย่างไร

```
midterm = [25,14,18,7]
final = [30,24,10,15]
```

ปกติ

```
result = []
for idx in range(len(midterm)):
    result.append( midterm[idx] + final[idx] )
print(result)
```

```
[55, 38, 28, 22]
```

ผลลัพธ์

map

```
def my_add(a,b): return a+b
print(list(map(my_add, midterm, final)))
```

แปลง map ให้เป็น list

ฟังก์ชัน

argument list 1

argument list 2

copyright Sethavidh Gertphol

Lambda

* lambda เป็น expression ที่ใช้สร้างฟังก์ชัน
  * เน้นการเขียนง่าย ๆ อย่างรวดเร็ว ไม่ต้อง def
  * ไม่เน้นการใช้ซ้ำ
  * มักใช้ในการส่งฟังก์ชันอ๊อปเจ็ค
  * เขียน lambda ตรงจุดที่จะใช้เลย
* รูปแบบ

```
lambda x,y : x+y
```

keyword arguments expression

copyright Sethavidh Gertphol

ตัวอย่าง

map

```
def my_add(a,b): return a+b
print(list(map(my_add, midterm, final)))
```

lambda

```
list(map(lambda x,y : x+y, midterm, final))
```

ไม่ต้องไป **def** สร้างฟังก์ชันข้างนอก สร้างเป็น **lambda** เลย

```
def my_calculation(calc_fn, *rest):
    result = rest[0]
    for x in rest[1:]:
        result = calc_fn(result,x)
    return result
```

```
my_calculation(lambda x,y : x*y, 1,2,3,4,5)
```

```
my_calculation(lambda x,y : x+y, 1,2,3,4,5)
```

copyright Sethavidh Gertphol

Reference

* Mark Lutz, "Learning Python, 5th Edition", O'Reilly Media, Inc., June 2013

copyright Sethavidh Gertphol