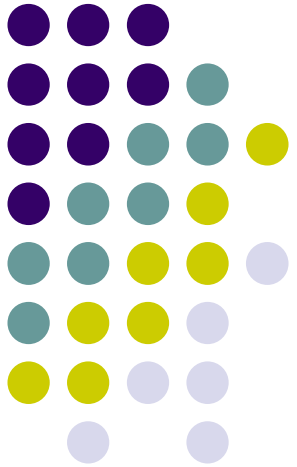
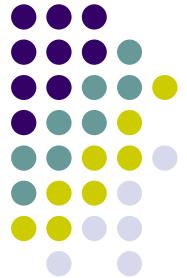
Python程式設計入門
函式(1/4)

葉難

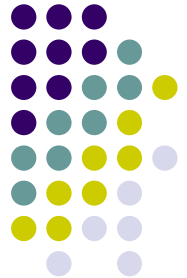


大綱



- 函式定義與呼叫，def述句和lambda運算式
- 參數傳遞
- 範圍，命名空間，環境模式
- 遞迴（recursion）
- 高階函式（higher-order function）
- 裝飾器（decorator）
- 產生器（generator）
- 函數式程式設計

函式「定義」與「呼叫（call）」



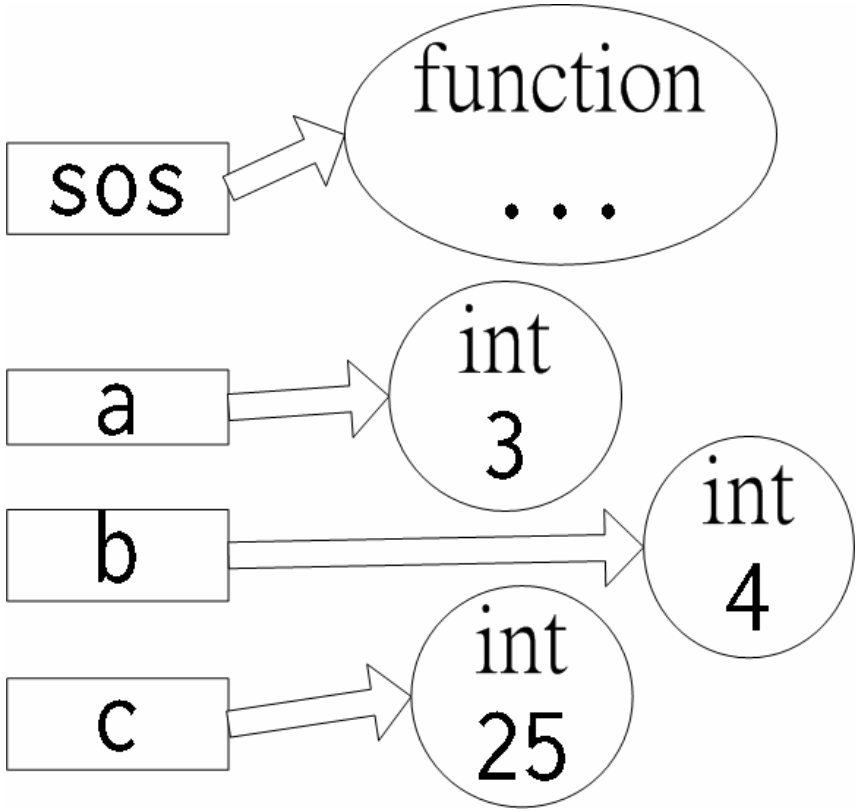
- 是兩回事，譬喻：食譜與做菜
- 呼叫機制：進入區域範圍、參數傳遞、跳躍（流程轉移）、結束返回（回傳）
- 語法

```
def 函式名稱(參數0, 參數1, ...):   # 定義
    述句
    述句...

函式名稱(參數0, 參數1, ...)          # 呼叫
```

函式也是物件

- 在Python裡，什麼都是物件

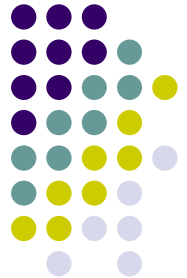
```
def sos(x, y):
    z = x**2 + y **2
    return z

a, b = 3, 4
c = sos(a, b)
```



抽象型別：可被呼叫者（Callable）

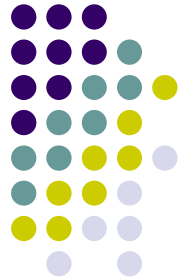
- def述句可定義函式（建立函式物件）
- 函式物件「可被呼叫」
- 但「可被呼叫」的物件不只有函式
- int、str、list：既代表型別、也可被呼叫（作爲建構式）
- 可被呼叫者：內建函式、使用者定義的函式、型別（類別）、物件的方法、等等

範例：int、int()、callable



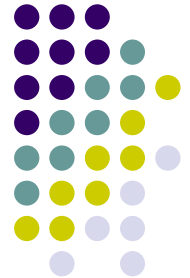
```
>>> callable(sos)       # 函式sos可被呼叫
True
>>> a = 3; type(a)      # 名稱a指向的物件，
<class 'int'>           # 型別是int
>>> type(a) is int      # 「int」是個型別
True
>>> int('0x64', 16)     # 「int」作爲建構式
100
>>> callable(int)       # 「int」可被呼叫
True
```

特殊方法__call__



```
>>> def sq(x): return x**2
...
>>> sq
<function sq at 0x00C868A0>

>>> sq(9)
81
>>> sq.__call__(9)
81
```

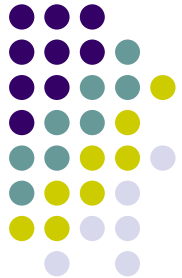
回傳



- return述句
- 函式結束時，一定會回傳某物件，若不指定則回傳None
- 只能回傳「一個」物件，若想回傳多個東西，須放在容器內

```
def foo(x, y):
    return x+y, x-y          # 回傳tuple物件
    # return (x+y, x-y)      # 此處可省略小括號
```

參數



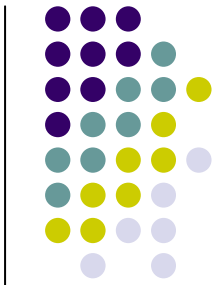
- 形式參數，實際參數
- 形參指定形式：預設值、不定個數、限制只能使用關鍵字參數
- 形參與星號「*」、雙星號「**」：收集
- 實參指定形式：位置參數、關鍵字參數
- 實參與星號「*」、雙星號「**」：解開

形式參數和實際參數

- 形參（formal argument、parameter）
- 實參（actual argument、argument）
- 參數傳遞機制，骨子裡就是指派（綁定關係）
- 呼叫：先執行「實際參數的運算式」，得到實參（物件），然後傳給（指派給）形參名稱

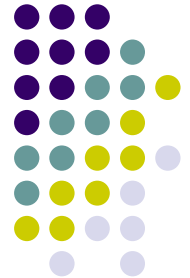
```
def sos(x, y): return x**2+y**2
sos(a+3, len([0, 1, 2])*2)
```

別名現象



```
li = [1, 2, 3, 4]     # 可變

def foo(data):
    for i, x in enumerate(data):
        data[i] = x * 2

foo(li)   # li與data指向同一個物件

# li 被修改成 [2, 4, 6, 8]
```

實參指定形式：
位置參數、關鍵字參數

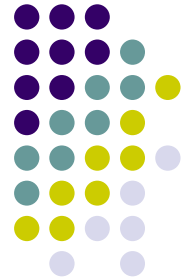


```
def f(x, y, z): print(x, y, z)

f(1, 2, 3)          # 根據位置先後，對應到x、y、z
f(1, 2)             # 出錯，個數不足

f(y=2, x=1, z=3)    # 以關鍵字指定
f(1, 2, z=3)        # 位置參數在前，關鍵字參數在後
f(1, y=2, 3)        # 出錯，模糊不清
```

形參可以設定「預設值」

```
def f(x, y, z=3): print(x, y, z)
f(1, 2, 9)          # 不使用預設值
f(1, 2)             # 使用預設值
f(1)                # 出錯，個數不足
f(y=2, x=1)         # 以關鍵字參數指定
f(y=2)              # 出錯，個數不足

def f(x, y=2, z): print(x, y, z)    # 出錯
# 預設值參數之後，都要有預設值。
#（另有特殊指定方式）
```

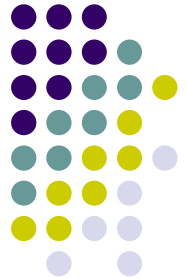
實參與
星號「*」、雙星號「**」



- 「解開」的意思
- 星號「*」解開序列型別、可迭代者
- 雙星號「**」解開映射型別、字典

```
def f(x, y, z): print(x, y, z)
li = [1, 2, 3]
f(*li)                  # 如同傳入位置參數
f(*range(1, 3+1)) # 可迭代者
d = {'x':1, 'y':2, 'z':3}
f(**d)                  # 如同傳入關鍵字參數
```

範例

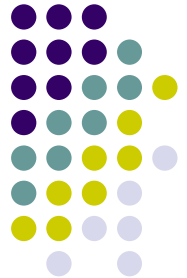


```
def f(name, age, score):
    print(name, age, score)
data = ('Amy', 25, 90)
f(*data)  # 從某處取得資料，解開、傳入

# 資料順序，不符合形參順序
data = (25, 90, 'Amy')
k = ('age', 'score', 'name')
f(**dict(zip(k, data)))
```

結合「*」與「**」

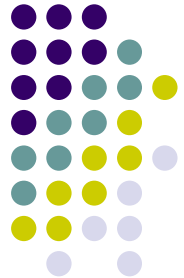
```
def f(a, b, c, d): print(a, b, c, d)

d = {'d':4, 'c':3}
f(*range(1, 2+1), **d)
f(**d, *range(1, 2+1))    # 出錯，語法錯誤

f(*range(1, 3+1), **d)    # 出錯，c得到太多值
f(**d)                    # 出錯，個數不足
d2 = {'a':1, 'b':2, 'c':3, 'd':4, 'e':5}
f(**d2)                   # 出錯，太多了
```

3.5版：增強「*」與「**」



```
def f(a, b, c, d): print(a, b, c, d)

f(*range(1, 2+1), *range(3, 4+1))

d1 = {'a':1, 'b':2, 'c':3}
d2 = {'d':4}
f(**d1, **d2)
```

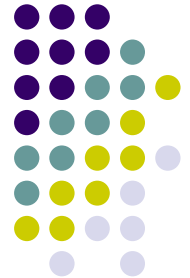
參數傳遞規則



- 如何指定預設值，如何接受與傳入不定個數的參數、如何以關鍵字指定參數、如何規定只能以關鍵字形式傳入參數

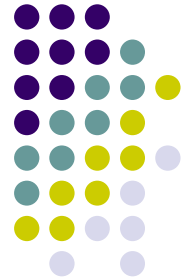
- 規則複雜，但有理可循
- 記住常用形式

參數傳遞規則（大致）



- 實參的指定順序，位置參數（先）、關鍵字參數（後），可使用「*」與「**」解開
- 若需要的話，使用預設值
- 比對形參與實參，個數不同（太多或太少）會出錯；實參形式不符合形參的規定，會出錯
- 形參可使用「*」或「**」 進行「收集」
- 「*」須在「**」之前，不論是實參或形參

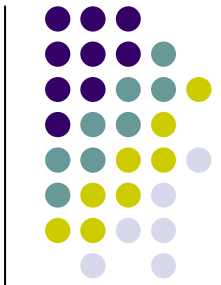
形參與
星號「*」、雙星號「**」



- 「收集」的意思
- 星號「*」收集位置參數，放進tuple
- 雙星號「**」收集關鍵字參數，放進dict

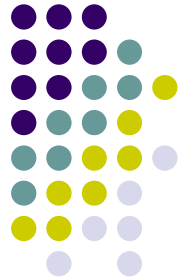
```
>>> def f(x, y, *t): print(x, y, t)
>>> f(1, 2, 3, 4, 5, 6)
1 2 (3, 4, 5, 6)
>>> def f(x, y, **d): print(x, y, d)
>>> f(1, y=2, k1=3, k2=4)
1 2 {'k1': 3, 'k2': 4}
```

範例：



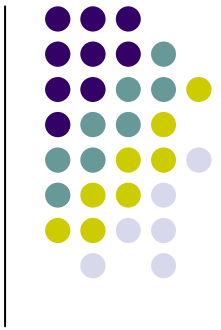
```
def avg(first, *rest):
    return (first + sum(rest)) / (1+len(rest))
avg(1)                  # 1.0
avg(1, 2)               # 1.5
avg(1, 2, 3, 4, 5, 6) # 3.5

def color(r, g, b, *, alpha=1.0): pass
color(0, 0.5, 0.5)
color(1, 0.7, 0.8, 0.5)  # 出錯
color(1, 0.7, 0.8, alpha=1.0)
```

範例



```
def sorted(iterable, *, key=None,
                        reverse=False): pass

def fruit(name, value, count, **others): pass
fruit('Apple', 30, 5,
    size='large', color='red', source='Japan')

def color(*, r, g, b, alpha=1.0, **d): pass
c1 = {'r':1, 'g':0.7, 'b':0.3}
color(**c1)
c2 = {'r':1, 'g':0.7, 'b':0.3, 'k1':'v1', 'k2':'v2'}
color(**c2)
```

語法（1/3）



```
def f(p1, p2,...pN, *t)
```

額外的實參，放進t（tuple）裡

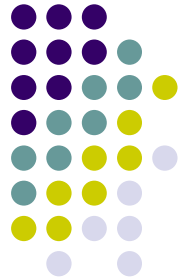
```
def f(p1, p2,...pN, *t, k1, k2,...kM) #3.x
def f(p1, p2,...pN, *, k1, k2,...kM)  #3.x
def f(*t, k1, k2,...kM)               #3.x
```

之後的參數，必須是關鍵字參數

```
def f(*, k1, k2,...kM)                #3.x
```

限定只能使用關鍵字參數

語法（2/3）



`def f(**d)`　限定使用關鍵字參數，通通放進d（dict）裡

`def f(p1, p2,...pN, **d)`

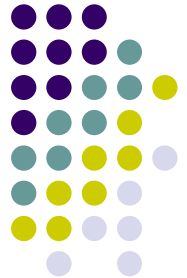
位置參數與關鍵字參數，先擺放到p1~pN，額外的關鍵字參數，放進d裡

`def f(p1, p2,...pN, *t, **d)`

額外的位置參數放進t裡，額外的關鍵字參數放進d裡

`def f(*t, **d)` 位置參數都放進t，關鍵字參數都放進d

語法（3/3）



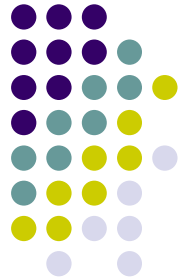
```
def f(p1, p2,...pN, *t, k1, k2,...kM, **d)#3.x
```
多的位置參數放進t，多的關鍵字參數放進d

```
def f(p1, p2,...pN, *, k1, k2,...kM, **d) #3.x
```
位置參數之後，限定使用關鍵字參數，額外的放進d

```
def f(*t, k1, k2,...kM, **d) # 3.x
```
位置參數都放進t，額外的關鍵字參數放進d

```
def f(*, k1, k2,...kM, **d)  # 3.x
```
限定只能使用關鍵字參數，額外的放進d

範例：相容舊的函式介面

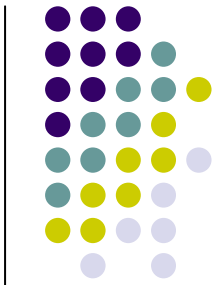


```
def f0(x, y, z): pass

def f1(x, y, z, extra=1.0): pass
def f2(x, y, z, *t): pass
def f3(x, y, z, **d): pass
def f4(x, y, z, extra=1.0, *, **d): pass
```

問題：指派述句與星號

```
>>> a, *b, c = ['Amy', 3, 4, 5, 'abc']
>>> b
[3, 4, 5]
>>> *x, = range(4)          # x會是什麼？

>>> a, b, c, d = *range(4)
>>> y = *range(4),          # 3.5版
>>> y                       # y會是什麼？
?
```

lambda運算式



- 同樣可以建立「函式物件」
- 無名（anonymous），但可指派名稱
- 裡頭只能含一個運算式，其結果就是回傳值
- 語法：

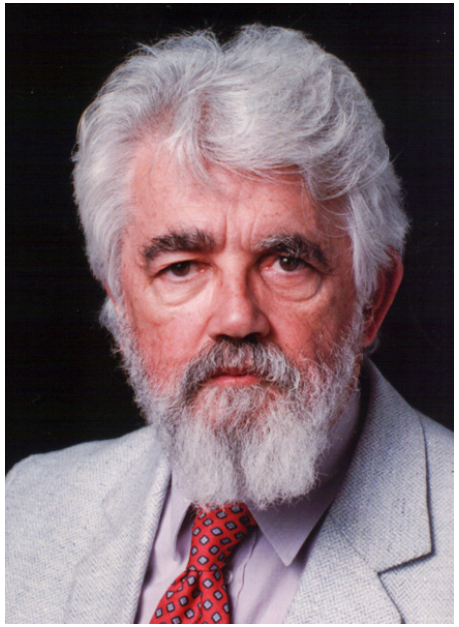
```
lambda 參數0, 參數1, ...: 運算式
```

- 例子：

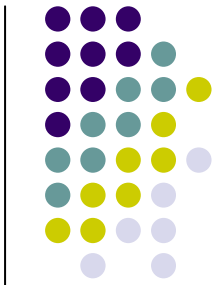
```
sq = lambda x: x ** 2
print(sq(3))                # 印出9
```

lambda之名（希臘字母λ）

- Alonzo Church的Lambda calculus理論
- John McCarthy的Lisp語言
- 其他語言也有lambda，有些類似、有些不同

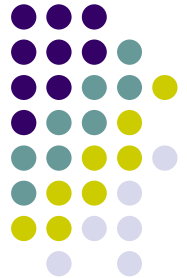


lambda是個運算式



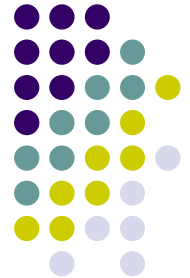
```
>>> print( (lambda x: x**2)(3) )
9                       # 定義後，立刻呼叫
>>> li = [('C', 180, 65), ('B', 160, 60), ('A', 170, 40)]
>>>        # 名字、身高、體重

>>> sorted(li)
[('A', 170, 40), ('B', 160, 60), ('C', 180, 65)]
>>> sorted(li, key=lambda x: x[1])
[('B', 160, 60), ('A', 170, 40), ('C', 180, 65)]
```

範例：sorted



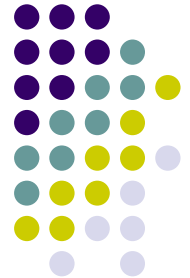
```
>>> def kf(x): return x[2]
...
>>> sorted(li, key=kf, reverse=True)
[('C', 180, 65), ('B', 160, 60), ('A', 170, 40)]

>>> sorted(li,                          # 男性標準體重
       key=lambda x: x[2] - ((x[1] - 80) * 0.7))
[('A', 170, 40), ('C', 180, 65), ('B', 160, 60)]
#  過輕 -> 標準 -> 過重
```

2.x的sorted，參數cmp



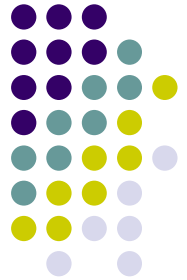
```
li = [('C', 170, 65), ('B', 160, 50), ('A', 150, 40)]
# 回傳負數代表x比y小，回傳0代表相等，
# 回傳正數代表x比y大。
def cmp_f(x, y):
    return x[1] - y[1]

print(sorted(li, cmp=cmp_f))

# 會印出
# [('A', 150, 40), ('B', 160, 50), ('C', 170, 65)]
```

可以寫的很難懂



```
(lambda x: (lambda y: print(x<y<5))(4))(3)

(lambda f: print(f(13)))(lambda x: x+5)
```

問題



```
li = [30, 41, 52, 63]

print(
 (lambda li:
  (lambda f, r, li: f(f, r, li))
   (lambda f, r, li:
     f(f, r+li[0], li[1:]) if li else r,0, li))
    (li))
```

Q&A