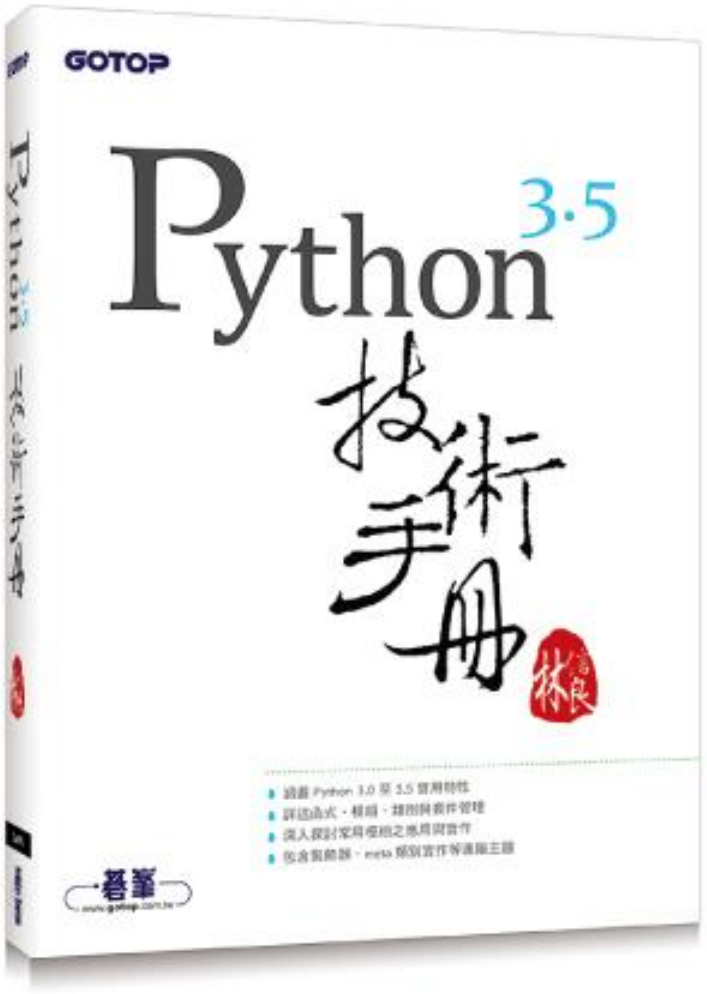


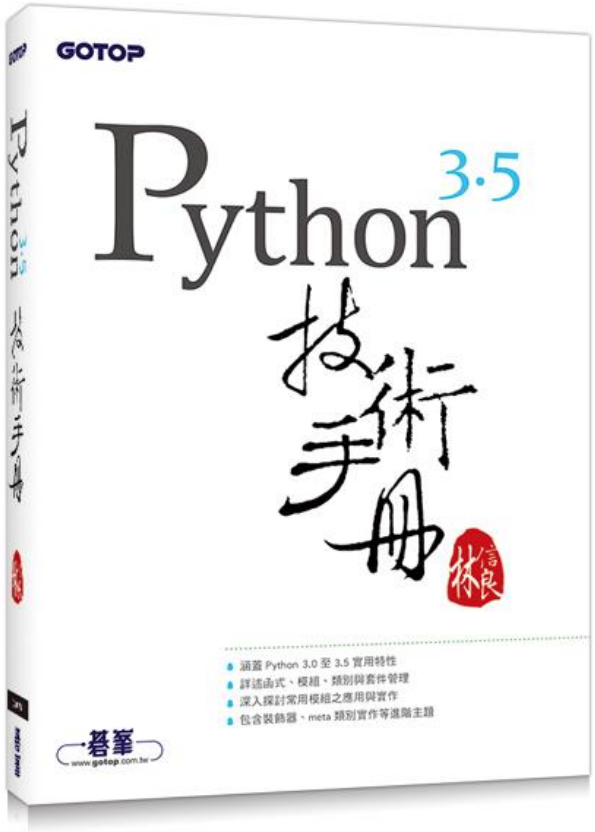
Python 3.5 技術手冊

林信良

CHAPTER

4. 流程語法與函式

- 學習目標
- 認識基本流程語法
- 使用 `for` Comprehensior
- 認識函式與變數範圍
- 運用一級函式特性
- 使用 `yield` 建立產生器



if 分支判斷

- 程式區塊使用冒號「:」開頭
- 之後同一區塊範圍要有相同的縮排
  - 不可混用不同空白數量，不可混用空白與Tab，
- Python 的建議是使用四個空白作為縮排

basic hello.py

```
import sys

name = 'Guest'
if len(sys.argv) > 1:
    name = sys.argv[1]
print('Hello, {}'.format(name))
```

- if 可以搭配 else

basic is_odd.py

```
import sys

number = int(sys.argv[1])
if number % 2:
    print('{} 爲奇數'.format(number))
else:
    print('{} 爲偶數'.format(number))
```

- if..elif..else

basic grade.py

```
score = int(input('輸入分數：'))
if score >= 90:
    print('得 A')
elif 90 > score >= 80:
    print('得 B')
elif 80 > score >= 70:
    print('得 C')
elif 70 > score >= 60:
    print('得 D')
else:
    print('不及格')
```

- if..else 運算式語法

basic is_odd2.py

```
import sys

number = int(sys.argv[1])
print('{} 爲 {}'.format(number, '奇數' if number % 2 else '偶數'))
```

while 迴圈

```
while 條件式:
    陳述句
else:
    陳述句
```

basic lucky5.py

```
import random

number = 0
while number != 5: ◀── ❶ 如果不是 5 就執行迴圈
    number = random.randint(0, 9) ◀── ❷ 隨機產生 0 到 9 的數
    print(number)
    if number == 5:
        print('我碰到 5 了....Orz')
```

- 跟 while 搭配的 else

```
>>> while False:
...     print('while')
... else:
...     print('else')
...
else
>>> while num == 0:
...     print('while')
...     num = 1
... else:
...     print('else')
...
while
else
>>>
```

- 若不想讓 else 執行，必須是 while 中因為 break 而中斷迴圈

basic gcd.py

```
print('輸入兩個數字...')

m = int(input('數字 1: '))
n = int(input('數字 2: '))

while n != 0:
    r = m % n
    m = n
    n = r
    if m == 1:
        print('互質')
        break      ◄── break 可用來中斷迴圈
else:
    print("最大公因數：", m)
```

- 建議別使用 **while** 與 **else** 的形式

basic gcd2.py

```
print('輸入兩個數字...')

m = int(input('數字 1: '))
n = int(input('數字 2: '))

while n != 0:
    r = m % n
    m = n
    n = r

if m == 1:
    print('互質')
else:
    print("最大公因數：", m)
```

for in 迭代

- 想要循序迭代某個序列

basic_uppers.py

```
import sys
for arg in sys.argv:
    print(arg.upper())
```

- 使用 range() 函式

```
>>> name = 'Justin'
>>> for i in range(len(name)):
...     print(i, name[i])
...
0 J
1 u
2 s
3 t
4 i
5 n
>>>
```

- 使用 zip() 函式

```
>>> list(zip([1, 2, 3], ['one', 'two', 'three']))
[(1, 'one'), (2, 'two'), (3, 'three')]
>>>
```

```
name = 'Justin'
for i, c in zip(range(len(name)), name):
    print(i, c)
```

- 使用 enumerate() 函式

```
>>> name = 'Justin'
>>> list(enumerate(name))
[(0, 'J'), (1, 'u'), (2, 's'), (3, 't'), (4, 'i'), (5, 'n')]
>>>
```

- 迭代時具有索引資訊

```
name = 'Justin'
for i, c in enumerate(name):
    print(i, c)
```

- 預設 enumerate() 會從 0 開始計數

```
name = 'Justin'
for i, c in enumerate(name, 1):
    print(i, c)
```

- set 也實作了__iter__()方法，可以進行迭代
- 想要迭代 dict 鍵值的話，可以使用它的 keys()、values()或 items()方法

```
>>> passwds = {'Justin' : 123456, 'Monica' : 54321}
>>> for name, passwd in passwds.items():
...     print(name, passwd)
...
Justin 123456
Monica 54321
>>>
```

- for in 也有個與 else 配對的形式
- **建議別使用 for in...else 的形式**

basic is_prime.py

```
number = int(input('輸入數字：'))
half = number // 2
for num in range(2, half + 1):
    if number % num == 0:
        print(number, '不是質數')
        break    ← break 可用來中斷迭代
else:
    print(number, '是質數')
```

pass、break、continue

- pass 就真的是 pass，什麼都不做

```
if is_prime:
    print('找到質數')
else:
    pass
```

- break 可用來中斷 while 迴圈、for in 的迭代
- 在 while 迴圈中遇到 continue 的話，此次不執行後續程式碼，直接進行下次迴圈

basic show_uppers.py

```
text = input('輸入一個字串：')
for letter in text:
    if letter.isupper():
        continue
    print(letter, end='')
```

```
>python show_uppers.py
輸入一個字串：This is a Question!
his is a uestion!
```

for Comprehension

```
import sys

squares = []
for arg in sys.argv[1:]:
    squares.append(int(arg) ** 2)
print(squares)
```

basic_square.py

```
import sys

squares = [int(arg) ** 2 for arg in sys.argv[1:]]
print(squares)
```

```
>python square.py 10 20 30
[100, 400, 900]
```

```
import sys

odds = []
for arg in sys.argv[1:]:
    if int(arg) % 2:
        odds.append(arg)
print(odds)
```

basic_odds.py

```
import sys

odds = [arg for arg in sys.argv[1:] if int(arg) % 2]
print(odds)
```

```
>python odds.py 11 8 9 5 4 6 3 2
['11', '9', '5', '3']
```

- 巢狀結構也是可行，不過建議別太過火

```
>>> matrix = [
...     [1, 2, 3],
...     [4, 5, 6],
...     [7, 8, 9]
... ]
>>> array = [element for row in matrix for element in row]
>>> array
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>>
```

- 取得兩個序列的排列組合：

```
>>> [letter1 + letter2 for letter1 in 'Justin' for letter2 in 'momor']
['Jm', 'Jo', 'Jm', 'Jo', 'Jr', 'um', 'uo', 'um', 'uo', 'ur', 'sm', 'so', 'sm',
'so', 'sr', 'tm', 'to', 'tm', 'to', 'tr', 'im', 'io', 'im', 'io', 'ir', 'nm',
'no
', 'nm', 'no', 'nr']
>>>
```

- 在 `for` Comprehension 兩旁放上 `()`，這樣的話就會建立一個 `generator` 物件，具有惰性求值特性
  - `sum([n for n in range(1, 10001)])`
  - `sum(n for n in range(1, 10001))`
  - `g = (n for n in range(1, 10001))`

- 也可以用來建立 set

```
>>> text = 'Your Right brain has nothing Left. Your Left brain has nothing
Right'
>>> {c for c in text if c.isupper()}
{'Y', 'R', 'L'}
>>>
```

- 建立 dict 實例

```
>>> names = ['Justin', 'Monica', 'Irene']
>>> passwds = [123456, 654321, 13579]
>>> {name : passwd for name, passwd in zip(names, passwds)}
{'Justin': 123456, 'Irene': 13579, 'Monica': 654321}
>>>
```

- 建立 `tuple`
- 將 `for` Comprehension 產生器運算式傳給 `tuple()`。

```
>>> tuple(n for n in range(10))
(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
>>>
```

使用 def 定義函式

- 發現到程式中...

```
# 其他程式片段...
max1 = a if a > b else b
# 其他程式片段...
max2 = x if x > y else y
# 其他程式片段...
```

- 定義函式

```
def max(num1, num2):
    return num1 if num1 > num2 else num2
```

- 原先的程式片段就可以修改為：

```
max1 = max(a, b)
# 其他程式片段...
max2 = max(x, y)
# 其他程式片段...
```

- 函式是一種抽象，對流程的抽象
- 函式也可以呼叫自身，這稱之為遞迴（Recursion）

func_gcd.py

```
def gcd(m, n):
    if n == 0:
        return m
    else:
        return gcd(n, m % n)

print('輸入兩個數字...')

m = int(input('數字 1: '))
n = int(input('數字 2: '))

r = gcd(m, n)
if r == 1:
    print('互質')
else:
    print("最大公因數：", r)
```

- 區域函式（Local function）

func sele_sort.py

```
import sys

def sele_sort(number):
    # 找出未排序中最小值
    def min_index(left, right):
        if right == len(number):
            return left
        elif number[right] < number[left]:
            return min(right, right + 1)
        else:
            return min(left, right + 1)

    for i in range(len(number)):
        selected = min_index(i, i + 1)
        if i != selected:
            number[i], number[selected] = number[selected], number[i]

number = [int(arg) for arg in sys.argv[1:]]
sele_sort(number)
print(number)
```

參數與引數

- 不支援函式重載（Overload）

```
>>> def sum(a, b):
...     return a + b
...
>>> def sum(a, b, c):
...     return a + b + c
...
>>> sum(1, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: sum() missing 1 required positional argument: 'c'
>>>
```

- 在 Python 中可以使用預設引數來有限度地模仿函式重載

```
def account(name, number, balance = 100):
    return {'name' : name, 'number' : number, 'balance' : balance}

# 顯示 {'name': 'Justin', 'balance': 100, 'number': '123-4567'}
print(account('Justin', '123-4567'))
# 顯示 {'name': 'Monica', 'balance': 1000, 'number': '765-4321'}
print(account('Monica', '765-4321', 1000))
```

- 在執行到 def 時，就會依定義建立了相關的資源

```
>>> def prepend(elem, lt = []):
...     lt.insert(0, elem)
...     return lt
...
>>> prepend(10)
[10]
>>> prepend(10, [20, 30, 40])
[10, 20, 30, 40]
>>> prepend(20)
[20, 10]
>>>
```

- 可以將 prepend() 的 lt 參數預設值設為 None，並在函式中指定真正的預設值

```
>>> def prepend(elem, lt = None):
...     lt = lt if lt else []
...     lt.insert(0, elem)
...     return lt
...
>>> prepend(10)
[10]
>>> prepend(10, [20, 30, 40])
[10, 20, 30, 40]
>>> prepend(20)
[20]
>>>
```

- 可以指定參數名稱來設定其引數值，稱之為關鍵字引數

```
def account(name, number, balance):
    return {'name' : name, 'number' : number, 'balance' : balance}

# 顯示 {'name': 'Monica', 'balance': 1000, 'number': '765-4321'}
print(account(balance = 1000, name = 'Monica', number = '765-4321'))
```

- *與**

```
def account(name, number, balance):
    return {'name' : name, 'number' : number, 'balance' : balance}

# 顯示 {'name': 'Justin', 'balance': 1000, 'number': '123-4567'}
print(account(*('Justin', '123-4567', 1000)))


def sum(*numbers):
    total = 0
    for number in numbers:
        total += number
    return total

print(sum(1, 2))         # 顯示 3
print(sum(1, 2, 3))      # 顯示 6
print(sum(1, 2, 3, 4))   # 顯示 10
```

```
def account(name, number, balance):
    return {'name' : name, 'number' : number, 'balance' : balance}

params = {'name' : 'Justin', 'number' : '123-4567', 'balance' : 1000}
# 顯示 {'name': 'Justin', 'balance': 1000, 'number': '123-4567'}
print(account(**params))


def ajax(url, **user_settings):
    settings = {
        'method' : user_settings.get('method', 'GET'),
        'contents' : user_settings.get('contents', ''),
        'datatype' : user_settings.get('datatype', 'text/plain'),
        # 其他設定 ...
    }
    print('請求 {}'.format(url))
    print('設定 {}'.format(settings))

ajax('http://openhome.cc', method = 'POST', contents = 'book=python')
my_settings = {'method' : 'POST', 'contents' : 'book=python'}
ajax('http://openhome.cc', **my_settings)
```

- 可以在一個函式中，同時使用*與**
- 如果想要設計一個函式接受任意引數，就可以加以運用

```
>>> def some(*arg1, **arg2):
...     print(arg1)
...     print(arg2)
...
>>> some(1, 2, 3)
(1, 2, 3)
{}
>>> some(a = 1, b = 22, c = 3)
()
{'a': 1, 'c': 3, 'b': 22}
>>> some(2, a = 1, b = 22, c = 3)
(2,)
{'a': 1, 'c': 3, 'b': 22}
>>>
```

一級函式的運用

- 函式不單只是個定義，還是個值，是 function 的實例

```
>>> def max(num1, num2):
...     return num1 if num1 > num2 else num2
...
>>> maximum = max
>>> maximum(10, 5)
10
>>> type(max)
<class 'function'>
>>>
```

- 函式跟數值、`list`、`set`、`dict`、`tuple`等一樣，都被 Python 視為一級公民來對待
- 可以自由地在變數、函式呼叫時指定，也被稱一級函式（First-class function）
- 函式代表著某個可重用流程的封裝，這表示可以將某個可重用流程進行傳遞

- 過濾出字串長度大於 6 的元素：

```
lt = ['Justin', 'caterpillar', 'openhome']
result = []
for elem in lt:
    if len(elem) > 6:
        result.append(elem)
print(result)
```

- 可能會多次進行這類的比較，因此定義出函式，以重用這個流程：

```
def len_greater_than_6(lt):
    result = []
    for elem in lt:
        if len(elem) > 6:
            result.append(elem)
    return result

lt = ['Justin', 'caterpillar', 'openhome']
print(len_greater_than_6(lt))
```

func_filter_demo.py

```
def filter_lt(predicate, lt):
    result = []
    for elem in lt:
        if predicate(elem):
            result.append(elem)
    return result

def len_greater_than_6(elem):
    return len(elem) > 6

def len_less_than_5(elem):
    return len(elem) < 5

def has_i(elem):
    return 'i' in elem

lt = ['Justin', 'caterpillar', 'openhome']
print('大於 6：', filter_lt(len_greater_than_6, lt))
print('小於 5：', filter_lt(len_less_than_5, lt))
print('有個 i：', filter_lt(has_i, lt))
```

func filter_demo2.py

```
def filter_lt(predicate, lt):
    result = []
    for elem in lt:
        if predicate(elem):
            result.append(elem)
    return result

def len_greater_than(num):
    def len_greater_than_num(elem):
        return len(elem) > num
    return len_greater_than_num

lt = ['Justin', 'caterpillar', 'openhome']
print('大於 5：', filter_lt(len_greater_than(5), lt))
print('大於 7：', filter_lt(len_greater_than(7), lt))
```

- 想將元素全部轉為大寫後傳回新的清單

```
lt = ['Justin', 'caterpillar', 'openhome']
result = []
for ele in lt:
    result.append(ele.upper())
print(result)
```

func map_demo.py

```
def map_lt(mapper, lt):
    result = []
    for ele in lt:
        result.append(mapper(ele))
    return result

lt = ['Justin', 'caterpillar', 'openhome']
print(map_lt(str.upper, lt))
print(map_lt(len, lt))
```

- Python 就內建有 `filter()`、`map()` 函式可以直接取用
- 傳回的實例並不是 `list`，分別是 `map` 與 `filter` 物件

func filter_map_demo.py

```
def len_greater_than(num):
    def len_greater_than_num(elem):
        return len(elem) > num
    return len_greater_than_num

lt = ['Justin', 'caterpillar', 'openhome']
print(list(filter(len_greater_than(6), lt)))
print(list(map(len, lt)))
```

- 有時會想將其中的元素進行排序

```
>>> sorted([2, 1, 3, 6, 5])
[1, 2, 3, 5, 6]
>>> sorted([2, 1, 3, 6, 5], reverse = True)
[6, 5, 3, 2, 1]
>>> sorted(('Justin', 'openhome', 'momor'), key = len)
['momor', 'Justin', 'openhome']
>>> sorted(('Justin', 'openhome', 'momor'), key = len, reverse = True)
['openhome', 'Justin', 'momor']
>>>
```

- `sorted()` 會傳回新的 `list`，其中包含了排序後的結果
- `key` 參數可用來指定針對什麼特性來迭代

- 如果是可變動的 `list`，本身也有個 `sort()` 方法，這個方法會直接在 `list` 本身排序

```
>>> lt = [2, 1, 3, 6, 5]
>>> lt.sort()
>>> lt
[1, 2, 3, 5, 6]
>>> lt.sort(reverse = True)
>>> lt
[6, 5, 3, 2, 1]
>>> names = ["Justin", "openhome", "momor"]
>>> names.sort(key = len)
>>> names
['momor', 'Justin', 'openhome']
>>>
```

lambda 運算式

- 本體很簡單，只有一句簡單的運算，對於這類情況，可以考慮使用 lambda 運算式

func filter_demo3.py

```
def filter_lt(predicate, lt):
    result = []
    for elem in lt:
        if predicate(elem):
            result.append(elem)
    return result

lt = ['Justin', 'caterpillar', 'openhome']
print('大於 6：', filter_lt(lambda elem: len(elem) > 6, lt))
print('小於 5：', filter_lt(lambda elem: len(elem) < 5, lt))
print('有個 i：', filter_lt(lambda elem: 'i' in elem, lt))
```

- 若需要兩個以上的參數，中間要使用逗號「,」區隔

```
>>> max = lambda n1, n2: n1 if n1 > n2 else n2
>>> max(10, 5)
10
>>>
```

- 結合 dict 與 lambda 來模擬 switch

func grade.py

```
score = int(input('請輸入分數：'))
level = score // 10
{
    10 : lambda: print('Perfect'),
    9  : lambda: print('A'),
    8  : lambda: print('B'),
    7  : lambda: print('C'),
    6  : lambda: print('D')
}.get(level, lambda: print('E'))()
```

初探變數範圍

- 一個名稱在指定值時，就可以成為變數，並建立起自己的作用範圍（Scope）
- 在取用一個變數時，會看看目前範圍中是否有指定的變數名稱，若無則向外尋找

```
>>> x = 10
>>> def func():
...     print(x)
...
>>> func()
10
>>>
```

- 如果在 func() 中，對名稱 x 作了指定值的動作呢？

```
>>> x = 10
>>> def func():
...     x = 20
...     print(x)
...
>>> func()
20
>>> print(x)
10
>>>
```

- 變數可以在內建、全域、外包函式、區域函式中尋找或建立

func_scope_demo.py

```
x = 10                          # 建立全域 x

def outer():
    y = 20                      # 建立全域 y

    def inner():
        z = 30                  # 建立全域 z
        print('x = ', x)        # 取用全域 x
        print('y = ', y)        # 取用 outer()函式的 y
        print('z = ', z)        # 取用 inner()函式的 z

    inner()

    print('x = ', x)            # 取用全域 x
    print('y = ', y)            # 取用 outer()函式的 y
outer()
print('x = ', x)                # 取用全域 x
```

- Python 中的全域，實際上是以模組檔案為
- `builtins` 中的名稱範圍，橫跨各個模組

```
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError',
'ImportWarning', 'IndentationError', 'IndexError', 'InterruptedError',
'IsADirectoryError', 'KeyError', …略
```

- `locals()` 函式可用來查詢區域變數名稱與值。

func scope_demo2.py

```
x = 10

def outer():
    y = 20

    def inner():
        z = 30
        print('inner locals:', locals())

    inner()
    print('outer locals:', locals())

outer()
```

```
inner locals: {'z': 30}
outer locals: {'inner': <function outer.<locals>.inner at 0x01E11270>, 'y':
20}
```

- global() 可以取得全域變數的名稱與值
- 如果對變數指定值時，希望是針對全域範圍的話，可以使用 global 宣告

```
>>> x = 10
>>> def func():
...     global x, y
...     x = 20
...     y = 30
...
>>> func()
>>> x
20
>>> y
30
>>>
```

- 來看看以下這個會發生什麼事情？

```
>>> x = 10
>>> def func():
...     print(x)
...     x = 20
...
>>> func()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 2, in func
UnboundLocalError: local variable 'x' referenced before assignment
>>>
```

- 在 Python 3 中新增了`nonlocal`，可以指明變數並非區域變數

func scope_demo3.py

```
x = 10
def outer():
    x = 100          # 這是在 outer() 函式範圍的 x
    def inner():
        nonlocal x
        x = 1000     # 改變的是 outer() 函式的 x
    inner()
    print(x)         # 顯示 1000

outer()
print(x)             # 顯示 10
```

yield 與 yield from

- 函式並不會因為 yield 而結束，只是將流程控制權讓給函式的呼叫者

func yield_demo.py

```
def xrange(n):
    x = 0
    while x != n:
        yield x
        x += 1

for n in xrange(10):
    print(n)
```

- 當函式中使用 yield 指定一個值時，呼叫該函式會傳回一個 generator 物件
- 該物件具有 __next__() 方法，通常會使用 next() 函式呼叫
- 若無法產生下一個值，則會發生 StopIteration 例外

```
>>> g = xrange(2)
>>> type(g)
<class 'generator'>
>>> next(g)
0
>>> next(g)
1
>>> next(g)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
>>>
```

- 可以透過 send() 方法指定值，令其成為 yield 運算結果

func producer_consumer.py

```
import sys
import random

def producer():
    while True:
        data = random.randint(0, 9)
        print('生產了：', data)
        yield data          ← ❶ 產生下個值，流程回到呼叫者

def consumer():
    while True:
        data = yield        ← ❷ 呼叫產生器 send()方法時的指定值，會成爲 yield 的運算結果
        print('消費了：', data)

def clerk(jobs, producer, consumer):
    print('執行 {} 次生產與消費'.format(jobs))
    p = producer()
    c = consumer()
    next(c)                 ← ❸ 令消費者執行至 yield 處
    for i in range(jobs):
        data = next(p)      ← ❹ 取得生產者的產生值
        c.send(data)        ← ❺ 將值傳給消費者

clerk(int(sys.argv[1]), producer, consumer)
```

- 產生器的資料來源是直接從另一個產生器取得，那會怎麼樣呢？

```
def np_range(n):
    for i in range(0 - n, 0):
        yield i

    for i in range(1, n + 1):
        yield i
# 顯示[-5, -4, -3, -2, -1, 1, 2, 3, 4, 5]
print(list(np_range(10)))
```

- 從 Python 3.4 開始，新增了 `yield from`

```
def np_range(n):
    yield from range(0 - n, 0)
    yield from range(1, n + 1)
# 顯示[-5, -4, -3, -2, -1, 1, 2, 3, 4, 5]
print(list(np_range(10)))
```