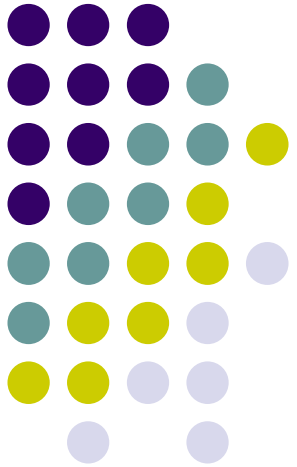
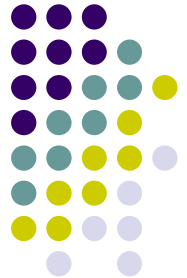
Python程式設計入門
函式(4/4)

葉難

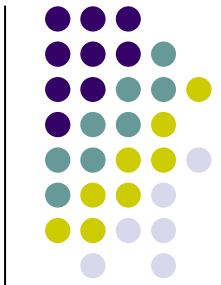


大綱



- 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，環境模型
- 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

產生器（generator）



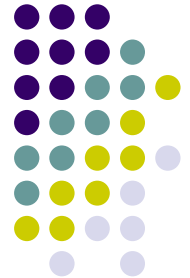
- 產生器運算式（generator expression）
- 產生器函式（generator function）
- 產生器物件符合迭代器介面
- 產生器：
    1. 實作Iterator的簡便方式
    2. 支援委託（delegation）概念：yield from
    3. coroutine：send

產生器函式（generator function）

- def述句裡有yield述句，便是gf
- 呼叫gf，回傳物件的型別是：產生器-迭代器，簡稱產生器，符合迭代器介面

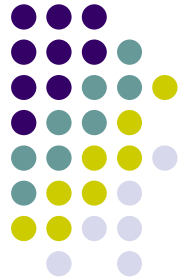
```
>>> def gf(): yield 1
>>> g = gf()      # 呼叫gf
>>> g             # 得到產生器
<generator object foo_gf at 0x00C775D0>
>>> next(g)       # 拿下一個
1
>>> next(g)       # 耗盡，引發異常StopIteration
StopIteration
```

產生器函式



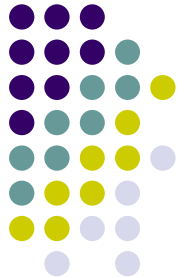
1) 產生器函式是函式，回傳物件的型別是產生器（generator），符合迭代器介面
2) 呼叫產生器函式，此時尚未執行裡頭的程式
3) 向產生器要下一個東西時（呼叫next），才會執行裡頭的程式，直到碰到yield，產出yield後面運算式的結果交給呼叫方（next）；此時會凍結產生器的執行流程、停在該處
4) 下次呼叫next時，從yield之後的述句開始執行
5) 當產生器遇到return或結束，引發異常StopIteration

範例：**Frange**（類別）



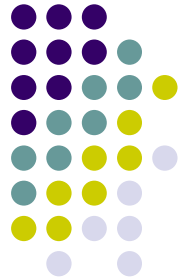
- 內建型別range，只能是int
- 需要float，例如0.0、0.1、0.2、...、0.9
- 注意：float有精確度的問題
- frange_class.py（類別）
- frange_gf.py（產生器函式）

範例



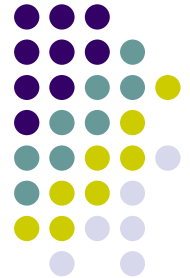
- 改寫MyDeck：MyDeck_gf.py
- 改寫unique：unique_gf.py

問題



- 給定n與step，下次產出n，下下次產出n+step，下下下次產出n+step*2，依此類推
- 給定iterable（含數字），產出總和與平均；例給定(2, 4, 6, 8)，依序產出(2, 2)、(6, 3)、(12, 4)、(20, 5)
- 實作take(n, iterable)，只產出iterable的前n個東西（最多）

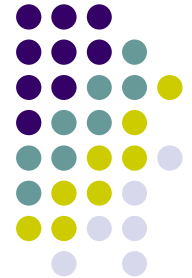
問題：到目前爲止的最大值



- 不斷傳入數字，回傳到目前爲止看到的最大值
- 使用例：

```
def max_so_far(): ...省略...
msf = max_so_far()
print(msf(3))   # 印出3
print(msf(5))   # 印出5
print(msf(-1))  # 印出5
print(msf(6))   # 印出6
```

範例：詞典順序的排列組合



- 「0、1、2」根據詞典順序的排列，依序是012、021、102、120、201、210
- 請找出「0、1、2...、9」詞典順序的排列的第100萬個數字
- lex_perm.py

範例：三角數、五角數、六角數（**tph.py**）

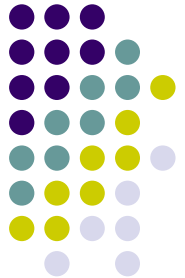
- 三角數T(n) = n * (n+1) / 2，
  1, 3, 6, 10...
- 五角數P(n) = n * (3*n-1) / 2，
  1, 5, 12, 22...
- 六角數H(n) = n * (2*n-1)，
  1, 6, 15, 28, 45...
- 已知T(1) = P(1) = H(1) = 1
- 已知T(285) = P(165) = H(143) = 40755
- 請找出下一個n，使得T(n)、P(n)、H(n)皆相等

yield from（3.3版）

- 讓產生器的產出、交由另一個產生器（或迭代器）負責

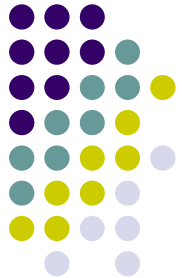
```
def gf(n):
    for i in range(1, n):
        yield i
    for i in range(-1, -n, -1):
        yield i

# 得到 1, 2, 3, 4, -1, -2, -3, -4
```

yield from範例：chain



```
def chain(*iterables):
    for itb in iterables:
        yield from itb
        # for e in itb:
            # yield e

r = range(3)
li = ['a', 'b', 'c']
t = (0.1, 0.2, 0.3)
for x in chain(r, li, t):
    print(x)
# 印出0、1、2、a、b、c、0.1、0.2、0.3
```

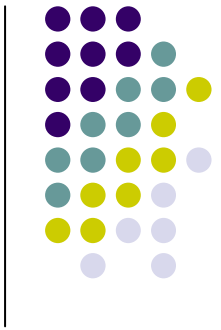
yield from範例



- 改寫爲產生器函式
- 改寫perm（排列）
- 改寫queen（八后問題）

- 若原本是迭代形式，改寫時較爲直覺
- 若原本是遞迴形式，改寫時需要稍加思考

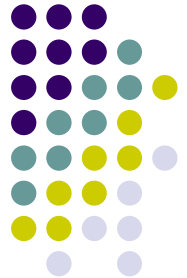
coroutine



- 產生器具備coroutine介面，使用方能夠送入東西給產生器
- coroutine使用方：x = co.send(y) # 給、拿
- coroutine方：y = yield x # 給、拿

- 產生器使用方：x = next(g) # 拿
- 產生器方：yield x # 給

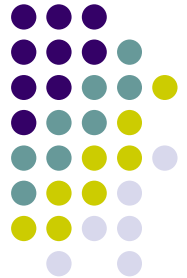
範例：coroutine



- 累加

```
def acc_gf(n):
    while True:
        y = yield n
        n += 1 if y is None else y
acc_g = acc_gf(100)

print(next(acc_g))          # y會是None
print(acc_g.send(3))        # y會是3
print(next(acc_g))
print(acc_g.send(5))
```

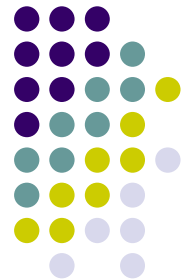
提醒：詞彙混淆



- 產生器函式
- 產生器
- 產生器-迭代器

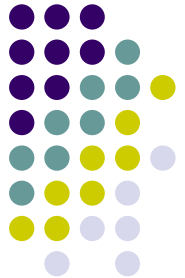
- 有些人混用上述所有詞彙

函數式（**functional**）程式設計



- 程式設計範式（paradigm）：程序式、物件導向、函數式、資料驅動式、邏輯式、等等
- 希望函式（function）能像數學函數（function）一樣
- 資料在一連串的函式之中流動
- 高階函式：map、filter、reduce
- 運算子模組operator

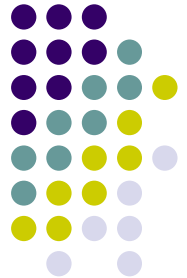
副作用（side effect）



- 攝氏轉華氏

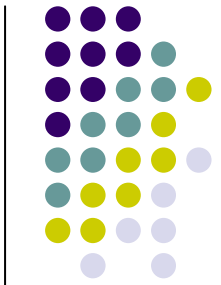
```
tc = [0, 100, 30.2, 22.5, -15.8, 23.7]

def ctof():     # 修改外界的可變物件
    for i in range(len(tc)):
        tc[i] = tc[i] * 9.0 / 5.0 + 32

ctof()
```

概略原則



- 每支函式各自獨立且行爲清楚
- 不要存取外界的物件
- 不要修改傳進來的參數（若是可變物件）
- 輸出僅與輸入有關：給什麼樣的東西，就會得到什麼樣的結果
- 採用一致的介面：如串列、可迭代者、迭代器

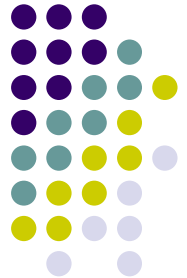
map(func, iterable, ...)



- 拿著函式func，作用到iterable裡每一個元素；由回傳的迭代器負責交給使用方

```
tc = (0, 100, 30.2, 22.5, -15.8, 23.7)

def ctof(c): # 攝氏轉華氏
    return c * 9.0 / 5.0 + 32

itr_f = map(ctof, tc)

# 請看fl_ctof.py
```

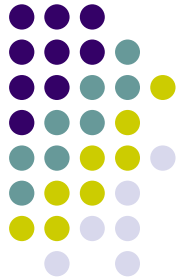
把迴圈換成map



- 拿出一個個元素，套用函式，以迭代器介面交給使用方

```
for e in itb:
    func(e)

def my_map(func, itb):
    for e in itb:
        yield func(e)
```

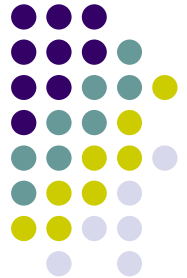
filter(func, iterable)



- 資料在一連串的函式之中流動
- map：映射、轉換
- filter：過濾、條件判斷

```
scores = (30, 45, 60, 80, 20)
def failed(score):
    return score < 60

filter(failed, scores)
```

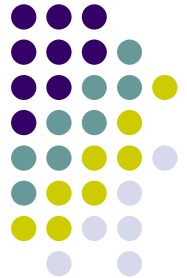
範例



- 1~n的偶數平方

```
def sq(n): return n**2
def even(n): return n%2 == 0

def sq_even(n):
    return map(sq, filter(even, range(1, n)))
```

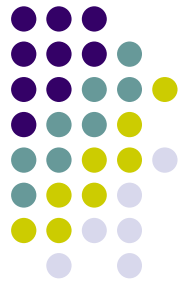
範例



- 找出奇數的費氏數

```
def odd_fibs(n):
    r = range(0, n+1)

    return filter(lambda x: x[1]%2 != 0,
                  zip(r,
                      map(fib_m, r)))
```

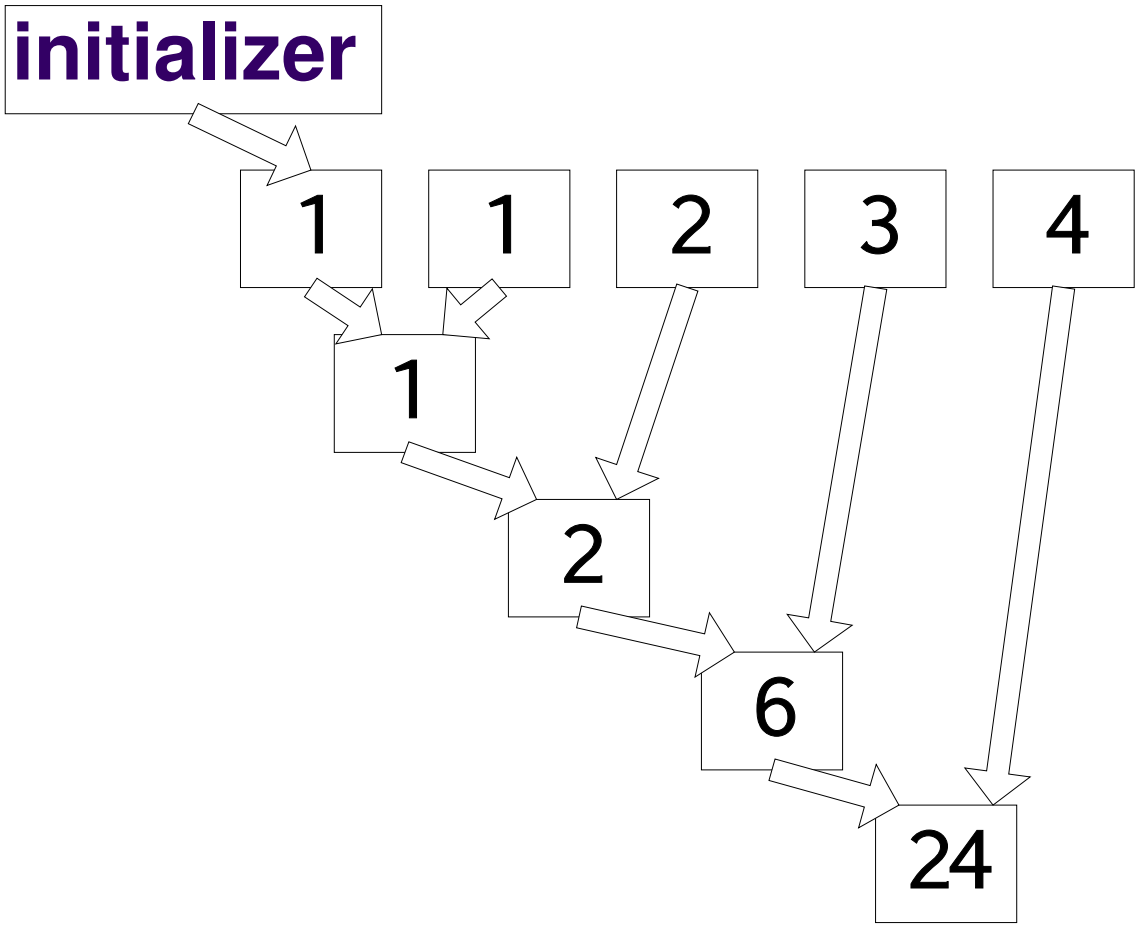
reduce(func, iterable [, initializer])



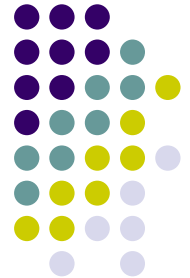
- 3.x版：移到functools模組裡
- 階乘

```
from functools import reduce
def mul(x, y): return x * y

def fact(n):
    return reduce(mul, range(1, n+1), 1)

# 模組operator裡已經有mul
```

reduce(mul, range(1, 4+1), 1) 示意圖

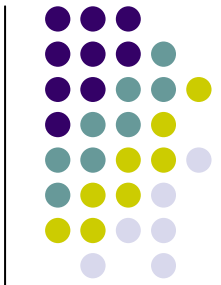


運算子與operator模組的函式



- 加法，「a + b」，「add(a, b)」
- 乘法，「a * b」，「mul(a, b)」
- 相等，「a == b」，「eq(a, b)」
- 索引，「obj[k]」，「getitem(obj, k)」
- 索引指派，「obj[k] = v」，「setitem(obj, k, v)」
- 小於，「a < b」，「lt(a, b)」
- 其他

reduce與sum：1~n的偶數平方的總和

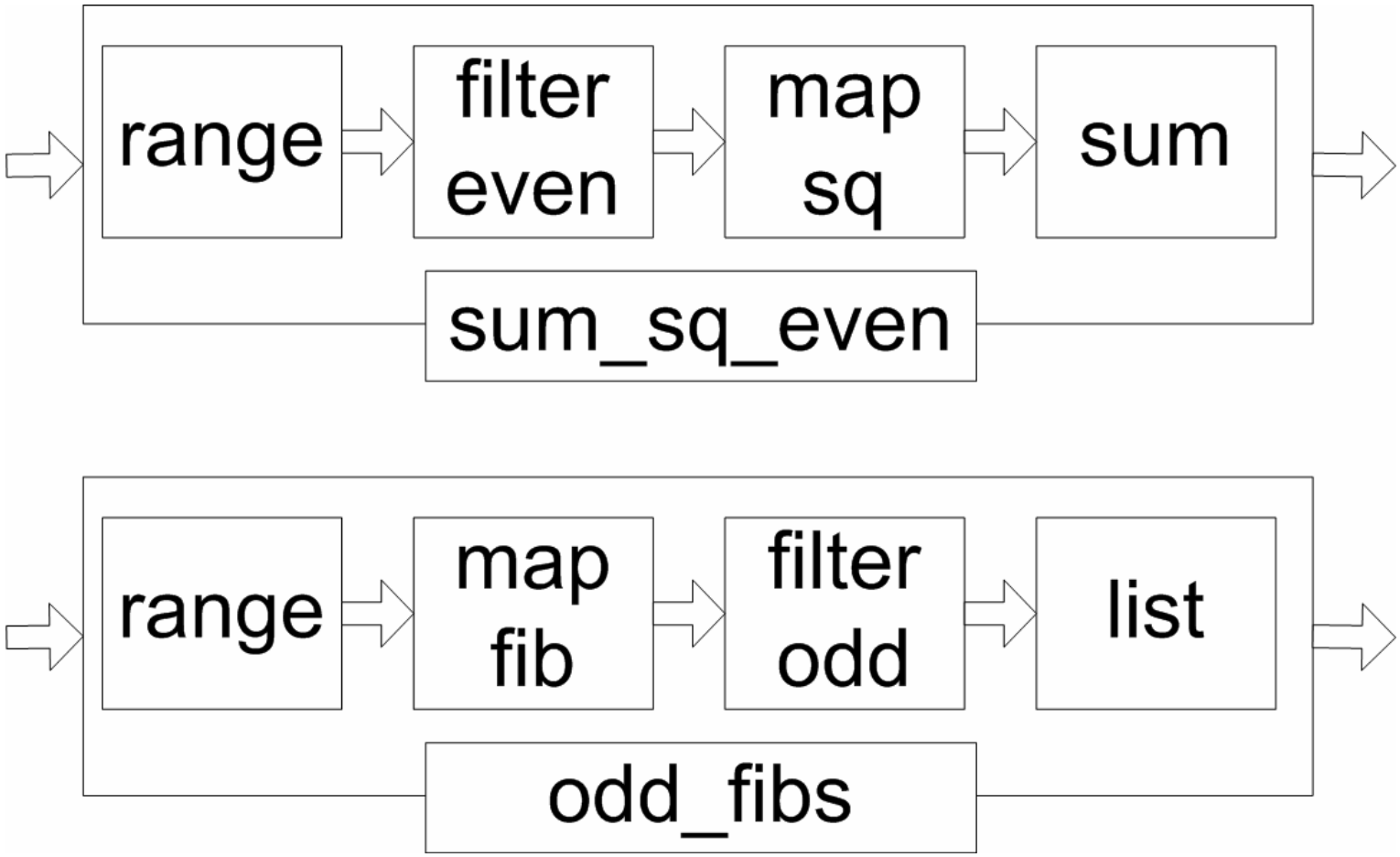


```
def sum_sq_even(n):
    return sum(map(sq,
                   filter(even,
                          range(1, n))))
def sum_sq_even(n):
    return reduce(add,
                  map(sq,
                      filter(even,
                             range(1, n))))
```

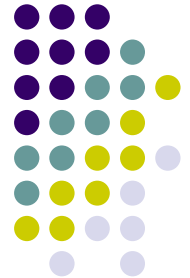
reduce與list：
找出奇數的費氏數，放入list

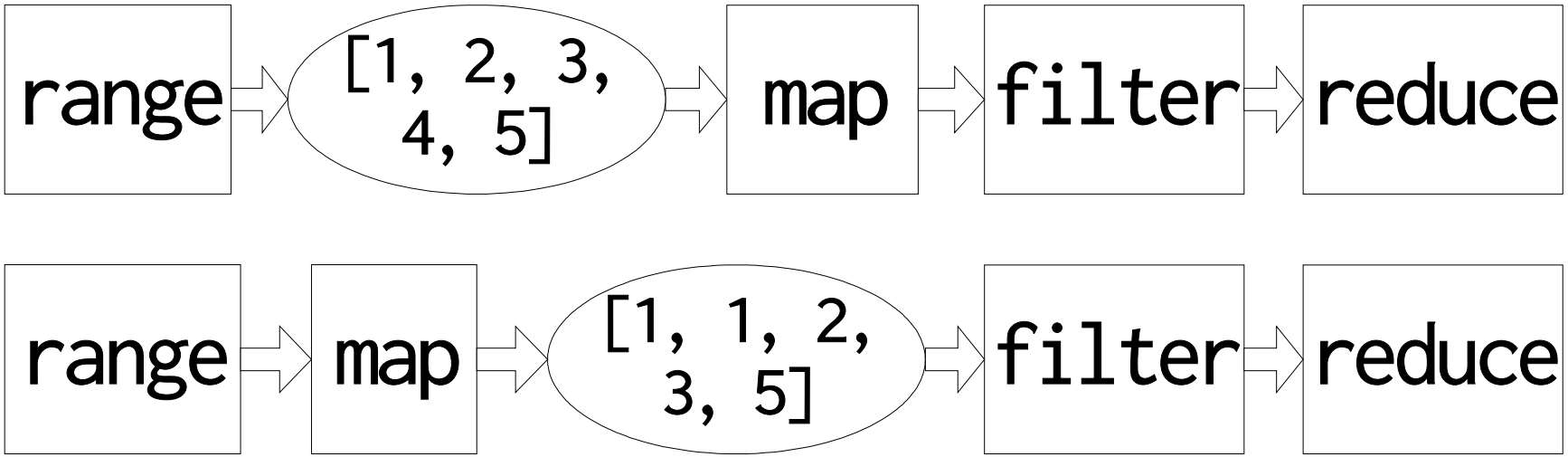
```
def odd_fibs(n):
    r = range(0, n+1)
    return list(filter(lambda x: x[1]%2 != 0,
                       zip(r,
                           map(fib_m, r))))
def odd_fibs(n):
    r = range(0, n+1)
    return reduce(lambda x, y: x + [y],
                  filter(lambda x: x[1]%2 != 0,
                         zip(r,
                             map(fib_m, r))),
                  [])
```

猶如一道道關卡

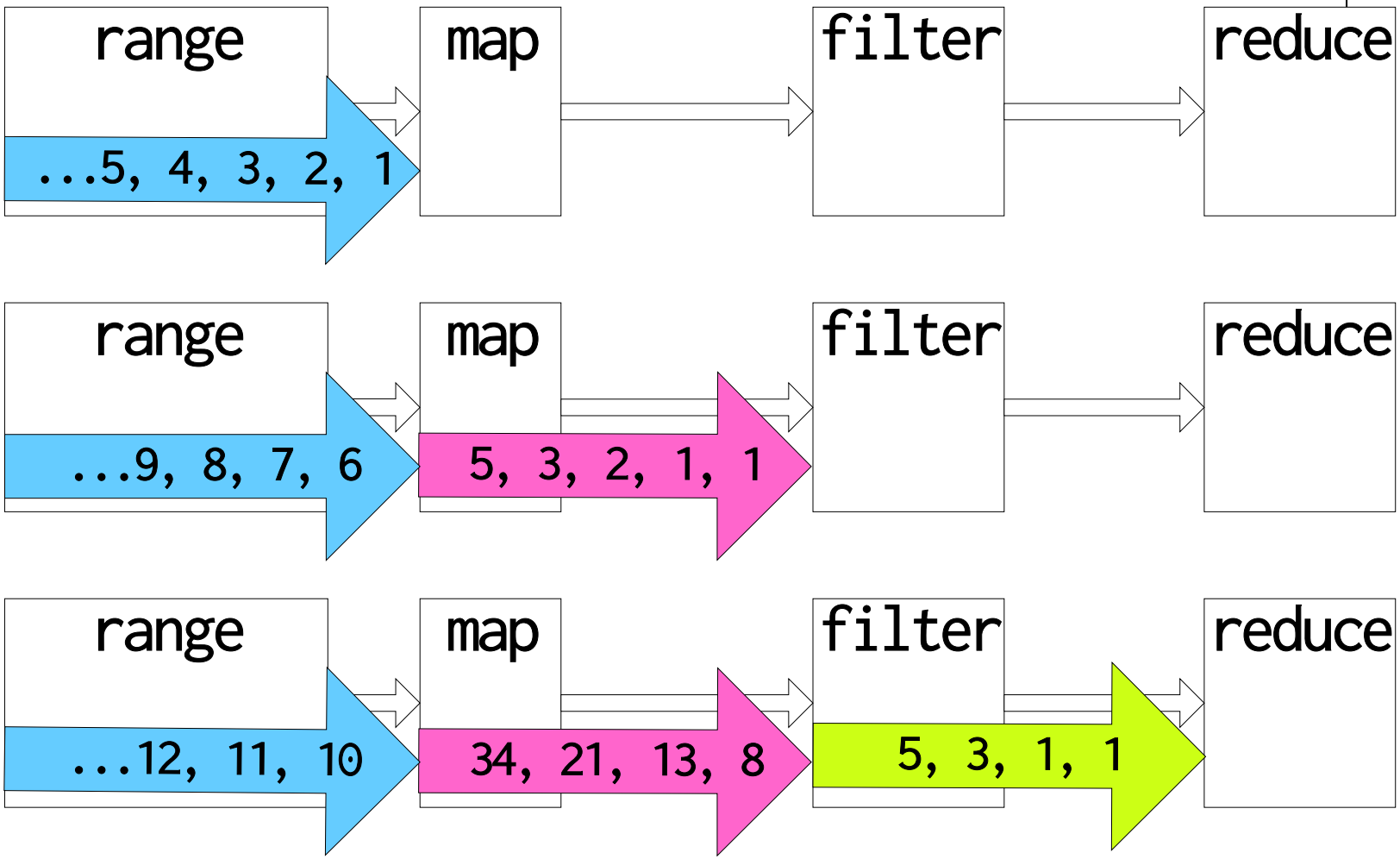

range
filter
even
map
sq
sum
sum_sq_even
range
map
fib
filter
odd
list
odd_fibs

整批處理vs延遲處理（串流）





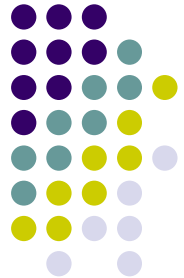
整批處理vs延遲處理（串流）



範例：三角數、五角數、六角數（**fl_tph.py**）

- 三角數T(n) = n * (n+1) / 2，
  1, 3, 6, 10...
- 五角數P(n) = n * (3*n-1) / 2，
  1, 5, 12, 22...
- 六角數H(n) = n * (2*n-1)，
  1, 6, 15, 28, 45...
- 已知T(1) = P(1) = H(1) = 1
- 已知T(285) = P(165) = H(143) = 40755
- 請找出下一個n，使得T(n)、P(n)、H(n)皆相等

問題



```
(lambda k: reduce(mul, range(1, k+1), 1))(8)


n = 50
print(sorted(set(range(2,n+1)).difference(set(
  (p * f) for p in range(2,int(n**0.5) + 2)
  for f in range(2, int(n/p)+1)))))
```

Q&A