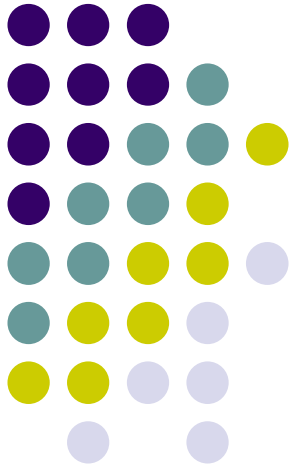
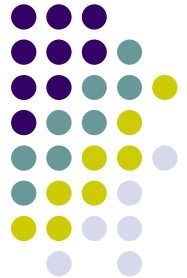
Python程式設計入門
函式(3/4)

葉難



大綱



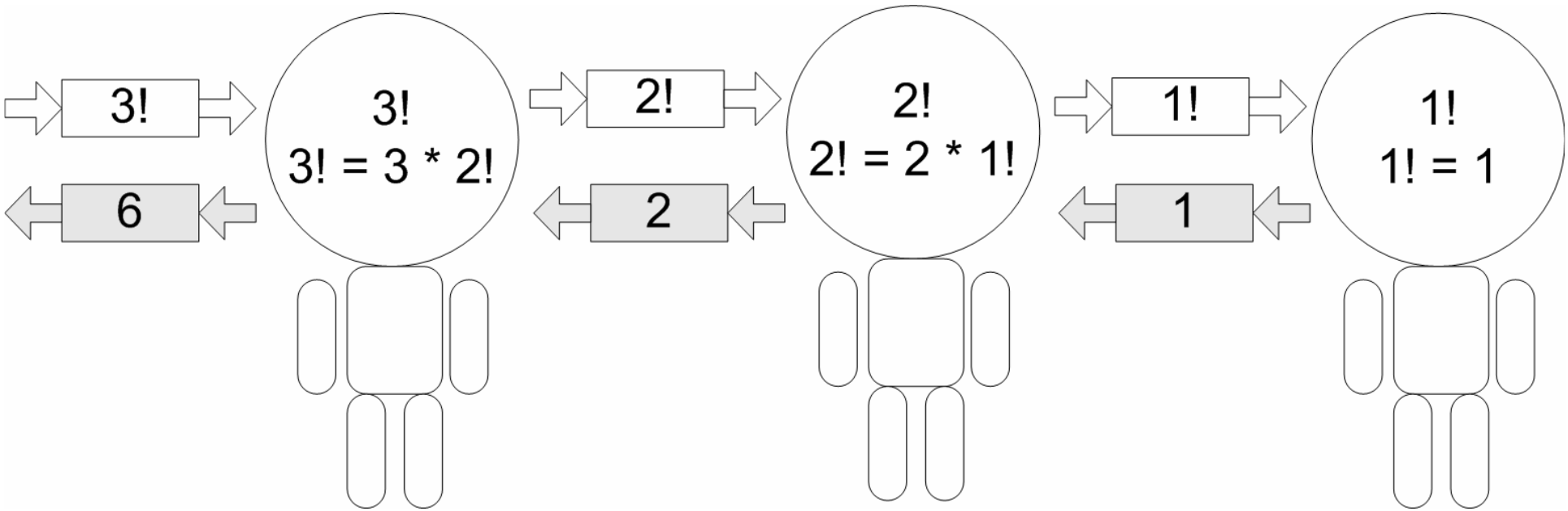
- 函式定義與呼叫，def述句和lambda運算式
- 參數傳遞
- 範圍，命名空間，環境模型
- 遞迴（recursion）
- 高階函式（higher-order function）
- 裝飾器（decorator）
- 產生器（generator）
- 函數式程式設計

遞迴

- 階乘定義（迭代形式）：
  n! = n * (n-1) * (n-2) * ... * 1
  0! = 1
- 階乘定義（遞迴形式）
  n! = n * (n-1)!　　縮減問題
  0! = 1! = 1　　終止條件

```
def fact_r(n):
    if n == 0 or n == 1:                # 終止條件
        return 1
    else:
        return n * fact_r(n-1)          # 縮減問題
```

階乘示意圖



Fibonacci（費波那契）數

- 定義

  fib(0) = 0

  fib(1) = 1

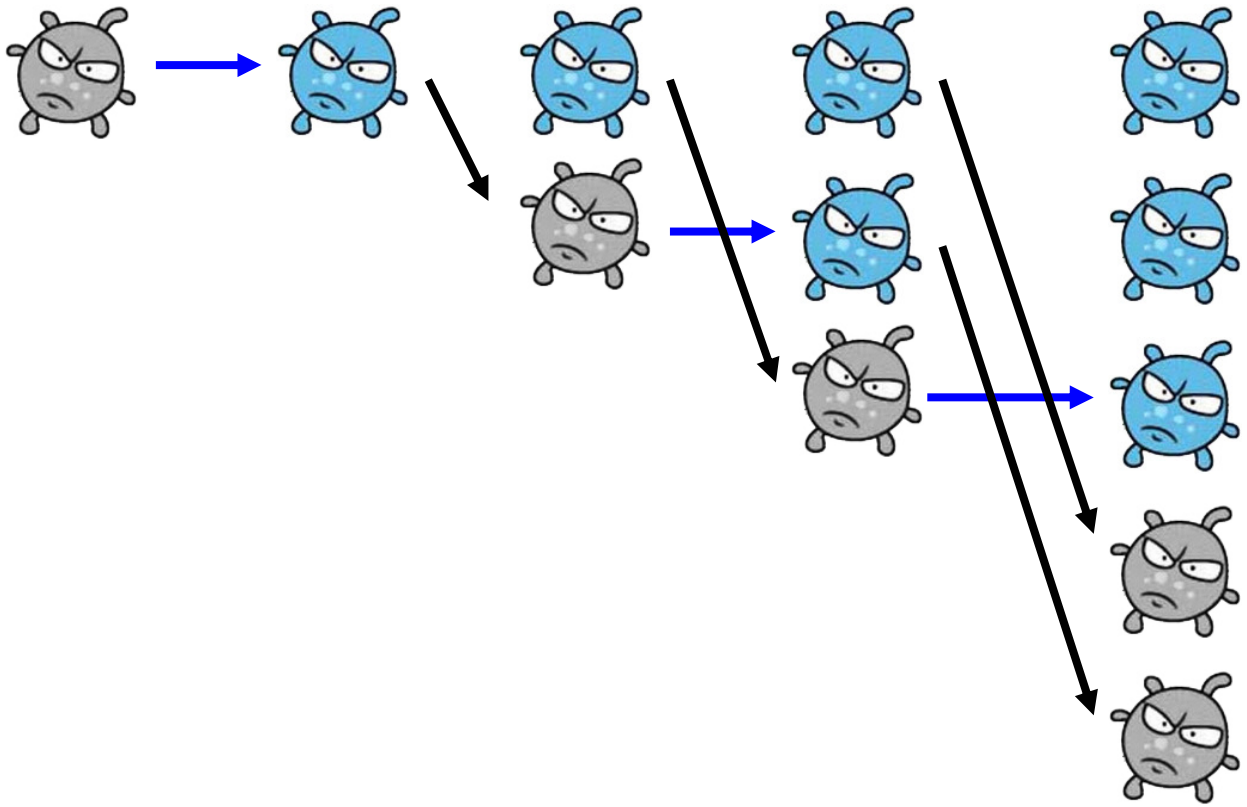
  fib(n) = fib(n-1) + fib(n-2)　若n≧2

- 細菌譬喻

n	0	1	2	3	4	5	6	7	9
fib(n)	0	1	1	2	3	5	8	13	21

細菌譬喻

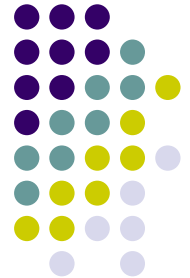
n	0	1	2	3	4	5
fib(n)	0	1	1	2	3	5



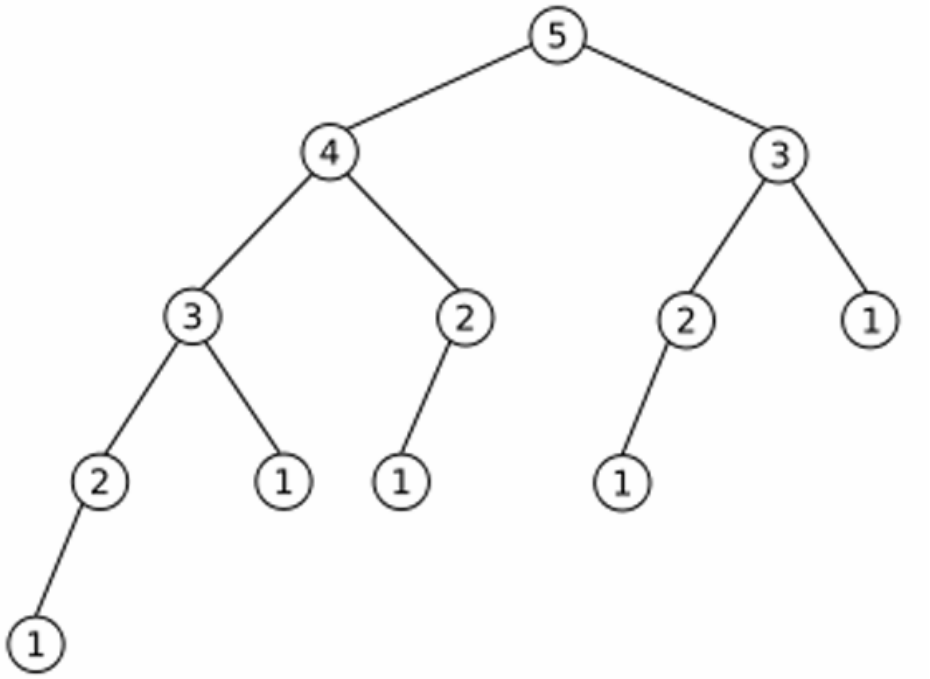
迭代與遞迴（**fib.py**）

```
def fib_i(n):
    a, b = 0, 1
    for i in range(n):
        a, b = b, a+b
    return a

def fib_r(n):
    if n == 0 or n == 1:   # 終止條件
        return n
    else:                  # 縮減問題
        return fib_r(n-1) + fib_r(n-2)
```

fib_r的問題：重複計算



- 原因：子結構重疊
- 技巧：記憶（memoization ）
- 運用高階函式（函式回傳函式），把記憶區放到外圍範圍

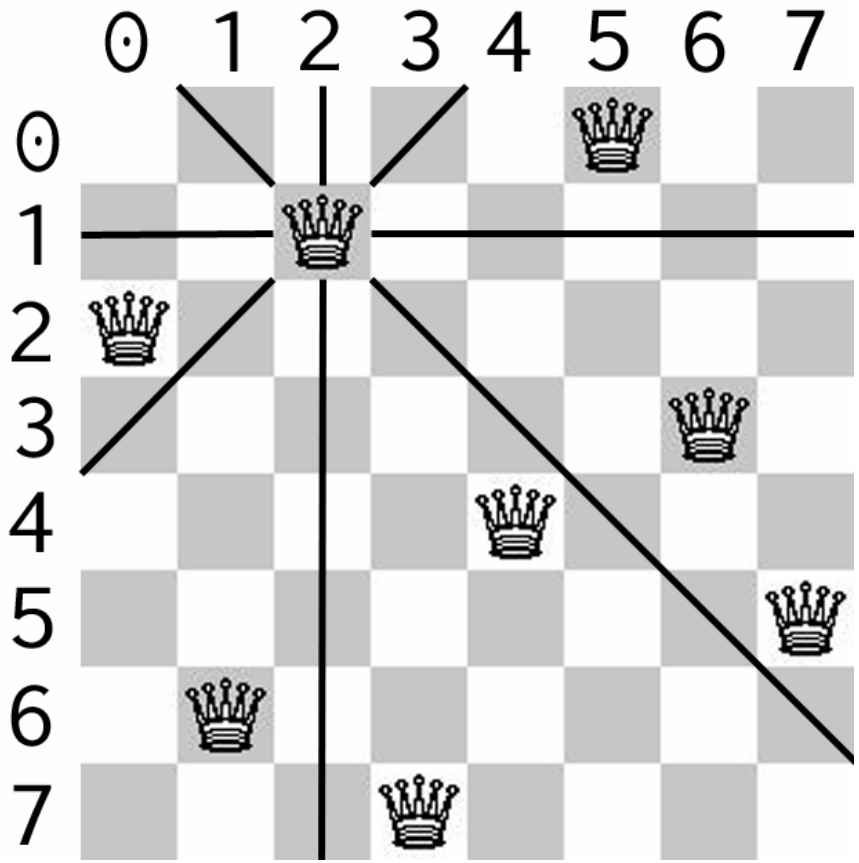


排列（permutation）
perm.py

- 給定(1, 2, 3)，回傳所有的排列，
  [(1, 2, 3), (1, 3, 2), (2, 1, 3),
  (2, 3, 1), (3, 1, 2), (3, 2, 1)]
- 想法：
  拿出1，（遞迴）取得(2, 3)的排列，然後在前頭分別加上1。
  拿出2，（遞迴）取得(1, 2)的排列，然後在前頭分別加上2。
  拿出3，（遞迴）取得(1, 2)的排列，然後在前頭分別加上3。

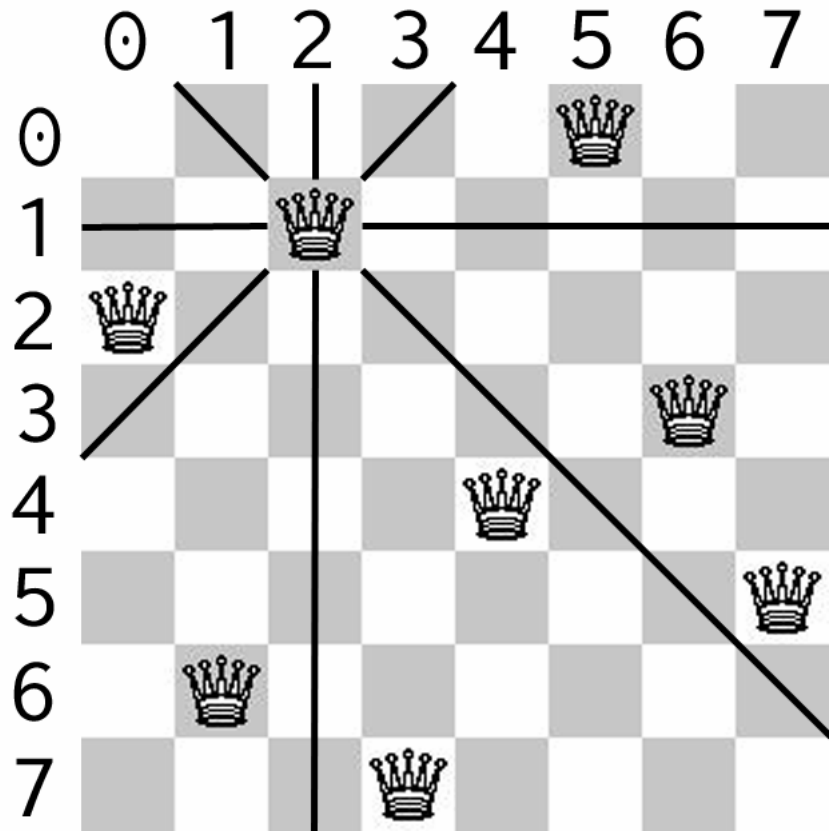
八后問題

- 8x8棋盤，擺放8個皇后，互不攻擊
- 皇后可以攻擊的地方：同一橫列、同一直欄、兩條斜角線
- 想法：從第0列開始擺，迭代列上所有可擺位置，然後遞迴

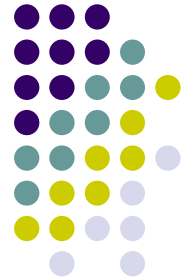


表示擺法

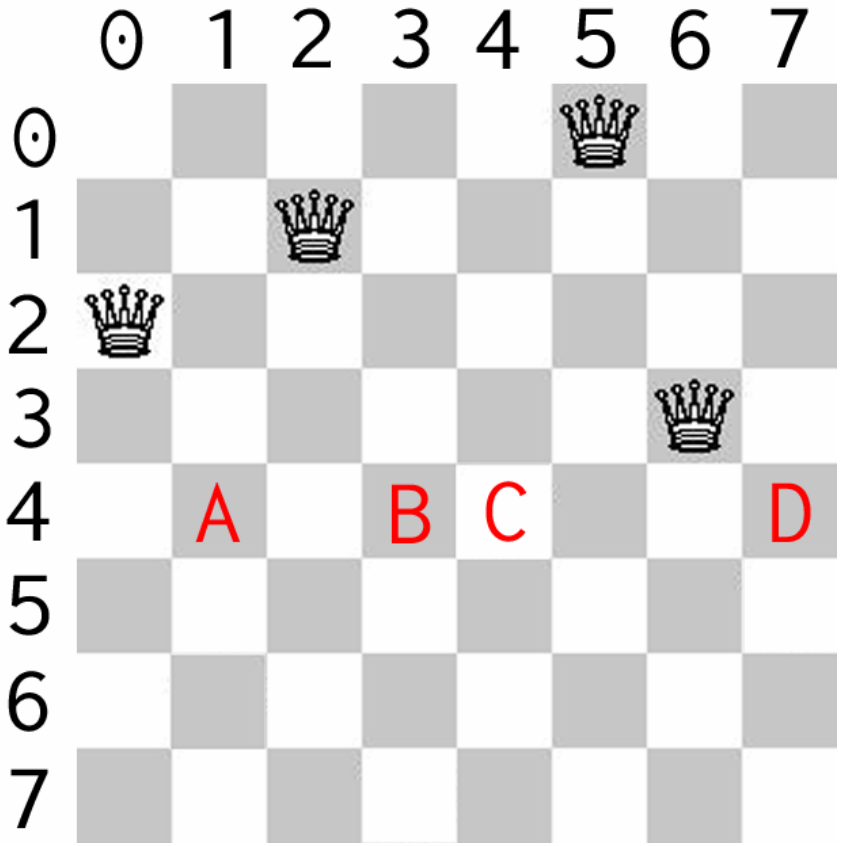
- 以tuple表示擺法，
  「索引」代表「列」，
  「值」代表「欄」
- 右圖擺法表示爲
  (5, 2, 0, 6, 4, 7, 1, 3)
- 例：索引3的值是6，代表第3列的皇后在第6欄



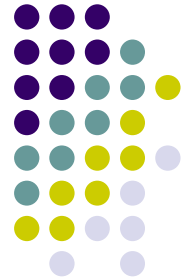
檢查下個皇后可擺處(1/2)



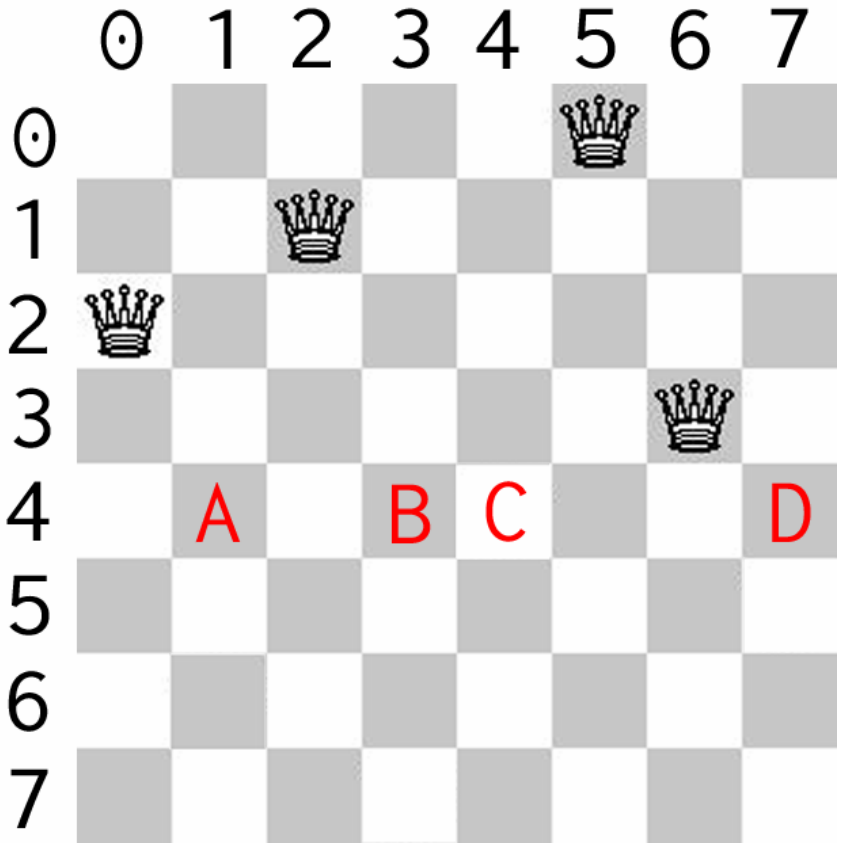
- 已有的部分擺法：
  (5, 2, 0, 6)
- 接下來要擺第4列（索引4），不必檢查橫列攻擊
- 檢查直欄攻擊，也就是檢查索引4的值是否已存在於之前的部分擺法



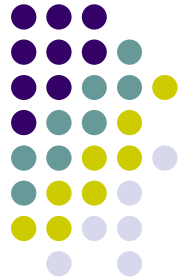
檢查下個皇后可擺處(2/2)



- （兩條）斜角線攻擊：也就是檢查兩后座標斜率是否爲1或-1
- 也就是：兩后列座標相減，欄座標相減，取絕對值後是否相等
- 例A(4,1)攻擊(0,5)，abs(4-0) == abs(1-5)
- 例B(4,3)不攻擊(1,2)，abs(4-1) != abs(3-2)



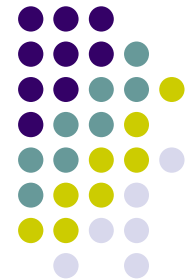
八后問題的程式



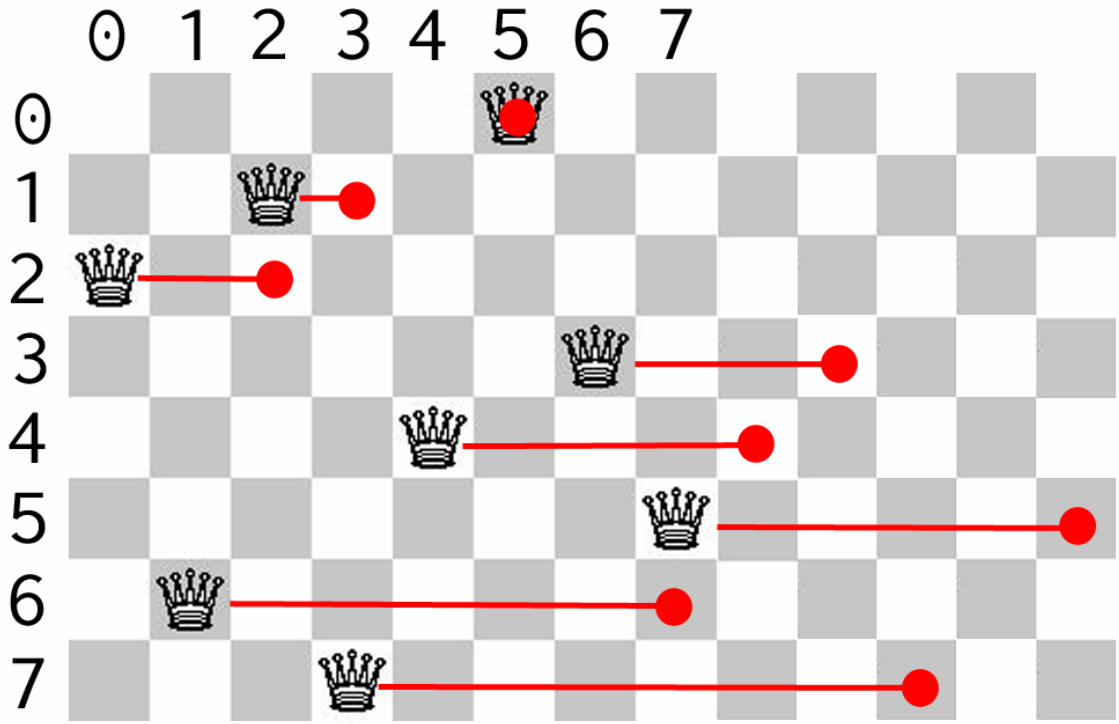
- queen_basic.py（解法如前所述）
- queen_hettingers.py（先排列出所有擺法）
- queen_howell.py（非遞迴）

- queen_c++.cpp（冗長，並未解出？）

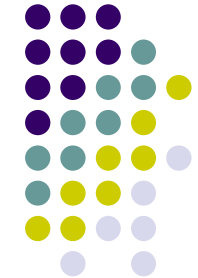
queen_hettingers.py
檢查可能擺法的斜角線攻擊



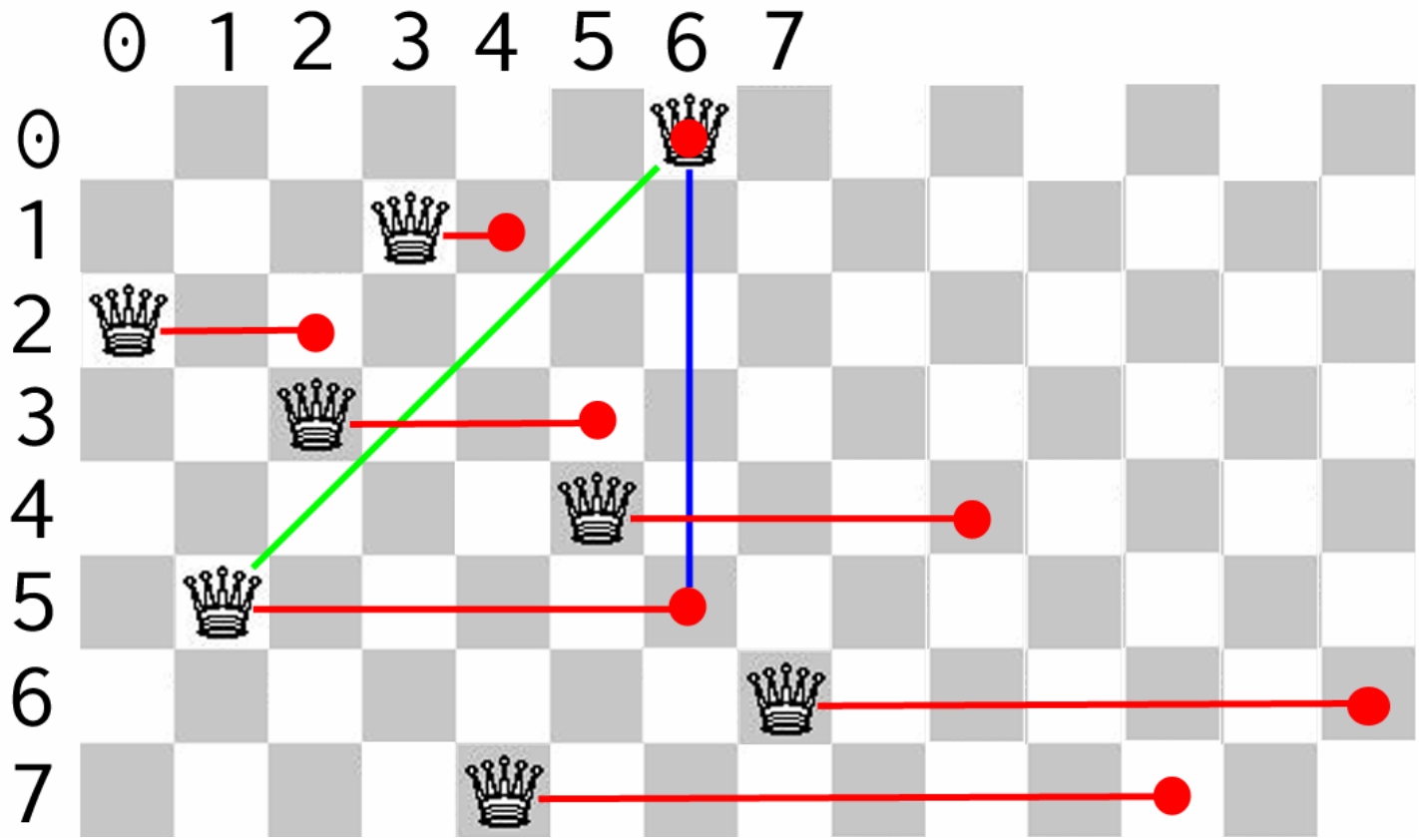
- 每個皇后向「右」位移，位移量是該皇后的列座標，若原先互相攻擊，那麼位移後的欄座標將相等
- 向「右」位移檢查斜率-1，向「左」位移檢查斜率1



queen_hettingers.py
檢查可能擺法的斜角線攻擊



- (0,6)與(5,1)會互相攻擊，位移後位於同一欄

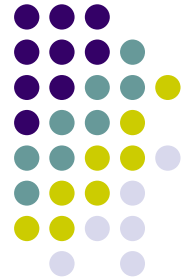


排列：取巧方式

- perm_product.py
- 使用itertools.product、運用set

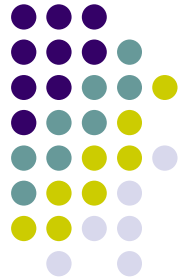
```
>>> from itertools import product
>>> tuple(product( ('a','b'), (1,2) ))
(('a', 1), ('a', 2), ('b', 1), ('b', 2))

>>> tuple(product( (1,2,3), repeat=3 ))
((1, 1, 1), (1, 1, 2), (1, 1, 3), (1, 2, 1),
  (1, 2, 2), (1, 2, 3), (1, 3, 1), (1, 3, 2),
  (1, 3, 3),...省略... (3, 2, 3), (3, 3, 1),
  (3, 3, 2), (3, 3, 3))
```

高階（**higher-order**）函式



- 函式也是物件，可指派名稱
- 函式作爲參數，傳入另一個函式
- 函式作爲回傳值（函式回傳函式）
- 包含外圍命名空間的函式，叫做「閉包」（closure）
- 想抓住重複出現的「pattern」（樣式、形態）

加總（summation）
hf_sum.py



- 從某數start，一直加，加到end（包括）

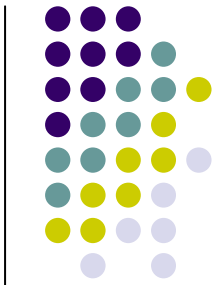
```
def sum_number_r(start, end):
    if start > end:     # 終止
        return 0
    else:               # 縮減，遞迴
        return start + sum_number_r(
                              start+1, end)
```

加總的項目：平方

- start**2 + (start+1)**2 + ... + end**2

```
def sum_square_r(start, end):
    if start > end:
        return 0
    else:
        return start**2 + sum_square_r(
                                start+1, end)
```

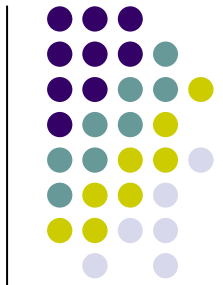
∑（sigma）



$$\sum_{i=start}^{end} i$$

$$\sum_{i=start}^{end} i^2$$

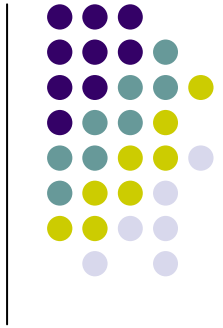
圓周率pi



- i每次要跳4

$$\sum_{i=start\,(by\,4)}^{end} \frac{1}{i(i+2)} = \frac{1}{1*3} + \frac{1}{5*7} + \frac{1}{9*11} + \ldots = \frac{pi}{8}$$

圓周率pi



- 下個i 、下個項目

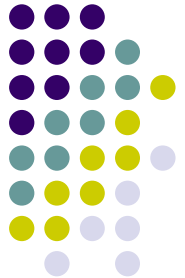
```
def sum_pi_r(start, end):
    if start > end:
        return 0
    else:
        return (1.0 / (start * (start+2))) +
                        sum_pi_r(start+4, end)
```

抓住上述的pattern

- 下個i 、下個項目：以更高階的形式表達

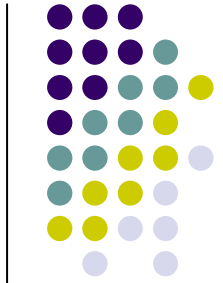
```
def sum_hf_r(item, start, next, end):
    if start > end:
        return 0
    else:
        return item(start) +
      sum_hf_r(item, next(start), next, end)
```

使用sum_hf_r



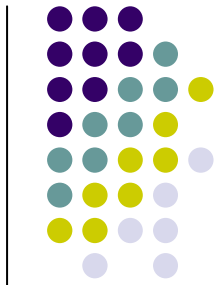
```
id = lambda x: x
next_1 = lambda i: i+1

def sum_number_r(start, end):
    return sum_hf_r(id, start, next_1, end)

#### 另一種寫法
sum(x for x in range(1, 100+1))
```

使用sum_hf_r



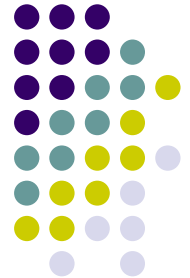
```
sq = lambda x: x**2
next_1 = lambda i: i+1

def sum_sq_r(start, end):
    return sum_hf_r(sq, start, next_1, end)

#### 另一種寫法
sum(x**2 for x in range(1, 100+1))
```

使用sum_hf_r



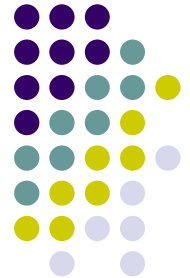
```
item = lambda x: 1.0 / (x * (x+2))
next_4 = lambda i: i+4

def sum_pi_r(start, end):
    return sum_hf_r(item, start, next_4, end)

#### 另一種寫法
sum(1/(x*(x+2)) for x in range(1, 100+1, 4))
```

fib_memo：函式回傳函式（fib.py）



```
def fib_memo():
    memo = {0: 0, 1: 1}
    def sub(n):
        if n not in memo:
            memo[n] = sub(n-1) + sub(n-2)
        return memo[n]
    return sub

fib_m = fib_memo()
x = fib_m(2)
```

計數器counter （scope_counter.py）



- 高階函式：函式回傳函式，閉包（closure）

```
def counter(n):
    li = [n]
    def bar(x):
        li[0] += x      # 在bar裡，原地修改counter的li
        return li[0]
    return bar          # counter回傳「函式」

c0 = counter(0)         # 名稱c0與c100都指向函式物件
c100 = counter(100)

print(c0(1))          # 印出1
print(c100(10))       # 印出110
```

計數器counter：使用nonlocal

- nonlocal，指定名稱的所屬範圍爲「外圍」

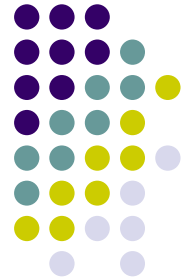
```
def counter(n):
    def bar(x):
        nonlocal n
        n += x
        return n
    return bar
```

裝飾器（decorator）

- 裝飾器是函式（可被呼叫者）
- 裝飾器的參數：另一支函式
- 裝飾器回傳新函式：爲那支函式加裝功能後的函式
- 語法「@decorator」

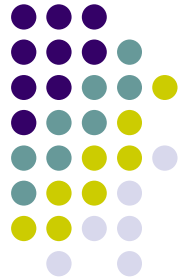
- fib_memo與記憶功能

檢查參數是否爲自然數



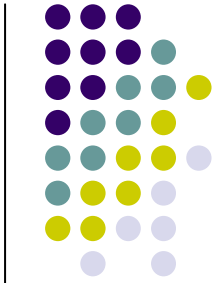
```
def dec_natural(func):
    def wrapper(n):
        if type(n) == int and n >= 0:
            return func(n)
        else:
            raise TypeError('Argument is
                        not a natural number')

    return wrapper
```

使用



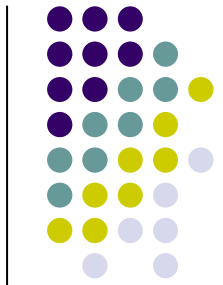
```
@dec_natural
def fact_i(n):
    result = 1
    for i in range(1, n+1):
        result *= i
    return result

print(fact_i(5))    # 印出120
print(fact_i(-1))  # 引發異常TypeError
```

@語法



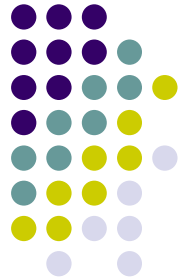
```
@dec_natural
def fact_i(n):
    ...省略...

#### 上下寫法意思相同

def fact_i(n):
    ...省略...
fact_i = dec_natural(fact_i)
```

量測函式執行花費時間
dec_time.py



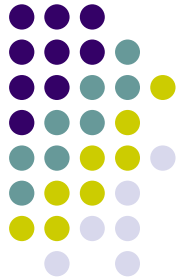
```
def dec_time(func):
    def wrapper(*args, **kwargs):
        import time
        t_start = time.clock()

        res = func(*args, **kwargs)

        t_end = time.clock()
        print(func.__name__, t_end-t_start)
        return res

    return wrapper
```

裝飾器是函式，也可有參數



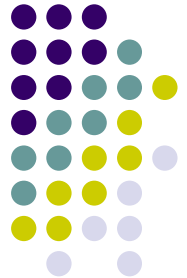
```
@decorator(argA, argB, ...)
def func(...):
    ...

#### 上下寫法意思相同

func = decorator(argA, argB, ...)(func)
```

記憶功能
dec_memo.py



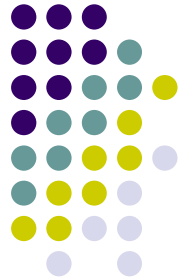
```
def dec_memo(func):
    cache = func.cache = {}

    def wrapper(*args, **kwargs):
        key = str(args) + str(kwargs)
        if key not in cache:
            cache[key] = func(*args, **kwargs)
        return cache[key]

    return wrapper
```

記憶功能：容量限制
dec_memo_limit.py



```
def dec_memo_limit(func, limit=None):
    ...省略...

@dec_memo_limit           # 一般
def fib_r(n):
    ...省略...

@dec_memo_limit(200)      # 有參數
def fib_r(n):
    ...省略...
```

補充



- 巢狀裝飾器

```
@dec2
@dec1
def f(...): ...
#
f = dec2(dec1(f))
```

- 原函式的資訊：functools.wraps
- 不一定回傳函式

Q&A