

1. 作業介紹

本作業使用程式語言 python 實作費氏數列，並使用 decorators 模式為費氏數列加上快取功能，比較原始費氏數列與加上取模式後的費氏數列的效能差距。

2. 實作方法

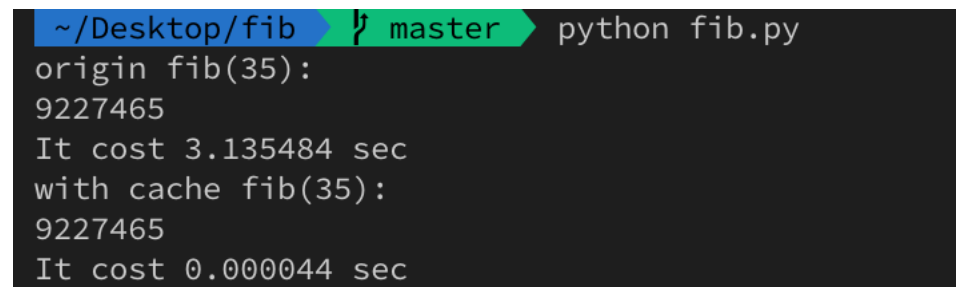
如圖所示，fib_o 與 fib 為費氏數列函式，傳入值為費氏數列的計算長度，回傳費氏數列運算結果，用函式 memoize 為 fib 加上快取功能，可以看到在 fib 函式前方使用“@”為函式加上 cache 函式功能，而程式碼 23 行後，使用套件 time 測試兩者執行時間的差異。

```
2  # coding: utf-8
3  import time
4
5  def cache(f):
6      cache_list = {}
7      def add_cache(*args):
8          if args in cache_list:
9              return cache_list[args]
10         else:
11             cache_list[args] = f(*args)
12             return cache_list[args]
13     return add_cache
14
15  def fib_o(n):
16      return n if n < 2 else fib_o(n-2) + fib_o(n-1)
17
18  @cache
19  def fib(n):
20      return n if n < 2 else fib(n-2) + fib(n-1)
21
22
23  tStart = time.time()
24  print('origin fib(35):')
25  print(fib_o(35))
26  tEnd = time.time()
27  print("It cost %f sec" % (tEnd - tStart))
28
29  tStart = time.time()
30  print('with cache fib(35):')
31  print(fib(35))
32  tEnd = time.time()
33  print("It cost %f sec" % (tEnd - tStart))
```

程式碼連結 <https://github.com/frogben/fib>

3. 執行結果

如圖所示，當 $N=35$ 時，原始的費式數列，耗時 3.135484 秒，而加入 cache 優化後，只需要 0.000044 秒，提升了 71261 倍，加入快取後效能提升非常明顯。



```
~/Desktop/fib master python fib.py
origin fib(35):
9227465
It cost 3.135484 sec
with cache fib(35):
9227465
It cost 0.000044 sec
```

4. 結論

推測由於費氏數列在遞迴時，會不斷計算到重複的值，加入快取功能後，效能提升非常明顯，而 python 方便的使用“@”即可使用 decorators 模式，也為程式設計師未來要擴充程式功能時，增加了許多選擇。