

科學計算程式設計實習

Python 檔案、模組、例外

王昱景 Brian Wang

brian.wang.frontline@gmail.com

檔案

- 內建函數 (function) `open()` ，讀取檔案並回傳檔案串流物件，參數 (parameter) `file` 為檔名字串

函數	描述
<code>open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True)</code>	讀取檔案並回傳檔案串流物件

- mode 有以下數種

'r'	讀取（預設）
'w'	寫入
'a'	附加
'b'	二進位模式
't'	文字模式（預設）
'+'	更新磁碟檔案
'U'	通用新行模式

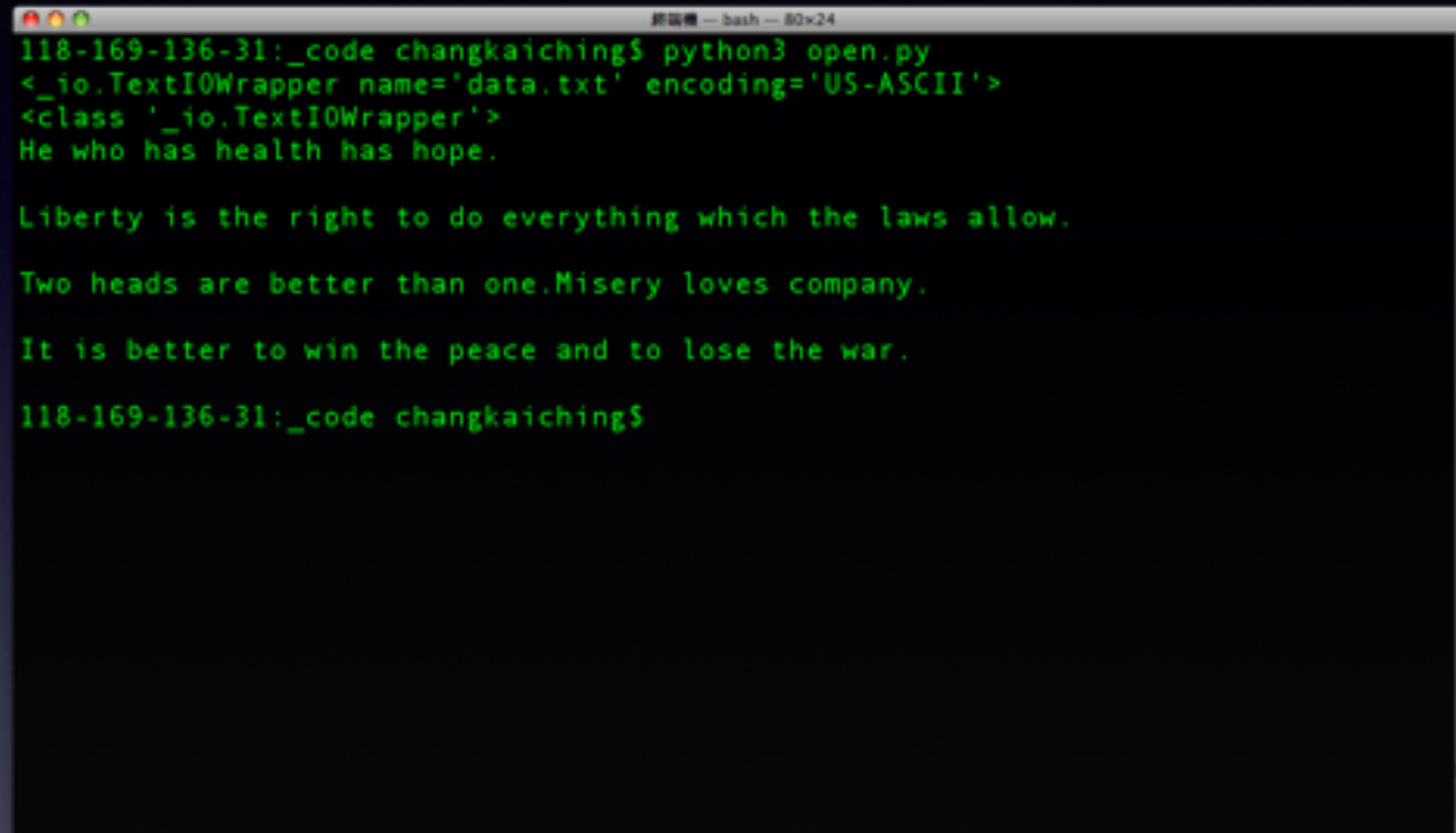
- 例如我們有以下的純文字檔案，檔名為 data.txt

```
1 He who has health has hope.  
2 Liberty is the right to do everything which the laws allow.  
3 Two heads are better than one.Misery loves company.  
4 It is better to win the peace and to lose the war.  
5 God's mill grinds slow but sure.  
6 It takes all sorts to make a world.There are two sides to every question.  
7 Rome was not built in a day.  
8 Self-trust is the first secret of success.  
9 Every man has his taste.  
10 Experience is the extract of suffering.  
11 Imagination is more important than knowledge.  
12 The End
```


- 以下程式讀取 data.txt ，印出檔案物件、檔案物件的型態及 data.txt 的前四行

```
1 f = open("data.txt", "r")
2 print(f)
3 print(type(f))
4 print(f.readline())
5 print(f.readline())
6 print(f.readline())
7 print(f.readline())
```

- 執行結果如下



```
118-169-136-31:_code changkaiching$ python3 open.py
<_io.TextIOWrapper name='data.txt' encoding='US-ASCII'>
<class '_io.TextIOWrapper'>
He who has health has hope.

Liberty is the right to do everything which the laws allow.

Two heads are better than one.Misery loves company.

It is better to win the peace and to lose the war.
118-169-136-31:_code changkaiching$
```

- 用 `open` 開檔之後要執行的動作至少有以下三種：讀檔、寫檔及關檔，分別以 `read()`、`write()`、`close()` 為代表
- 而讀檔有 `read()`、`readline()`、`readlines()` 三種方法

- `read()` 一次可以讀取整份文件，通常會將整份文件的文字放在一個 `string` 裡面，對於文件中“行”的概念處理起來相當麻煩
- `readline()` 跟 `readlines()` 的差異點在於 `readlines()` 跟 `read()` 一樣會一次讀取整份文件，但 `readlines()` 會將文件依行為單位存成 `list` 就可以使用 `for` 迴圈進行處理

- `readline()` 比起 `readlines()` 來說慢的許多，因為 `readline()` 一次只讀取一行
- 如果記憶體的空間足夠建議使用 `readlines()` 一次性讀取完畢

- `write()` 方法可將任何字串寫入目前已經被打開的檔案
- Python 字串可以是二進位資料，而不僅僅是文字內容
- 通常 `write()` 方法不會自動在字串的結尾中添加分行符號“`\n`”，因此要換行則必須自行加上分行符號

模組

- 一個 .py 檔案就是一個模組 (module) ，
例如以下程式

```

1  class Demo:
2      __x = 0
3
4      def __init__(self, i):
5          self.__i = i
6          Demo.__x += 1
7
8      def __str__(self):
9          return str(self.__i)
10
11     def hello(self):
12         print("hello " + self.__str__())
13
14     @classmethod
15     def getX(cls):
16         return cls.__x
17
18 class Other:
19     def __init__(self, k):
20         self.k = k
21
22     def __str__(self):
23         return str(self.k)
24
25     def hello(self):
26         print("hello, world")
27
28     def bye(self):
29         print("Good-bye!", self.__str__())
30
31 class SubDemo(Demo, Other):
32     def __init__(self, i, j):
33         super().__init__(i)
34         self.__j = j
35
36     def __str__(self):
37         return super().__str__() + "+" + str(self.__j)
38
39 a = SubDemo(12, 34)
40 a.hello()
41 a.bye()
42 b = SubDemo(56, 78)
43 b.hello()
44 b.bye()

```


- `cla21.py` 即是一個模組
- 我們可以在其他 Python 程式檔案中使用 `cla21.py` 所定義的類別
- 最簡單的方式就是利用關鍵字 (keyword) `import` 引入 `cla21.py` 的內容，如下

```
import cla21
```

- 引入模組名稱不需要 .py 的副檔名，使用檔名 cla21 即可，引入後便可使用 cla21 所定義的內容，如下

```
1  import cla21
2
3  a = cla21.SubDemo(12, 34)
4  a.hello()
5  a.bye()
6  b = cla21.SubDemo(56, 78)
7  b.hello()
8  b.bye()
```

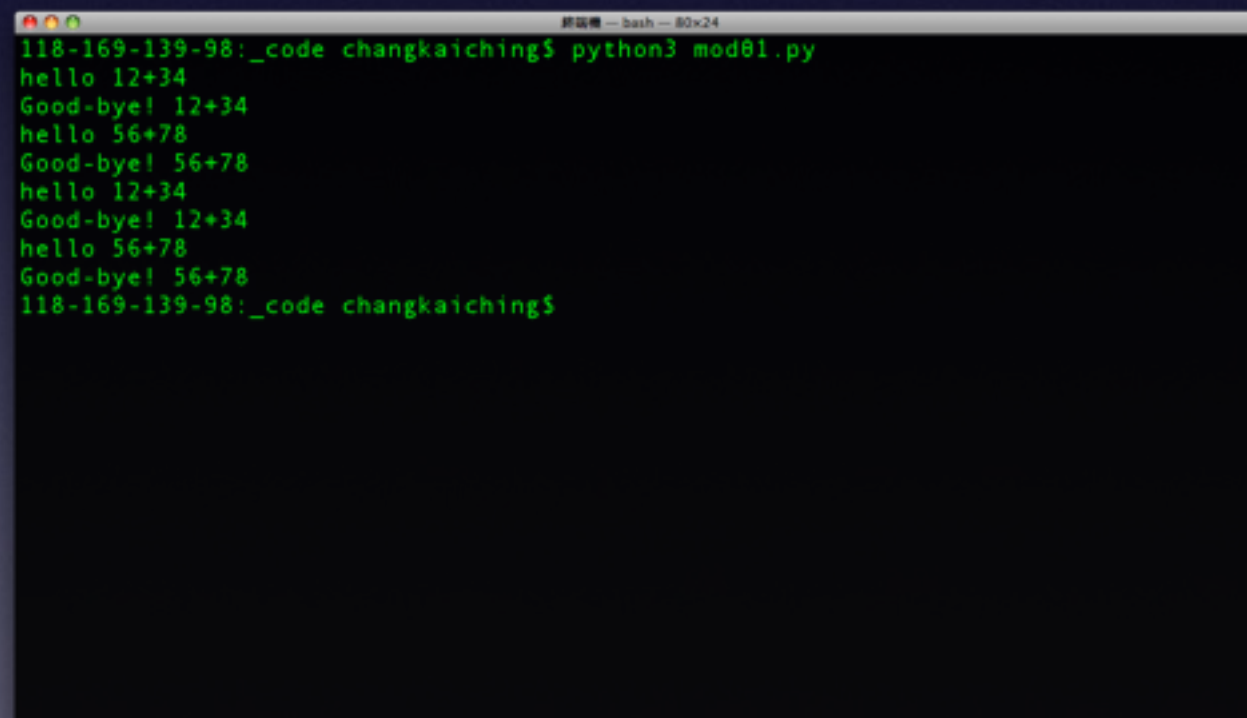
- 注意兩個，首先， mod01.py 與 cla21.py 必須在相同資料夾內，其次，第三行

```
3 | a = cla21.SubDemo(12, 34)
```

- 利用 `import` 引入 `cla21` 所定義的類別 `SubDemo`，須連用模組名稱 `cla21`，以辨明是哪個模組。這裡，若是我們想要直接使用類別 `SubDemo` 的名稱，我們可以連用另一個關鍵字 `from`，例如

```
from cla21 import *
```


- 因此，當引入的名稱數量很多時，星號可以省下許多打字的時間
- `mod01.py` 執行結果如下



```
118-169-139-98:~_code changkaiching$ python3 mod01.py
hello 12+34
Good-bye! 12+34
hello 56+78
Good-bye! 56+78
hello 12+34
Good-bye! 12+34
hello 56+78
Good-bye! 56+78
118-169-139-98:~_code changkaiching$
```

- 總共執行了兩次，這是為什麼呢？
- 因為 `cla2l.py` 有沒有縮排的程式碼，所以執行 `import` 的時候就先執行 `cla2l.py` 中沒有縮排的程式碼

- 如果模組中除了類別、函數或其他的定義外，還具有可執行的程式碼，我們不打算在引入模組後執行這些程式碼的話，當作模組的檔案，於可執行程式碼前應該加入額外 if 陳述 (statement)，如下

```
if __name__ == "__main__":  
    #模組中可執行的程式碼
```

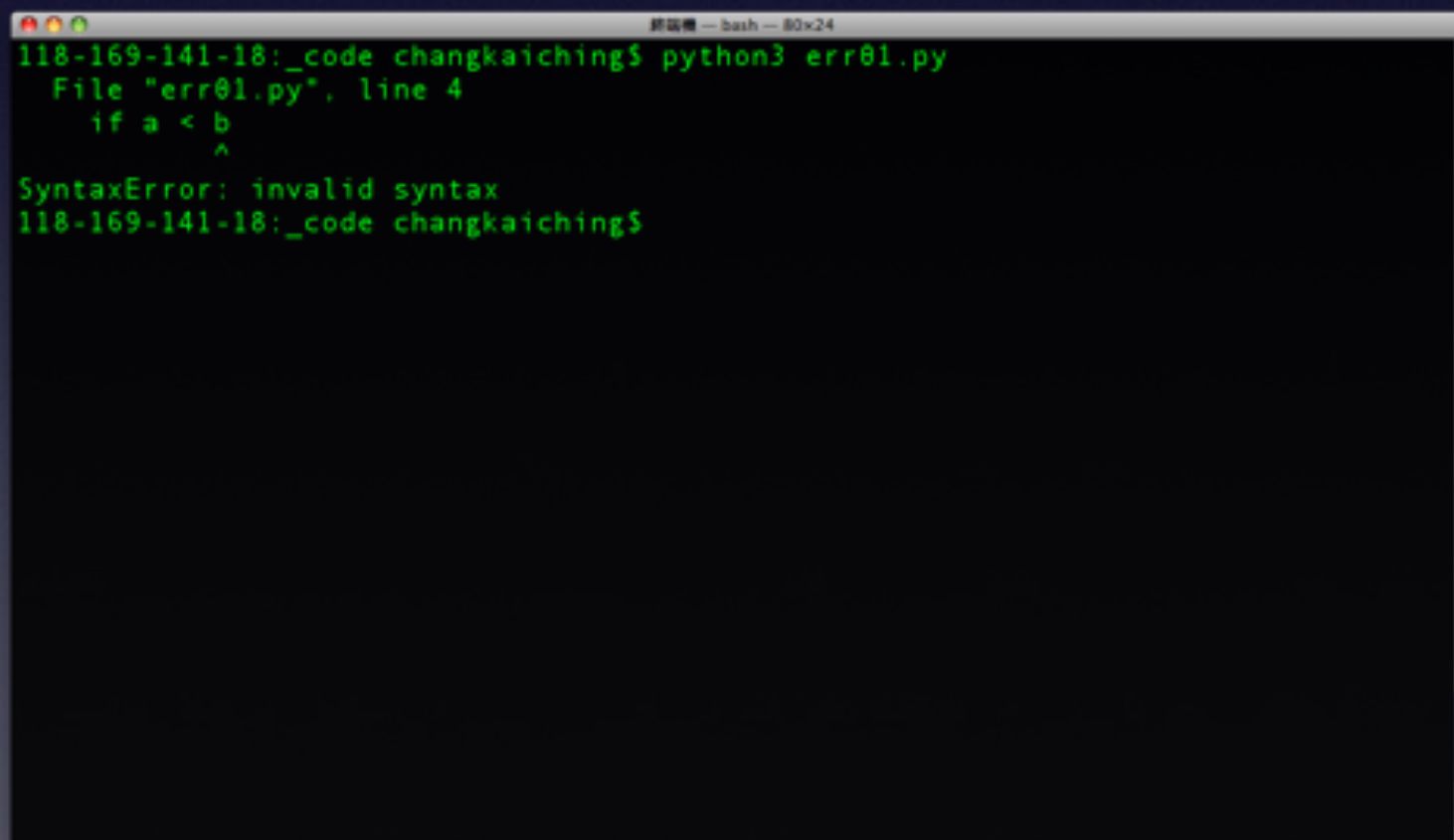
- `__name__` 是模組特別的屬性 (attribute)，當 Python 檔案以直譯器直接執行時，`__name__` 就會被設置成 `"__main__"`
- 因此只有當此模組直接被執行，if 陳述底下的程式碼才會執行
- 若是在其他 Python 檔案中引入該模組，直譯器就會跳過這段 if 陳述

例外

- 寫程式難免會有錯誤 (error) ，例如以下程式

```
1  a = 22
2  b = 33
3
4  if a < b
5      print("Hello world!")
6
```

- 會發生 `SyntaxError` 也就是語法錯誤 (syntax error)



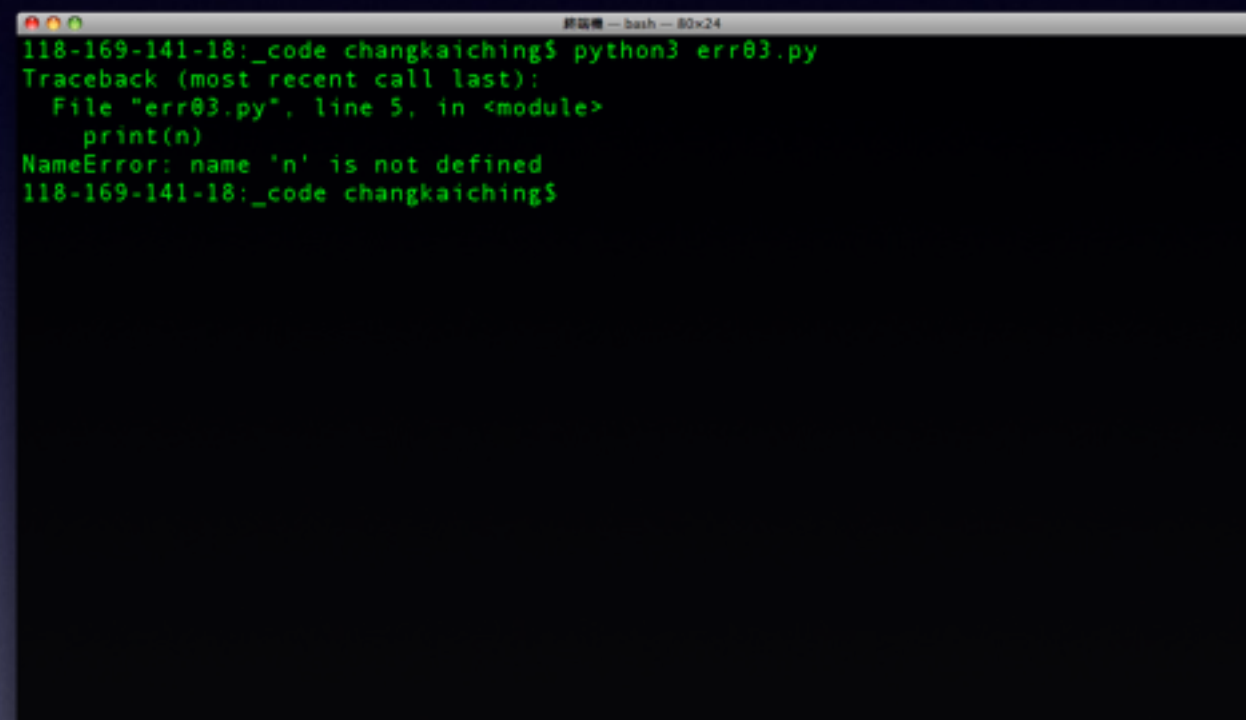
```
118-169-141-18:_code changkaiching$ python3 err01.py
File "err01.py", line 4
    if a < b
        ^
SyntaxError: invalid syntax
118-169-141-18:_code changkaiching$
```

- 這是因為第 4 行

```
4 | if a < b
```

- 我們在 if 陳述 (if statement) 漏打了冒號，直譯器 (interpreter) 在執行時便會直接抓出這個錯誤，然後停止程式的執行

- 執行結果如下

A terminal window with a title bar that reads "終端機 -- bash -- 80x24". The terminal shows a command prompt "118-169-141-18:_code changkaiching\$" followed by the command "python3 err03.py". The output is a traceback: "Traceback (most recent call last):", "File \"err03.py\", line 5, in <module>", "print(n)", and "NameError: name 'n' is not defined". The prompt "118-169-141-18:_code changkaiching\$" appears again at the end.

```
118-169-141-18:_code changkaiching$ python3 err03.py
Traceback (most recent call last):
  File "err03.py", line 5, in <module>
    print(n)
NameError: name 'n' is not defined
118-169-141-18:_code changkaiching$
```


- 程式有可能會發生三種錯誤，語法錯誤是其中之一，有一種語意錯誤 (semantic error)，程式會順利執行，但會得到錯誤的結果。如下例

```
1 sum = 0
2 i = 1
3 while i < 100:
4     sum += i
5     i += 1
6
7 print("1 + 2 + + 3 ... + 98 + 99 + 100 =", sum)
8
```

- 此例中我們想計算 1 到 100 所有正整數的合，正確的結果應該是 5050，可是程式卻算出 4950，為什麼呢？因為第 3 行

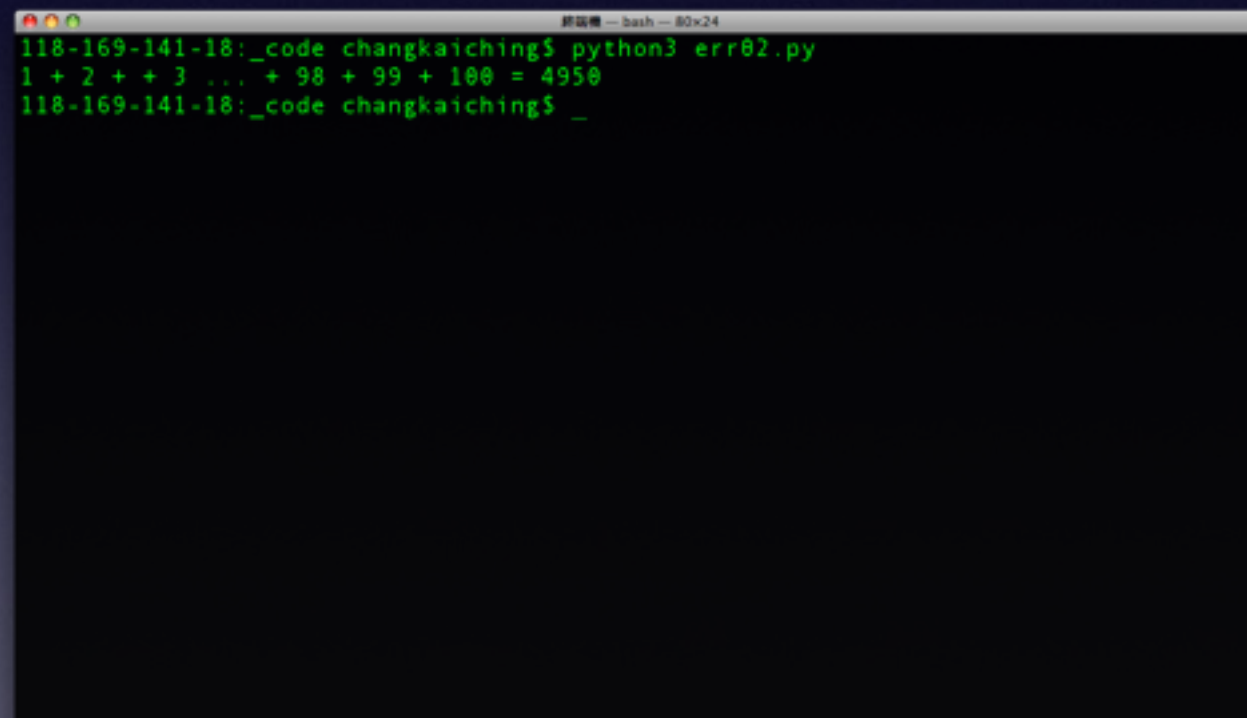
```
3 | while i < 100:
```

- 這裡的條件 (condition) 只會讓 sum 加到 99 為止
- 若 i 等於 100 就會結束迴圈，因此 while 迴圈 (while loop) 的結束條件應該是 $i \leq 100$ 或 $i < 101$

- 語法錯誤會被直譯器直接挑出，而語意錯誤需要依賴程式設計師自己清楚規劃程式的邏輯
- 另外一種錯誤通常在執行中發生錯誤，直譯器會以發起例外 (exception) 的方式終止程式的進行，如下例


```
1 a = 22
2 b = 33
3
4 if a < b:
5     print(n)
```

- 執行結果如下



```
118-169-141-18:~_code changkaiching$ python3 err02.py
1 + 2 + + 3 ... + 98 + 99 + 100 = 4950
118-169-141-18:~_code changkaiching$ _
```

A terminal window with a title bar showing standard window controls and the text "終端機 -- bash -- 80x24". The terminal displays the execution of a Python script named `err02.py`. The output is a single line of text: `1 + 2 + + 3 ... + 98 + 99 + 100 = 4950`. The prompt `118-169-141-18:~_code changkaiching$` appears before and after the command.

- 這裡發生的是 `NameError`，這是名稱錯誤，因為程式中沒有定義過 `n`，第 5 行

```
3 | print(n)
```

- 沒有變數 `n` 的存在，所以直譯器發起 `NameError` 的例外

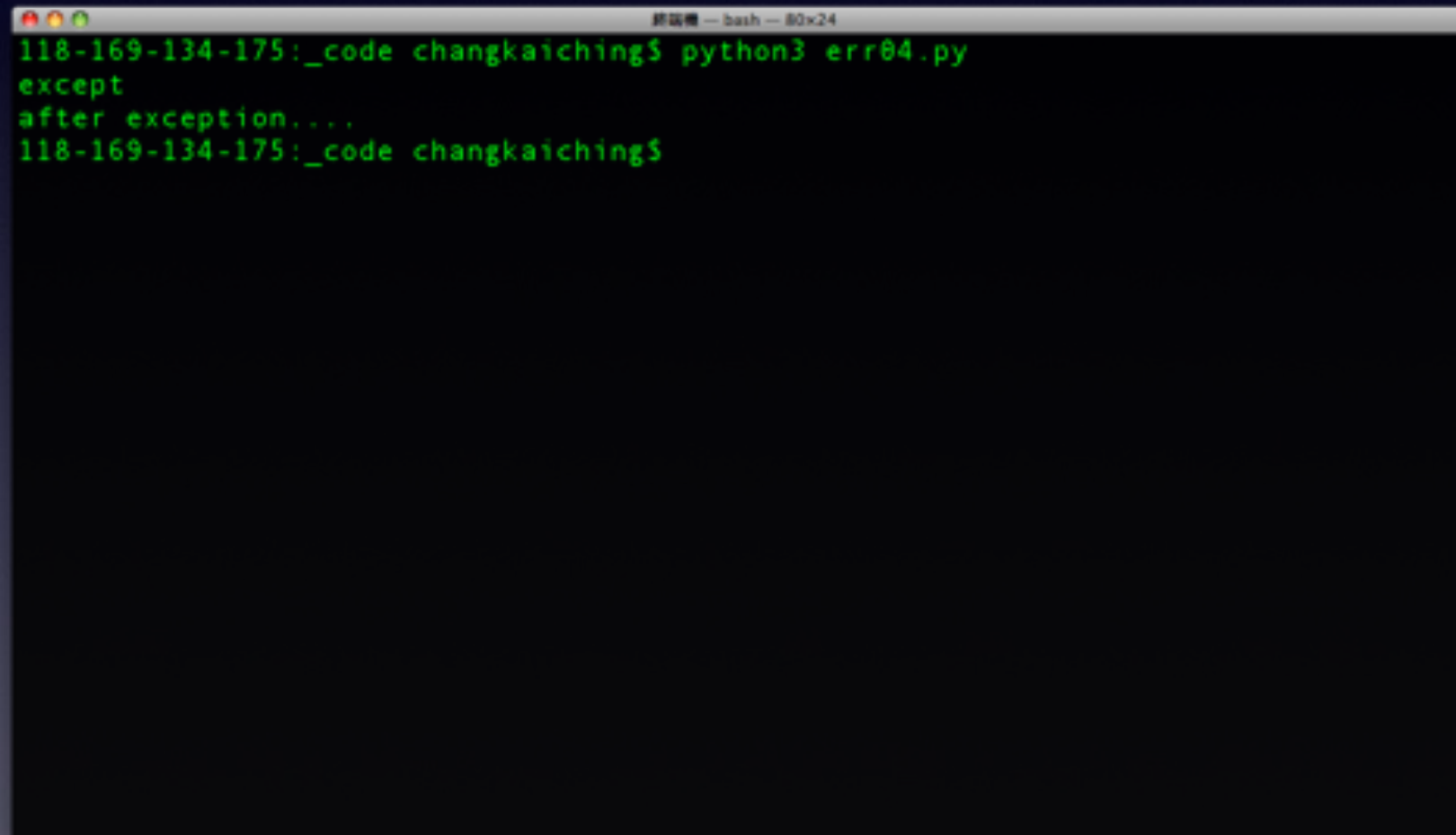
- 大部分執行中的錯誤，Python 直譯器 (interpreter) 會以發起例外 (exception) 的方式來中斷程式的執行
- 實際上，很多情況下我們需要自行控制可能會產生例外的程式碼，因為例外並不全然是程式的邏輯錯誤
- 例如程式中打算開啟檔案，然而實際檔名並不存在，這種情況下，我們需要的是例外發生後的處理動作，而非中止程式的執行

- 凡是可能會產生例外的程式碼，Python 利用 `try-except` 陳述 (`try-except statement`) 讓程式設計師自行處理例外
- `try-except` 為關鍵字 (`keyword`) 之一，專門用來例外處理 (`exception handling`) 的

- 基本形式就是把可能會產生例外的程式碼放在 `try` 之後的程式區塊 (block) ，
`except` 則放例外發生時的處置，如下例

```
1  a = 22
2  b = 33
3
4  try:
5      if a < b:
6          print(n)
7  except:
8      print("except")
9
10 print("after exception...")
```

- 執行後結果如下



```
終端機 — bash — 80x24
118-169-134-175:_code changkaiching$ python3 err04.py
except
after exception....
118-169-134-175:_code changkaiching$
```

- 由於第 6 行，這裡原本會產生 `NameError`，但因為有 `try-except` 的例外處理，所以發生例外是執行 `except` 部份印出 "except" 的字串 (string)
- 如果沒有處理例外，程式執行到

```
6 | print(n)
```

- 就會停止，而最後 "after exception..." 的字串也不會被印出

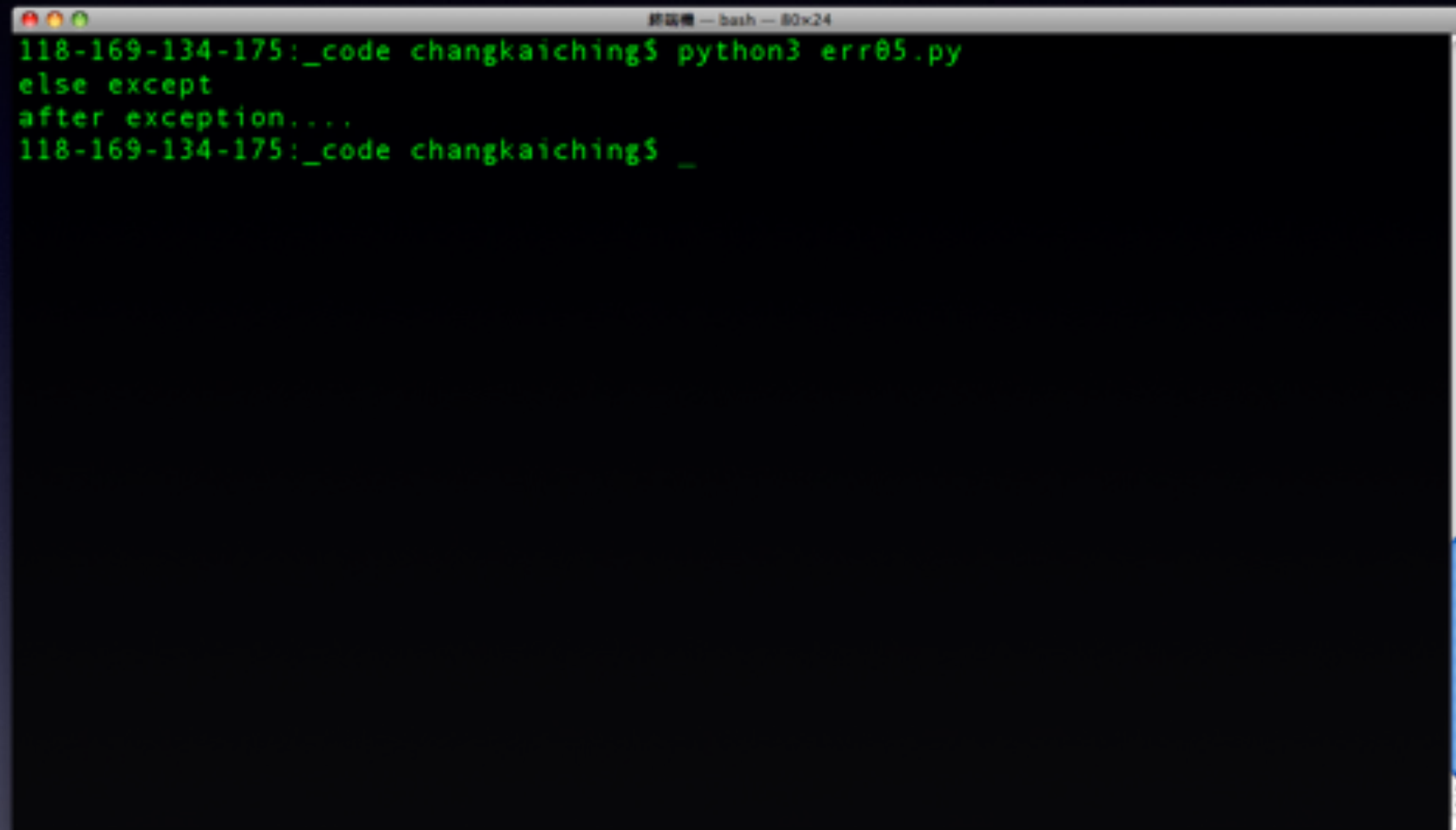
- 例外的名稱也可以寫在 `except` 之後，如

```
7 | except NameError:
```

- 如果一段程式碼有可能會發生多種例外，這樣的寫法可以分別處理不同種類的例外
- `try-except` 也可以和 `else` 連用，`else` 後的程式區塊放的是沒有發生例外，程式所執行的工作，例如

```
1  a = 22
2  b = 33
3
4  try:
5      if a > b:
6          print(n)
7  except:
8      print("except")
9  else:
10     print("else except")
11
12 print("after exception....")
```

- 執行後結果如下



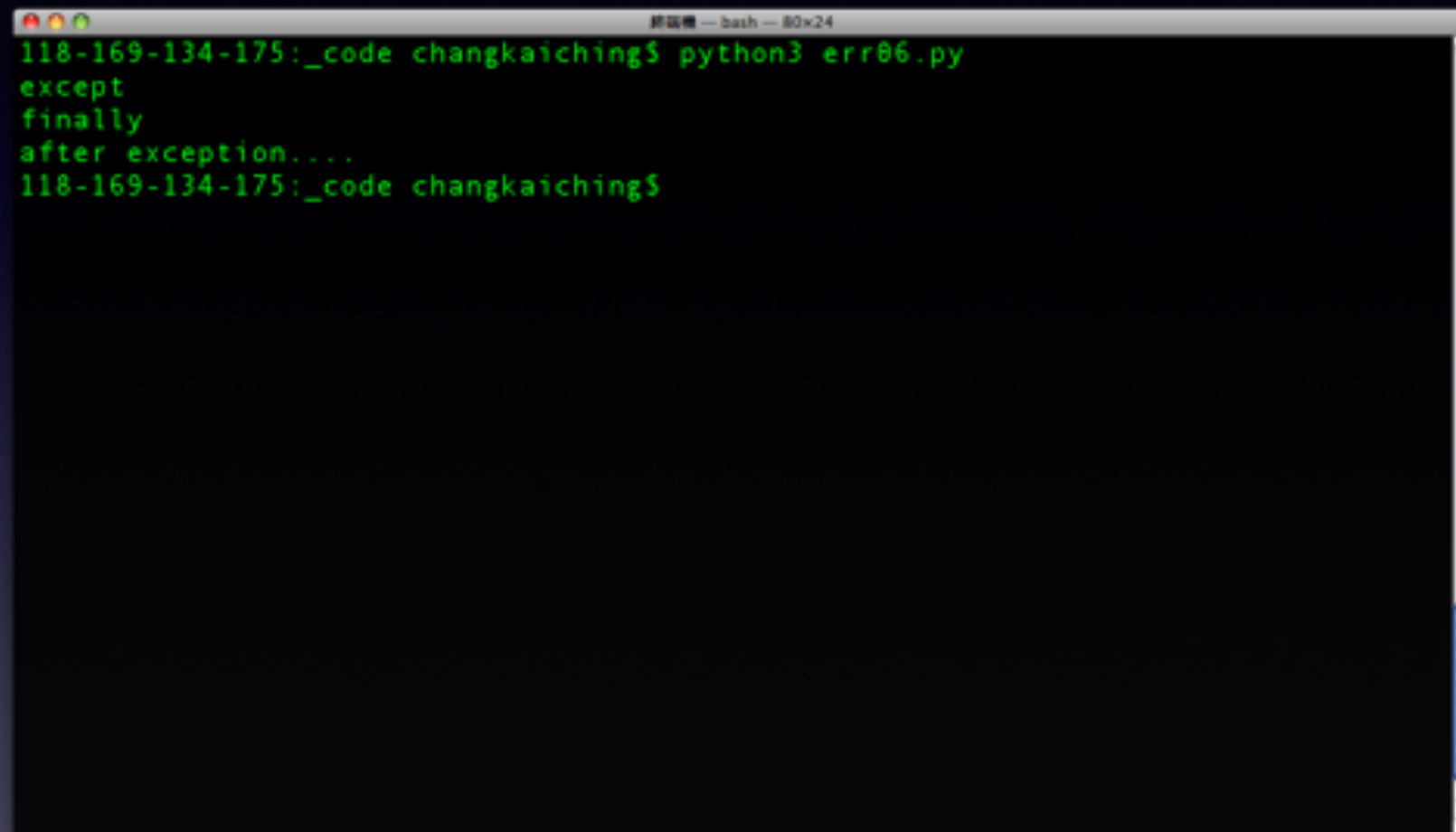
```
118-169-134-175:_code changkaiching$ python3 err05.py  
else except  
after exception....  
118-169-134-175:_code changkaiching$ _
```

A terminal window with a title bar that reads "終端機 — bash — 80x24". The window has a black background with green text. The text shows a command being executed and its output. The output consists of three lines: "else except", "after exception....", and a prompt character "_".

- 這裡因為 $a > b$ 為假，所以例外不會發生，因此程式執行 `else` 的部份
- 若加入另一個關鍵字 `finally`，無論例外有沒有發生都會執行 `finally` 後的程式區塊。例如


```
1  a = 22
2  b = 33
3
4  try:
5      if a < b:
6          print(n)
7  except:
8      print("except")
9  else:
10     print("else except")
11 finally:
12     print("finally")
13
14 print("after exception...")
```

- 執行後結果如下

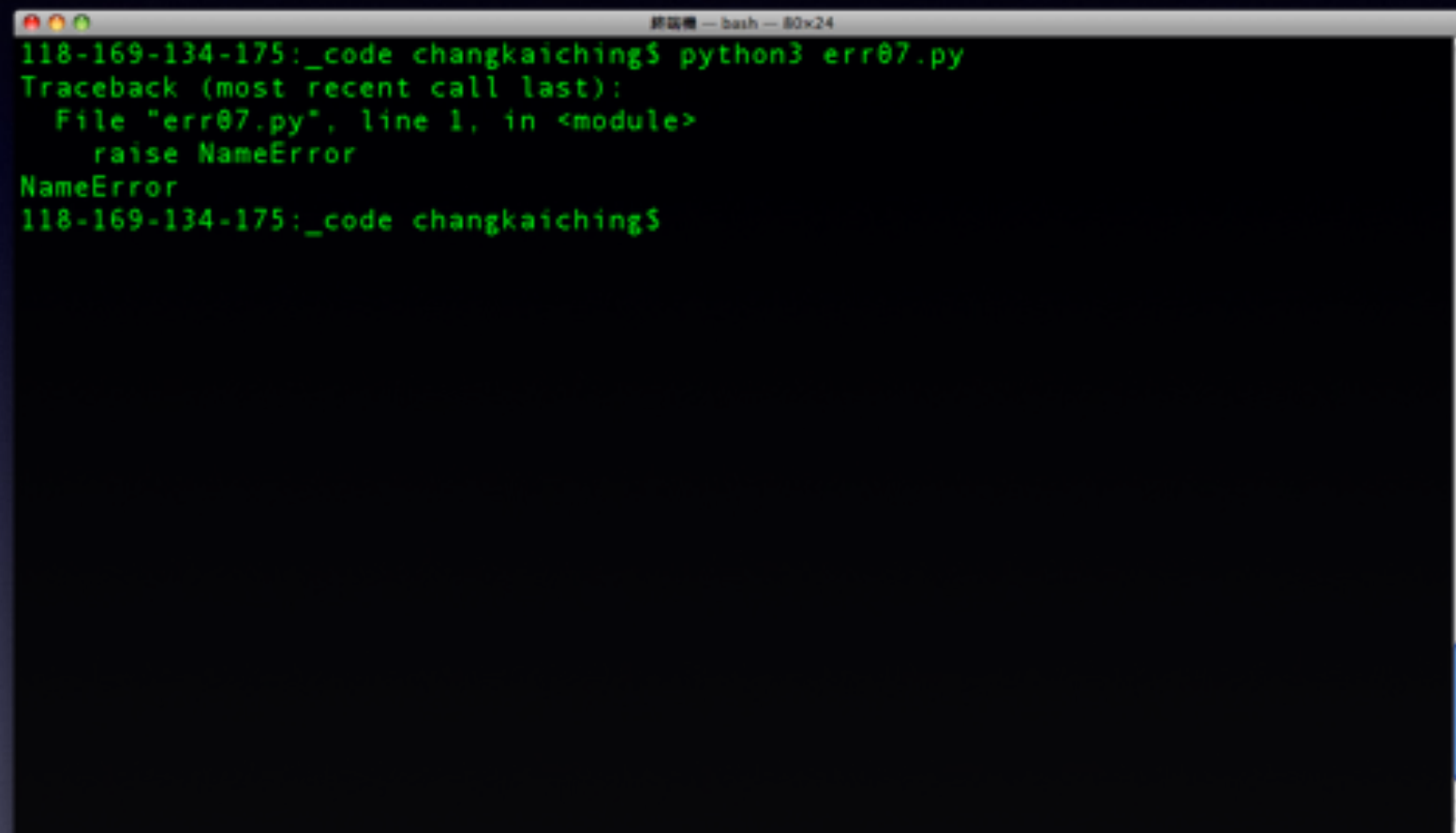


```
終端機 — bash — 80x24
118-169-134-175:_code changkaiching$ python3 err06.py
except
finally
after exception....
118-169-134-175:_code changkaiching$
```

- 例外 (except) 除了會由直譯器 (interpreter) 自動發生外，我們也可以自己在程式中利用 raise 陳述 (raise statement) 引起例外
- raise 為關鍵字 (keyword) 之一，用來引起例外
- 例如，最簡單的形式為單一的 raise 陳述

```
1 raise NameError
```

- 執行後結果如下



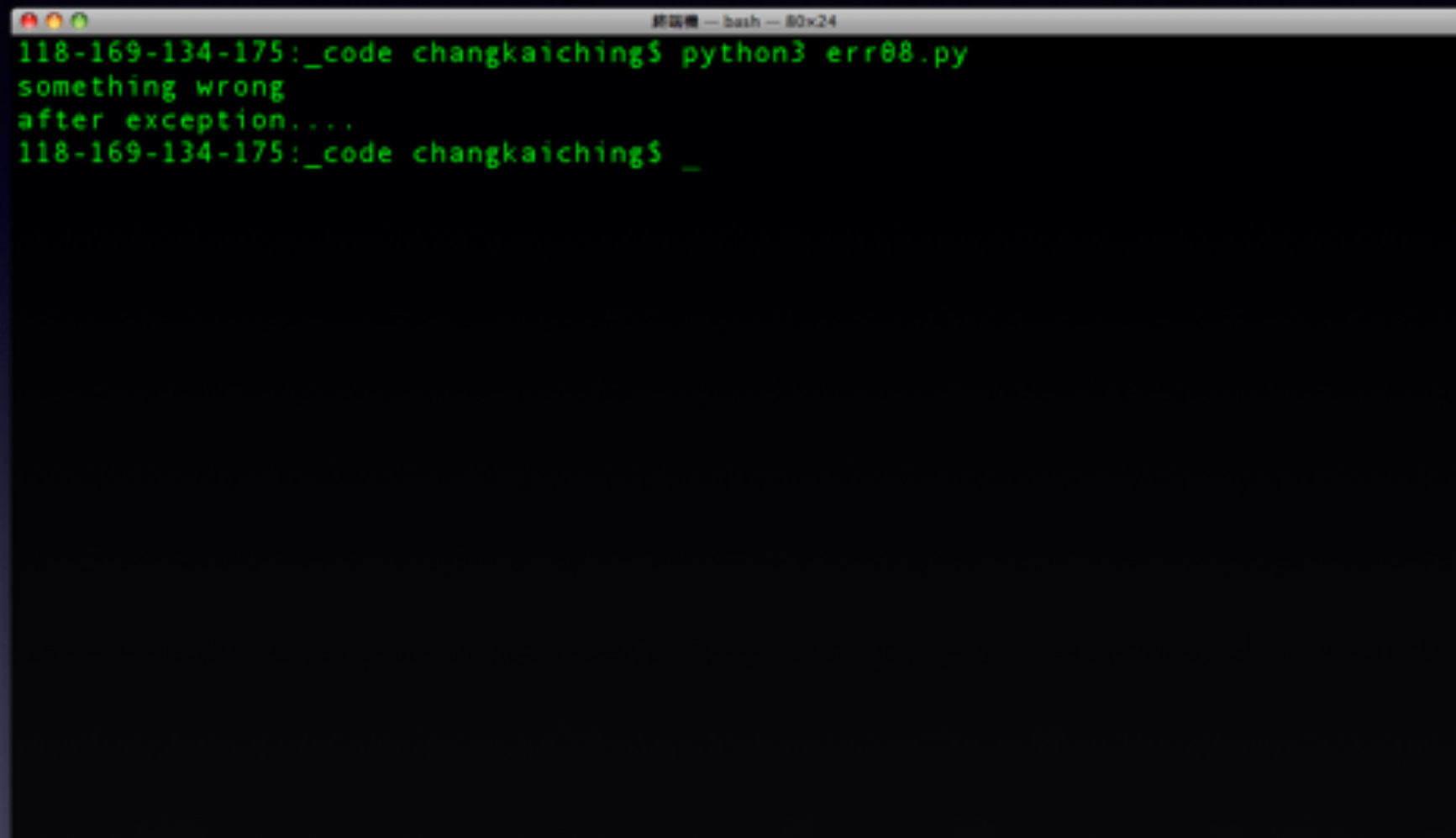
```
118-169-134-175:_code changkaiching$ python3 err07.py
Traceback (most recent call last):
  File "err07.py", line 1, in <module>
    raise NameError
NameError
118-169-134-175:_code changkaiching$
```

A terminal window with a title bar containing window control icons and the text "終端機 — bash — 80x24". The terminal displays the execution of a Python script named `err07.py`. The output shows a `NameError` exception being raised on line 1 of the script. The prompt `118-169-134-175:_code changkaiching$` is shown both before and after the error.

- 若加入 try-except 便可處理例外

```
1  try:
2      raise NameError
3  except NameError:
4      print("something wrong")
5
6  print("after exception....")
```

- 執行後結果如下



```
118-169-134-175:~$ python3 err08.py
something wrong
after exception....
118-169-134-175:~$
```

A terminal window with a title bar containing three colored window control buttons (red, yellow, green) and the text "終端機 — bash — 80x24". The terminal displays the command `python3 err08.py` and its output: `something wrong`, `after exception....`, and a prompt `_`. The prompt is followed by a space and a tilde character.