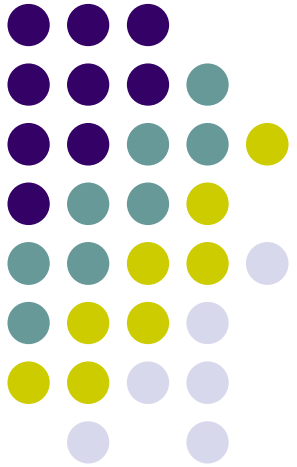
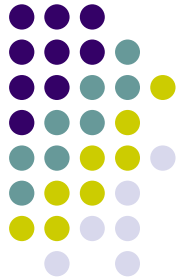
Python程式設計入門
異常

葉難

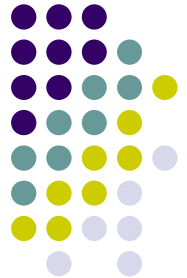


大綱



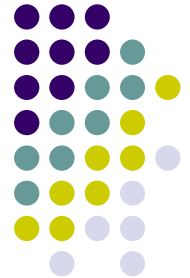
- 錯誤種類
- 異常（exception）處理機制
- try述句，except、else、finally
- raise述句
- 異常型別：內建、自訂
- assert

錯誤種類



- 語法錯誤、打錯字、縮排錯誤
- 想要的行爲跟程式執行結果不同：Python述句語意認知錯誤
- 設想演算法有錯，自然無法得到正確答案
- 靜態、動態、每次都發生、視情況而定
- 執行時各項參數不定，有時正常、有時出錯

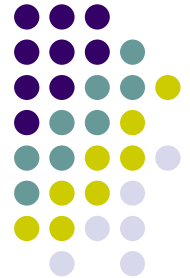
傳統方式：特定回傳值



- 以特定回傳值告知狀況
- 例，-2代表型別不對，-1代表n爲負數

```
def fact(n):
    if type(n) is not int:
        return -2
    if n < 0:
        return -1

    ...計算n!並回傳...
```

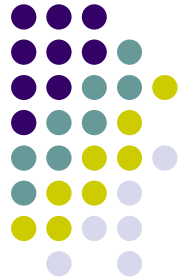
傳統方式：檢查回傳值



- 根據回傳值做相對應的動作

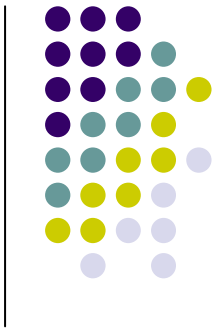
```
while True:
    s = input('input n: ')
    n = fact(int(s))    # int()會引發異常！
    ans = fact(n)
    if ans == -2:
        print('Must input integer')
    elif ans == -1:
        print('Must be non-negative')
    else:
        print('%d! = %d' % (n, ans))
```

困難



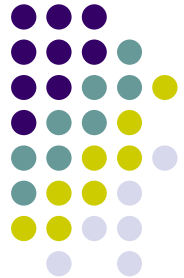
- Happy程式碼很短很清楚，加上處理錯誤的程式碼，變得很長且難懂
- 原本很熟悉的程式執行流程：循序、條件判斷、迴圈、函式；現在要加上另一套
- 「異常處理機制」，另一套程式執行流程
- 處理錯誤情況，原本就很煩：會發生哪些錯誤，要在哪裡處理，該怎麼處理

異常基本知識



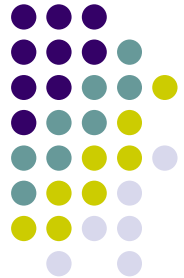
- 發生錯誤無法繼續執行，「引發」異常，終止原本的執行流程
- 從呼叫堆疊一層一層往外跳躍，直到某地方「捕抓」住異常
- 最後若由Python直譯器捕抓，預設行為是印出資訊並終止
- raise述句：引發異常
- try/except述句：捕抓異常

呼叫堆疊



```
def foo(li): li[99]
def bar(li): tmp = foo(li)
bar([0, 1, 2])
####
Traceback (most recent call last):
  File "a.py", line 5, in <module>
    bar([0, 1, 2])
  File "a.py", line 4, in bar
    def bar(li): tmp = foo(li)
  File "a.py", line 3, in foo
    def foo(li): li[99]
IndexError: list index out of range
```

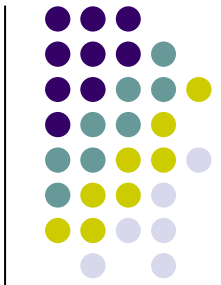
raise述句：引發異常



- 建立異常物件，啓動異常處理機制（流程）
- 語法

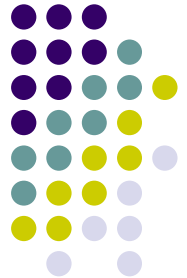
```
raise 異常型別名
raise 異常物件

# 再次引發目前的異常
raise
# 再次引發，鏈接舊異常物件
raise 異常型別名或物件 from 異常型別名或物件
```

內建異常型別



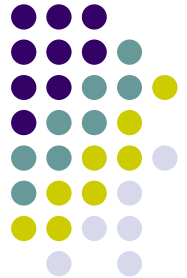
```
BaseException
    SystemExit
    KeyboardInterrupt
    Exception
        StopIteration
        ArithmeticError
        AssertionError
            LookupError
                IndexError
            NameError
```

try述句：捕抓異常



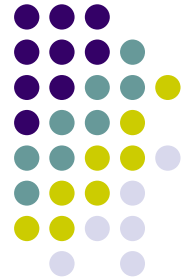
```
try:
    ...可能引發異常的述句...
except 異常型別名【as 名稱】:
    ...處理...
except (異常型別名1,異常型別名 2)【as 名稱】:
    ...處理...
except:
    ...引發異常且上述except子句未能捕抓，執行此處...
else:
    ...若未引發異常，執行此處
finally:
    ...不論如何，一定會執行...
```

範例



- fact_exception_1.py，int()與fact()都會引發ValueError，但原因不同
- fact_exception_2.py，改寫，自行定義異常型別
- fact_exception_2.py，巢狀（雙重）try述句

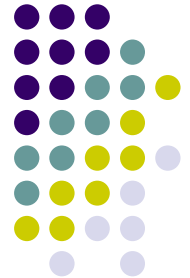
範例：開啓文字檔並印出



- open_file.py
- 檔案不存在、是個目錄、權限不足、解碼錯誤

- 另外：iter()可接受兩個參數，前者是無參數的可呼叫者，若回傳物件等於後者就停止迭代

異常處理流程

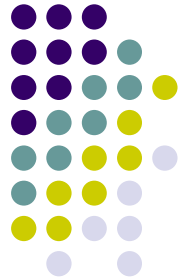


- try述句裡若無異常，執行else子句，然後是finally子句

- 若引發異常，從呼叫堆疊一層一層往外跳躍，直到某try述句的except子句「捕抓」住異常
- 然後執行finally子句（若有的話）
- 然後執行try述句之後的程式碼

- 例：except_foo_bar.py

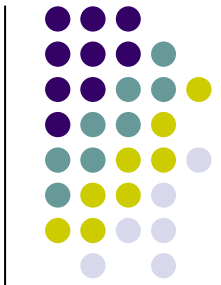
再次引發異常



- except子句處理異常時又再次發生異常
- 被視爲整個try述句引發異常

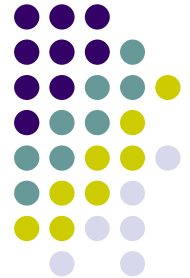
- 自行再次引發異常

範例



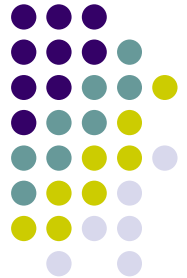
```
def reraise():
    try:
        int('!@#')
    except ValueError as e:
        raise RuntimeError('parse error') from e:
        # print(exc)

reraise()
```

做記錄或警告，繼續讓別人處理異常



```
import warnings
def reraise():
    try:
        int('!@#')
    except ValueError as e:
        warnings.warn('Warning: can not parse')
        raise

reraise()
```

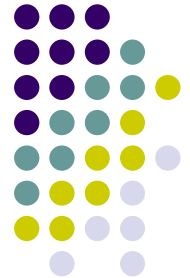
except子句的名稱



- except子句離開時，會刪除as後指派的名稱

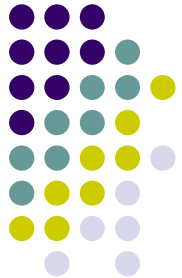
```
try:
    [1, 2, 3][99]
except IndexError as e:
    print(e)    # 離開except子句時，會del e
finally:
    print(e)    # 出錯
print(e)        # 出錯
```

多個**return**，以最後執行的爲準



```
def bar(): raise ValueError
def foo():
    try:
        bar()
        return 'try'
    except ValueError:
        return 'except'
    else:
        return 'else'
    finally:
        return 'finally'

print(foo())
```

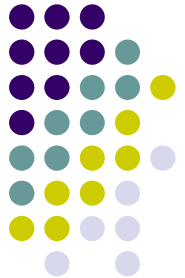
assert述句



- 只會引發AssertionError異常
- 結合if與raise
- 語法，expr2可有可無

```
assert expr1, expr2
#上下寫法相同
if __debug__:
    if not expr1: raise AssertionError(expr2)
```

範例



```
def fact(n):
    assert type(n) is int
    assert n < 0

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

Q&A