3.9.1. Basics

Fig. 3.9.1.1 Photo by Karim MANJRA on Unsplash

Note

Outline

1. Introduction
    a. Syntax
    b. Activity Diagram
    c. Error Types
        i. Exception Hierarchy
        ii. Part of Frequent Errors
        iii. Part of Frequent Warnings
2. Examples
    a. Ex1: Divide Zero
    b. Ex2: Pass Divide Zero
    c. Ex3: else and finally
3. Performance
    a. Ex4: while vs. try...except
        i. Similar Codes
        ii. Comparison: strftime()
        iii. Comparison: timeit()
    b. Ex5: National Exam: 2019

Note

Roadmap

1. This topic: TryExcept

> myBlock
>
> | Exception Handling | | | | |
> |---|---|---|---|---|
> | try Statement | except Statement | finally Statement | Error Type | with Statement |

2. Course: Python 1
3. Subject: Programming
4. Field
    a. Software Engineering (SE)
    b. Computer Science and Information Engineering (CSIE)
    c. Electrical/Electronics Engineering (EE)

1. 正所謂人生不如意事，十之八九！程式碼在執行時，也可能會發生錯誤。
2. 這代表著在設計這支程式的時候，有些地方不甚周詳；又或者是在執行這支程式時，發生了意想不到的錯誤。
3. 在這種時候，如果不希望程式就此終止，就需要在程式裡面將可預期的錯誤另外撰寫程式來預防之，而所使用的指令就是try…except。

3.9.1.1. Introduction

3.9.1.1.1. Syntax

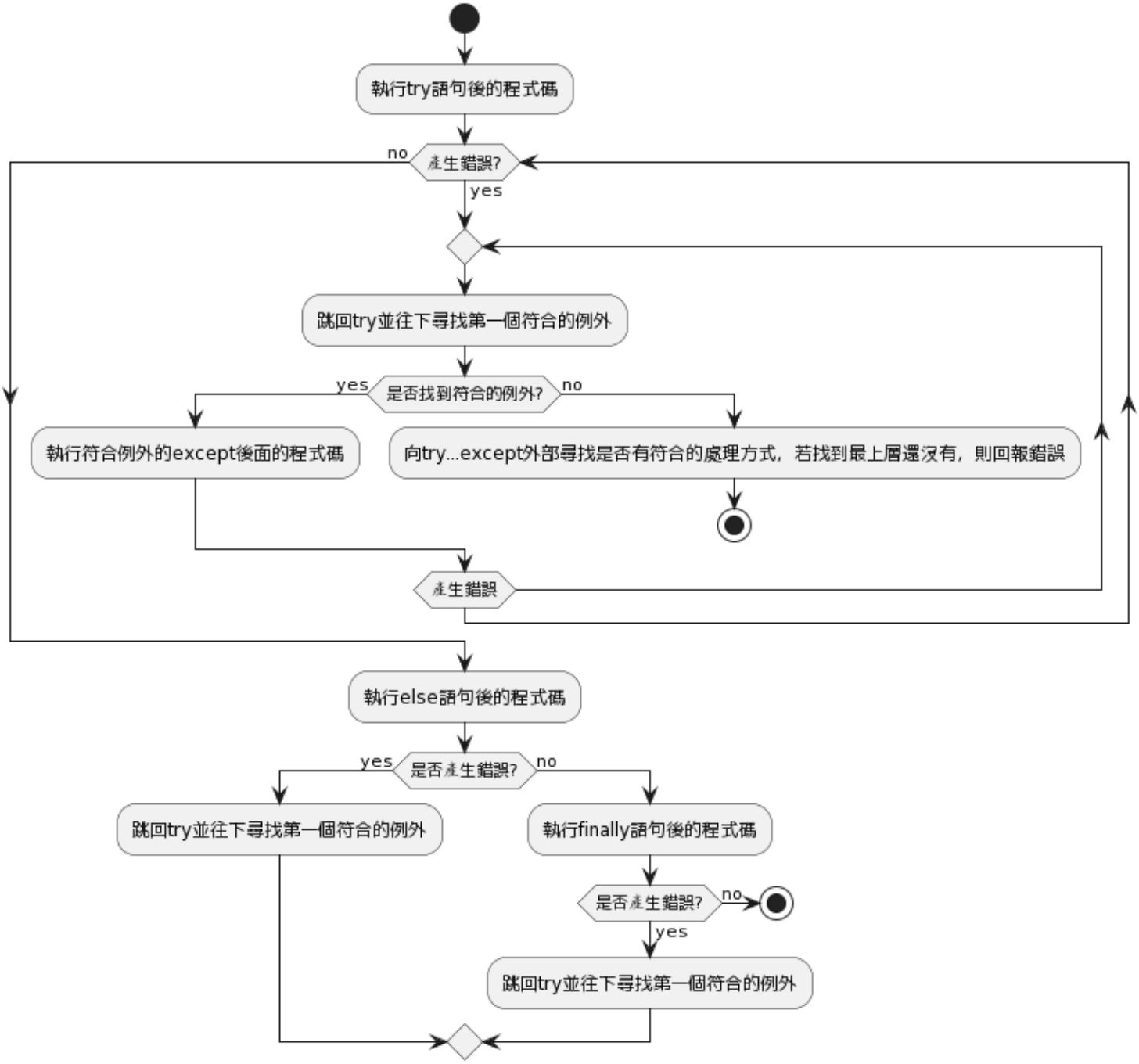
```
try:
    欲執行的程式碼T
except 例外情形1 [, 參數]:
    欲執行的程式碼1
except 例外情形2 [, 參數]:
    欲執行的程式碼2
…
except 例外情形N [, 參數]:
    欲執行的程式碼N
else:
    欲執行的程式碼E
finally:
    欲執行的程式碼F
```

1. 在try…except語法裡面， 至少要存在一個except敘述， [else]跟[finally]則是可選擇要或不要的。
2. try的工作原理是，當Python開始一個try指令後， 就在當前程式的上下文中作標記。
    a. 而當異常出現前， 便會先回到這裡執行try後面的程式碼， 進而預防錯誤的發生。
3. 如果當try後面的程式碼執行時發生異常， Python就跳回到try的起始位置， 並執行第一個符合該異常的except敘述。
4. 等異常處理完畢後， 程式流程就執行整個try區塊中的程式碼， 除非在處理異常時又引發新的異常。

5. 如果在try後面的程式碼裡發生了異常， 卻沒有匹配的except敘述， 異常將被轉送到上一層的try區塊處理， 或者到程式的最上層。
6. 若是遇到後者情況，將結束程式，並印出錯誤資訊。
7. 如果在try之後的程式碼執行並無引發異常， 如果有[else]的區塊內容的話， Python 便會去執行[else]區塊內部的程式碼， 然後再執行[finally]區塊內部的程式碼， 如此便算順利執行完整個try…except指令區。

3.9.1.1.2. Activity Diagram

1. 值得注意的是，不管有沒有發生異常，finally: 後面的程式碼都會被執行。



3.9.1.1.3. Error Types

1. Python 有內建相當多的錯誤類型，每一種對應的情況都不同，像上面範例所示範的ZeroDivisionError便是專門對應0 在除數的除法錯誤。
2. 另外，還有對應type錯誤的TypeError、對應導入模組錯誤的ImportError 等等，大部分都可以從錯誤名稱看出錯誤的原因。

3.9.1.1.3.1. Exception Hierarchy

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
      |    |    +-- ConnectionResetError
      |    +-- FileExistsError
      |    +-- FileNotFoundError
      |    +-- InterruptedError
      |    +-- IsADirectoryError
      |    +-- NotADirectoryError
      |    +-- PermissionError
      |    +-- ProcessLookupError
      |    +-- TimeoutError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      |    +-- RecursionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
```

```
64            +-- EncodingWarning
65            +-- ResourceWarning
```

See also

Exception Hierarchy, Python Documentation

1. Built-in Exceptions: Exception hierarchy, Python Documentation, v3.11.5. [Web]

3.9.1.1.3.2. Part of Frequent Errors

1. 在Error 之中還有所謂的母子關係，也就是母Error 跟子Error。
2. 舉例來說，在使用除法時不小心將0 放在分母， 會出現ZeroDivisionError，而這個ZeroDivisionError 就是ArithmeticError 的子錯誤
3. 要清楚知道這些母子關係的目的是因為如果在try...except的錯誤捕捉沒有辦法做到很細的分類， 那可以在錯誤分類時，使用較大的母Error 來作為捕捉條件。
4. 它可以包含比較多的錯誤，讓程式碼在執行時， 不會因為沒有定義到較為精細的Error，而導致沒有做到完善的錯誤處理動作。
5. 以下列出Python常見的錯誤類型。

錯誤名稱	描述
ArithmeticError	所有數值計算錯誤的基礎類型
AssertionError	斷言語句失敗
AttributeError	物件沒有這個屬性
BaseException	所有異常的基礎類型
EOFError	沒有內建輸入,到達EOF(End Of File) 標記
EnvironmentError	作業系統錯誤的基礎類型
Exception	常規錯誤的基礎類型
FloatingPointError	浮點計算錯誤
GeneratorExit	generator發生異常來通知退出
ImportError	導入模組/對象失敗
IndentationError	縮排錯誤
IndexError	序列中沒有此索引(index)
IOError	輸入/輸出操作失敗

錯誤名稱	描述
KeyboardInterrupt	用戶中斷執行(通常是輸入^C)
KeyError	集合中沒有這個鍵
LookupError	無效數據查詢的基礎類型
MemoryError	內存溢出錯誤(對於Python 編譯器不是致命的)
NameError	未宣告/初始化對象(沒有屬性)
NotImplementedError	尚未實作的方法
OSError	作業系統錯誤
OverflowError	數值運算超出最大限制
ReferenceError	Weak reference試圖訪問已經垃圾回收了的對象
RuntimeError	一般的執行階段錯誤
StandardError	所有的內建標準異常的基礎類型
StopIteration	疊代器沒有更多的值
SyntaxError	Python 語法錯誤
SystemError	一般的編譯器系統錯誤
SystemExit	Python編譯器請求退出
TabError	Tab和空格混用
TypeError	對類型無效的操作
UnboundLocalError	訪問未初始化的本地變數
UnicodeError	Unicode相關的錯誤
UnicodeDecodeError	Unicode解碼時的錯誤
UnicodeEncodeError	Unicode編碼時錯誤
UnicodeTranslateError	Unicode轉換時錯誤
ValueError	傳入無效的參數

錯誤名稱	描述
WindowsError	系統調用失敗
ZeroDivisionError	除(或取餘數)零(所有資料類型)

3.9.1.1.3.3. Part of Frequent Warnings

1. 以下列出Python常見的警告類型。

警告名稱	描述
DeprecationWarning	關於被棄用的特徵的警告
FutureWarning	關於構造將來語義會有改變的警告
OverflowWarning	溢位警告
PendingDeprecationWarning	關於特性將會被廢棄的警告
RuntimeWarning	可疑的runtime behavior警告
SyntaxWarning	可疑的語法的警告
UserWarning	用戶代碼生成的警告
Warning	警告的基礎類型

3.9.1.2. Examples

3.9.1.2.1. Ex1: Divide Zero

1. Source

Listing 3.9.1.2.1.1 /src/ExceptionHandling/DivideZero/__init__.py

```
i = 10
for j in range(3, -4, -1):
    print("%d / %d = %0.3f" %(i, j, i / j))
```

2. Output

```
10/3=3.333
10/2=5.000
10/1=10.000
Traceback (most recent call last):
   File "C:\examples\Ch9\ex09_05.py", line 3, in <module>
   print("%d / %d = %0.3f" %(i, j, i / j))
ZeroDivisionError: division by zero
```

3.9.1.2.2. Ex2: Pass Divide Zero

1. Source

Listing 3.9.1.2.2.1 /src/ExceptionHandling/ExceptionHandling/__init__.py

```
i = 10
for j in range(3, -4, -1):
    try:
        print("%d / %d = %0.3f" %(i, j, i / j))
    except ZeroDivisionError:
        print("除數為0，無法進行除法。")
```

2. Output

```
10 / 3 = 3.333
10 / 2 = 5.000
10 / 1 = 10.000
除數為0，無法進行除法。
10 / -1 = -10.000
10 / -2 = -5.000
10 / -3 = -3.333
```

3. 從以上的第一個示範程式碼可以看到，在沒有使用try…except來避免錯誤發生時，把0 當作除數就會產生「ZeroDivisionError」的錯誤；但是第二個示範程式碼有使用try…except 將此錯誤預先抓出，如此後面的程式碼也能順利執行下去。
4. 在except 語句中，如果沒有加上錯誤的類型，則所產生任何錯誤都會被歸類到某個特定的except 語句來處理。
5. 好處是沒有錯誤會被遺漏，程式一定可以正常執行完畢；壞處是如此一來，程式設計人員和使用者便無法得知到底發生什麼錯誤，也就沒有辦法輕易地修改程式碼來避免錯誤的發生。因此，菲絲恩建議最好不要把except 後面的

   錯誤類型留白。

3.9.1.2.3. Ex3: else and finally

1. 終於到了兩人要分離的時候。為了確保普羅能夠安全地回到現實世界，在施法過程中是不允許被中斷的。
2. 因此，菲絲恩展開了一道結界，用以保護法術能夠安全地被施展完畢。
3. Source

Listing 3.9.1.2.3.1 /src/ExceptionHandling/ElseFinally/__init__.py

```
try:
    print("進入try區塊")
    for i in range(5, -5, -1):
        print("10 / " + str(i) + " = ", end = "")
        print(str(10 / i))
except:
    print("發生錯誤，進入except區塊")
    print("(except) i = " + str(i))
else:
    print("其他錯誤，進入else區塊")
    print("(else) i = " + str(i))
finally:
    print("進入finally 區塊")
    print("(finally) i = " + str(i))

```

4. Output

```
進入try 區塊
10 / 5 = 2.0
10 / 4 = 2.5
10 / 3 = 3.3333333333333335
10 / 2 = 5.0
10 / 1 = 10.0
10 / 0 = 發生錯誤，進入except 區塊
(except) i = 0
進入finally 區塊
(finally) i = 0
```

3.9.1.3. Performance

3.9.1.3.1. Ex4: while vs. try...except

1. Consider the following program,

3.9.1.3.1.1. Similar Codes

1. If we input the following test data to the following programs,

```
Andy
World
Everyone
```

2. Their outcomes are shown as below.

```
hello, Andy
hello, World
hello, Everyone
```

3. Source 1: sys.stdin.readline()

Listing 3.9.1.3.1.1.1 /src/ExceptionHandling/PerformanceHello/__init__.py

```
'''
Created on  20171207
@author:    cph
@note:      PyDev Remote Debugger
                http://www.pydev.org/manual_adv_remote_debugger.html
            pip install pydevd

@note:      ZeroJudge a001-哈囉
                https://zerojudge.tw/ShowProblem?problemid=a001
    內容 :
        學習所有程式語言的第一個練習題
        請寫一個程式，可以讀入指定的字串，並且輸出指定的字串。
    輸入說明：
        輸入指定的文字
    輸出說明
        輸出指定的文字
    範例輸入
        world
        C++
        mary
    範例輸出
        hello, world
        hello, C++
        hello, mary
```

```python
'''
@ref:       ZeroJudge高中生程式解題系統
                https://zerojudge.tw/
'''
import sys                                                  # 匯入系統模組

#-------------------------------------------------------------------------------
# 函數
#-------------------------------------------------------------------------------
def Calc(sInput):                                           # 定義函數
    sOutput = "hello, " + sInput                            # 組合輸出字串
    return(sOutput)                                         # 回傳輸出字串

#-------------------------------------------------------------------------------
# 主程式
#-------------------------------------------------------------------------------
sInput = sys.stdin.readline()                               # 讀第1行的輸入資料
while (sInput != ""):                                       # 判斷輸入資料是否為空字串
    sInput = sInput.replace("\r", "").replace("\n", "")     # 移除字串中的跳行字元, \n
    print(Calc(sInput))                                     # 列印呼叫函數後的回傳值
    sInput = sys.stdin.readline()                           # 讀第2行以後的輸入資料

```

Important

\r vs. \n

Code 1 | Input | Output

Listing 3.9.1.3.1.1.2 /src/ExceptionHandling/PerformanceHello/Hello_TeachUsage.py

```
import sys

def Calc(sInput):
    sOutput = "hello, " + sInput + "!"
    return(sOutput)

sInput = sys.stdin.readline()
while (sInput != ""):
    print("1: [%d][%s]" %(len(sInput), sInput))
    sInput = sInput.replace("\r", "")
    print("2: [%d][%s]" %(len(sInput), sInput))
    sInput = sInput.replace("\n", "")
    print("3: [%d][%s]" %(len(sInput), sInput))
    print(Calc(sInput))
    sInput = sys.stdin.readline()
```

Code 2: More detail | Input | Output

Listing 3.9.1.3.1.1.3 /src/ExceptionHandling/PerformanceHello2/Hello2_TeachUsage.py

```
import sys

def Calc(sInput):
    sOutput = "hello, " + sInput + "!"
    return(sOutput)

sInput = sys.stdin.readline()
while (sInput != ""):
    print("1: [%d][%s]" %(len(sInput), sInput))
    print("1.n: [%d][%d][%s]" %(len(sInput), sInput.find("\n"), sInput))
    print("1.r: [%d][%d][%s]" %(len(sInput), sInput.find("\r"), sInput))
    sInput = sInput.replace("\r", "")
    print("2: [%d][%s]" %(len(sInput), sInput))
    sInput = sInput.replace("\n", "")
    print("3: [%d][%s]" %(len(sInput), sInput))
    print(Calc(sInput))
    sInput = sys.stdin.readline()
```

4. Source 2: try/except with function

Listing 3.9.1.3.1.1.4 /src/ExceptionHandling/PerformanceHello2/__init__.py

```
#-------------------------------------------------------------------------------
# 函數
#-------------------------------------------------------------------------------
def Calc(sInput):                       # 定義函數
    sOutput = "hello, " + sInput        # 組合輸出字串
    return(sOutput)                     # 回傳輸出字串

#-------------------------------------------------------------------------------
# 主程式
#-------------------------------------------------------------------------------
```

```
try:
    while(1):
        print(Calc(input()))          # 列印呼叫函數後的回傳值
except (EOFError):
    pass
```

5. Source 3: try/except

Listing 3.9.1.3.1.1.5 /src/ExceptionHandling/PerformanceHello3/__init__.py

```
try:
    while(1):
        print("hello, " + input())
except (EOFError):
    pass

```

3.9.1.3.1.2. Comparison: strftime()

1. Add Time Logs: datetime.datetime.now().strftime
2. Code+Output

Code | Output

Listing 3.9.1.3.1.2.1 /src/ExceptionHandling/PerformanceHello3/Hello3_TryExcept_TeachUsage.py

```
import datetime

print("1: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))
try:
    print("2: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))
    while(1):
        print("3: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))
        print("hello, " + input())
        print("4: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))

except (EOFError):
    print("5: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))
    pass
    print("6: " + datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f'))


```

3.9.1.3.1.3. Comparison: timeit()

1. Code+Output: sys.stdin.readline() with function

Code | Output

```
hello, world
hello, C++
hello, mary
hello, world
hello, C++
hello, mary
...
hello, world
hello, C++
hello, mary
```

```
timeit[stdin][FileOpen]: 0.041614399990066886
timeit[stdin][FileClose]: 0.04242349999549333
timeit[stdin][Fun]: 0.005667999997967854
timeit[stdin][All]: 15.19845080000232
timeit[stdin][Stdin]: 15.108744900018792
```

2. Code+Output: try/except with function

Code | Output

```
hello, world
hello, C++
hello, mary
hello, world
hello, C++
hello, mary
...
hello, world
hello, C++
hello, mary
timeit[except][Fun]: 0.0048540999996475875
timeit[except][File]: 0.004684800005634315
timeit[except][All]: 13.66475770001125
timeit[except][Try]: 13.655218800005969
```

3. Code+Output: try/except without function

Code | Output

```
hello, world
hello, C++
hello, mary
hello, world
hello, C++
hello, mary
...
hello, world
hello, C++
hello, mary
timeit[except][File]: 0.004902800006675534
timeit[except][All]: 13.339870199997677
timeit[except][Try]: 13.334967399991001
```

Important

Summary: stdin vs. try/except

1. It shows that the following situation for Python v3.11.5.
    a. sys.stdin is the slowest one.
    b. try/except is the fastest one.
    c. Even try/except with function is also faster than sys.stdin method.
2. Our comparison still needs to be fine tune, Because of our limited maximum execution count.

3.9.1.3.2. Ex5: National Exam: 2019

1. 20190721【考選部特考考題：檢察事務官電子資訊組】
2. Question+Answer

> **Question** | **Python Code 1** | **Python Code 2** | **Answer**
>
> 請問以下兩個Python程式，
> 若分別輸入Andy, World, Everyone後，
> (1)其執行後的輸出分別為何?
> (2)若兩個程式都可以被正常執行，請說明其執行方式與優缺點?

Important

Summary: Exception Handling

1. Here is a summary of built-in Python functions related to try/except error handling.

Build-in	Description
try	Defines a block of code to test for errors.
except	Defines a block of code to execute if an error occurs in the try block. Can specify the error type(s) to handle.
else	Optional block executed if no error occurred in try block.
finally	Optional block that executes after try/except regardless of error.
raise	Manually raises an exception.
assert	Verify condition is true, raise AssertionError if not.
hasattr()	Check if an object has a given attribute.
getattr()	Get attribute from an object if present.
callable()	Check if an object is callable like a function.
eval()	Evaluate a dynamically created Python expression.
repr()	Return printable representation of object for debugging.

2. The try/except construct allows gracefully handling exceptions in Python.
3. finally always runs after try/except to release resources.
4. Built-ins like hasattr(), repr() are useful in try/except for introspection and diagnostics.
5. Proper use of try/except/finally blocks is important for robust error handling.

Note

1. Start: 20170719
2. System Environment:

Listing 3.9.1.3.2.3 requirements.txt

```
sphinx==7.1.2                           # Sphinx
graphviz>=0.20.1                        # Graphviz
sphinxbootstrap4theme>=0.6.0            # Theme: Bootstrap
sphinx-material>=0.0.35                 # Theme: Material
sphinxcontrib-plantuml>=0.25            # PlantUML
sphinxcontrib.bibtex>=2.5.0             # Bibliography
sphinx-autorun>=1.1.1                   # ExecCode: pycon
sphinx-execute-code-python3>=0.3        # ExecCode
btd.sphinx.inheritance-diagram>=2.3.1   # Diagram
sphinx-copybutton>=0.5.1                # Copy button
sphinx_code_tabs>=0.5.3                 # Tabs
sphinx-immaterial>=0.11.3               # Tabs

#-----------------------------------------------------
#-- Library Upgrade Error by Library Itself
#   >> It needs to fix by library owner
#   >> After fixed, we need to try it later
#-----------------------------------------------------
pydantic==1.10.10                       # 2.0: sphinx compiler error, 20230701

#-----------------------------------------------------
#-- Minor Extension
#-----------------------------------------------------
sphinxcontrib.httpdomain>=1.8.1         # HTTP API

#sphinxcontrib-blockdiag>=3.0.0         # Diagram: block
#sphinxcontrib-actdiag>=3.0.0           # Diagram: activity
#sphinxcontrib-nwdiag>=2.0.0            # Diagram: network
#sphinxcontrib-seqdiag>=3.0.0           # Diagram: sequence

#-----------------------------------------------------
#-- Still Wait For Upgrading Version
#-----------------------------------------------------

#-----------------------------------------------------
#-- Still Under Testing
#-----------------------------------------------------
#numpy>=1.24.2                          # Figure: numpy

#-----------------------------------------------------
#-- NOT Workable
#-----------------------------------------------------
#sphinxcontrib.jsdemo==0.1.4    # ExecCode: Need replace add_js_file()
#jupyter-sphinx==0.4.0          # ExecCode: Need gcc compiler
#sphinxcontrib.slide==1.0.0     # Slide: Slideshare
#hieroglyph==2.1.0              # Slide: make slides
#matplotlib>=3.7.1              # Plot: Need Python >= v3.8
#manim==0.17.2                  # Diagram: scipy, numpy need gcc
#sphinx_diagrams==0.4.0         # Diagram: Need GKE access
#sphinx-tabs>=3.4.1                     # Tabs: Conflict w/ sphinx-material
```