LẬP TRÌNH PYTHON TỪ CƠ BẢN ĐẾN NÂNG CAO

Thực hiện: TS.Trần Bình Long

File

1. Làm việc với File
2. Quản lý File và thư mục
3. Error và Exception
4. Xử lý ngoại lệ - Exception Handling

Làm việc với File

File còn gọi là tệp, tập tin. File là tập hợp của các thông tin được đặt tên và lưu trữ.

Khi muốn đọc hoặc ghi file, phải mở file. Khi hoàn thành, file phải đóng lại để giải phóng tài nguyên.

Trong Python, thao tác với file diễn ra theo thứ tự sau.

1. Mở tập tin
2. Đọc hoặc ghi
3. Đóng tập tin

Mở File

Hàm mở file: **hàm *open()***. Hàm này trả về đối tượng file còn gọi là "handle" thực hiện đọc, ghi, sửa đổi trên file.

```
>>> f = open("test.txt") # mở file cùng thư mục với file
>>> f = open("C:/Python33/README.txt") # mở file ở thư mục khác, có đường dẫn
```

Chế độ mở tập tin để làm việc: **read**, **write**, **append**,... đây là chế độ tùy chọn.

File có thể mở dạng văn bản hay dạng nhị phân.

Chế độ truy cập file mặc định là **read** (r). Khi dùng mode này giá trị chuỗi trả về dạng văn bản.

Mở File

Nếu giá trị trả về dạng byte thì file được mở ra là ảnh hoặc exe.
Danh sách các chế độ (mode) khi mở file:

MODE	MÔ TẢ
‘r’	Chế độ chỉ được phép đọc.
‘r+’	Chế độ được phép đọc và ghi
‘rb’	Mở file chế độ đọc cho định dạng nhị phân. Con trỏ tại phần bắt đầu của file
‘rb+’ ‘r+b’	Mở file để đọc và ghi trong định dạng nhị phân. Con trỏ tại phần bắt đầu của file
‘w’	Mở file để ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ

Mở File

‘w+’	Mở file để đọc và ghi. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘wb’	Mở file để ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘wb+’ ‘w+b’	Mở file để đọc và ghi cho dạng nhị phân. Nếu file không tồn tại thì sẽ tạo mới file và ghi nội dung, nếu file đã tồn tại thì sẽ bị cắt bớt (truncate) và ghi đè lên nội dung cũ
‘a’	Mở file chế độ ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.

Mở File

‘a+’	Mở file chế độ đọc và ghi tiếp. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘ab’,	Mở file chế độ ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘ab+’ ‘a+b	Mở file chế độ đọc và ghi tiếp ở dạng nhị phân. Nếu file đã tồn tại rồi thì nó sẽ ghi tiếp nội dung vào cuối file, nếu file không tồn tại thì tạo một file mới và ghi nội dung vào đó.
‘x’	Mở file chế độ ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi

Mở File

‘x+’	Mở file chế độ đọc và ghi. Tạo file độc quyền mới (exclusive creation) và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘xb’	Mở file chế độ ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘xb+’ ‘x+b’	Mở file chế độ đọc và ghi dạng nhị phân. Tạo file độc quyền mới và ghi nội dung, nếu file đã tồn tại thì chương trình sẽ báo lỗi
‘b’	Mở file ở chế độ nhị phân
‘t’	Mở file ở chế độ văn bản (mặc định)

Mở File

```
>>> f = open("test.txt") # mở file mode 'r' hoặc 'rt' để đọc
>>> f = open("test.txt",'w') # mở file mode ‘w’ để ghi
>>> f = open("img.bmp",'r+b') # mở file mode ‘r+b’ để đọc và ghi dạng nhị phân
```

Khi làm việc ở chế độ văn bản, nên chỉ định loại mã hóa.

```
>>> f = open("test.txt",mode = 'r',encoding = 'utf-8')
```

Đóng File

Sau khi thực hiện xong các thao tác cần đóng file lại.

Đóng file để đảm bảo quy định đóng mở và giải phóng bộ nhớ cho chương trình.

Đóng file sử dụng **hàm *close()***.

Python sẽ tự động đóng file khi đối tượng tham chiếu của file đã được gán cho một file khác. Tuy nhiên, nên sử dụng **hàm *close()*** để đóng file.

```
>>> f = open("test.txt",encoding = 'utf-8')
    # thực hiện các thao tác với file
    f.close()
```

Đóng File

Trường hợp ngoại lệ, xảy ra khi thực hiện các thao tác với file khiến chương trình tự động thoát ra mà không đóng file.
Để đảm bảo đóng file nên sử dụng khối lệnh try...finally (**finally** sẽ luôn được thi hành bất chấp có ngoại lệ hay không).

```
>>> try:
        f = open("test.txt",encoding = 'utf-8')
        # thực hiện các thao tác với file
    finally:
        f.close()
```

Đóng File

Cách khác để đóng file là sử dụng lệnh ***with***. Lệnh ***with*** bảo đảm file luôn được đóng.

```
>>> with open("test.txt",encoding = 'utf-8') as f:
    # thực hiện các thao tác với file
```

So sánh hai cách đóng file thì cách sử dụng ***with*** viết ngắn gọn hơn.

Ghi File

Để ghi file sử dụng mode write 'w', append 'a' hoặc mode tạo độc quyền 'x'
Với chế độ 'w' cần cẩn thận, vì sẽ ghi đè.

File dạng nhị phân, nếu ghi vào các chuỗi văn bản, chuỗi dạng byte, thì kết quả là kí tự.

```
>>> with open("test.txt",'w',encoding = 'utf-8') as f:
        f.write("KTLT\n")
        f.write("CTDL + Thuật toán = chương_trình\n\n")
        f.write("KTLT nâng cao\n")
```

Chương trình sẽ tạo file có tên là *test.txt* nếu file chưa tồn tại, nếu có rồi sẽ bị ghi đè.

Ghi File

Sử dụng kí tự *'\n'* để xuống dòng.
Kết quả:

```
KTLT
CTDL + Thuật toán = chương_trình
KTLT nâng cao
```

Đọc File

Đọc file: mở file bằng mode read ‘r’.

Phương thức *read(size)* lấy dữ liệu có kích thước size. Nếu size để trống sẽ đọc hết file, file quá lớn sẽ đọc theo giới hạn của bộ nhớ.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
    a = f.read(5) # đọc 5 kí tự đầu tiên
    print('Nội dung 4 kí tự đầu là:\n', (a))
    b = f.read(34) # đọc 35 kí tự tiếp theo
    print('Nội dung 34 kí tự tiếp theo là:\n', (b))
    c = f.read() # đọc phần còn lại
    print('Nội dung phần còn lại là:\n', (c))
```

Đọc File

Kết quả:

```
    Nội dung 5 kí tự đầu là:
    KTLT
    Nội dung 34 kí tự tiếp theo là:
    CTDL + Thuật toán = chương_trình
    Nội dung phần còn lại là:
     KTLT nâng cao
```

Đọc File

Phương thức *tell()* cho biết vị trí trong file.
Phương thức *seek()* thay đổi vị trí trong file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
    a = f.read(5) # đọc 5 kí tự đầu tiên
    print('Nội dung là: \n', (a))
    b = f.tell() # Kiểm tra vị trí hiện tại
    print ('Vị trí hiện tại: ', (b))
    f.seek(0) # Đặt lại vị trí con trỏ tại vị trí đầu file
    c = f.read()
    print('Nội dung mới là: \n', (c))
```

Đọc File

Kết quả:

```
>>> Nội dung là:
    KTLT
    Vị trí hiện tại: 6
    Nội dung mới là:
    KTLT
    CTDL + Thuật toán = chương_trình
    KTLT nâng cao
```

Đọc File

readline(): đọc từng dòng trong file:

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
    a = f.readline()
    print ('Nội dung dòng đầu: ', (a))
    b = f.readline()
    print ('Nội dung dòng 2: ', (b))
    c = f.readline()
    print ('Nội dung dòng 3: ', (c))
    d = f.readline()
    print ('Nội dung dòng 4: ', (d))
```

Đọc File

Kết quả:

```
>>> Nội dung dòng đầu: KTLT
    Nội dung dòng 2: CTDL + Thuật toán =
chương_trình
    Nội dung dòng 3:
    Nội dung dòng 4: KTLT nâng cao
```

Đọc File

Phương thức *readlines()* trả về toàn bộ các dòng còn lại trong file và trả về giá trị rỗng khi kết thúc file.

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
    a = f.readline()
    print ('Nội dung dòng đầu: ', (a))
    b = f.readlines()
    print ('Nội dung các dòng còn lại: \n', (b))
    c = f.readlines()
    print ('Nội dung các dòng còn lại: \n', (c))
```

Đọc File

Kết quả:

```
>>> Nội dung dòng đầu: KTLT
    Nội dung các dòng còn lại:
    ['CTDL + Thuật toán = chương_trình\n', '\n', 'KTLT
nâng cao\n']
    Nội dung các dòng còn lại:
    [ ]
```

Phương thức làm việc với File

PHƯƠNG THỨC	MÔ TẢ
close()	Đóng một file đang mở. Nó không thực thi được nếu tập tin đã bị đóng.
fileno()	Trả về một số nguyên mô tả file (file descriptor).
flush()	Xóa sạch bộ nhớ đệm của luồng file.
isatty()	Trả về TRUE nếu file được kết nối với một thiết bị đầu cuối.
read(n)	Đọc n kí tự trong file.
readable()	Trả về TRUE nếu file có thể đọc được.
readline(n=-1)	Đọc và trả về một dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.

Phương thức làm việc với File

readlines(n=-1)	Đọc và trả về một danh sách các dòng từ file. Đọc nhiều nhất n byte/ký tự nếu được chỉ định.
seek(offset,from=SEEK_SET)	Thay đổi vị trí hiện tại bên trong file.
seekable()	Trả về TRUE nếu luồng hỗ trợ truy cập ngẫu nhiên.
tell()	Trả về vị trí hiện tại bên trong file.
truncate(size=None)	Cắt gọn kích cỡ file thành kích cỡ tham số size.
writable()	Trả về TRUE nếu file có thể ghi được.
write(s)	Ghi s kí tự vào trong file và trả về.
writelines(lines)	Ghi một danh sách các dòng và file.

Quản lý thư mục và tập tin

Thư mục là nơi lưu trữ các file.
Các phương thức xử lý các hoạt động liên quan tới thư mục trong **Module os**

Module os có các phương thức tạo, xóa, thay đổi thư mục.

Hiển thị thư mục hiện tại

Phương thức *getcwd()* hiển thị thư mục hiện tại, trả về kết quả dạng chuỗi.
Phương thức *getcwdb()* trả về kết quả dạng byte.

```
>>> import os
>>> os.getcwd()
    'C:\\Program Files\\PyScripter'
>>> os.getcwdb()
    b'C:\\Program Files\\PyScripter'
```

Thay đổi thư mục hiện tại

Phương thức *chdir()*: thay đổi Thư mục hiện tại.

chdir() nhận tham số là tên thư mục muốn tới từ thư mục hiện hành. Dấu gạch chéo (/) hoặc dấu gạch chéo ngược (\) dùng để phân chia các phần tử trong đường dẫn.

```
>>> os.chdir('C:\\Python33')
>>> print(os.getcwd())
    C:\Python33
```

Danh sách thư mục và file

Phương thức *listdir()*: liệt kê các file và thư mục con.

Phương thức này trả về danh sách các thư mục con và các file có trong thư mục.

Nếu không có đường dẫn chỉ định, kết quả trả về sẽ truy xuất từ thư mục hiện hành.

```
>>> print(os.getcwd())
    C:\Python33
>>> os.listdir()
    ['DLLs', 'Doc', 'include', 'Lib', 'libs', 'LICENSE.txt',
'NEWS.txt', 'python.exe', 'pythonw.exe', 'README.txt',
'Scripts', 'tcl', 'Tools']
>>> os.listdir('G:\\')
['$RECYCLE.BIN', 'Movies', 'Music', 'Photos', 'Series']
```

Tạo một thư mục mới

Phương thức *mkdir():* tạo thư mục mới,
Chọn nơi chứa thư mục (nhập đường dẫn tới nơi muốn tạo). Nếu không chỉ định đường dẫn, thư mục mới sẽ được tạo trong thư mục hiện hành.

```
>>> os.mkdir('test')
>>> os.listdir()
      ['test']
```

Đổi tên thư mục hoặc tên tập tin

Phương thức *rename():* đổi tên thư mục hoặc tập tin.

```
>>> os.listdir()
      ['test']
>>> os.rename('test','new_one')
>>> os.listdir()
      ['new_one']
```

Xóa thư mục hoặc tập tin

Phương thức *remove():* Xóa tập tin (file)

Phương thức *rmdir():* Xóa thư mục

```
>>> os.listdir()
    ['new_one', 'old.txt']
>>> os.remove('old.txt')
>>> os.listdir()
    ['new_one']
>>> os.rmdir('new_one')
>>> os.listdir()
    [ ]
```

Lưu ý: phương thức ***rmdir()*** chỉ xóa các thư mục rỗng.

Xóa thư mục hoặc tập tin

Phương thức *rmtree*() trong **module *shutil*** *dùng* xóa thư mục không rỗng.

```
>>> os.listdir()
    ['test']
>>> os.rmdir('test')
    Traceback (most recent call last):
    ...
OSError: [WinError 145] The directory is not empty:
'test'
>>> import shutil
>>> shutil.rmtree('test')
>>> os.listdir()
    [ ]
```

Error và Exception

Python thường sinh ra các ngoại lệ (exception) khi có lỗi xảy ra trong quá trình thực thi.

Lỗi (Error)

Lỗi do không theo cấu trúc của cú pháp gọi là lỗi cú pháp (*syntax error* hoặc *parsing error*).

```
>>> if a < 3
      File "<interactive input>", line 1
      if a < 3
      ^
      SyntaxError: invalid syntax
```

Lệnh trên thiếu dấu hai chấm trong câu lệnh IF lập tức chương trình báo lỗi cú pháp *invalid syntax*

Ngoại lệ (Exception)

Lỗi phát sinh khi đang thực thi chương trình (*runtime error*) được gọi là ngoại lệ (Exception).

Ngoại lệ được Python tạo ra để xử lý vấn đề, tránh cho chương trình bị hỏng. Ngoại lệ (Exception):

- Mở một tệp không tồn tại *(FileNotFoundError)*
- Chia một số cho 0 *(ZeroDivisionError)*
- Không tìm thấy module được import *(ImportError).*
- Truyền giá trị vào hàm, đúng kiểu dữ liệu nhưng giá trị không thích hợp *(ValueError).*

Bất cứ khi nào có *runtime error* xảy ra, Python sẽ tạo một ngoại lệ.

Ngoại lệ (Exception)

Chương trình báo lỗi và chi tiết về lý do lỗi xảy ra.

```
>>> 1/0
    Traceback (most recent call last):
    File "<string>", line 301, in runcode
    File "<interactive input>", line 1, in <module>
    ZeroDivisionError: division by zero
>>> open("imaginary.txt")
    Traceback (most recent call last):
    File "<string>", line 301, in runcode
    File "<interactive input>", line 1, in <module>
    FileNotFoundError: [Errno 2] No such file or
directory:
```

Ngoại lệ được Python tạo ra

Có nhiều ngoại lệ được Python tạo ra khi gặp lỗi tương ứng.

Xem Exception có sẵn bằng hàm local():

```
locals()['__builtins__']
```

Hàm này trả về danh sách các ngoại lệ, chức năng và thuộc tính được Python tích hợp sẵn.

Một số ngoại lệ được xây dựng sẵn trong Python cùng với lỗi gây ra :

Ngoại lệ được Python tạo ra

Ngoại lệ	Lý do gây ra
AssertionError	Xảy ra khi câu lệnh assert thất bại.
AttributeError	Xảy ra khi gán thuộc tính hoặc tham chiếu thất bại
EOFError	Xảy ra khi hàm input () chạm vào điều kiện end-of-file.
FloatingPointError	Xảy ra khi một số thực dấy phẩy động thực thi không thành công
GeneratorExit	Xảy ra khi phương thức close() của hàm generator được gọi.
ImportError	Xảy ra khi không tìm thấy module được import.
IndexError	Xảy ra khi một chỉ số trong chuỗi (sequence) nằm ngoài phạm vi.

Ngoại lệ được Python tạo ra

KeyError	Xảy ra khi không tìm thấy khóa ánh xạ (từ điển) trong tập hợp các khóa hiện có.
KeyboardInterrupt	Xảy ra khi người dùng nhấn phím ngắt (thông thường là Ctrl-C hoặc Delete).
MemoryError	Xảy ra khi một operation hết bộ nhớ nhưng tình huống vẫn có thể được sửa chữa (bằng cách xóa một số đối tượng).
NameError	Xảy ra khi không tìm thấy tên cục bộ hoặc toàn cầu của biến.
NotImplementedError	Xảy ra bằng các phương thức trừu tượng khi chúng yêu cầu các lớp dẫn xuất ghi đè phương thức.
OSError	Xảy ra khi một hàm trả về lỗi liên quan đến hệ thống

Ngoại lệ được Python tạo ra

OverflowError	Xảy ra khi kết quả của phép toán số học quá lớn không thể biểu diễn.
ReferenceError	Xảy ra khi một proxy tham chiếu yếu sử dụng để truy cập một thuộc tính của tham chiếu sau khi thu thập rác.
RuntimeError	Xảy ra khi phát hiện thấy lỗi không thuộc bất kỳ danh mục nào khác.
StopIteration	Xảy ra bằng phương thức next() của một vòng lặp để báo hiệu rằng không có giá trị nào được trả về bởi iterator
SyntaxError	Xảy ra khi gặp lỗi cú pháp.
IndentationError	Xảy ra khi có lỗi thụt lề không chính xác.

Ngoại lệ được Python tạo ra

TabError	Xảy ra khi thụt lề sử dụng các tab và dấu cách không nhất quán.
SystemError	Xảy ra khi trình thông dịch tìm thấy các lỗi nội bộ nhưng tình hình không quá nghiêm trọng.
SystemExit	Xảy ra bởi hàm sys.exit().
TypeError	Xảy ra khi một hàm hoặc phép thực thi (operation) áp dụng kiểu không chính xác cho một đối tượng.
UnboundLocalError	Xảy ra khi tham chiếu tạo thành một biến cục bộ trong một hàm hoặc phương thức, nhưng không có giá trị nào bị ràng buộc với biến đó.

Ngoại lệ được Python tạo ra

UnicodeError	Xảy ra khi có lỗi liên quan đến Unicode
UnicodeEncodeError	Xảy ra khi lỗi liên quan đến Unicode diễn ra trong quá trình mã hóa.
UnicodeDecodeError	Xảy ra khi lỗi liên quan đến Unicode diễn ra trong quá trình giải mã.
UnicodeTranslateError	Xảy ra khi lỗi liên quan đến Unicode trong quá trình dịch.
ValueError	Xảy ra khi một phép toán hoặc hàm nhận được một đối số có kiểu đúng nhưng giá trị không phù hợp
ZeroDivisionError	Xảy ra khi đối số thứ hai của phép chia hoặc phép toán modulo bằng 0

Xử lý ngoại lệ - Exception Handling

Xử lý các ngoại lệ trong Python bằng cách sử dụng các câu lệnh **Try**, **Except** và **Finally**.

Cú pháp:

```
try:
# khối code lệnh try
except exceptionName:
# khối code lệnh except
```

Phần thân của ***try*** gồm lệnh có thể tạo ra **exception**, khi có exception, các câu lệnh trong khối sẽ bị bỏ qua.

Phần thân của ***except*** được gọi bởi exception handler, dùng để bắt lỗi khi lỗi xãy ra, nếu không sẽ được bỏ qua. Dùng exception có sẵn trong Thư viện Python.

Xử lý ngoại lệ

exceptionName là tên của exception có thể xảy ra.
try có thể có nhiều ***except***.
Khi khối lệnh *try* có 1 lỗi xảy ra, chương trình sẽ tìm đến các *except* phía dưới, *except* nào thỏa mãn thì sẽ thực thi lệnh trong khối *except* đó.

```
>>> try :
        # khối lệnh try
        except ValueError:
        # lệnh xử lý ValueError
        except RuntimeError:
        # lệnh xử lý RuntimeError
```

Xử lý ngoại lệ

```
>>> # import module sys để gọi các ngoại lệ
        Import sys
        randomList = ['a', 0, 2]
        for nhap in randomList:
            try:
                print("Phần tử:", nhap)
                r = 1/int(nhap)
                break
            except:
                print("Có ngoại lệ ",sys.exc_info()[0],“ xảy ra.")
            print("Nhập phần tử tiếp theo")
            print()
        print("Kết quả với phần tử ",nhap,"là:",r)
```

Xử lý ngoại lệ

Kết quả:

```
>>> Phần tử: a
    Có ngoại lệ <class 'ValueError'> xảy ra.
    Nhập phần tử tiếp theo
    Phần tử: 0
    Có ngoại lệ <class 'ZeroDivisionError'> xảy ra.
    Nhập phần tử tiếp theo
    Phần tử: 2
    Kết quả với phần tử 2 là: 0.5
```

Chương trình sẽ thực thi đến khi số được nhập là số nguyên.

Xử lý ngoại lệ

Nếu không có ngoại lệ xảy ra, khối *except* sẽ được bỏ qua và chương trình thực hiện bình thường, nhưng nếu có bất cứ ngoại lệ nào nó sẽ bị chặn lại bởi *except*.

Chương trình cũng in tên của exception bằng hàm *ex_info()* bên trong module *sys,* kết quả trả về là giá trị 'a' gây ra *ValueError* và 0 gây ra *ZeroDivisionError*

Xử lý ngoại lệ

Có thể đặt nhiều exception trong một lần khai báo *except* bằng cách đặt các ngoại lệ cách nhau bởi dấu phẩy ‘,’

```
>>> try :
        # khối lệnh try
        except (TypeError, ZeroDivisionError):
        # lệnh xử lý nhiều ngoại lệ
        # TypeError, ZeroDivisionError
```

Xây dựng ngoại lệ (Exception)

Dùng *raise* để xử lí ngoại lệ (exception). Có thể tạo exception riêng khi vấn đề nằm ngoài phạm vi dự kiến lỗi xảy ra.

```
>>> try:
        x = input('Nhập một số trong khoảng 1-10: ')
        if x<1 or x>10:
            raise Exception
            print (‘Số nhập hợp lệ :’, x)
    except:
        print ('Số vừa nhập nằm ngoài khoảng cho phép’)
```

Trong ví dụ trên, nếu nhập một số ngoài phạm vi cho phép, lệnh print trong khối *except* sẽ thực hiện.

Xây dựng ngoại lệ (Exception)

Module traceback dùng để in lỗi của chương trình khi exception xảy ra. Traceback gồm thông báo lỗi, dòng gây ra lỗi và call stack của hàm gây ra lỗi.

```
>>> raise KeyboardInterrupt
    Traceback (most recent call last):
    File "<pyshell#0>", line 1, in <module>
    raise KeyboardInterrupt
    KeyboardInterrupt
>>> raise MemoryError("Đây là một tham số.")
    Traceback (most recent call last):
    File "<pyshell#1>", line 1, in <module>
    raise MemoryError("Đây là một tham số.")
    MemoryError: Đây là một tham số.
```

Xây dựng ngoại lệ (Exception)

```
>>> try:
... a = int(input("Nhập một số nguyên dương: "))
... if a <= 0:
... raise ValueError("Số vừa nhập không phải số
nguyên dương”)
... except ValueError as ve:
... print(ve)
...
Nhập một số nguyên dương: -2
Số vừa nhập không phải số nguyên dương
```

Try…Finally

Try...finally

Finally là mệnh đề *clean-up/termination* vì luôn luôn chạy bất kể có lỗi xảy ra trong khối **try**.

Câu lệnh trong ***finally*** thường được dùng để thực hiện giải phóng tài nguyên.

Sau khi thực hiện xong các thao tác với file thì cần đóng lại. Đóng file để đảm bảo quy trình đóng mở và giải phóng bộ nhớ cho chương trình.

Try…Finally

```
>>> try:
        f = open("test.txt",encoding = 'utf-8')
        # thực hiện các thao tác với file
    finally:
        f.close()
```

File sẻ được đóng ngay cả khi phát sinh ngoại lệ khiến chương trình dừng đột ngột.

```
>>> mauso = input (“Nhập giá trị mẫu số:  ”)
    try:
        ketqua = 15/int(mauso)
        print("Kết quả là:",ketqua)
    finally:
        print(“Số đã nhập không thể thực hiện được.")
```

Try...Finally

Finally luôn chạy bất kể có lỗi xảy ra hay không.
Nhập input là 5, kết quả:

```
>>> Nhập giá trị mẫu số: 5
    Kết quả là: 3.0
    Số đã nhập không thể thực hiện được.
```

Khi input là 0, kết quả:

```
>>> Nhập giá trị mẫu số: 0
    Số đã nhập không thể thực hiện được.
```

Bài tập

1. Đưa ra một RuntimeError exception.
Gợi ý: Sử dụng raise() để đưa ra exception.
2. Viết hàm để tính 5/0 và sử dụng try/exception để bắt lỗi.
3. Định nghĩa một class exception tùy chỉnh, nhận một thông báo là thuộc tính.
(**định nghĩa một class kế thừa từ Exception**)