**Sử dụng logging trong Python**

Thói quen của mình và có lẽ cũng là thói quen của nhiều người lúc mới học Python cũng như khi viết những script nhỏ để lấy thông tin chương trình lúc runtime đó là sử dụng print(), print() ở bất cứ đâu và bất cứ khi nào. Thường thì chương trình sẽ có nhiều lệnh print như thế này:

try:

print("Find PID in current url")

wait.until(lambda mydriver: "{}".format(PID) in mydriver.current\_url)

except:

print("Could not find PID in url")

print("Current url: {}".format(mydriver.current\_url))

mydriver.find\_element\_by\_xpath('//select[@id="aex-view-3"]/option[text()="Vietnam"]').click()

mydriver.find\_element\_by\_xpath('//button[@id="create-profile-submit-btn"]').click()

Tuy nhiên khi chương trình lớn dần lên, một vấn đề khá đau đầu nảy sinh đó là thông tin được print ra khá điên loạn, không được phân loại, mất thời gian xác định cũng như giảm hiệu suất của chương trình. Hàm print mặc định sẽ xuất nội dung ra stdout và nội dung log in ra thường rất hay bị nhầm với dữ liệu khác.

Giải pháp mình muốn giới thiệu trong bài viết này đó là sử dụng module logging. Đây là module tiêu chuẩn (được pack kèm bộ cài mặc định) do cộng đồng Python phát triển, dễ dàng sử dụng và cực kỳ linh hoạt. Logging cung cấp cho lập trình viên những tiện ích như:

* Phân chia level cho các thông báo lỗi, cho phép tuỳ chọn mức độ nghiêm trọng của các thông báo, cho phép hoặc không cho phép hiển thị một thông báo đã được phân loại.
* Cho phép cấu hình output của thông báo lỗi là trên console hoặc file hoặc [nguồn khác](https://docs.python.org/3/library/logging.handlers.html)

Lập trình viên có thể chọn các mức sau khi viết nội dung log, các mức được liệt kê có độ nghiêm trọng lớn dần:

| **Mức** | **Nội dung** |
| --- | --- |
| **DEBUG** | Thông tin chi tiết, thường là thông tin để tìm lỗi |
| **INFO** | Thông báo thông thường, các thông tin in ra khi chương trình chạy theo đúng kịch bản |
| **WARNING** | Thông báo khi nghi vấn bất thường hoặc lỗi có thể xảy ra, tuy nhiên chương trình vẫn có thể hoạt động |
| **ERROR** | Lỗi, chương trình có thể không hoạt động được một số chức năng hoặc nhiệm vụ nào đó, thường thì nên dùng ghi bắt được Exception |
| **CRITICAL** | Lỗi, chương trình gặp lỗi nghiêm trọng không thể giải quyết được và bắt buộc phải dừng lại |

**Sử dụng module logging cơ bản**

Ví dụ đơn giản nhất khi sử dụng logging:

# source: https://docs.python.org/3.6/howto/logging.html#a-simple-example

import logging

logging.warning('Watch out!') # sẽ in ra warning trên console

logging.info('I told you so') # không in gì cả

Mặc định, mức thông báo sẽ được đặt ở mức WARNING. Các thông báo có mức dưới nó (DEBUG, INFO) vì vậy sẽ không được in ra. Để thay đổi mức này, bạn đơn giản chỉ việc set lại thông số logging level như sau.

import logging

logging.basicConfig(level=logging.DEBUG)

logging.warning('Watch out!') # sẽ in ra warning trên console

logging.info('I told you so') # sẽ in ra warning trên console

Bạn cũng có thể log ra file:

import logging

logging.basicConfig(filename='example.log',level=logging.DEBUG)

logging.debug('This message should go to the log file')

logging.info('So should this')

logging.warning('And this, too')

Nội dung được in ra sẽ tương tự khi bạn in ra console:

DEBUG:root:This message should go to the log file

INFO:root:So should this

WARNING:root:And this, too

Cần chú ý là nội dung log mới sẽ ghi nối tiếp vào cùng 1 file với nội dung log cũ (lần chạy trước đó), trong trường hợp nội dung cũ không còn quan trọng nữa, có thể setup để logging luôn tạo log mới bằng cách chỉnh chế độ filemode khi config:

logging.basicConfig(level=logging.DEBUG, filename='runtime.log', filemode='w')

Một ví dụ phức tạp hơn về việc dùng hỗn hợp các log với mức độ khác nhau:

# source: https://code.tutsplus.com/tutorials/error-handling-logging-in-python--cms-27932

import logging

# initialize the log settings

logging.basicConfig(filename='app.log',level=logging.INFO)

try:

logging.info('Trying to open the file')

filePointer = open('appFile','r')

try:

logging.info('Trying to read the file content')

content = filePointer.readline()

finally:

filePointer.close()

except IOError as e:

logging.error('Error occurred ' + str(e))

**Log ra nhiều nguồn cùng lúc**

Việc sử dụng logger cơ bản chỉ cho phép chúng ta in log ra một nguồn tại một thời điểm, trong ví dụ trên, khi đặt filename='app.log'thì console sẽ không in nội dung log ra nữa. Một cách cho phép lập trình viên thực hiện việc in log ra nhiều nguồn khác nhau đó là sử dụng handles, tham khảo ví dụ in log ra FileHandler sau:

# source: https://www.smartfile.com/blog/what-is-good-logging-in-python/

import logging

# create the logging instance for logging to file only

logger = logging.getLogger('SmartfileTest')

# create the handler for the main logger

file\_logger = logging.FileHandler('smartfile\_test.log')

NEW\_FORMAT = '[%(asctime)s] - [%(levelname)s] - %(message)s'

file\_logger\_format = logging.Formatter(NEW\_FORMAT)

# tell the handler to use the above format

file\_logger.setFormatter(file\_logger\_format)

# finally, add the handler to the base logger

logger.addHandler(file\_logger)

# remember that by default, logging will start at 'warning' unless

# we set it manually

logger.setLevel(logging.DEBUG)

# log some stuff!

logger.debug("This is a debug message!")

logger.info("This is an info message!")

logger.warning("This is a warning message!")

Đoạn code trên sẽ tạo file log và in nội dung log vào file đó tương tự như việc bổ xung config filename='smartfile\_test.log', để in đồng thời ra console, bạn add thêm StreamHandler như sau:

# source: https://www.smartfile.com/blog/what-is-good-logging-in-python/

# now we can add the console logging

console = logging.StreamHandler()

console.setLevel(logging.INFO)

logging.getLogger('SmartfileTest').addHandler(console)

Ngoài FileHandler và StreamHandler, logging còn hỗ trợ rất nhiều handles khác nữa. [Chi tiết ở đây](https://docs.python.org/3/library/logging.handlers.html).

**In nội dung biến**

Để in nội dung biến, bạn chỉ cần đưa chuỗi format vào sau đó truyền các biến tương ứng với format trong đối số. Xem ví dụ sau để hiểu rõ hơn.

>>> import logging

>>> logString = "Hi"

>>> logging.warning('%s, this i a %s', logString, 'warning!')

WARNING:root:Hi, this i a warning!

Chú ý là bạn không cần format chuỗi trước khi truyền vào hàm logging.warning() vì logging sẽ tự làm việc này. Có nghĩa là việc bạn viết: logging.warning('%s, this i a %s' % (logString, 'warning!')) hoàn toàn thừa thãi.

**Format nội dung và thêm các thông tin chi tiết về log**

Logging cho phép thêm các thông tin chi tiết hơn về chương trình khi gọi log như: Vị trí gọi log (dòng code), tên module gọi log, file đang chạy ...

Một config đầy đủ có thể như sau:

import logging

logging.basicConfig(level=logging.DEBUG, filename='runtime.log', filemode='w', format = (

'%(levelname)s:\t'

'%(filename)s:'

'%(funcName)s():'

'%(lineno)d\t'

'%(message)s'

)

)

Thông tin in ra sẽ như sau:

INFO: main.py:main():113 Hi, this is a log text

DEBUG: main.py:main():281 Processing file something.txt

WARNING: main.py:main():294 File too big something.txt

Việc phân loại chi tiết như trên rất hữu ích khi bạn phải phân tích các file log dài, khi đó bạn có thể dùng các chương trình soạn thảo code để filter hoặc search các thông tin tương ứng một cách nhanh chóng.

**Logging trong exception (in thông tin Traceback)**

Việc ghi log ra khi có exception là rất cần thiết, tuy nhiên ghi log ra mà không có thông tin chi tiết về exception đó thì không có ý nghĩa lắm. Để bổ xung thông tin khi bắt được exception chúng ta chỉ cần thêm config exc\_info=True

# source: fangpenlin.com/posts/2012/08/26/good-logging-practice-in-python#capture-exceptions-and-record-them-with-traceback

try:

open('/path/to/does/not/exist', 'rb')

except (SystemExit, KeyboardInterrupt):

raise

except Exception, e:

logger.error('Failed to open file', exc\_info=True)

Thông tin in ra:

ERROR:\_\_main\_\_:Failed to open file

Traceback (most recent call last):

File "example.py", line 6, in <module>

open('/path/to/does/not/exist', 'rb')

IOError: [Errno 2] No such file or directory: '/path/to/does/not/exist'

Bạn cũng có thể gọi logger.exception(msg, \*args), có chức năng tương đương với logger.error(msg, exc\_info=True, \*args).

**Sử dụng logging trong nhiều module**

[chi tiết](https://docs.python.org/3.6/howto/logging.html#logging-from-multiple-modules)

Nếu phần mềm của bạn có nhiều modules thì đây là một cách đơn giản để setup hệ thống logging xuyên suốt cả phần mềm:

# myapp.py

# source: https://docs.python.org/3.6/howto/logging.html#logging-from-multiple-modules

import logging

import mylib

def main():

logging.basicConfig(filename='myapp.log', level=logging.INFO)

logging.info('Started')

mylib.do\_something()

logging.info('Finished')

if \_\_name\_\_ == '\_\_main\_\_':

main()

Trong module:

# mylib.py

import logging

def do\_something():

logging.info('Doing something')

Kết quả:

INFO:root:Started

INFO:root:Doing something

INFO:root:Finished

Kết hợp với việc format nội dung logging như đã nêu trước đó, bạn có thể in ra thông tin log được gọi từ module nào.

except:

log.error(traceback.format\_exc())

**Giúp cho ta phát hiện lỗi ở chính xác dòng nào trong chương trình**